Generative AI and Large Language Models for Cyber Security: All Insights You Need

Mohamed Amine Ferrag, Fatima Alwahedi, Ammar Battah, Bilel Cherif, Abdechakour Mechri, and Norbert Tihanyi

***Abstract*—This paper provides a comprehensive review of the future of cybersecurity through Generative AI and Large Language Models (LLMs). We explore LLM applications across various domains, including hardware design security, intrusion detection, software engineering, design verification, cyber threat intelligence, malware detection, and phishing detection. We present an overview of LLM evolution and its current state, focusing on advancements in models such as GPT-4, GPT-3.5, Mixtral-8x7B, BERT, Falcon2, and LLaMA. Our analysis extends to LLM vulnerabilities, such as prompt injection, insecure output handling, data poisoning, DDoS attacks, and adversarial instructions. We delve into mitigation strategies to protect these models, providing a comprehensive look at potential attack scenarios and prevention techniques. Furthermore, we evaluate the performance of 42 LLM models in cybersecurity knowledge and hardware security, highlighting their strengths and weaknesses. We thoroughly evaluate cybersecurity datasets for LLM training and testing, covering the lifecycle from data creation to usage and identifying gaps for future research. In addition, we review new strategies for leveraging LLMs, including techniques like Half-Quadratic Quantization (HQQ), Reinforcement Learning with Human Feedback (RLHF), Direct Preference Optimization (DPO), Quantized Low-Rank Adapters (QLoRA), and Retrieval-Augmented Generation (RAG). These insights aim to enhance real-time cybersecurity defenses and improve the sophistication of LLM applications in threat detection and response. Our paper provides a foundational understanding and strategic direction for integrating LLMs into future cybersecurity frameworks, emphasizing innovation and robust model deployment to safeguard against evolving cyber threats.**

***Index Terms*—Generative AI, LLM, Transformer, Security, Cyber Security.**

M. A. Ferrag is the corresponding author.

M. A. Ferrag is with Technology Innovation Institute, 9639 Masdar City, Abu Dhabi, UAE Email: mohamed.ferrag@tii.ae

F. Alwahedi is with Technology Innovation Institute, 9639 Masdar City, Abu Dhabi, UAE Email: fatima.alwahedi@tii.ae

A. Battah is with Technology Innovation Institute, 9639 Masdar City, Abu Dhabi, UAE Email: ammar.battah@tii.ae

B. Cherif is with Technology Innovation Institute, 9639 Masdar City, Abu Dhabi, UAE Email: bilel.cherif@tii.ae

A. Mechri is with Technology Innovation Institute, 9639 Masdar City, Abu Dhabi, UAE Email: abdechakour.mechri@tii.ae

N. Tihanyi is with Technology Innovation Institute, 9639 Masdar City, Abu Dhabi, UAE Email: norbert.tihanyi@tii.ae

List of Abbreviations

AI	Artificial Intelligence
AIGC	Artificial Intelligence Generated Content
APT	Advanced Persistent Threat
CNN	Convolutional Neural Network
CTG	Controllable Text Generation
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
FNN	Feed-Forward Neural Network
FRR	False Refusal Rate
GPT	Generative Pre-trained Transformers
GRU	Gated Recurrent Units
GQA	Grouped-Query Attention
HPC	High-Performance Computing
HLS	High-Level Synthesis Design Verification
HQQ	Half-Quadratic Quantization
IDS	Intrusion Detection System
LLM	Large Language Model
LoRA	Low-rank Adapters
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
MQA	Multi-Query Attention
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NLU	Natural Language Understanding
ORPO	Odds Ratio Preference Optimization
PEFT	Parameter Efficient Fine-Tuning
PLM	Pre-trained Language Model
PPO	Proximal Policy Optimization
RAG	Retrieval Augmentation Generation
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Networks
RTL	Register-Transfer Level
SARD	Software Assurance Reference Dataset
SFT	Supervised Fine-Tuning
SVM	Support Vector Machine
TRPO	Trust Region Policy Optimization

I. Introduction

The history of Natural Language Processing (NLP) dates back to the 1950s when the Turing test was developed. However, NLP has seen significant advancements in recent decades with the introduction of Recurrent Neural Networks (RNN) [1], Long Short-Term Memory (LSTM) [2], Gated Recurrent Units (GRU) [3], and Transformer methods [4]. RNN was first introduced in the 1990s to model data sequences. LSTM, a

variant of RNN, was introduced in 1997, which addressed the vanishing gradient problem and allowed for longer-term memory in NLP models. GRU, another variant of RNN, was introduced in 2014, which reduced the number of parameters and improved computational efficiency [5]. The latest breakthrough in NLP was the introduction of Transformers in 2017, enabling parallel processing of sequential data and revolutionizing tasks like machine translation. These methods have significantly improved various NLP tasks, including sentiment analysis, language generation, and translation [4], [6], [7].

Cybersecurity is an ever-evolving field, with threats becoming increasingly sophisticated and complex. As organizations and individuals rely on digital technologies for communication, commerce, and critical infrastructure, the need for robust cybersecurity measures has never been greater [8]. The scale and diversity of cyber threats make it a daunting challenge for security professionals to effectively identify, detect, and defend against them. In this context, Large Language Models (LLMs) have emerged as a game-changing technology with the potential to enhance cybersecurity practices significantly [9]–[13]. These models, powered by advanced NLP and Machine Learning (ML) techniques, offer a new frontier in the fight against cyber threats [14], [15]. This article explores the motivations and applications of LLMs in cybersecurity.

Cybersecurity professionals often need to sift through a vast amount of textual data, including security alerts, incident reports, threat feeds, and research papers, to stay ahead of evolving threats. LLMs, like Falcon 180b [16], possess natural language understanding capabilities that enable them to parse, summarize, and contextualize this information efficiently [7], [17], [18]. They can assist in rapidly identifying relevant threat intelligence, allowing analysts to make more informed decisions and prioritize responses [19]. LLMs can excel in various domains within cybersecurity. Figure 1 highlights the top nine use cases and applications for LLMs in this field [20].

1) **Threat Detection and Analysis:** LLMs can analyze vast network data in real-time to detect anomalies and potential threats. They can recognize patterns indicative of cyber attacks, such as malware, phishing attempts, and unusual network traffic [19].
2) **Security Automation:** LLMs can facilitate the automation of routine security tasks such as patch management, vulnerability assessments, and compliance checks. This reduces the workload on cybersecurity teams and allows them to focus on more complex tasks [9].
3) **Phishing Detection and Response:** LLMs can identify phishing emails by analyzing the text for malicious intent and comparing it to known phishing examples. They can also generate alerts and recommend preventive actions [21].
4) **Cyber Forensics:** LLMs can help in forensic analysis by parsing through logs and data to determine the cause and method of attack, thus aiding in the recovery process and future prevention strategies [22].
5) **Penetration Testing:** LLMs can help generate scripts or modify existing ones to automate certain parts of the penetration testing process. This includes scripts for vulnerability scanning, network mapping, and exploiting known vulnerabilities [23].
6) **Security Protocols Verification:** LLMs can help verify the security of protocols such as TLS/SSL, IPSec, . . . etc.
7) **Incident Response:** During a cybersecurity incident, LLMs can assist by providing rapid analysis of the situation, suggesting mitigation strategies, and automating responses where applicable [24].
8) **Chatbots:** LLMs significantly enhance the capabilities of chatbots in cybersecurity environments by providing User Interaction, Incident Reporting and Handling, Real-time Assistance, Training and Simulations, and FAQ Automation [25].
9) **Security Training and Awareness:** LLMs can generate training materials tailored to an organization's needs. They can also simulate phishing attacks and other security scenarios to train employees to recognize and respond to security threats [26].

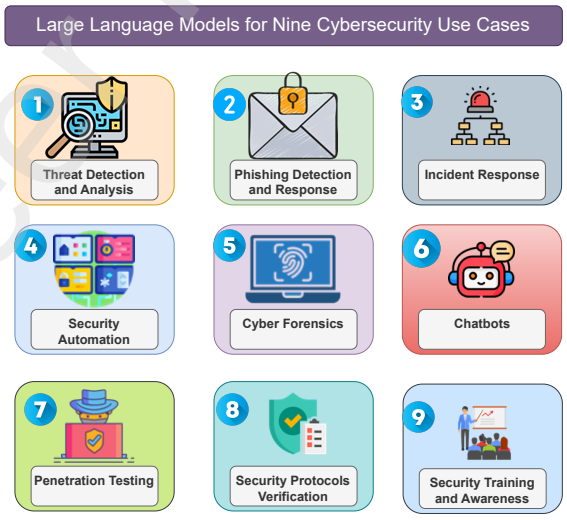

Fig. 1: LLM Use Cases And Applications for Cybersecurity.

The primary aim of this paper is to provide an in-depth and comprehensive review of the future of cybersecurity using Generative AI and LLMs, covering all relevant topics in the cyber domain. The contributions of this study are summarized below:

- We review LLMs' applications for cybersecurity use cases, such as hardware design security, intrusion detection, software engineering, design verification, cyber threat intelligence, malware detection, phishing, and spam detection, etc., providing a nuanced understanding of LLM capabilities across different cybersecurity domains;
- We present a comprehensive overview of LLMs in cybersecurity, detailing their evolution and current state, including advancements in 42 specific models, such as GPT-4o, GPT-4, BERT, Falcon, and LLaMA models;
- We analyze the vulnerabilities associated with LLMs, including prompt injection, insecure output handling, training data poisoning, inference data poisoning, DDoS attacks, and adversarial natural language instructions. We

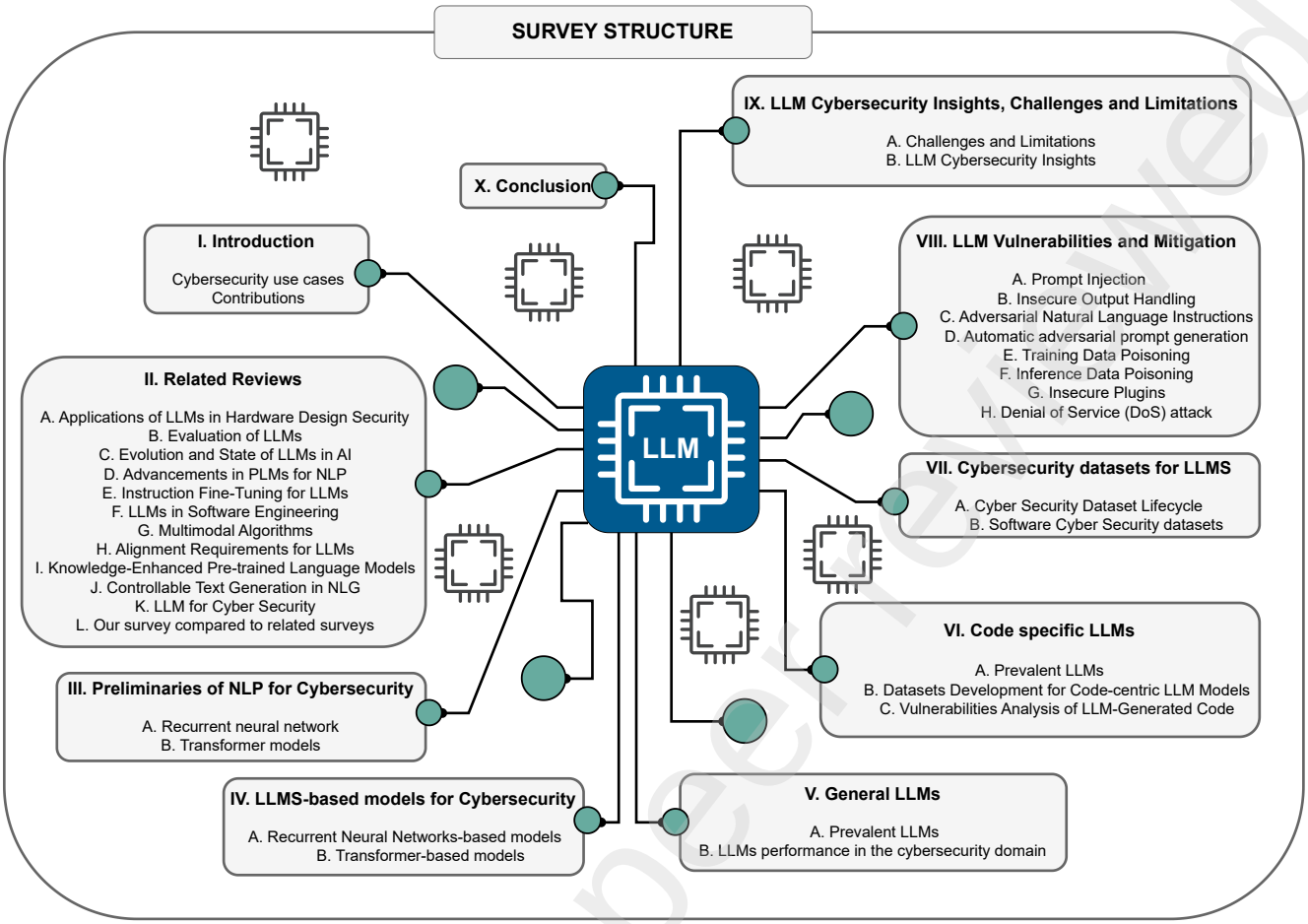

Fig. 2: Survey Structure (From Section I. to Section X.)

also examine the mitigation strategies to safeguard these models from such vulnerabilities, providing a comprehensive look at potential attack scenarios and prevention techniques;

- We evaluated the performance of 42 LLM models in different datasets in the cybersecurity domain.
- We thoroughly evaluate cybersecurity datasets tailored for LLM training and testing. This includes a lifecycle analysis from dataset creation to usage, covering various stages such as data cleaning, preprocessing, annotation, and labeling. We also compare cybersecurity datasets to identify gaps and opportunities for future research;
- We provide the challenges and limitations of employing LLMs in cybersecurity settings, such as dealing with adversarial attacks and ensuring robustness. We also discuss the implications of these challenges for future LLM deployments and the development of secure, optimized models;
- We discuss novel insights and strategies for leveraging LLMs in cybersecurity, including advanced techniques such as Half-Quadratic Quantization (HQQ), Reinforcement Learning with Human Feedback (RLHF), Direct Preference Optimization (DPO), Odds Ratio Preference Optimization (ORPO), GPT-Generated Unified Format (GGUF), Quantized Low-Rank Adapters (QLoRA), and Retrieval-Augmented Generation (RAG). These insights aim to enhance real-time cybersecurity defenses and improve the sophistication of LLM applications in threat detection and response.

The rest of this paper is organized as follows. Section II presents an in-depth analysis of related reviews in the field, charting the evolution and state of LLMs in artificial intelligence. Section III delves into the preliminaries of NLP applications for cybersecurity, covering foundational models and their advancements. Section IV discusses LLM-based solutions specific to cybersecurity. Section V reviews general LLM models. Section VI reviews Code-specific LLMs models. Section VII explores various cybersecurity datasets designed for LLM training and evaluation, detailing their development lifecycle and specific attributes. Section VIII focuses on the vulnerabilities associated with LLMs and the strategies for their mitigation, introducing a classification of potential threats and defense mechanisms. Section IX offers comprehensive insights into the challenges and limitations of integrating LLMs into cybersecurity frameworks, including practical considerations and theoretical constraints. Finally, Section X concludes the paper by summarizing the key findings and proposing directions for future research in LLMs and cybersecurity. A

brief overview of the paper's structure is illustrated in Figure 2.

II. Related reviews

This section delves into a curated collection of recent articles that significantly contribute to the evolving landscape of LLMs and their multifaceted applications. These reviews offer a comprehensive and insightful exploration into various dimensions of LLMs, including their innovative applications in hardware design security, evaluation methodologies, and evolving role in artificial intelligence. Further, they cover cutting-edge advancements in Pre-trained Language Models (PLMs) for NLP, delve into the intricacies of instruction fine-tuning for LLMs, and explore their impactful integration into software engineering. The section also encompasses an in-depth look at multimodal algorithms, examines the critical aspect of alignment requirements for LLMs, and discusses integrating external knowledge into PLMs to enhance NLP tasks. Lastly, it sheds light on the burgeoning field of Controllable Text Generation (CTG) in Natural Language Generation (NLG), highlighting the latest trends and challenges in this dynamic and rapidly advancing area of research [41]–[43].

A. Applications of LLMs in Hardware Design Security

Saha *et al.* [27] discussed several key applications of LLMs in the context of hardware design security. The paper illustrates how LLMs can intentionally introduce vulnerabilities and weaknesses into RTL (Register-Transfer Level) designs. This process is guided by well-crafted prompts in natural language, demonstrating the model's ability to understand and manipulate complex technical designs. The authors explore using LLMs to assess the security of hardware designs. The model is employed to identify vulnerabilities, weaknesses, and potential threats. It's also used to pinpoint simple coding issues that could evolve into significant security bugs, highlighting the model's ability to evaluate technical designs critically. In this application, LLMs verify whether a hardware design adheres to specific security rules or policies. The paper examines the model's proficiency in calculating security metrics, understanding security properties, and generating functional testbenches to detect weaknesses. This part of the study underscores the LLM's ability to conduct thorough and detailed verification processes. Finally, the paper investigates how effectively LLMs can be used to develop countermeasures against existing vulnerabilities in a design. This aspect focuses on the model's capability to solve problems and create solutions to enhance the security of hardware designs. Overall, the paper presents an in-depth analysis of how LLMs can be a powerful tool in various stages of hardware design security, from vulnerability introduction and assessment to verification and countermeasure development.

B. Evaluation of LLMs

Chang *et al.* [26] offers a comprehensive analysis of LLM evaluations, addressing three key aspects: the criteria for evaluation (what to evaluate), the context (where to evaluate), and the methodologies (how to evaluate). It thoroughly reviews various tasks across different domains to understand the successes and failures of LLMs, contributing to future research directions. The paper also discusses current evaluation metrics, datasets, and benchmarks and introduces novel approaches, providing a deep understanding of the current evaluation landscape. Additionally, it highlights future challenges in LLM evaluation and supports the research community by open-sourcing related materials, fostering collaborative advancements in the field.

C. Evolution and State of LLMs in AI

Zhao *et al.* [24] provides an in-depth survey of LLMs' evolution and current state in artificial intelligence. It traces the progression from statistical language models to neural language models, specifically focusing on the recent emergence of pre-trained language models (PLMs) using Transformer models trained on extensive corpora. The paper emphasizes the significant advancements achieved by scaling up these models, noting that LLMs demonstrate remarkable performance improvements beyond a certain threshold and exhibit unique capabilities not found in smaller-scale models. The survey covers four critical aspects of LLMs: pre-training, adaptation tuning, utilization, and capacity evaluation, providing insights into both their technical evolution and the challenges they pose. Additionally, the paper discusses the resources available for LLM development and explores potential future research directions, underlining the transformative effect of LLMs on AI development and application.

D. Advancements in PLMs for NLP

Min *et al.* [28] surveys the latest advancements in leveraging PLMs for NLP, organizing the approaches into three main paradigms. Firstly, the "Pre-train then Fine-tune" method involves general pre-training on large unlabeled datasets followed by specific fine-tuning for targeted NLP tasks. Secondly, "Prompt-based Learning" uses tailored prompts to transform NLP tasks into formats akin to a PLM's pre-training, enhancing the model's performance, especially in few-shot learning scenarios. Lastly, the "NLP as Text Generation" paradigm reimagines NLP tasks as text generation problems, fully capitalizing on the strengths of generative models like GPT-2 and T5. These paradigms represent the cutting-edge methods in utilizing PLMs for various NLP applications.

E. Instruction Fine-Tuning for LLMs

Zhang *et al.* [29] delves into the field of instruction fine-tuning for LLMs, offering a detailed exploration of various facets of this rapidly advancing area. It begins with an overview of the general methodologies used in instruction fine-tuning, then discusses the construction of commonly-used, representative datasets tailored for this approach. The survey highlights a range of instruction-fine-tuned models, showcasing their diversity and capabilities. It also examines multi-modality techniques and datasets, including those involving images, speech, and video, reflecting the broad applicability of instruction tuning. The adaptation of LLMs to different

TABLE I: Summary of Related Reviews on Large Language Models

Focused Area of Study	Year	Authors	Key Points	Data.	Vuln.	Comp.	Optim.	Hardw.
LLMs in Enhancing Hardware Design Security	2023	Saha *et al.* [27]	Discuss applications of LLMs in hardware design security, including vulnerability introduction, assessment, verification, and countermeasure development.	✘	✘	✘	✘	✘
Comprehensive Evaluation Methodologies for LLMs	2023	Chang *et al.* [26]	Provides an analysis of LLM evaluations focusing on criteria, context, methodologies, and future challenges.	✘	✘	✘	✘	✘
The Evolutionary Path of LLMs in AI	2023	Zhao *et al.* [24]	Surveys the evolution of LLMs in AI, focusing on pre-training, adaptation tuning, utilization, and capacity evaluation.	✘	✘	✘	✘	✘
Recent Advancements in PLMs for NLP	2023	Min *et al.* [28]	Reviews advancements in PLMs for NLP, covering paradigms like Pre-train then Fine-tune, Prompt-based Learning, and NLP as Text Generation.	✘	✘	✘	✘	✘
Exploring Instruction Fine-Tuning in LLMs	2023	Zhang *et al.* [29]	Explores instruction fine-tuning for LLMs, covering methodologies, datasets, models, and multi-modality techniques.	✘	✘	✘	✘	✘
Applying LLMs in Software Engineering	2023	Fan *et al.* [30]	Survey the use of LLMs in Software Engineering, discussing applications, challenges, and hybrid approaches.	✘	✘	✘	✘	✘
Understanding Multimodal Algorithms	2023	Wu *et al.* [31]	Provides an overview of multimodal algorithms, covering definition, evolution, technical aspects, and challenges.	✘	✘	✘	✘	✘
Defining Alignment Requirements for LLMs	2023	Liu *et al.* [32]	Proposes a taxonomy of alignment requirements for LLMs and discusses harmful content concepts.	✘	✘	✘	✘	✘
Incorporating External Knowledge in PLMs	2023	Hu *et al.* [33]	Reviews KE-PLMs, focusing on incorporating different types of knowledge into PLMs for NLP.	✘	✘	✘	✘	✘
Advances in Controllable Text Generation	2023	Zhang *et al.* [34]	Reviews CTG in NLG, focusing on Transformer-based PLMs and challenges in controllability.	✘	✘	✘	✘	✘
LLM for Blockchain Security	2024	He *et al.* [35]	Analyze existing research to understand how LLMs can improve blockchain systems' security.	◐	◐	✘	✘	✘
LLM for Critical Infrastructure Protection	2024	Yigit *et al.* [36]	Proposing advanced strategies using Generative AI and Large Language Models to enhance resilience and security.	✘	✘	✘	✘	✘
Software Testing with Large Language Models	2024	Wang *et al.* [37]	Explore how Large Language Models (LLMs) can enhance software testing, examining tasks, techniques, and future research directions.	✘	✘	✘	✘	✘
Malicious Insider Threat Detection Using Machine Learning Methods	2024	Alzaabi *et al.* [22]	Recommends advanced ML methods like deep learning and NLP for better detection and mitigation of insider threats in cybersecurity, emphasizing the need for integrating time-series techniques.	◐	◐	✘	✘	✘
Advancements in Large Language Models	2024	Raiaan *et al.* [25]	Reviews the evolution, architectures, applications, societal impacts, and challenges of LLMs, aiding practitioners, researchers, and experts in understanding their development and prospects.	✘	✘	✘	✘	✘
Applications of LLMs in cybersecurity tasks	2024	Xu *et al.* [38]	Highlights the diverse applications of LLMs in cybersecurity tasks such as vulnerability detection, malware analysis, and intrusion and phishing detection.	◐	◐	✘	✘	✘
Retrieval-Augmented Generation for LLMs	2024	Zhao *et al.* [25]	Reviews how RAG has been integrated into various AIGC scenarios to overcome common challenges such as updating knowledge, handling long-tail data, mitigating data leakage, and managing costs associated with training and inference.	✘	✘	✘	✘	✘
Provides an overview of Parameter Efficient Fine-Tuning (PEFT)	2024	Han *et al.* [39]	Reviews various PEFT algorithms, their effectiveness, and the computational overhead involved.	✘	✘	✘	✘	✘
LLM for Cyber Security	2024	Zhang *et al.* [40]	The paper conducts a systematic literature review of over 180 works on applying LLMs in cybersecurity.	◐	◐	✘	✘	✘
LLM with security and privacy issues	2024	Yao *et al.* [9]	Explores the dual impact of LLMs on security and privacy, highlighting their potential to enhance cybersecurity and data protection while also posing new risks and vulnerabilities.	◐	◐	✘	✘	✘
THIS SURVEY	2024	Ferrag *et al.*	This paper provides an in-depth review of using Generative AI and Large Language Models (LLMs) in cybersecurity.	✔	✔	✔	✔	✔

✘ : Not covered; ◐: Partially covered; ✔: Covered; Data.: Datasets used for training and fine-tuning LLMs for security use cases; Vuln.: LLM Vulnerabilities and Mitigation ; Comp.: Experimental Analysis of LLMs Models' Performance in Cyber Security Knowledge; Optim.: Optimization Strategies for Large Language Models in Cybersecurity; Hardw. : Experimental Analysis of LLMs Models' Performance in Hardware Security.

domains and applications using instruction tuning strategies is reviewed, demonstrating the versatility of this method. Additionally, the survey addresses efforts to enhance the efficiency of instruction fine-tuning, focusing on reducing computational and time costs. Finally, it evaluates these models, including performance analysis and critical perspectives, offering a holistic view of the current state and potential of instruction fine-tuning in LLMs.

F. LLMs in Software Engineering

Fan *et al.* [30] present a survey on using LLMs in Software Engineering (SE), highlighting their potential applications and open research challenges. LLMs, known for their emergent properties, offer novel and creative solutions across various Software Engineering activities, including coding, design, requirements analysis, bug fixing, refactoring, performance optimization, documentation, and analytics. Despite these advantages, the paper also acknowledges the significant technical challenges these emergent properties bring, such as the need

for methods to eliminate incorrect solutions, notably hallucinations. The survey emphasizes the crucial role of hybrid approaches, which combine traditional Software Engineering techniques with LLMs, in developing and deploying reliable, efficient, and effective LLM-based solutions for Software Engineering. This approach suggests a promising pathway for integrating advanced AI models into practical software development processes.

G. Multimodal Algorithms

Wu *et al.* [31] addresses a significant gap in understanding multimodal algorithms by providing a comprehensive overview of their definition, historical development, applications, and challenges. It begins by defining multimodal models and algorithms, then traces their historical evolution, offering insights into their progression and significance. The paper serves as a practical guide, covering various technical aspects essential to multimodal models, such as knowledge representation, selection of learning objectives, model construction, information fusion, and prompts. Additionally, it reviews current algorithms employed in multimodal models and discusses commonly used datasets, thus laying a foundation for future research and evaluation in this field. The paper concludes by exploring several applications of multimodal models and delving into key challenges that have emerged from their recent development, shedding light on both the potential and the limitations of these advanced computational tools.

H. Alignment Requirements for LLMs

Liu *et al.* [32] propose a taxonomy of alignment requirements for LLMs to aid practitioners in understanding and effectively implementing alignment dimensions and inform data collection efforts for developing robust alignment processes. The paper dissects the concept of "harmful" generated content into specific categories, such as harm to individuals (like emotional harm, offensiveness, and discrimination), societal harm (including instructions for violent or dangerous behaviors), and harm to stakeholders (such as misinformation impacting business decisions). Citing an imbalance in Anthropic's alignment data, the paper points out the uneven representation of various harm categories, like the high frequency of "violence" versus the marginal appearance of "child abuse" and "self-harm." This observation supports the argument that alignment techniques heavily dependent on data cannot ensure that LLMs will uniformly align with human behaviors across all aspects. The authors' own measurement studies reveal that aligned models do not consistently show improvements across all harm categories despite the alignment efforts claimed by the model developers. Consequently, the paper advocates for a framework that allows a more transparent, multi-objective evaluation of LLM trustworthiness, emphasizing the need for a comprehensive and balanced approach to alignment in LLM development.

I. Knowledge-Enhanced Pre-trained Language Models

Hu *et al.* [33] offers a comprehensive review of Knowledge-Enhanced Pre-trained Language Models (KE-PLMs), a burgeoning field aiming to address the limitations of standard PLMs in NLP. While PLMs trained on vast text corpora demonstrate impressive performance across various NLP tasks, they often fall short in areas like reasoning due to the absence of external knowledge. The paper focuses on how incorporating different types of knowledge into PLMs can overcome these shortcomings. It introduces distinct taxonomies for Natural Language Understanding (NLU) and Natural Language Generation (NLG) to distinguish between these two core areas of NLP. For NLU, the paper categorizes knowledge types into linguistic, text, knowledge graph (KG), and rule knowledge. In the context of NLG, KE-PLMs are classified into KG-based and retrieval-based methods. By outlining these classifications and exploring the current state of KE-PLMs, the paper provides not only clear insights into this evolving domain but also identifies promising future directions for the development and application of KE-PLMs, highlighting their potential to enhance the capabilities of PLMs in NLP tasks significantly.

J. Controllable Text Generation in NLG

Zhang [34] provides a critical and systematic review of Controllable Text Generation (CTG), a burgeoning field in NLG that is essential for developing advanced text generation technologies tailored to specific practical constraints. The paper focuses on using large-scale pre-trained language models (PLMs), particularly those based on transformer architecture, which have established a new paradigm in NLG due to their ability to generate more diverse and fluent text. However, the limited interpretability of deep neural networks poses challenges to the controllability of these methods, making transformer-based PLM-driven CTG a rapidly evolving and challenging research area. The paper surveys various approaches that have emerged in the last 3-4 years, each targeting different CTG tasks with varying controlled constraints. It provides a comprehensive overview of common tasks, main approaches, and evaluation methods in CTG and discusses the current challenges and potential future directions in the field. Claiming to be the first to summarize state-of-the-art CTG techniques from the perspective of Transformer-based PLMs, this paper aims to assist researchers and practitioners in keeping pace with the academic and technological developments in CTG, offering them an insightful landscape of the field and a guide for future research.

K. LLM for Cyber Security

Zhang *et al.* [40] examines the integration of LLMs within cybersecurity. Through an extensive literature review involving over 127 publications from leading security and software engineering venues, this paper aims to shed light on LLMs' multifaceted roles in enhancing cybersecurity measures. The survey pinpoints various applications for LLMs in detecting vulnerabilities, analyzing malware, and managing network intrusions and phishing threats. It highlights the current limitations regarding the datasets used, which often lack size and diversity, thereby underlining the necessity for more robust datasets tailored to these security tasks. The paper also identifies promising methodologies like fine-tuning and

domain-specific pre-training, which could better harness the potential of LLMs in cybersecurity contexts.

Yao *et al.* [9] explores the dual role of LLMs in security and privacy, highlighting their benefits in enhancing code security and data confidentiality and detailing potential risks and inherent vulnerabilities. The authors categorize the applications and challenges into "The Good," "The Bad," and "The Ugly," where they discuss LLMs' positive impacts, their use in offensive applications, and their susceptibility to specific attacks, respectively. The paper emphasizes the need for further research on threats like model and parameter extraction attacks and emerging techniques such as safe instruction tuning, underscoring the complex balance between leveraging LLMs for improved security and mitigating their risks.

L. Our survey compared to related surveys

Our paper presents a more specialized and technical exploration of generative artificial intelligence and large language models in cybersecurity than the previous literature review. Focusing on a broad array of cybersecurity domains such as hardware design security, intrusion detection systems, and software engineering, it targets a wider professional audience, including engineers, researchers, and industrial practitioners. This paper reviews 35 leading models like GPT-4, BERT, Falcon, and LLaMA, not only highlighting their applications but also their developmental trajectories, thereby providing a comprehensive insight into the current capabilities and future potentials of these models in cybersecurity.

The paper also delves deeply into the vulnerabilities associated with LLMs, such as prompt injection, adversarial natural language instructions, and insecure output handling. It presents sophisticated attack scenarios and robust mitigation strategies, offering a detailed analysis crucial for understanding and protecting against potential threats. Additionally, the lifecycle of specialized cybersecurity datasets—covering creation, cleaning, preprocessing, annotation, and labeling—is scrutinized, providing essential insights into improving data integrity and utility for training and testing LLMs. This level of detail is vital for developing robust cybersecurity solutions that can effectively leverage the power of LLMs.

Lastly, the paper examines the challenges associated with deploying LLMs in cybersecurity contexts, emphasizing the necessity for model robustness and the implications of adversarial attacks. It introduces advanced methodologies such as Reinforcement Learning with Human Feedback (RLHF) and Retrieval-Augmented Generation (RAG) to enhance real-time cybersecurity operations. This focus not only delineates the current state of LLM applications in cybersecurity but also sets the direction for future research and practical applications, aiming to optimize and secure LLM deployments in an evolving threat landscape. This makes the paper an indispensable resource for anyone involved in cybersecurity and AI, bridging the gap between academic research and practical applications.

III. Preliminaries of NLP for Cyber Security

This section presents the preliminaries of NLP for cybersecurity, including recurrent neural networks (LSTMs and GRUs) and transformer models.

A. Recurrent neural networks

Recurrent Neural Networks (RNNs) [44] are artificial neural networks that handle data sequences such as time series or NLP tasks. The RNN model consists of two linked recurrent neural networks. The first RNN encodes sequences of symbols into a fixed-length vector, while the second decodes this vector into a new sequence. This architecture aims to maximize the conditional probability of a target sequence from a given source sequence. When applied to cybersecurity, this model could be instrumental in threat detection and response systems by analyzing and predicting network traffic or log data sequences that indicate malicious activity. Integrating the conditional probabilities generated by this model could enhance anomaly detection frameworks, improving the identification of subtle or novel cyber threats. The model's ability to learn meaningful representations of data sequences further supports its potential to recognize complex patterns and anomalies in cybersecurity environments [45], [46].

1) Gated Recurrent Units: GRUs are a recurrent neural network architecture designed to handle the vanishing gradient problem that can occur with standard recurrent networks. Introduced by Cho et al. in 2014 [47], GRUs simplify the LSTM (Long Short-Term Memory) model while retaining its ability to model long-term dependencies in sequential data. GRUs achieve this through two main gates: the update gate, which controls how much a new state overwrites the old state, and the reset gate, which determines how much past information to forget. These gates effectively regulate the flow of information, making GRUs adept at tasks like time series prediction, speech recognition, and natural language processing. The main steps of GRUs are organized as follows:

- Update Gate: The update gate determines how much information from the previous hidden state should be passed to the new one. The update gate is calculated using the following formula:

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{1}$$

where z_t is the update gate at time step t, W_z and U_z are the weight matrices, x_t is the input at time step t, and h_{t-1} is the previous hidden state. The sigmoid function, represented by σ, squishes the equation's results between 0 and 1. The update gate allows the GRU to decide how much of the previous hidden state information should be passed on to the new hidden state. If the update gate is close to 1, it means that a lot of the previous hidden state information should be passed on, while if the update gate is close to 0, it means that very little of the previous hidden state information should be passed on. The Update Gate formula can be explored in the following three different parts:
Part 1: Linear combination of inputs:

$$z_t = W_r x_t + U_r h_{t-1} \tag{2}$$

Part 2: Application of the sigmoid function:

$$\tilde{r}_t = \sigma(z_t) \tag{3}$$

Part 3: Element-wise multiplication of $\tilde{r}_t$ and previous hidden state:

$$r_t = \tilde{r}_t \odot h_{t-1} \tag{4}$$

Where $\odot$ represents the Hadamard product, also known as the element-wise multiplication.

- Reset Gate: The reset gate determines how much of the previous hidden state should be forgotten. The reset gate is calculated using the following formula:

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{5}$$

where r_t is the reset gate at time step t, W_r and U_r are the weight matrices, x_t is the input at time step t, and h_{t-1} is the previous hidden state.

- Candidate Hidden State: The candidate's hidden state combines the input and the previous hidden state, filtered through the reset gate. The candidate's hidden state is calculated using the following formula:

$$\tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1})) \tag{6}$$

where $\tilde{h}_t$ is the candidate hidden state at time step t, W and U are the weight matrices, x_t is the input at time step t, and r_t is the reset gate at time step t. In this equation, the input at time step t (x_t) is combined with the previous hidden state (h_{t-1}) through the weight matrices W and U. The reset gate (r_t) is used to control the extent to which the previous hidden state (h_{t-1}) is passed to the candidate hidden state ($\tilde{h}_t$). The element-wise product between the reset gate (r_t) and the previous hidden state (h_{t-1}) is used to create the reset vector ($r_t \odot h_{t-1}$). The reset vector is combined with the input (x_t) through the weight matrix U. Finally, the result is passed through the hyperbolic tangent function to calculate the candidate hidden state ($\tilde{h}_t$).

- New Hidden State: The new hidden state combines the previous and candidate hidden states, filtered through the update gate. The new hidden state is calculated using the following formula:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{7}$$

where h_t is the new hidden state at time step t, z_t is the update gate at time step t, and $\tilde{h}_t$ is the candidate hidden state at time step t. The new hidden state (h_t) is calculated by taking a weighted combination of the previous hidden state (h_{t-1}) and the candidate hidden state ($\tilde{h}_t$). The weight of the previous hidden state is determined by the update gate (z_t) - if the update gate is close to 1, the new hidden state is primarily influenced by the previous hidden state. If the update gate is close to 0, the candidate's hidden state primarily influences the new hidden state. The element-wise product between the update gate (z_t) and the candidate hidden state ($\tilde{h}_t$) is used to create the update vector ($z_t \odot \tilde{h}_t$). The element-wise product between the complement of the update gate ($1 - z_t$) and the previous hidden state (h_{t-1}) is used to create the reset vector ($(1-z_t) \odot h_{t-1}$). Finally, the update and reset vectors are added to calculate the new hidden state (h_t).

2) Long Short-Term Memory: The LSTM [2] was designed to overcome the vanishing gradient problem that affects traditional recurrent neural networks (RNNs) during training, particularly over long sequences. By integrating memory cells that can maintain information over extended periods and gates that regulate the flow of information into and out of the cell, LSTMs provide an effective mechanism for learning dependencies and retaining information over time. This architecture has proved highly influential, becoming foundational to numerous applications in machine learning that require handling sequential data, such as natural language processing, speech recognition, and time series analysis. The impact of this work has been extensive, as it enabled the practical use and development of deep learning models for complex sequence modeling tasks. In cybersecurity, LSTMs can be used for anomaly detection, where they analyze network traffic or system logs to identify unusual patterns that may signify a security breach or malicious activity [48]–[50]. Their ability to learn from long sequences makes them particularly useful for detecting sophisticated attacks that evolve, such as advanced persistent threats (APTs) and ransomware. The main steps of LSTM models are organized as follows:

- Input Gate: The first step in an LSTM-based RNN involves calculating the input gate. The input gate determines the extent of new input to be added to the current state. The formula for the input gate is:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{8}$$

where i_t is the input gate at time step t, W_i is the weight matrix for the input gate, h_{t-1} is the hidden state from the previous time step, x_t is the input at time step t, and b_i is the bias for the input gate. The function $\sigma(x)$ is the sigmoid activation function. This formula calculates the input gate i_t by first concatenating the previous hidden state h_{t-1} with the current input x_t. This combined vector is multiplied by the weight matrix W_i, and the bias b_i is added. Finally, the sigmoid activation function is applied to produce i_t, which ranges from 0 to 1 and represents how much the current input updates the hidden state.

- Forget Gate: The second step calculates the forget gate, determining how much the previous state should be forgotten. The formula for the forget gate is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (9)$$

where f_t is the forget gate at time step t, W_f is the weight matrix for the forget gate, h_{t-1} is the hidden state from the previous time step, x_t is the input at time step t, and b_f is the bias for the forget gate. The forget gate f_t is calculated like the input gate. It involves concatenating the previous hidden state h_{t-1} with the current input x_t, multiplying by the weight matrix W_f, and adding the bias b_f. The resulting value is passed through the sigmoid activation function to determine the forget gate f_t, which ranges from 0 to 1 and represents the degree to which the previous hidden state is preserved or forgotten in the current hidden state.

- Candidate Memory Cell: The third step calculates the candidate memory cell, representing the potential memory state update. The formula for the candidate memory cell is:

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (10)$$

where $\tilde{c}_t$ is the candidate memory cell at time step t, W_c is the weight matrix for the candidate memory cell, h_{t-1} is the hidden state from the previous time step, x_t is the input at time step t, and b_c is the bias for the candidate memory cell. The function $\tanh(x)$ is the hyperbolic tangent activation function. In this formula, the candidate memory cell $\tilde{c}_t$ is calculated by concatenating the previous hidden state h_{t-1} and the input x_t, then multiplying by the weight matrix W_c and adding the bias b_c. The result is passed through the hyperbolic tangent activation function, which ranges from -1 to 1, to control the magnitude of the memory cell update.

- Current Memory Cell: The fourth step calculates the current memory cell, which is the updated state of the memory cell, combining the effects of the forget and input gates. The formula for the current memory cell is:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \quad (11)$$

where c_t is the current memory cell at time step t, f_t is the forget gate at time step t, c_{t-1} is the memory cell from the previous time step, i_t is the input gate at time step t, and $\tilde{c}_t$ is the candidate memory cell at time step t. This equation represents the new memory cell state as a combination of the old state (modulated by the forget gate) and the potential update (modulated by the input gate).

- Output Gate: The final step calculates the output gate, which determines the amount of information output from the LSTM cell. The details and formula for the output gate should follow.

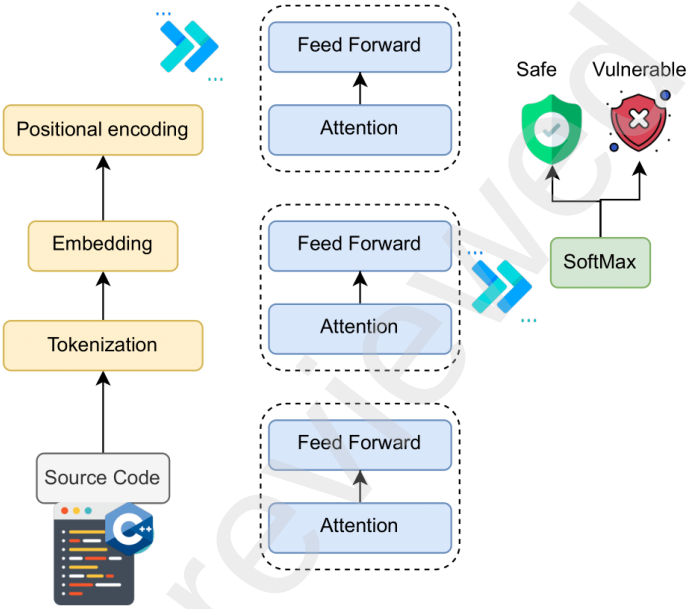

Fig. 3: How Transformer works for Software Security.

B. *Transformer models*

The Transformer architecture proposed by Vaswani et al. [4] in 2017 is a significant advancement in natural language processing built entirely around attention mechanisms. These mechanisms allow the model to assess the relevance of different words in a sentence, independent of their positional relationships. This foundational technology has enhanced the efficiency of tasks like translation and text summarization and has broad cybersecurity applications. In cybersecurity, Transformer models can detect and respond to threats by analyzing source code patterns and network traffic and identifying anomalies in system logs, as presented in Figure 3. They can also be used for the automated generation of security policies based on the evolving landscape of threats and for intelligent threat hunting, where the system predicts and neutralizes threats before they cause harm. This makes Transformers versatile in enhancing security protocols and defending against cyber attacks [19]. The main steps of Transformer models are organized as follows:

- Attention Mechanism: The attention mechanism in the Transformer model computes attention scores between the input and output representations. These scores are calculated using the scaled dot-product of the query and key representations and then normalized by a softmax function. The attention scores are subsequently used to compute a weighted sum of the value representations, forming the output of the attention mechanism.
The equation defines the attention scores:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (12)$$

Where:

Q, K, and V represent the matrices of queries, keys, and values transformed from the input representations. The dimension of the keys is denoted by d_k. The attention mechanism involves computing the dot product of Q and the transpose of K, which is then scaled by the inverse square root of d_k to stabilize the gradients. The result is passed through a softmax function to normalize the scores, ensuring they sum to 1. These scores, representing attention weights, compute a weighted sum of the values in V, resulting in the final attention output. This mechanism allows the model to dynamically focus on the most relevant parts of the input sequence for making predictions.

- Multi-Head Attention: In the Transformer model, multiple attention heads enhance the model's capability to simultaneously focus on different parts of the input sequence. The multi-head attention is calculated as follows:

$$\text{MHead}(Q, K, V) = \text{Concat}(hd_1, hd_2, \ldots, hd_h)W^O \quad (13)$$

Where: hd_i represents the output of the i-th attention head, computed using the attention formula $\text{Attention}(Q_i, K_i, V_i)$. Each Q_i, K_i, and V_i are different linear projections of the original inputs Q, K, and V. W^O is a linear transformation matrix applied to the concatenated results of all attention heads. The Concat function concatenates the outputs of each head along a specific dimension. The outputs of individual heads, hd_i, are each computed using the scaled dot-product attention mechanism:

$$hd_i = \text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \quad (14)$$

This approach enables the Multi-Head Attention mechanism to capture various aspects of the input sequence, simultaneously focusing on different subspace representations. As a result, it facilitates the model's capture of more complex relationships and improves performance across different types of tasks.

- Layer Normalization: In the Transformer model, layer normalization ensures the input is within a standard range. The layer normalization can be calculated as follows:

$$LN(x) = \frac{x - \text{mean}(x)}{\sqrt{\text{var}(x)}} \quad (15)$$

Where x is the input to the layer normalization. $textmean(x)$ and $textvar(x)$ are the mean and variance of x, respectively. Layer Normalization aims to mitigate the internal covariate shift, which arises when the distribution of activations in a layer changes during training. The normalization operation is performed by subtracting the mean of the activations and dividing by the square root of the variance. This ensures the activations have a zero mean and unit variance, leading to more stable training.

- Position-wise Feed Forward: The position-wise feed-forward network transforms the input and output representations in the Transformer model. The position-wise feedforward can be calculated as follows:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (16)$$

Where: x is the input to the feed-forward network. W_1, b_1, W_2, and b_2 are the weight and bias parameters of the feed-forward network. $\max(0, x)$ is the ReLU activation function. This equation represents a simple feed-forward neural network (FFN) operation in deep learning models. The FFN operation is a multi-layer perceptron (MLP) that transforms the input x into a new representation by passing it through two fully connected (dense) layers. The first layer is followed by a ReLU activation function, which applies a non-linear activation to the input by setting all negative values to zero. This activation function helps the model learn complex non-linear relationships between the input and output. The second layer is a linear transformation that produces the final output of the FFN. The weight and bias parameters of the two layers, W_1, b_1, W_2, and b_2, are learned during training and allow the model to learn different representations of the input data.

- Encoder and Decoder Blocks: In the Transformer model, the encoder and decoder blocks transform the input sequences into the output sequences. The encoder and decoder blocks can be calculated as follows:

$$\text{Enc}(x) = \text{LN}(x + \text{MHead}(x, x, x)) \quad (17)$$

$$\text{Dec}(x, y) = \text{LN}(x + \text{MHead}(x, y, y) + \text{MHead}(x, x, x)) \quad (18)$$

Where: x is the input to the encoder/decoder block. y is the output from the previous encoder/decoder block. The Encoder block $\text{Enc}(x)$ takes the input x and applies the Multi-Head Attention mechanism to compute the attention scores between the input and itself. The result is then added to the input and passed through a Layer Normalization operation. The output of the encoder block is the new representation of the input after processing through the Multi-Head Attention and Layer Normalization operations. The Decoder block $\text{Dec}(x, y)$ is similar to the encoder block but also takes the output from the previous decoder block, y, as input. The Multi-Head Attention mechanism is applied to compute the attention scores

between the input and the previous output and between the input and itself. The results are added to the input and passed through a Layer Normalization operation. The output of the decoder block is the new representation of the input after processing through the Multi-Head Attention and Layer Normalization operations.

IV. LLMs-based models for Cyber Security

This section reviews recent studies employing LLM-based models (i.e., Recurrent Neural Networks-based and transformer-based models) for threat detection, malware classification, intrusion detection, and software vulnerability detection. Tables II, III, and Figure 4 present the LLM-based solutions for Cyber Security Use Cases.

A. Recurrent Neural Networks-based models

1) Intrusion Detection: Yin *et al.* [55] propose a deep learning approach for intrusion detection using recurrent neural networks (RNN-ID) and study its performance in binary and multiclass classification tasks. The results show that the RNN-ID model outperforms traditional machine learning methods in accuracy. Chawla *et al.* [67] presented an anomaly-based intrusion detection system that leverages recurrent neural networks (RNNs) with gated recurrent units (GRUs) and stacked convolutional neural networks (CNNs) to detect malicious cyber attacks. The system establishes a baseline of normal behavior for a given system by analyzing sequences of system calls made by processes. It identifies anomalous sequences based on a language model trained on normal call sequences from the ADFA dataset of system call traces. The authors demonstrate that using GRUs instead of LSTMs results in comparable performance with reduced training times and that combining GRUs with stacked CNNs leads to improved anomaly detection. The proposed system shows promising results in detecting anomalous system call sequences in the ADFA dataset. However, further research is needed to evaluate its performance in other datasets and real-world scenarios and address issues related to adversarial attacks.

Ullah *et al.* [68] introduce the deep learning models to tackle the challenge of managing cybersecurity in the growing realm of IoT devices and services. The models utilize Recurrent Neural Networks, Convolutional Neural Networks, and hybrid techniques to detect anomalies in IoT networks accurately. The proposed models are validated using various datasets (i.e., IoT-DS2, MQTTset, IoT-23, and datasets) and achieve high accuracy, precision, recall, and F1 score. However, the models need to be tested on more extensive and diverse datasets, and further research is necessary to enhance their scalability for practical applications in cybersecurity. Donkol *et al.* [69] presents a technique, ELSTM-RNN, for improving security in intrusion detection systems. Using likely point particle swarm optimization (LPPSO) and enhanced LSTM classification, the proposed system addresses gradient vanishing, generalization, and overfitting issues. The system uses an enhanced particle swarm optimization technique to select efficient features, which are used for effective classification using an enhanced LSTM framework. The proposed system outperformed other methods, such as LPBoost and DNNs, in accuracy, precision, recall, and error rate. The NSL-KDD dataset was used for validation and testing, and further verification was done on other datasets. While the paper provides a comprehensive solution, future research could explore the applicability of the proposed system to other datasets and real-world scenarios. Additionally, a more detailed analysis of the computational cost of the proposed system compared to other methods could be beneficial.

Zhao *et al.* [70] presents ERNN, an end-to-end RNN model with a novel gating unit called session gate, designed to address network-induced phenomena that may result in misclassifications in traffic detection systems used in cybersecurity. The gating unit includes four types of actions to simulate network-induced phenomena during model training and the Mealy machine to adjust the probability distribution of network-induced phenomena. The paper demonstrates that ERNN outperforms state-of-the-art methods by 4% accuracy and is scalable in terms of parameter settings and feature selection. The paper also uses the Integrated Gradients method to interpret the gating mechanism and demonstrates its ability to reduce dependencies on local packets. Althubiti *et al.* [57] propose a deep learning-based intrusion detection system (IDS) that uses a Long Short-Term Memory (LSTM) RNN to classify and predict known and unknown intrusions. The experiments show that the proposed LSTM-based IDS can achieve a high accuracy rate of 0.9997. Xu *et al.* [58] propose a novel IDS that consists of a recurrent neural network with gated recurrent units (GRU), multilayer perceptron (MLP), and softmax module. The experiments on the KDD 99 and NSL-KDD data sets show that the system has a high overall detection rate and a low false positive rate. Ferrag and Leandros [59] propose a novel deep learning and blockchain-based energy framework for smart grids, which uses a blockchain-based scheme and a deep learning-based scheme for intrusion detection. The deep learning-based scheme employs recurrent neural networks to detect network attacks and fraudulent transactions in the blockchain-based energy network. The performance of the proposed IDS is evaluated using three different data sources.

Polat *et al.* [72] introduce a method for improving the detection of DDoS attacks in SCADA systems that use SDN technology. The authors propose using a Recurrent Neural Network (RNN) classifier model with two parallel deep learning-based methods: Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). The proposed model is trained and tested on a dataset from an experimentally created SDN-based SCADA topology containing DDoS attacks and regular network traffic data. The results show that the proposed RNN model achieves an accuracy of 97.62% for DDoS attack detection, and transfer learning further improves its performance by around 5%.

2) Software Security: Wang *et al.* [71] propose a deep learning-based defense system called PatchRNN to automatically detect secret security patches in open-source software (OSS). The system leverages descriptive keywords in the commit message and syntactic and semantic features at the source-code level. The system's performance was evaluated

TABLE II: LLMs-based models for Cyber Security (Part I).

Study	Year	Type of Model	Dataset Used	Domain	Key Contributions	Open Issues
Parra *et al.* [51]	2022	Federated Transformer Log Learning Model	HDFS and CTDD datasets	Threat detection and forensics	The interpretability module integrated into the model provides insightful interpretability of the model's decision-making process	The paper briefly mentions the applicability of the proposed approach in edge computing systems but does not discuss the scalability of the approach to larger systems
Ziems *et al.* [52]	2021	Transformer Model, BERT, CANINE, Bagging-based random transformer forest (RTF)	Malware family datasets	Malware Classification	Demonstration that transformer-based models outperform traditional machine and deep learning models in classifying malware families	The experiments are conducted on preprocessed NIST NVD/SARD databases, which may not reflect real-world conditions
Wu *et al.* [53]	2022	Robust Transformer-based Intrusion Detection System (RTID)	CICID2017 and CIC-DDoS2019 datasets	Intrusion Detection	The proposed method outperforms classical machine learning algorithms such as support vector machine (SVM) and deep learning algorithms (i.e., RNN, FNN, LSTM) on the two evaluated datasets	There is no discussion in the paper regarding the scalability of the proposed method, particularly when dealing with large-scale and real-time network traffic
Demirkıran *et al.* [54]	2022	Transformer-based models	Catak dataset, Oliveira dataset, VirusShare dataset, and VirusSample dataset	Malware classification	The paper demonstrates that transformer-based models, specifically BERT and CANINE, outperform traditional machine and deep learning models in classifying malware families	The study only focuses on malware families that use API call sequences, which means that it does not consider other malware types that may not use API calls
Yin *et al.* [55]	2017	RNN-ID (Recurrent Neural Network-Intrusion Detection)	Benchmark data set	Intrusion Detection	The proposed model can improve the accuracy of intrusion detection	Other machine learning algorithms and deep learning models, such as convolutional neural networks and transformers are not considered in the comparison
Güera *et al.* [56]	2018	Temporal-aware Pipeline (CNN and RNN)	Large set of deepfake videos collected from multiple video websites	Detection of Deepfake Videos	The proposed method achieves competitive results in detecting deepfake videos while using a simple architecture	The proposed approach's effectiveness might be limited to the specific types of deepfakes present in the dataset
Althubiti *et al.* [57]	2018	LSTM RNN	CSIC 2010 HTTP dataset	Web Intrusion Detection	Proposal of LSTM RNN for web intrusion detection. High accuracy rate (0.9997) in binary classification.	The paper only uses the CSIC 2010 HTTP dataset, which may not be representative of all types of web application attacks
Xu *et al.* [58]	2018	GRU-MLP-Softmax (Gated Recurrent Unit, Multilayer Perceptron, Softmax)	KDD 99 and NSL-KDD data sets	Network Intrusion Detection	The system achieves leading performance with overall detection rates of 99.42% using KDD 99 and 99.31% using NSL-KDD, with low false positive rates	The paper does not provide information about the scalability of the proposed model
Ferrag and Leandros [59]	2019	DeepCoin (Blockchain and Deep Learning)	CICID2017 dataset, Power system dataset, Bot-IoT dataset	Energy framework for Smart Grids	Proposal of DeepCoin framework combining blockchain and deep learning for smart grid security	The paper does not address the potential scalability issues that may arise as the number of nodes in the network increases
Ghourbi *et al.* [60]	2022	An optimized LightGBM model and a Transformer-based model	ToN-IoT and Edge IIoTset datasets	Threat Detection	The experimental evaluation of the approach showed remarkable accuracies of 99%	The paper does not discuss the scalability of the proposed system for large-scale healthcare networks
Thapa *et al.* [61]	2022	Transformer-based language models	Software vulnerability datasets of C/C++ source codes	Software security and vulnerability detection in programming languages, specifically C/C++	The paper highlights the advantages of transformer-based language models over contemporary models	The paper only focuses on detecting vulnerabilities in C/C++ source code and does not explore the use of large transformer-based language models in detecting vulnerabilities in other programming languages
Ranade *et al.* [62]	2021	A transformer-based language model, specifically GPT-2	WebText dataset	Fake Cyber Threat Intelligence	The attack is shown to introduce adverse impacts such as returning incorrect reasoning outputs	Further research is needed to explore how to prevent or detect data poisoning attacks on cyber-defense system
Fu *et al.* [63]	2022	Transformer-based line-level vulnerability prediction model	Large-scale real-world dataset with more than 188k C/C++ functions	Software vulnerability prediction in safety-critical software systems	The proposed system is accurate for predicting vulnerable functions affected by the Top-25 most dangerous CWEs	The model's performance can be changed when applied to different programming languages or software systems
Mamede *et al.* [64]	2022	A transformer-based deep learning model	Software Assurance Reference Dataset (SARD) project, which contains vulnerable and non-vulnerable Java files	Software security in the context of Java programming language	The proposed system can identify up to 21 vulnerability types and achieved an accuracy of 98.9% in multi-label classification	The proposed method cannot be extended to other programming languages and integrated into existing software development processes
Evange *et al.* [65]	2021	A transformer-based model	DNRTI (Dataset for NER in Threat Intelligence)	Cybersecurity threat intelligence	The experimental results demonstrate that transformer-based techniques outperform previous state-of-the-art approaches for NER in threat intelligence	Further research is needed to test the effectiveness of transformer-based models on larger and more diverse datasets
Hashemi *et al.* [66]	2023	Transformer models (including BERT, XLNet, RoBERTa, and DistilBERT)	Labeled dataset from vulnerability databases	Vulnerability Information Extraction	The proposed approach outperforms existing rule-based and CRF-based models	The paper does not address the issue of bias in the labeled dataset

on a large-scale real-world patch dataset and a case study on NGINX. The results indicate that the PatchRNN system can effectively detect secret security patches with a low false positive rate.

3) Detection of Deepfake Videos: Güera *et al.* [56] propose a temporal-aware pipeline that automatically detects deepfake

TABLE III: LLMs-based models for Cyber Security (Part II).

Study	Year	Type of Model	Dataset Used	Domain	Key Contributions	Open Issues
Chawla *et al.* [67]	2019	GRU with CNN	ADFA (Australian Defence Force Academy) dataset	Intrusion Detection	Achieved improved performance by combining GRUs and CNNs	The proposed system is vulnerable to adversarial attacks
Ullah *et al.* [68]	2022	LSTM, BiLSTM, and GRU	IoT-DS2, MQTTset, IoT-23, and datasets	Intrusion Detection	Validation of the proposed models using various datasets, achieving high accuracy, precision, recall, and F1 score	Further research is necessary to enhance their scalability for practical applications in cybersecurity
Donkol *et al.* [69]	2023	LSTM	CSE-CIC-IDS2018, CICIDS2017, and UNSW-NB15 datasets	Intrusion Detection	The proposed system outperformed other methods such as LPBoost and DNNs in terms of accuracy, precision, recall, and error rate	Future research could explore the applicability of the proposed system to other datasets
Zhao *et al.* [70]	2023	End-to-End Recurrent Neural Network	IDS2017 and IDS2018 datasets	Intrusion attacks and malware	+ Address network-induced phenomena that may result in misclassifications in traffic detection systems used in cybersecurity	The proposed system is vulnerable to adversarial attacks
Wang *et al.* [71]	2021	RNN	A large-scale patch dataset PatchDB	Software Security	The PatchRNN system can effectively detect secret security patches with a low false positive rate	The PatchRNN system can only support C/C++
Polat *et al.* [72]	2022	LSTM and GRU	SDN-based SCADA system	Detection of DDoS attacks	The results show that the proposed RNN model achieves an accuracy of 97.62% for DDoS attack detection	The paper only focuses on detecting DDoS attacks and does not address other types of cyber threats (e.g., insider threats or advanced persistent threats)
Liu *et al.* [73]	2022	Transformer model	A commit benchmark dataset that includes over 7.99 million commits across 7 programming languages	Commit message generation (generation task) and security patch identification (understanding task)	The experimental results demonstrate that CommitBART significantly outperforms previous pre-trained models for code	The pre-training dataset used in the paper is limited to GitHub commits
Ahmad *et al.* [74]	2024	Transformer model	Set of 15 hardware security bug benchmark designs from three sources: MITRE website, OpenTitan System-on-Chip (SoC) and the Hack@DAC 2021 SoC	Hardware Security Bugs	Bug repair potential demonstrated by ensemble of LLMs, outperforming state-of-the-art automated tool	The need for designer assistance in bug identification, handling complex bugs, limited evaluations due to simulation constraints, and challenges with token limits and repair generation using LLMs
Wan *et al.* [75]	2024	Transformer model	Chrysalis dataset, comprising over 1,000 function-level HLS designs with injected logical bugs	Design Verification	Creating the Chrysalis dataset for HLS debugging, and enabling LLM-based bug detection and integration into development environments	Refining LLM techniques, integrating LLMs into development environments, and addressing scalability and generalization challenges
Jang *et al.* [76]	2024	Transformer model	Includes 150K online security articles, 7.3K security paper abstracts, 3.4K Wikipedia articles, and 185K CVE descriptions.	Threat Detection	Pre-trained language model for the cybersecurity domain, CyBERTuned incorporates non-linguistic elements (NLEs) such as URLs and hash values commonly found in cybersecurity texts.	The paper's limitations include a narrow focus on specific non-linguistic element (NLE) types, acknowledging the existence of more complex NLE types like code blocks and file paths that require future exploration
Bayer *et al.* [77]	2024	Transformer model	A dataset consisting of 4.3 million entries of Twitter, Blogs, Paper, and CVEs related to the cybersecurity domain	Intrusion attacks and malware	Created a high-quality dataset and a domain-adapted language model for the cybersecurity domain, which improves the internal representation space of domain words and performs best in cybersecurity scenarios	the model may not be suitable as a replacement for every type of cybersecurity model. They also state that the hyperparameters may not be generalizable to other language models, especially very large language models
Shestov *et al.* [78]	2024	Transformer model	The dataset comprises 22,945 function-level source code samples. It includes 13,247 samples for training, 5,131 for validation, and 4,567 for testing	Vulnerability detection	Finetuning the state-of-the-art code LLM, WizardCoder, increasing its training speed without performance harm.	The proposed study shows that the main bottlenecks of the task that limit performance lie in the field of dataset quality and suggests the usage of the project-level context information
He *et al.* [79]	2024	Transformer model	Used three datasets: one with over 100,000 entries from Ethereum mainnet contracts, another with 892,913 addresses labelled across five vulnerability categories, and a third with 6,498 smart contracts, including 314 associated with Ponzi schemes	blockchain technology and smart contracts	The introduction of a novel model, BERT-ATT-BiLSTM, for advanced vulnerability detection in smart contracts, and the evaluation of its performance against other models	Include the model's limitation in recognizing unseen contract structures or novel types of vulnerabilities, and the need to incorporate support for multiple programming languages to enhance universality and robustness
Jamal *et al.* [21]	2024	Transformer model	Two open-source datasets, 747 spam, 189 phishing, 4825 ham; class imbalance addressed with ADASYN	Phishing and spam detection	Proposing IPSDM, a fine-tuned version of DistilBERT and RoBERTA, outperforming baseline models and the demonstration of the effectiveness of LLMs in addressing cybersecurity challenges	Class imbalance, addressed with ADASYN, but potential bias remains

videos by using a convolutional neural network (CNN) to extract frame-level features and a recurrent neural network (RNN) to classify the videos. The results show that the system can achieve competitive results in this task with a simple architecture.

Overall, the reviewed studies demonstrate the potential of deep learning methods, particularly RNNs, for intrusion detection in various domains. The results show that the proposed deep learning-based models outperform traditional machine learning methods in accuracy. However, more research is

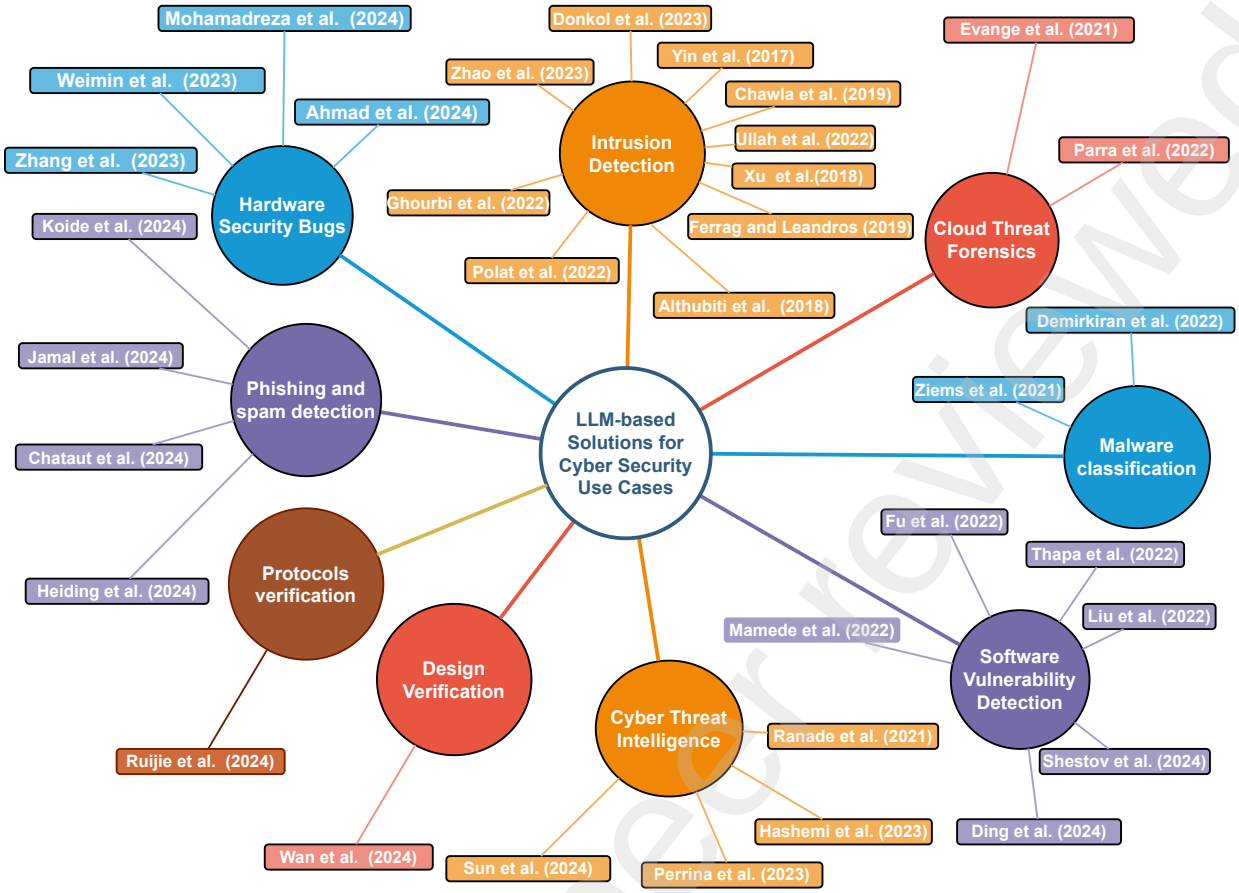

Fig. 4: LLM-based Solutions for Cyber Security Use Cases.

needed to address the limitations and challenges associated with these approaches, such as data scalability and interpretability.

B. Transformer-based models

1) Cloud Threat Forensics: Parra *et al.* [51] proposed an interpretable federated transformer log learning model for threat detection in syslogs. The model is generated by training local transformer-based threat detection models at each client and aggregating the learned parameters to generate a global federated learning model. The authors demonstrate the difference between normal and abnormal log time series through the goodness of fit test and provide insights into the model's decision-making process through an attention-based interpretability module. The results from the HDFS and CTDD datasets validate the proposed approach's effectiveness in achieving threat forensics in real-world operational settings. Evange *et al.* [65] discuss the importance of actionable threat intelligence in defending against increasingly sophisticated cyber threats. Cyber Threat Intelligence is available on various online sources, and Named Entity Recognition (NER) techniques can extract relevant information from these sources. The paper investigates the use of transformer-based models in NER and how they can facilitate the extraction of cybersecurity-related named entities. The DNRTI dataset, which contains over 300 threat intelligence reports, tests the effectiveness of transformer-based models compared to previous approaches. The experimental results show that transformer-based techniques are more effective than previous methods in extracting cybersecurity-related named entities.

2) Malware classification: Ziems *et al.* [52] explore transformer-based models for malware classification using API call sequences as features. The study compares the performance of the traditional machine and deep learning models with transformer-based models. It shows that transformer-based models outperform traditional models in terms of F1-score and AUC score. The authors also propose a bagging-based random transformer forest (RTF) model that reaches state-of-the-art evaluation scores on three out of four datasets. Demirkıran *et al.* [54] proposes using transformer-based models for classifying malware families, better suited for capturing sequence relationships among API calls than traditional machine and deep learning models. The experiments show that the proposed transformer-based models outperform traditional models such as LSTM and pre-trained models such as BERT or CANINE in classifying highly imbalanced malware families based on evaluation metrics like F1-score and AUC score. Additionally, the proposed bagging-based random transformer forest (RTF) model, an ensemble of BERT or CANINE, achieves state-of-the-art performance on three out of four datasets, including a state-of-the-art F1-score of 0.6149 on one of the commonly used benchmark datasets.

3) Intrusion Detection: Wu *et al.* [53] proposed an RTID that reconstructs feature representations in imbalanced datasets to make a trade-off between dimensionality reduction and feature retention. The proposed method utilizes a stacked encoder-decoder neural network and a self-attention mechanism for network traffic type classification. The results with CICID2017 and CIC-DDoS2019 datasets demonstrate the proposed method's effectiveness in intrusion detection compared to classical machine learning and deep learning algorithms. Ghourbi *et al.* [60] propose an intrusion and malware detection system to secure the entire network of the healthcare system independently of the installed devices and computers. The proposed solution includes two components: an intrusion detection system for medical devices installed in the healthcare network and a malware detection system for data servers and medical staff computers. The proposed system is based on optimized LightGBM and Transformer-based models. It is trained with four different datasets to ensure a varied knowledge of the different attacks affecting the healthcare sector. The experimental evaluation of the approach showed remarkable accuracies of 99%.

4) Software Vulnerability Detection: Thapa *et al.* [61] explores the use of large transformer-based language models in detecting software vulnerabilities in C/C++ source code, leveraging the transferability of knowledge gained from natural language processing. The paper presents a systematic framework for source code translation, model preparation, and inference. It conducts an empirical analysis of software vulnerability datasets to demonstrate the good performance of transformer-based language models in vulnerability detection. The paper also highlights the advantages of transformer-based language models over contemporary models, such as bidirectional long short-term memory and bidirectional gated recurrent units, in terms of F1-score. However, the paper does not discuss the limitations or potential drawbacks of using transformer-based language models for software vulnerability detection, and further research is needed in this area. Fu *et al.* [63] propose an approach called LineVul, which uses a Transformer-based model to predict software vulnerabilities at the line level. The approach is evaluated on a large-scale dataset (i.e., on a large-scale real-world dataset with more than 188k C/C++ functions). It achieves higher F1-measure for function-level predictions and higher Top-10 accuracy for line-level predictions compared to baseline approaches. The analysis also shows that LineVul accurately predicts vulnerable functions affected by the top 25 most dangerous CWEs. However, the model's performance can be changed when applied to different programming languages or software systems.

Mamede *et al.* [64] presented a transformer-based VS Code extension that uses state-of-the-art deep learning techniques for automatic vulnerability detection in Java code. The authors emphasize the importance of early vulnerability detection within the software development life cycle to promote application security. Despite the availability of advanced deep learning techniques for vulnerability detection, the authors note that these techniques are not yet widely used in development environments. The paper describes the architecture and evaluation of the VDet tool, which uses the Transformer architecture for multi-label classification of up to 21 vulnerability types in Java files. The authors report an accuracy of 98.9% for multi-label classification and provide a demonstration video, source code, and datasets for the tool.

Liu *et al.* [73] introduce CommitBART, a pre-trained Transformer model specifically designed to understand and generate natural language messages for GitHub commits. The model is trained on a large dataset of over 7.99 million commits, covering seven different programming languages, using a variety of pre-training objectives, including denoising, cross-modal generation, and contrastive learning, across six pre-training tasks. The authors propose a "commit intelligence" framework encompassing one understanding task and three generation tasks for commits. The experimental results demonstrate that CommitBART significantly outperforms previous pre-trained models for code, and the analysis suggests that each pre-training task contributes to the model's performance.

Ding et al. [80] discuss the effectiveness of code language models (code LMs) in detecting vulnerabilities. It identifies significant issues in current datasets, such as poor quality, low accuracy, and high duplication rates, which compromise model performance in realistic scenarios. To overcome these challenges, it introduces the PrimeVul dataset, which uses advanced data labeling, de-duplication, and realistic evaluation metrics to represent real-world conditions accurately. The findings reveal that current benchmarks, like the BigVul, greatly overestimate code LMs' capabilities, with much lower performance observed on PrimeVul. This significant discrepancy highlights the need for further innovative research to meet the practical demands of deploying code LMs in security-sensitive environments.

5) Cyber Threat Intelligence: Ranade *et al.* [62] presented a method for automatically generating fake Cyber Threat Intelligence (CTI) using transformers, which can mislead cyber-defense systems. The generated fake CTI is used to perform a data poisoning attack on a Cybersecurity Knowledge Graph (CKG) and a cybersecurity corpus. The attack introduces adverse impacts such as returning incorrect reasoning outputs, representation poisoning, and corruption of other dependent AI-based cyber defense systems. A human evaluation study was conducted with cybersecurity professionals and threat hunters, which reveals that professional threat hunters were equally likely to consider the generated fake CTI and authentic CTI as true.

Hashemi *et al.* [66] propose an alternative approach for automated vulnerability information extraction using Transformer models, including BERT, XLNet, RoBERTa, and DistilBERT, to extract security-related words and terms and phrases from descriptions of vulnerabilities. The authors fine-tune several language representation models similar to BERT on a labeled dataset from vulnerability databases for Named Entity Recognition (NER) to extract complex features without requiring domain-expert knowledge. This approach outperforms the CRF-based models and can detect new information from vulnerabilities with different description text patterns. The authors conclude that this approach provides a structured and unambiguous format for disclosing and disseminating vul-

nerability information, which is crucial for preventing security attacks.

6) Phishing and spam detection: Koide et al. introduced [81], a novel system leveraging LLMs to detect phishing emails. Despite advances in traditional spam filters, significant challenges such as oversight and false positives persist. The system transforms email data into prompts for LLM analysis, achieving a high accuracy rate (99.70%) and providing detailed reasoning for its determinations. This helps users make informed decisions about suspicious emails, potentially enhancing the effectiveness of phishing detection.

Jamal et al. [21] explored the potential of LLMs to address the growing sophistication of phishing and spam attacks. Their work, IPSDM, is an improved model based on the BERT family, specifically fine-tuned to detect phishing and spam emails. Compared to baseline models, IPSDM shows superior accuracy, precision, recall, and F1-score performance on both balanced and unbalanced datasets while addressing overfitting concerns.

Heiding et al. [82] compared automatically generated phishing emails by GPT-4, manually designed emails using the V-Triad method, and their combination. Their findings suggest that emails designed with the V-Triad achieved the highest click-through rates, indicating the effectiveness of exploiting cognitive biases. The study also evaluated the capability of four different LLMs to detect phishing intentions, with results often surpassing human detection. Furthermore, they discuss the economic impact of AI in lowering the costs of orchestrating phishing attacks.

Chataut et al. [83] focused on the effectiveness of LLMs in detecting phishing emails amidst threat actors' constant evolution of phishing strategies. Their study emphasizes the necessity for continual development and adaptation of detection models to keep pace with innovative phishing techniques. The role of LLMs in this context highlights their potential to significantly enhance email security by improving detection capabilities.

7) Hardware Security Evaluation: Ahmad *et al.* [74] delves into leveraging LLMs to automatically repair identified security-relevant bugs present in hardware designs, explicitly focusing on Verilog code. Hardware security bugs pose significant challenges in ensuring the reliability and safety of hardware designs. They curated a corpus of hardware security bugs through a meticulously designed framework. They explored the performance of various LLMs, including OpenAI's Codex and CodeGen, in generating replacement code to fix these bugs. The experiments reveal promising results, demonstrating that LLMs can effectively repair hardware security bugs, with success rates varying across different bugs and LLM models. By optimizing parameters such as instruction variation, temperature, and model selection, they achieved successful repairs for a significant portion of the bugs in their dataset. In addition, the results demonstrate that LLMs, including GPT-4, code-davinci-002, and code-cushman-001, yield successful repairs for simple security bugs, with GPT-4 achieving a success rate of 67% at variation e, temp 0.5. However, LLMs' performance varies across bugs, showing success rates over 75% with some bugs, while others are more challenging to repair, with success rates below 10%. The study emphasizes the importance of detailed prompt instructions, with variation d showing the highest success rate among OpenAI LLMs. Further investigation is needed to evaluate LLMs' scalability and effectiveness for diverse hardware security bug scenarios. Their findings underscore the potential of LLMs in automating the bug repair process in hardware designs, marking a crucial step towards developing automated end-to-end bug repair tools for hardware security.

Mohamadreza *et al.* [84] explored the potential of using large language models to enhance the input generation in the process of hardware design verification for security-related bugs. Mohamadreza et al. introduced Chatfuzz, a novel ML-based hardware fuzzer that leverages LLMs and reinforcement learning to generate complex and random machine code sequences for exploring processor security vulnerabilities. Chatfuzz introduces a specialized LLM model into a hardware fuzzing approach to enhance the input generation quality, outperforming the existing approaches regarding coverage, scalability, and efficiency. Utilizing LLMs to understand processor language and generate data/control flow entangled machine code sequences, Chatfuzz integrates RL to guide input generation based on code coverage metrics. Their experiment on real-world cores, namely RocketCore and BOOM cores, showed significantly faster coverage than state-of-the-art hardware fuzzes. ChatFuzz achieves 75% condition coverage in RocketCore in 52 minutes and 97.02% in BOOM in 49 minutes, identifying unique mismatches and new bugs and showcasing its effectiveness in hardware security testing.

Weimin *et al.* [85] introduces LLM4SECHW, a novel framework for hardware debugging that utilizes domain-specific Large Language Models. The authors addressed the limitations of out-of-the-shelf LLMs in the hardware security domain by gathering a dataset of hardware design defects and remediation steps. The collected dataset has been built by leveraging open-sourced hardware designs from GitHub; the data consists of different Hardware Description Language modules with their respective commits. By harnessing version control information from open-source hardware projects and processing it to create a debugging-oriented dataset, LLM4SECHW fine-tunes hardware domain-specific language models to locate and rectify bugs autonomously, enhancing bug localization. LLM4SECHW has been evaluated with two objectives: bug identification and design patching. The authors demonstrated that non-fine-tuned LLMs lack hardware domain knowledge, which makes them incapable of locating bugs in the hardware design of a popular security-specialized chip project named OpenTitan. The based models (falcon 7b, llama 2, Bard, chatbot, and stableLM) did not efficiently locate the introduced hardware bugs. The three fine-tuned models (falcon 7b, llama2, stableLM) successfully located the introduced bugs in the hardware design.

Zhang *et al.* [85] introduces Hardware Phi-1.5B, a large language model tailored for the hardware domain of the semiconductor industry, addressing the complexity of hardware-specific issues. The research focused on developing datasets specifically for the hardware domain to enhance the model's performance in comprehending complex terminologies. The

authors claim to surpass general code language models and natural language models like CodeLlama, BERT, and GPT-2 in the Hardware understanding tasks.

Madhav *et al.* [86] evaluated the security of the HDL code generated by ChatGPT. The authors introduced a similar taxonomy to the NIST CWE []. The authors conducted various experiments to explore the impact of prompt engineering on the security of the generated hardware design.

8) Hardware design & Verification: Lily *et al.* [87] introduced the application of LLMs into High-Level Synthesis Design Verification (HLS). The authors created a dataset named Chrysalis to solve the problem of the non-existence of specialized HLS bug detection and evaluation capabilities. The Chrysalis dataset comprises over 1000 function-level designs extracted from reputable sources with intentionally injected known bugs to evaluate and refine LLM-based HLS bug localization. The set of the introduced bugs was selected based on the most common human coding errors and has been shaped to elude most of the existing conventional HLS synthesis tools detection mechanisms. The paper's authors suggest that Chrysalis would contribute to the LLM-aided HLS design verification by offering a benchmarking to the existing and specialized models. The paper also suggests a prompt engineering approach that would enhance the efficiency of a large language model on the studied task. The proposed prompt structure introduces a separation of concern approach, where the used prompt deals with each class of bugs separately. The prompt starts by explicitly defining the context of the task, the functional description, the implementation context, and the task objective. The prompt is implemented through three main sections: context, requirements, and complementary rules. The highlighted works lay a foundation for a methodological, practical approach to benchmarking, evaluating, and deploying LLM tasks for HLS design verification. While the paper does not provide any conclusive results about LLMs' performance in such tasks, the authors believe that such methodology would accelerate the adoption of new techniques to integrate LLMs into the design verification flow.

Mingjie *et al.* [88] evaluated the LLMs' performance in solving Verilog related design tasks and generating design testbenchs by introducing VerilogEval. VerilogEval comprises different hardware design tasks ranging from module implementation of simple combinatorial circuits to complex finite state machines, code debugging, and testbench construction. VerilogEval suggests an end-to-end evaluation framework that fits better in the context of the hardware design verification process benchmarking. The VerilogEval framework validates the correctness of the prompted tasks by comparing the behavior simulation to an established golden model of the prompted design. The authors used pass@k metric instead of the generic NLP related metrics like the BLEU score probability metric. The study demonstrates that pre-trained language models' Verilog code generation capabilities can be improved through supervised fine-tuning. The experimental results show that fine-tuning LLMs on the hardware design tasks and using the pass@k metric helps assess the performance of the resulting models properly. The pass@k metric helps assess the performance of Large Language Models (LLMs) in Verilog code generation by quantifying the number of successful code completions out of k samples, offering a clear evaluation criterion. The used metric shows that a fine-tuned model could have equal or better performance than the state-of-the-art OpenAI models (gpt-3 and gpt-4). VerilogEval highlights the growing significance of Large Language Models (LLMs) and their application in various domains, emphasizing their potential in Verilog code generation for hardware design and verification. The findings underscore the importance of the proposed benchmarking framework in advancing the state of the art in Verilog code generation, highlighting the vast potential of LLMs in assisting the hardware design and verification process.

9) Protocols verification: Ruijie *et al.* [90] introduced ChatAFL, an LLM-based protocol fuzzer. ChatAFL introduces an LLM-guided protocol fuzzing to address the challenge of finding security flaws in protocol implementations without a machine-readable specification. The study suggests three strategies for integrating an LLM into a mutation-based protocol fuzzer, focusing on grammar extraction, seed enrichment, and saturation handling to enhance code coverage and state transitions. ChatAFL prototype implementation demonstrates that the LLM-guided stateful fuzzer outperforms state-of-the-art fuzzers like AFLNET [91] and NSFUZZ [92] in terms of protocol state space coverage and code coverage.

The experiments evaluated CHATAFL's improvement over the baselines in terms of transition coverage achieved in 24 hours, speed-up in achieving the same coverage, and the probability of outperforming the baselines in a random campaign. CHATAFL demonstrated significant efficacy by covering 47.60% and 42.69% more state transitions, 29.55% and 25.75% more states, and 5.81% and 6.74% more code than AFLNET and NSFUZZ, respectively.

CHATAFL discovered nine unique and previously unknown vulnerabilities in widely used and extensively tested protocol implementations on real widely used projects (live555, proFTPD, kamailio). The discovered vulnerabilities encompass various memory vulnerabilities, including use-after-free, buffer overflow, and memory leaks, which have potential security implications such as remote code execution or memory leakage. The study demonstrated the effectiveness of utilizing LLMs for guiding protocol fuzzing to enhance state and code coverage in protocol implementations.

Wang *et al.* [93] introduced LLMIF and LLM-aided fuzzing approach for IoT devices protocols. LLMIF introduces an LLM augmentation-based approach. The developed pipeline incorporates an enhanced seed generation strategy by building an augmentation based on domain knowledge. The domain knowledge structure is extracted from the various specifications of the under-fuzzing protocol. The flow starts by selecting a seed from the extracted augmentation set and then enriching the extracted seed by exploring the protocol specification. The enriching process is driven by the various ranges of input values extracted during the augmentation phase. Furthermore, LLMIF introduces a coverage approach by mutating the selected seed through the various enrichment and mutation operators that have been selected.

The evaluation part of LLMIF mainly aimed to evaluate

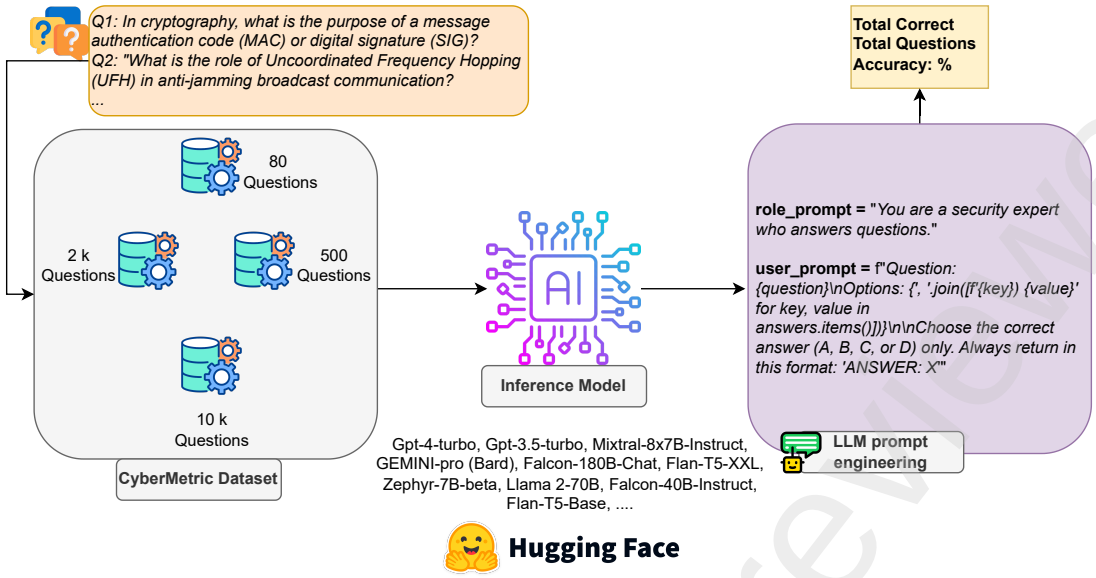

Fig. 5: LLMs Performance Steps in the cybersecurity domain using CyberMetric Dataset [89].

three axes: code coverage, ablation, and bug identification. The authors used an out-of-the-shelf popular SOC (CC2530) for the evaluation. 11 commercial devices have been selected to conduct the various experiments. While the ablation and bug detection could be easily evaluated, the code coverage is impossible using the custom firmware that ships with the selected devices. The authors used an open-source Zigbee stack to demonstrate the coverage capabilities. The authors claimed that LLMIF outperforms Z-FUZZER [94], and BOOFUZZ [95] in terms of code coverage for the target Zigbee stack. The authors claim that LLMIF achieved a notable increase in protocol message coverage and code coverage by 55.2% and 53.9%, respectively, outperforming other Zigbee fuzzers in these aspects.

LLMIF algorithm successfully uncovered 11 vulnerabilities on real-world Zigbee devices, including eight previously unknown vulnerabilities, showcasing its effectiveness in identifying security flaws in IoT devices. By incorporating the large language model into IoT fuzzing, LLMIF demonstrated enhanced capabilities in protocol message coverage and vulnerability discovery, highlighting its potential for improving the security testing of IoT devices.

V. General LLMs

Tables VI, VII, VIII compare general transformer-based Large Language Models. LLM models are generally trained on a diverse and broad range of data to provide a relatively comprehensive understanding. They can handle various language tasks like translation, summarization, and question-answering. In contrast, code-specific LLMs are specialized models trained primarily on programming languages and related technical literature, which makes their primary role in understanding and generating programming code well-suited for tasks like automated code generation, code completion, and bug detection.

A. Prevalent LLMs

1) GPT-3: GPT-3 (the third version of the Generative Pre-trained Transformer series by OpenAI) was developed to prove that scaling language models substantially improves their task-agnostic few-shot performance [96]. Based on transformer architecture, GPT-3 has eight variants ranging between 125M and 175B parameters, all trained for 300B tokens from datasets like Common Crawl, WebText, Books, and Wikipedia. Additionally, the models were trained on V100 GPU leveraging techniques like autoregressive training, scaled cross-entropy loss, and others. GPT-3, especially its most capable 175B version, has demonstrated strong performance on many NLP tasks in different settings (i.e., zero-shot, one-shot, and few-shots), suggesting it could significantly improve cybersecurity applications if appropriately fine-tuned. This could translate to more effective Phishing Detection through precise language analysis, faster Incident Response, and other critical applications to enhance digital security measures.

2) GPT-4: In 2023, the GPT-4 transformer-based model was released by OpenAI as the first large-scale multi-modal model, exhibiting unprecedented performance in various benchmarks. The model's capability of processing image and text inputs has shifted the AI paradigm to a new level, expanding beyond traditional NLP. [97] declared that GPT-4 was trained using a vast corpus of web-based data and data licensed from third-party sources with autoregressive techniques and Reinforcement Learning from Human Feedback (RLHF). However, other specifics, such as the model size, data size, and comprehensive training details, remain undisclosed. Although GPT-4 could potentially be leveraged by cybercriminals for a wide range of attacks, such as social engineering, if implemented strategically, it can also help reduce the likelihood of individuals and organizations falling prey to them.

3) T5: Motivated by the trend of applying transfer learning for NLP, researchers of Google have introduced T5 [98], an

TABLE IV: Comparison of 19 LLMs Models' Performance in Hardware Security Knowledge.

LLM model	Size	Design bug detection	Hardware CWE Number								
			1245	1221	1224	1298	1254	1209	1223	1234	1231
Llama 3-7b-instruct	8B	39.556%	Yes	No	No	No	Yes	No	No	Yes	No
Mixtral-8x7B-Instruct	8x7B	16.154%	No	No	No	No	No	No	No	No	No
Dolphin-mistral-7B	7B	16,024%	Yes	Yes	No	No	No	No	No	No	No
Codegemma-9b-instruct	9B	10.746%	No	No	No	No	No	No	No	No	Yes
CodeQwen-7b-instruct	7B	10.269%	No	No	No	No	No	No	No	No	No
Wizard-vicuna-uncensored-7b-instruct	7B	9.374%	No	No	No	No	No	No	No	No	No
Mistral-openorca-7b-instruct	7B	8.241%	No	No	No	No	No	Yes	No	No	No
Wizardlm2-7b-instruct	7B	5.646%	No	No	No	No	No	No	No	No	No
Llama2-uncensored-7b-instruct	7B	2.505%	No	No	No	No	No	No	No	No	No
Falcon-40b-instruct	40B	1.620%	No	No	No	No	No	No	No	No	No
Deepseek-coder-33b-instruct	33B	1.570%	No	No	No	No	No	No	No	No	No
Orca-mini-3b-instruct	3B	1.173%	No	No	Yes	No	No	No	No	No	No
Qwen2-4b-instruct	4B	0.576%	No	No	No	No	No	No	No	No	No
CodeLlama-7b-instruct	7B	0.218%	No	No	No	No	No	No	No	No	No
Phi3-4b-instruct	4B	0.019%	No	No	No	No	No	No	No	No	No
Hardware-Phi	1.5B	0%	No	No	No	No	No	No	No	No	No
Llava-13b-instruct	13B	0%	No	No	No	No	No	No	No	No	No
Gemma-9b-instruct	9B	0%	No	No	No	No	No	No	No	No	No
Starcoder2-15b-instruct	15B	0%	No	No	No	No	No	No	No	No	No

Yes: Detected the CWE sample by MITRE, No: Did not Detect the CWE sample by MITRE. CWE: Common Weakness Enumeration.

TABLE V: Comparison of 42 LLMs Models' Performance in Cyber Security Knowledge.

LLM model	Company	Size	License	Accuracy			
				80 Q	500 Q	2k Q	10k Q
GPT-4o	OpenAI	N/A	Proprietary	96.25%	93.40%	91.25%	88.89%
GPT-4-turbo	OpenAI	N/A	Proprietary	96.25%	93.30%	91.00%	88.50%
Mixtral-8x7B-Instruct	Mistral AI	45B	Apache 2.0	92.50%	91.80%	91.10%	87.00%
Falcon-180B-Chat	TII	180B	Apache 2.0	90.00%	87.80%	87.10%	87.00%
GEMINI-pro 1.0	Google	137B	Proprietary	90.00%	85.05%	84.00%	87.50%
GPT-3.5-turbo	OpenAI	175B	Proprietary	90.00%	87.30%	88.10%	80.30%
Yi-1.5-9B-Chat	01-ai	9B	Apache 2.0	87.50%	80.80%	77.15%	76.04%
Hermes-2-Pro-Llama-3-8B	NousResearch	8B	Open	86.25%	80.80%	77.95%	77.33%
Dolphin-2.8-mistral-7b-v02	Cognitive Computations	7B	Apache 2.0	83.75%	77.80 %	76.60%	75.01%
Mistral-7B-OpenOrca	Open-Orca	7B	Apache 2.0	83.75%	80.20%	79.00%	76.71 %
Gemma-1.1-7b-it	Google	7B	Open	82.50%	75.40%	75.75%	73.32%
Flan-T5-XXL	Google	11B	Apache 2.0	81.94%	71.10%	69.00%	67.50%
Meta-Llama-3-8B-Instruct	Meta	8B	Open	81.25 %	76.20%	73.05%	71.25%
Zephyr-7B-beta	HuggingFace	7B	MIT	80.94%	76.40%	72.50%	65.00%
Yi-1.5-6B-Chat	01-ai	6B	Apache 2.0	80.00%	75.80%	75.70%	74.84%
Mistral-7B-Instruct-v0.2	Mistral AI	7B	Apache 2.0	78.75%	78.40%	76.40%	74.82%
Llama 2-70B	Meta	70B	Apache 2.0	75.00%	73.40%	71.60%	66.10%
Qwen1.5-7B	Qwen	7B	Open	73.75%	60.60%	61.35%	59.79%
Qwen1.5-14B	Qwen	14B	Open	71.25%	70.00%	72.00%	69.96%
Mistral-7B-Instruct-v0.1	Mistral AI	7B	Apache 2.0	70.00%	71.80%	68.25%	67.29%
Llama-3-8B-Instruct-Gradient-1048k	Bartowski	8B	Open	66.25%	58.00%	56.30%	55.09%
Qwen1.5-MoE-A2.7B	Qwen	2.7B	Open	62.50%	64.60%	61.65%	60.73%
Phi-2	Microsoft	2.7B	MIT	53.75%	48.00%	52.90%	52.13%
Llama3-ChatQA-1.5-8B	Nvidia	8B	Open	53.75%	52.80%	49.45 %	49.64%
DeciLM-7B	Deci	7B	Apache 2.0	52.50%	47.20%	50.44%	50.75%
Flan-T5-Base	Google	0.25B	Apache 2.0	51.25%	50.40%	48.55%	47.09%
Deepseek-moe-16b-chat	Deepseek	16B	MIT	47.50%	45.80%	49.55%	48.76%
Mistral-7B-v0.1	Mistral AI	7B	Apache 2.0	43.75%	39.40%	38.15%	39.28%
Qwen-7B	Qwen	7B	Open	43.75%	58.00%	55.75%	54.09%
Gemma-7b	Google	7B	Open	42.50%	37.20%	36.00%	34.28%
Meta-Llama-3-8B	Meta	8B	Open	38.75%	35.80%	37.00%	36.00%
Genstruct-7B	NousResearch	7B	Apache 2.0	38.75%	40.60%	37.55%	36.93%
Qwen1.5-4B	Qwen	4B	Open	36.25%	41.20%	40.50%	40.29%
Llama-2-13b-hf	Meta	13B	Open	33.75%	37.00%	36.40%	34.49%
Dolly V2 12b BF16	Databricks	12B	MIT	33.75%	30.00%	28.75%	27.00%
Deepseek-llm-7b-base	DeepSeek	7B	MIT	33.75%	25.20%	27.00%	26.48%
Cerebras-GPT-2.7B	Cerebras	7B	Apache 2.0	25.00%	20.20%	19.75%	19.27%
Gemma-2b	Google	2B	Open	25.00%	23.20%	18.20%	19.18%
Stablelm-2-1_6b	Stability AI	6B	Open	16.25%	21.80%	19.55%	20.09%
ZySec-7B	ZySec-AI	7B	Apache 2.0	12.50%	16.40%	15.55%	14.04%
Phi-3-mini-4k-instruct	Microsoft	3.8B	MIT	5.00%	5.00%	4.41%	4.80%
Phi-3-mini-128k-instruct	Microsoft	3.8B	MIT	1.25%	0.20%	0.70%	0.88%

encoder-decoder-based model that operates within the unified text-to-text framework. Multiple variants of T5 with different sizes - ranging between 220M to 11B parameters- were developed to broaden the experimental scope and were trained on massive amounts of data from various sources, including C4, Web Text, and Wikipedia. Building on the foundation of these diverse model sizes and rich data sources, multiple approaches and different settings for pre-training and fine-tuning were examined and discussed, achieving performance that nearly matched human levels on one of the benchmarks. Considering that, the model's potential in cybersecurity applications is particularly promising. For instance, T5 can be utilized for threat intelligence by extracting critical information from vast security documents and then summarizing and organizing that information.

4) BERT: Bidirectional Encoder Representations from Transformers, commonly known as BERT, was presented by [99] to enhance fine-tuning-based approaches in NLP. It is available in two versions: BERT-Base, with 110M parameters, and BERT-Large, with 340M parameters, trained on 126GB of data from BooksCorpus and English Wikipedia. During its pre-training phase, BERT employed two key techniques: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). Building on these approaches, fine-tuning, and feature-based methods have led to competitive performance from BERT-Large in particular. Since encoder-only models like BERT are known for their robust contextual understanding, applying such models to tasks like malware detection and software vulnerability can be highly effective in cybersecurity.

5) ALBERT: Aiming to address the limitations related to GPU/TPU memory and training time in Large Language Models (LLMs), Google researchers developed A Lite BERT (ALBERT), a modified version of BERT with significantly fewer parameters [100]. And like other LLMs, ALBERT was introduced in various sizes, with options ranging from 12M to 235M parameters, all trained on data from BooksCorpus and English Wikipedia. Various methods and techniques were deployed during the pre-training stage, including Factorized Embedding Parameterization, Cross-layer Parameter Sharing, Inter-sentence Coherence Loss, and Sentence Order Prediction (SOP). As a result, one of the models (i.e., ALBERT-xxlarge) outperformed BERT-Large despite having fewer parameters. Thus, utilizing ALBERT in cybersecurity applications, such as phishing detection and malware classification, could significantly contribute to advancing cybersecurity infrastructure.

6) RoBERTa: RoBERTa, proposed by Meta, is an optimized replication of BERT that demonstrates how the choice of hyperparameters can significantly impact the model's performance [101]. RoBERTa has only one version with 355M parameters but is trained and tested in various data sizes and training steps. Similar to BERT, the training data was taken from the Books corpus and English Wikipedia. However, the key optimizations in this model were in the training techniques, which included multiple methods such as Dynamic Masking, training on Full Sentences without NSP loss, using Large Mini-Batches, and employing a Larger Byte-Level BPE. Consequently, RoBERTa achieved state-of-the-art results in some of the benchmarks. With proper fine-tuning, RoBERTa's ability to understand, interpret, and generate human-like text is leveraged to automate and enhance various tasks in the realm of cybersecurity.

7) XLNet: The advances and limitations of Masked Language Modeling (MLM) in bidirectional encoders and Autoregressive Language Modeling have inspired researchers at CMU and Google AI to develop XLNet [102]. Based on the Transformer-XL model, XLNet combines aspects of both approaches, enabling the learning of bidirectional contexts while addressing common MLM issues, such as neglecting dependencies between masked positions and the discrepancy between pretraining and finetuning phases. With 340M parameters, XLNet was pre-trained using data from English Wikipedia and utilizing techniques like Permutation Language Modeling (PLM), Two-stream attention, Segment Recurrence, and Relative Encoding. Due to the careful design of the model and strategic pre-training techniques, XLNet has achieved substantial performance over other popular models like BERT, making it -after appropriate fine-tuning- a capable tool for enhancing various aspects of the cybersecurity field.

8) ProphetNet: ProphetNet LLM, proposed by Microsoft, is a sequence-to-sequence pre-trained model that aims to address the issue of overfitting on strong local correlations by leveraging two novel techniques, namely: future n-gram prediction and n-stream self-attention [103]. Built on an encoder-decoder architecture and trained on 16GB base-scale and 160GB large-scale datasets sourced from web data and books, ProphetNet, with its 550M parameters, achieved new state-of-the-art results on multiple benchmarks. The model was also fine-tuned for two downstream tasks, Question Generation and Text Summarization, where it achieved the best performance. Therefore, utilizing ProphetNet in cybersecurity tasks such as automated security incident summarization could significantly enhance efficiency and decision-making.

9) Falcon: Falcon LLM, built on decoder-only architecture, was introduced by the Technology Innovation Institute (TII) as a proof-of-concept that enhancing data quality can significantly improve the LLM performance even with purely web-sourced data [104]. This insight is increasingly relevant as scaling in LLMs, which is becoming more prevalent, requires more data for processing. The model has three versions (i.e., 7B, 40B, 180B) pre-trained on the "RefinedWeb" dataset proposed by TII. RefinedWeb, sourced exclusively from web data, was subjected to various filtering and deduplication techniques to ensure high quality. Autoregressive training, Flash Attention, and ALiBi Positional encoding were the methods used for pre-training. With further fine-tuning, Falcon can advance cybersecurity, particularly in threat intelligence and analysis.

10) Reformer: Striving to address common memory limitations in LLMs, Google proposed the Reformer, an encoder-decoder memory-efficient LLM [105]. With up to 6B parameters, Reformer was pre-trained on web data using techniques including Locality-Sensitive Hashing (LSH) Attention, Chunked Processing, Shared-QK Attention Heads, and Reversible layers. These techniques were proven to have a negligible impact on the training process compared to the standard Transformer, as the Reformer achieved results that matched

the full Transformer but with much faster processing and better memory efficiency. Subsequently, employing Reformer for tasks like large-scale data analysis could serve the cybersecurity field by enabling more efficient processing and analysis of extensive datasets.

11) PaLM: Driven by the advancement in machine learning and natural language processing, Google has developed PaLM to examine the impact of scale on few-shot learning [106]. PaLM, built on decoder-only architecture, was trained with 540B parameters using Pathways, a new system that utilizes highly efficient training across multiple TPU pods. The model was trained on 2TB of data from multiple sources, including news articles, Wikipedia, source code, etc. SwiGLU Activation, Parallel Layers, and other techniques were deployed for pre-training three different parameter scales, 8B, 62B, and 540B, to understand the scaling behavior better. An observed discontinuous improvement indicated that as LLMs reach a certain level of scale, they exhibit new abilities. Furthermore, these emerging capabilities continue to evolve and become apparent even beyond the scales that have been previously explored and documented. Subsequently, PaLM achieved a breakthrough by outperforming the finetuned state-of-the-art and average human on some benchmarks, proving that when scaling is combined with chain-of-thought prompting, basic few-shot evaluation has the potential to equal or surpass the performance of fine-tuned state-of-the-art models across a broad spectrum of reasoning tasks. With such strong capabilities, utilizing PaLM for tasks like generating security policies and incident response automation can enhance the efficiency and effectiveness of cybersecurity operations.

12) PaLM2: PaLM2 is an advanced variant of the PaLM model that is more compute-efficient, although it offers better multilingual and reasoning capabilities [107]. The key enhancements in the model are the improved dataset mixtures, the compute-optimal scaling, and architectural and objective improvements. The significant evaluation results of PaLM2 indicate that various approaches could elaborate on the model's enhancement besides scaling, such as meticulous data selection and efficient architecture/objectives. Moreover, the fact that PaLM2 outperformed the predecessor PaLM despite its significantly smaller size shows that the model quality has a greater influence on the performance than the model size as it could enable more efficient inference, reducing serving costs and potentially allowing for broader applications and accessibility to more users.

13) LLaMA: Proposed by Meta, the LLaMA decoder-only model is a proof-of-concept that it's possible to achieve state-of-the-art performance by training exclusively on publicly available data [108]. LLaMA, with multiple variants ranging between 7 and 65 billion parameters, was trained on 1400B tokens of publicly available datasets, including CommonCrawl, C4, arXiv, and others. Interestingly, the techniques used for training the model were inspired by multiple popular models like GPT-3 (Pre-normalization), PaLM (SwiGLU activation function), and GPTNeo (Rotary Embedding). As a result of this incorporation, LLaMA-13B was able to outperform GPT-3(175B) on most benchmarks despite it being more than ten times smaller, while LLaMA-65B has shown to be competitive with Chinchilla-70B and PaLM-540B. Given its relatively small size and superior performance, fine-tuning LLaMA on cyber threat intelligence tasks could significantly enhance the security of edge devices.

14) LLaMA2: LLaMA2 is an optimized version of LLaMA developed by Meta and a collection of pre-trained and fine-tuned LLMs with sizes ranging from 7 to 70B parameters [109]. In the pre-training, a mixture of publicly available data was used for up to 2000B training tokens. Moreover, multiple techniques were used in the predecessor LLaMA, such as Pre-normalization, SwiGLU activation function, and Rotary positional embeddings. Two additional methods, namely increased context length and group-query attention (GQA), were also used. After pre-training, variants of the model (i.e., LLaMA2-Chat) were optimized for dialog use cases by supervised fine-tuning and reinforcement learning with human feedback (RLHF). The model evaluation, which focused on helpfulness and safety, showed superiority over the other open-source models and competitive performance to some closed-source models.

15) GShard: GShard LLM was introduced by Google in 2020, aiming to address neural network scaling issues related to computation cost and training efficiency [110]. Based on a Mixture-of-Experts (MoE) transformer with 600B parameters, GShard was pre-trained on 1000B tokens of web data. Multiple techniques were deployed for the training stage, such as conditional computation, XLA SPMD partitioning, position-wise MoE, and parallel execution using annotation APIs. Subsequently, GShard outperformed prior models in translation tasks and exhibited a favorable trade-off between scale and computational cost, resulting in a practical and sample-efficient model. These results highlight the importance of considering training efficiency when scaling LLMs, which makes it more viable in the real world.

16) ELECTRA: The extensive computation cost of MLM pre-training methods has inspired Google to propose ELECTRA LLM, which is a 335M parameters' encoder-only transformer model that utilizes a novel pre-training approach called "replaced token detection" [111]. This technique allows the model to learn from the entire sequence rather than just a small portion of masked tokens. Given that the quality and diversity of ELECTRA training data play a pivotal role in its ability to generalize across tasks, the model was trained on a vast Books Corpus and English Wikipedia. Pre-training techniques were utilized, including replaced token detection, generator-discriminator framework, token replacement, and weight-sharing. As a result, ELECTRA was able to perform comparably to popular models like RoBERTa and XLNet when using less than 25% of their compute and outperform them when using equivalent compute. Deploying such a robust model in the security field after fine-tuning can provide an efficient solution for detecting and mitigating sophisticated cyber threats, thanks to its nuanced understanding of context and language patterns.

17) MPT-30B: MPT-30B LLM is a decoder-only transformer introduced by MosaicML after the notable success of MPT-7B [112]. The model has multiple variants, the base model and two fine-tuned variants, namely MPT-30B-

Instruct and MPT-30B-Chat. Training the model on a variety of datasets such as C4, CommonCrawl, and arXiv, among others, besides the strategic selection of pre-training methods like FlashAttention and ALiBi positional encoding, have contributed to a robust performance, surpassing even the original GPT-3 benchmarks. MPT-30B has also significantly performed in programming tasks, outperforming some open-source models designed specifically for code generation. With these capabilities, deploying MPT-30B in cybersecurity could substantially enhance threat detection and response systems. Its adeptness at understanding and generating programming languages promises advancements in automated vulnerability assessment and developing sophisticated security protocols.

18) Yi-34B: The newly released LLM Yi-34B developed by 01.AI is getting attention as one of the best open-source LLMs [113]. Given the recent release of the model, its technical paper has not yet been published; hence, the available information is limited. The model has multiple variants: base and chat models, some quantized. All variants are trained on a dataset containing Chinese and English only, and the chat versions have gone through supervised fine-tuning, resulting in more efficient models for downstream tasks. The base model outperformed many open LLMs in certain benchmarks, including renowned ones like LLaMA2-70B and Falcon-180B. Even the quantized versions have demonstrated impressive performance, paving the way for their deployment in cybersecurity applications, such as edge security solutions.

19) Falcon2-11B: Falcon2-11B LLM [114] built by TII, is a decoder-only model with 11 billion parameters, trained on an immense corpus of text data totaling over 5,000 billion tokens. In terms of performance, Falcon2-11B showcases impressive capabilities, supporting 11 languages: English, German, Spanish, French, Italian, Portuguese, Polish, Dutch, Romanian, Czech, and Swedish. While it excels in generating human-like text, it also carries the biases and stereotypes prevalent in its training data, a common challenge LLMs face. To address this, TII recommends fine-tuning the model for specific tasks and implementing guardrails for production use. In the training process of Falcon2-11B, they utilized a four-stage strategy with increasing context lengths; in the final stage, they reached 8162 context lengths. This stage focused on enhancing performance using high-quality data. Additionally, the training leveraged 1024 A100 40GB GPUs and a custom distributed training codebase named Gigatron, which employs a 3D parallelism approach combined with ZeRO, high-performance Triton kernels, and FlashAttention-2 for efficient and effective training.

B. LLMs performance in the hardware cybersecurity

Table IV compares 19 publicly available LLMs' performance in Hardware design-related bug detection and security issues identification using samples from various sources. A portion of the Chrystalis dataset [87] has been used to evaluate the performance of the LLM models in bug detection tasks. A set of faults has been injected intentionally into a functional code and labeled as faulty. The sample size that has been processed comprises 10K of hardware design-related code samples. The prompt that has been used instructs the model to check the concerned code for any issue or bug and respond only with yes or no. The result is presented as the ratio of the responses where the model successfully identified a buggy code from the total samples used. The hardware CWE column evaluates the capability of the models to link a buggy code to its CWE number. The prompt has been designed to ask for a well-defined CWE number on the buggy design. This evaluation process asses the capability of an LLM model in bug detection and classification into the correct CWE class number.

The top performers in this evaluation in terms of design bug detection are LLama3 and Mixtral. While the LLama3 model performs better in the bug detection tasks, it lacks the proper identification of the CWE issue related to the faulty section. Mixtral models show less performance at identifying bugs but higher diversity in identifying a bug's security impact on the overall design implementation. The outcomes of this experiment reveal that some models cannot identify the right issues with the source code, which might require further refinement of the used prompt and/or fine-tuning the general-purpose models on bug-locating tasks. The results also show that the model size doesn't greatly impact the model performance at locating the bugs nor reasoning about their according impact (CWE class identification). While the samples that have been picked do not exceed the context length of the selected models, the token size of the model itself might reveal a superiority for the larger models when dealing with large source codes. However, superior bug identification and reasoning are also required to provide the required performance.

In conclusion, the highlighted results reveal that the existing models might be subject to weaknesses in identifying bugs in Hardware designs that might lead to security-related issues. The two-step evaluation process gives better visibility in building more robust dedicated LLMs for Hardware design security evaluation. Models that properly locate bugs do not show similar performance in classifying the bug's impact on the overall design. The outcomes could be evaluated with a larger sample size and a more dedicated study at a large scale to get conclusive results.

C. LLMs performance in the cybersecurity knowledge

Table V compares various 42 LLMs performance in the cybersecurity domain using CyberMetric dataset [89]. Figure 5 presents the LLMs performance steps. The models are evaluated based on their accuracy across four question sets: 80 questions, 500 questions, 2000 questions, and 10,000 questions. The performance is represented in percentage accuracy, offering a comprehensive view of each model's proficiency in handling cybersecurity-related queries.

The top performers in this evaluation are the GPT-4 and GPT-4-turbo models by OpenAI. These models demonstrate exceptional performance, with GPT-4 achieving 96.25% accuracy on the 80-question set and maintaining high accuracy with 88.89% on the 10,000-question set. GPT-4-turbo closely follows with similar accuracy percentages. Both models are proprietary and developed by OpenAI, indicating a high

optimization level for specialized tasks within a controlled environment. Another strong performer is the Mixtral-8x7B-Instruct by Mistral AI, which boasts accuracy of 92.50% on the 80-question set and 87.00% on the 10,000-question set. This model is open-source under the Apache 2.0 license, demonstrating the potential of community-driven development in achieving high performance. Additionally, GEMINI-pro 1.0 by Google shows robust performance, achieving 90.00% accuracy on the 80-question set and 87.50% on the 10,000-question set, highlighting the capabilities of large-scale corporate research and development in LLMs.

Mid-tier performers include models like Yi-1.5-9B-Chat by 01-ai and Hermes-2-Pro-Llama-3-8B by NousResearch. Yi-1.5-9B-Chat performs reasonably well with an 87.50% accuracy on the 80-question set, tapering to 76.04% on the 10,000-question set. Under the Apache 2.0 license, this model shows a balance between open-source collaboration and performance. Hermes-2-Pro-Llama-3-8B achieves 86.25% accuracy on the 80-question set and 77.33% on the 10,000-question set, further underscoring the effectiveness of collaborative research efforts.

Lower-tier performers include models like Qwen1.5-7B by Qwen. Qwen1.5-7B scores 73.75% on the 80-question set, dropping to 59.79% on the 10,000-question set. As an open model, Qwen1.5-7B indicates the challenges faced by smaller models in maintaining high accuracy with increasing question set sizes. Falcon-40B-Instruct achieves 67.50% accuracy on the 80-question set and 64.50% on the 10,000-question set. Licensed under Apache 2.0, it highlights the competitive landscape of open-source LLMs.

The lowest-tier performers include models such as Phi-3-mini-128k-instruct by Microsoft and Stablelm-2-1_6b by Stability AI. Phi-3-mini-128k-instruct has the lowest performance, with only 1.25% accuracy on the 80-question set and 0.88% on the 10,000-question set. Despite being from a major company like Microsoft and licensed under MIT, this model underscores the importance of continuous development and optimization in LLMs. Stablelm-2-1_6b scores 16.25% on the 80-question set, decreasing to 20.09% on the 10,000-question set, demonstrating smaller models' difficulties in scaling up effectively.

In conclusion, the table reveals that proprietary models perform better than open-source models, suggesting that controlled environments and dedicated resources may significantly enhance model performance. However, larger models do not always guarantee higher performance, as seen with some mid and lower-tier performers. Additionally, many models show a decline in accuracy as the number of questions increases, highlighting the challenges in maintaining performance consistency across larger datasets. The analysis indicates that while top-tier proprietary models lead in performance, there is significant potential within the open-source community to develop competitive models. Continuous improvements in model architecture, training data quality, and optimization techniques are crucial for advancing state-of-the-art cybersecurity knowledge within LLMs.

VI. Code-specific LLMs

The rapid evolution of technology and software development has increased the demand for specialized tools that aid in coding, debugging, and enhancing software security [115], [116]. Recognizing this need, various organizations have developed Code-specific LLMs, each offering unique features and capabilities. These models leverage advanced machine learning techniques to understand, generate, and manipulate code, thereby revolutionizing the field of software development [117], [118]. This section delves into several notable Code-specific LLMs, exploring their architectures, training methods, and potential applications in cybersecurity and beyond [119]–[122]. Table IX and Table X compare Code-specific Large Language Models.

A. Prevalent LLMs

1) SantaCoder: As part of the BigCode project, HuggingFace and ServiceNow have proposed SantaCoder LLM [123]. Based on the decoder-only architecture and with a 1.1B parameter, SantaCoder was trained on 268GB of Python, Java, and JavaScript subsets of The Stack dataset. Multiple filtering techniques were used for the training data without much impact except for one (i.e., filtering files from repositories with 5+ GitHub stars), significantly deteriorating the performance on text2code benchmarks. Pre-training methods included Multi-Query-Attention (MQA) and Fill-in-the-Middle (FIM). Although these techniques have led to a slight drop in the model's performance compared to Multi-Head-Attention (MHA) and training without FIM, the model could still outperform previous multi-lingual code models like CodeGen0Multi-2.7B and InCoder-6.7B despite being substantially smaller. Such performance can be promising if deployed in cybersecurity for tasks like software vulnerability and secure code generation.

2) StarCoder: StarCoder is another decoder-only model developed within the BigCode project [124]. With 15.5B parameters, StarCoder was pre-trained on 1000B tokens from over 80 different programming languages. The pre-training utilized techniques such as FIM and MQA and Learned Absolute Positional Embeddings. After pre-training, the base model was fine-tuned on an additional 35B tokens of Python. Compared to other Code LLMs, StarCoder outperformed all fine-tuning models on Python. Moreover, the base model outperformed OpenAI code-cushman-001. StarCoder's exceptional performance in Python and its broad training in multiple programming languages position it as a highly versatile tool for various coding tasks.

3) StarChat-Alpha: StarChat Alpha is a variant of StarCoder fine-tuned to act as a helpful coding assistant that accepts natural language prompting (considering that StarCoder needs specific structured prompting) [125]. With 16B parameters, the model was fine-tuned on a mixture of oasst1 and databricks-dolly-15k datasets. The model has not undergone RLHF or similar methods, which would have helped align it with human preferences. Nevertheless, the comprehensive pre-training of the base model contributed to the model's ability to

interpret various coding tasks and provide accurate code suggestions. This capability makes it an invaluable programming tool, simplifying code development and problem-solving.

4) CodeGen-2: Developed by Salesforce AI research, CodeGen2 was proposed as a product of extensive research in the field of LLM aimed at optimizing model architectures and learning algorithms to enhance the efficiency and reduce the costs associated with LLMs [126]. The final findings were examined in multiple variants with parameters ranging from 1B to 16B, where the 16B model is trained on 400B tokens from the Stack dataset. Causal language modeling, cross-entropy loss, and other techniques were used for pre-training, resulting in a robust program synthesis model. CodeGen2's proficiency in program synthesis makes it a valuable asset in cybersecurity applications, such as aiding in vulnerability detection and enhancing code security analysis. Its ability to understand and generate complex small models can be trained for multiple epochs with specific settings, efficient security protocols, and automated threat detection systems.

5) CodeGen-2.5: Another version of the CodeGen family is CodeGen 2.5 [127]. The 7B parameters model was introduced to prove that good models don't necessarily have to be big, especially with the trend of scaling up LLMs and the data size limitations. CodeGen 2.5 was trained on 1400B training tokens from StarCoderData. A strategic selection of pre-training techniques, such as Flash Attention, Infill Sampling, and Span Corruption, enhanced the model's performance. Moreover, that led to a good performance that is on par with popular LLMs of larger size. The results indicated that small models can be trained for multiple epochs with specific settings and achieve comparable results to bigger models.

6) CodeT5+: CodeT5+ is an encoder-decoder transformer proposed by Salesforce AI Research to address some code LLMs limitations [128]. Specifically, those related to the architecture being either inflexible or serving as a single system and pre-training task limitations related to a limited set of pre-training objectives can result in a substantial degradation in performance. The proposed model has different variants ranging from 220M to 16B parameters. Trained on 51.5B tokens from CodeSearchNet and GitHub code datasets using techniques like span denoising, contrastive learning, and others, the model achieved new state-of-the-art results on various code-related tasks like code generation, code completion, etc. A model with such capabilities can be valuable to cybersecurity for threat intelligence and software vulnerability.

7) XGen-7B: Another production of Salesforce AI Research is XGen-7B LLM, a decoder-only transformer with 7B parameters [129]. The model was developed to address the problem of sequence length constraints in the available open-source LLMs as many tasks require inference over an input context. XGen-7B, with up to 8K sequence length, was trained on 1500B tokens from a mixture of text and code data. Techniques like standard dense attention and a two-stage training strategy were utilized for pre-training. Additionally, the model was enhanced with instructional tuning, a technique that refines its responses to align closely with specific user instructions. As a result, XGen-7B achieved comparable or better results than other 7B state-of-the-art open-source LLMs.

8) Replit code v1: Proposed by Replit, Inc., the 2.7B parameters causal language model Replit-code-v1-3b, with a focus on code completion, was trained on 525B tokens from a subset of the stack Dedup v1.2 dataset [130]. The model underwent advanced pre-training techniques such as Flash Attention for efficient computation, AliBi positional embeddings for enhanced context interpretation, and the LionW optimizer for improved training dynamics. The Replit code v1 model is also available in two quantization options: 8-bit and 4-bit. The Replit-code-v1-3b model's capabilities in understanding and generating code make it particularly suited for cybersecurity applications, such as automating the detection of code vulnerabilities and generating secure coding patterns. Additionally, its quantized versions can be utilized for edge security.

9) DeciCoder-1B: DeciCoder-1B is an open-source 1B parameter decoder-only transformer developed by Deci AI with a 2048-context window [131]. Subsets of Python, Java, and JavaScript from the StarCoderData dataset were used for training. The model architecture was built using Automated Neural Architecture Construction (AutoNAC) developed by the company, which is a technology designed to automatically create and optimize deep learning models, particularly neural networks, for specific tasks and hardware environments. Moreover, Grouped Query Attention (GQA) and FIM were utilized to pre-train the model. Consequently, the model has shown smaller memory usage compared to popular code LLMs like StarCoder and outperformed SantaCoder in the languages it was trained on with remarkable inference speed.

10) CodeLLAMA: Based on LLAMA 2, CodeLLAMA was introduced by Meta as a decoder-only transformer code LLM [132]. With variants ranging from 7 to 34B parameters of base, python specialized, and instruction-following models, all trained on text and code from multiple datasets, CodeLLAMA emerges as a comprehensive suite of models, adept at handling a wide array of programming-related tasks. Causal infilling, Long-context fine-tuning, and other techniques were utilized for pre-training and fine-tuning. CodeLLAMA models' family achieved state-of-the-art performance in multiple benchmarks, indicating their potential for transformative applications in cybersecurity. Their advanced code analysis and generation capabilities could be crucial in automating threat detection and enhancing vulnerability assessments.

11) CodeQwen1.5-7B: CodeQwen1.5-7B-Chat [133] is a transformer-based decoder-only language model trained on 3 trillion tokens of code data. It supports 92 coding languages and has strong code-generation capabilities. The model can understand and generate long contexts of up to 64,000 tokens and has shown excellent performance in text-to-SQL and bug-fixing tasks. It is based on Qwen1.5, which offers eight model sizes, including 0.5B, 1.8B, 4B, 7B, 14B, 32B, and 72B dense models, and an MoE model of 14B with 2.7B activated.

12) DeepSeek Coder-33B-instruct: Deepseek Coder [134] is a series of code language models, with each model trained from scratch on 2 trillion tokens, 87% of which are code and 13% natural language in English and Chinese. The model comes in various sizes, ranging from 1B to 33B, with the 33B model being fine-tuned on 2 billion tokens of instruction

data. It achieves state-of-the-art performance among open-source code models on multiple programming languages and benchmarks.

13) CodeGemma-7B: CodeGemma [135] is a collection of lightweight open code models built on top of Gemma. It is a text-to-text and text-to-code decoder-only model with 7 billion parameters, specializing in code completion and generation tasks. It can answer questions about code fragments, generate code from natural language, or discuss programming or technical problems. CodeGemma was trained on 500 billion tokens of primarily English language data from publicly available code repositories, open-source mathematics datasets and synthetically generated code.

14) Granite 8B Code: IBM released a family of Granite code models [136], including the Granite-8B-Code-Base, to make coding more accessible and efficient for developers. Granite-8B-Code-Base is a decoder-only code model designed for code generation, explanation, and fixing. It is trained in two phases: first on 4 trillion tokens from 116 programming languages, then on 500 billion tokens from a carefully designed mixture of high-quality code and natural language data. This two-phase training strategy ensures the model can reason and follow instructions while understanding programming languages and syntax.

15) DeepSeek-V2: DeepSeek-V2 [137] is a mixture-of-experts (MoE) language model with 236 billion parameters, of which 21 billion are activated for each token. It is a significant upgrade from the previous DeepSeek model, offering stronger performance while reducing training costs by 42.5%. The model was pre-trained on a vast and diverse corpus of 8.1 trillion tokens, followed by supervised fine-tuning and reinforcement learning to maximise its capabilities. DeepSeek-V2 excels at live coding tasks and open-ended generation, supporting both English and Chinese.

B. Datasets Development for Code-centric LLM Models

The development of large-scale datasets has played a crucial role in advancing LLM models, especially those focused on understanding and generating code. Table XI presents the datasets used for pre-training foundation models in Coding. Datasets like CodeSearchNet [145] and The Pile [146] have been instrumental in bridging the gap between natural language and code, improving semantic search capabilities, and enhancing language model training across diverse domains. These datasets provide a rich source of real-world code in multiple programming languages and include expert annotations and natural language queries that challenge and push the boundaries of LLM performance in code-related tasks.

Over time, the focus has shifted towards increasing the size, diversity, and ethical considerations of the data used in training AI models. Introducing datasets such as ROOTS and The Stack v2 [149] reflects a growing emphasis on responsible LLM development. These newer datasets encompass a broader range of programming languages and coding scenarios, and they incorporate governance frameworks to ensure the ethical use of the data. In addition, these datasets are designed to address the needs of large multilingual language models and the specific challenges of code generation and comprehension, demonstrating the evolving landscape of LLM research driven by enhanced dataset quality and scope.

C. Vulnerabilities Analysis of LLM-Generated Code

The evolution of LLMs in software development has brought significant advancements and new security challenges [158]. Table XII presents a comparative analysis of vulnerabilities in LLM-generated code.

Schuster et al. [150] demonstrate how LLMs employed in code autocompletion are susceptible to poisoning attacks, which can manipulate the model's output to suggest insecure code. This vulnerability is intensified by the ability to target specific developers or repositories, making the attacks more effective and difficult to detect. Despite defenses against such attacks, their effectiveness remains limited, raising concerns over the secure deployment of these technologies [150].

Recent studies, such as those by Asare et al. [151] and Sandoval et al. [152], provide an empirical and comparative analysis of the security aspects of code generated by LLMs like GitHub's Copilot and OpenAI Codex. Asare et al. [151] find that while Copilot occasionally replicates vulnerabilities known from human-written code, it does not consistently do so across different vulnerabilities. In contrast, Sandoval et al. [152] report a minimal increase in security risks when developers use LLMs in coding, indicating that LLMs do not necessarily degrade the security of the code more than human developers would.

Moreover, Perry et al. [153] reveal a concerning trend where users interacting with AI code assistants tend to write less secure code but believe otherwise. Their findings underscore the need for heightened awareness and better design of user interfaces to foster critical engagement with the code suggestions provided by LLMs [153]. In a similar vein, Hamer et al. [154] emphasize the educational gap among developers regarding the security implications of using code snippets from AI like ChatGPT or traditional sources like StackOverflow, highlighting that both sources can propagate insecure code.

Lastly, novel tools like DeVAIC introduced by Cotroneo et al. [155] and comprehensive vulnerability evaluations in LLM-generated web application code by Tóth et al. [156] and Tihanyi et al. [157] illustrate ongoing efforts to understand better and mitigate the risks associated with AI-generated code. DeVAIC, for instance, offers a promising approach to detecting vulnerabilities in incomplete Python code snippets, potentially enhancing the security assessment capabilities for AI-generated code.

VII. CyberSecurity Datasets for LLMs

A. Cyber Security Dataset Lifecycle

Creating a cybersecurity dataset for use with LLMs involves several steps that ensure the dataset is comprehensive, accurate, and effective for training or evaluating the models. Figure 6 presents the cyber security dataset lifecycle for LLM development.

TABLE VI: Comparison of Large Language Models

Model	Architecture	Base Model	Parameters	Training Tokens	Pre-training	Corpus Volume	Released By	Applications	Use Cases in Cybersecurity	Training Scheme	Key Training Techniques	Quantization	Ref
GPT-3	Decoder-only	NA	175B	300B	Books, Web text, Wikipedia, Common Crawl	+570GB	Open AI	Language Modeling, Text Completion, QA	Malware Detection, Threat Intelligence, Social Engineering Detection	Pre-training, In-context learning	Autoregressive training, Scaled Cross Entropy Loss, Backpropagation and gradient descent, Mixed precision training.	NA	[96]
GPT-4	Decoder-only	NA	NA	NA	Web Data, Third-party licensed data	NA	Open AI	Language Modeling, Text Completion, QA	Malware Detection, Threat Intelligence, Social Engineering Detection	Pre-training, RLHF	Autoregressive training	NA	[97]
T5	Encoder-decoder	NA	11B	1000B	C4, Web Text, Wikipedia	750GB	Google	Language Modeling, Summarization, Translation	Malware Detection, Threat Intelligence, Social Engineering Detection	Pre-training, Fine-tuning	Text-to-text framework, Denotation-based pretraining	NA	[98]
BERT	Encoder-only	NA	340M	250B	BooksCorpus, English Wikipedia	126GB	Google	Language Modeling, Classification, QA, NER	Malware Detection, Threat Intelligence, Intrusion Detection, Phishing Detection	Pre-training	Masked LM(MLM), Next-sentence prediction(NSP)	NA	[99]
ALBERT	Encoder-only	BERT	235M	+250B (calculated)	BooksCorpus, English Wikipedia	NA	Google	Language Modeling, Classification	Malware Detection, Threat Intelligence, Intrusion Detection, Phishing Detection	Pre-training	Factorized embedding parameterization, Cross-layer parameter sharing, Inter-sentence coherence loss, Sentence order prediction (SOP)	NA	[100]
RoBERTa	Encoder-only	BERT	355M	2000B	BooksCorpus, English Wikipedia	NA	Meta	Language Modeling, Classification, QA, NER	Malware Detection, Threat Intelligence, Intrusion Detection, Phishing Detection	Pre-training	Dynamic Masking, Full-Sentences without NSP loss, Large mini-batches, Larger byte-level BPE	NA	[101]
XLNet	Encoder-only	Transformer-XL	340M	+2000B (calculated)	English Wikipedia	158GB (calculated)	CMU, Google	Language Modeling, Classification, QA	Malware Detection, Threat Intelligence, Intrusion Detection, Phishing Detection	Pre-training	Permutation LM(PLM), Two-stream self-attention, Segment Recurrence and Relative Encoding	NA	[102]
ProphetNet	Encoder-decoder	NA	550M	+260B (calculated)	Web Data, Books	160GB	Microsoft Research Asia	Language Modeling, Question Generation, Summarization	Cybersecurity Reporting, Threat Intelligence	Pre-training, Fine-tuning	Masked Sequence generation, Autoregressive training, Denoising Autoencoder objective, Shared Parameters between encoder and decoder, Maximum Likelihood Estimation (MLE)	NA	[103]
Falcon	Decoder-only	NA	7-180B	5000B	Web Data	NA	TII	Language Modeling, Text Completion, QA	Malware Detection, Threat Intelligence, Social Engineering Detection	Pre-training	Autoregressive training, FlashAttention, ALiBi Positional encoding	NA	[104]
Reformer	Encoder-decoder	NA	Up to 6B	+150B (calculated)	Web Data	NA	Google	Language Modeling, Classification	Malware Detection, Threat Intelligence, Intrusion Detection, Phishing Detection	Pre-training	Locality-Sensitive Hashing (LSH) Attention, Chunked Processing, Shared-QK Attention Heads, Reversible layers	NA	[105]

TABLE VII: Continued

Model	Architecture	Base Model	Parameters	Training Tokens	Pre-training	Corpus Volume	Released By	Applications	Use Cases in Cybersecurity	Training Scheme	Key Training Techniques	Quantization	Ref
PaLM	Decoder-only	NA	540B	780B	Webpages, books, Wikipedia, news articles, source code, social media conversations, GitHub	2TB	Google	Language Modeling, QA, Translation	Threat Intelligence, Security Policies Generation	Pre-training	SwiGLU Activation, Parallel Layers, Multi-Query attention (MQA), RoPE embeddings, Shared Input-Output embedding	NA	[106]
PaLM2	Decoder-only	NA	NA	NA	web documents, books, code, mathematics, conversational data	NA	Google	Language Modeling, QA, Summarization	Threat Intelligence, Security Policies Generation	Pre-training	Compute optimal scaling, Canary token sequences, Control tokens for inference	NA	[107]
LLaMA	Decoder-only	NA	7-65B	1400B	CommonCrawl, C4, GitHub, Wikipedia, Books, arXiv, StackExchange	177GB	Meta	Language Modeling, Text Completion, QA	Threat Intelligence, Malware Detection	Pre-training	Pre-normalization, SwiGLU activation function, Rotary Embedding, Model and sequence parallelism	NA	[108]
LLaMA2	Decoder-only	NA	7-70B	2000B	Mix of pulically available data	NA	Meta	Language Modeling, Text Completion, QA	Threat Intelligence, Malware Detection	Pre-training, Fine-tuning, RLHF	Optimized autoregressive training, Grouped Query Attention (GQA)	NA	[109]
GShard	MoE	NA	600B	1000B	Web Data	NA	Google	Language Modeling	Threat Intelligence, Intrusion Detection, Malware Detection	Pre-training	Conditional Computation, Lightweight Annotation APIs, XLA SPMD partitioning, Position-wise MoE	NA	[110]
ELECTRA	Encoder-only	NA	335M	+1800B (calculated)	BooksCorpus, English Wikipedia	158GB	Google	Language Modeling, Classification	Threat Intelligence, Intrusion Detection, Malware Detection, Phishing Detection	Pre-training, Fine-tuning	Replaced token detection, Generator-discriminator framework, Token replacement, Weight-sharing	NA	[111]
MPT-30B	Decoder-only	NA	30B	1000B	C4, mC4, CommonCrawl, Wikipedia, Books, arXiv	NA	MosaicML	Language Modeling, Text Completion, QA	Threat Intelligence, Malware Detection, Software Vulnerability	Pre-training	FlashAttention, ALiBi positional encoding	NA	[112]
Yi-34B	NA	NA	34B	3000B	Chinese and English dataset	NA	01.AI	Language Modeling, Question Answering	Threat Intelligence, Phishing Detection, Vulnerability Assessment	Pre-training, Fine-tuning	NA	GPTQ, AWQ	[113]
Phi-3-mini	Decoder-only	NA	3.8B	3.3T	Phi-3 datasets (Public documents, synthetic, chat formats)	NA	Microsoft	Language Modeling, Text Completion, QA	Threat Intelligence, Intrusion Detection, Malware Detection	Pre-training, Fine-tuning	LongRope, Query Attention (GQA)	NA	[138]
Mistral 7B	Decoder-only	NA	7.24B	NA	NA	NA	Mistral AI	Language Modeling, Text Completion, QA	Threat Intelligence, Intrusion Detection, Malware Detection	Pre-training, Fine-tuning	Sliding Window Attention, Query Attention (GQA), Byte-fallback BPE tokenizer	NA	[139]
Cerebras-GPT 2.7B	Decoder-only	NA	2.7B	371B	The Pile Dataset	825 GB	Cerebras	Language Modeling, Text Completion, QA	Threat Intelligence, Intrusion Detection, Malware Detection	Pre-training	standard trainable positional embeddings and GPT-2 transformer, GPT-2/3 vocabulary and tokenizer block	NA	[140]
ZySec-AI/ZySec 7B	Decoder-only	NA	7.24B	NA	Trained across 30+ domains in cybersecurity	NA	ZySec AI	Language Modeling, Text Completion, QA	Expert guidance in cybersecurity issues	Pre-training	NA	NA	[141]
DeciLM 7B	Decoder-only	NA	7.04	NA	NA	NA	Deci	Language Modeling, Text Completion, QA	Threat Intelligence, Intrusion Detection, Malware Detection	Pre-trained	Grouped-Query Attention (GQA)	NA	[142]

TABLE VIII: Continued

Model	Architecture	Base Model	Parameters	Training Tokens	Pre-training	Corpus Volume	Released By	Applications	Use Cases in Cybersecurity	Training Scheme	Key Training Techniques	Quantization	Ref
Zephyr 7B Beta	Decoder-only	Mistral 7B	7.24B	NA	NA	NA	Hugging-Face	Language Modeling, Text Completion, QA	Threat Intelligence, Intrusion Detection, Malware Detection	Fine-tuning	Flash Attention, Direct Preference Optimization (DPO)	NA	[143]
Dolly v2 12B	Decoder-only	Pythia 12B	12B	3T	The Pile Dataset	825GiB	Databricks	Language Modeling, Text Completion, QA	Threat Intelligence, Intrusion Detection, Malware Detection	Fine-tuning	NA	NA	[144]
Falcon2 11B	Decoder-only	NA	11.1B	5T	RefinedWeb enhanced with curated corpora.	NA	TII	Language Modeling, Text Completion, QA	Malware Detection, Threat Intelligence, Social Engineering Detection	Pre-training	ZeRO, high-performance Triton kernels, FlashAttention-2	NA	[114]

1) Define Objectives: Defining the objectives for a cybersecurity dataset for LLMs is crucial as it dictates its construction and application. For training purposes, the dataset should cover various cybersecurity topics and incorporate various data types like text, code, and logs, aiming to develop a robust and versatile LLM capable of understanding diverse threats (e.g., Edge-IIoT dataset [159] for Network Security and FormAI dataset [157], [160] for Software Security). For evaluation, the focus narrows to specific areas, such as benchmarking the LLMs' knowledge in cybersecurity (e.g., CyberMetric [89]).

2) Scope and Content Gathering: For the scope and content gathering stage of building a cybersecurity dataset aimed at training and fine-tuning LLMs, selecting a broad range of topics is essential to ensure comprehensive coverage. Key areas include network security, malware analysis, software security, cryptographic protocols, cloud security, and incident response. The data should be sourced from diverse and reliable origins, such as public and private databases such as Common Weakness Enumeration (CWE) and Common Vulnerabilities and Exposures (CVE) [161], [162].

3) Data Cleaning and Preprocessing: This process involves filtering out irrelevant content to maintain a focus on cybersecurity and standardizing formats across the dataset. For example, processing the Starcoder 2 dataset [149] involves several meticulous steps to refine GitHub issues collected from GHArchive. Initially, auto-generated texts from email replies and brief messages under 200 characters are removed, along with truncating longer comments to maintain a maximum of 100 lines while preserving the last 20 lines. This step alone reduced the dataset volume by 17%. The dataset then undergoes further cleaning to remove comments by bots identified through specific keywords in usernames, eliminating an additional 3% of the issues. A notable focus is placed on the interaction quality within the issues; conversations with two or more users are prioritized, and those with extensive text under a single user are preserved if they stay under 7,000 characters. Issues dominated by a single user with more than ten events are excluded, recognizing them as potentially low-quality or bot-driven, resulting in a 38% reduction of the remaining dataset. For privacy, usernames are anonymized by replacing them with a sequential participant counter, maintaining confidentiality while preserving the integrity of conversational dynamics.

4) Annotation and Labeling: A sophisticated hybrid approach can be adopted to ensure precision and scalability in the annotation and labeling stage of developing a cybersecurity dataset for LLMs. Cybersecurity experts manually annotate the dataset, meticulously labeling complex attributes such as threat type, guaranteeing high accuracy. Concurrently, automated tools like static analyzers (e.g., Clang for C/C++ and Bandit for Python), formal verification methods (e.g., ESBMC), and dynamic tools are employed to handle the large volume of data efficiently. These tools initially tag the data, which human experts carefully review and correct [163].

B. Software Cyber Security datasets

In software cyber security, datasets play a crucial role in understanding, detecting, and mitigating vulnerabilities in software systems. This sub-section explores several significant software cybersecurity datasets, each offering unique insights and methodologies for vulnerability analysis in cybersecurity. From the extensive BigVul dataset, which links vulnerabilities in the CVE database to specific code commits, to the innovative FormAI dataset, leveraging AI-generated C programs and advanced verification methods for precise vulnerability classification, each dataset contributes uniquely to the field. These datasets range from manually labeled by security experts to those generated using state-of-the-art automated tools, providing diverse resources for researchers and practitioners. Table XIII provides an overview of software vulnerability datasets that can be used for fine-tuning LLMs for software security.

1) Sate IV - Juliet dataset: Nist has developed the SARD Juliet dataset to assess the capabilities of static analysis tools on C/C++ program code out of many other programming languages. The dataset contains the source files, with each test case containing bad functions and good functions that patch the vulnerable "bad" code. Test cases are labeled with CWEs to indicate the type of vulnerability exposed in the program.

TABLE IX: Comparison of Code-specific Large Language Models

Model	Architecture	Base Model	Parameters	Training Tokens	Pre-training	Corpus Volume	Released By	Applications	Use Cases in Cybersecurity	Training Scheme	Key Training Techniques	Quantization	Ref
SantaCoder	Decoder-only	NA	1.1B	236B	The Stack v1.1 dataset (Python, Java, and JavaScript)	268GB	HuggingFace, ServiceNow	Code Generation, Code Completion, Code Analysis, QA	Threat Intelligence, Software Vulnerability, Source Code Generation	Pre-training	Multi Query Attention (MQA), Fill-in-the-Middle (FIM)	NA	[123]
StarCoder	Decoder-only	NA	15.5B	PT 1000B, FT 35B	80+ programming languages, Git commits, GitHub issues, and Jupyter notebooks	+800GB	HuggingFace, ServiceNow	Code Generation, Code Completion, Code Analysis, QA	Threat Intelligence, Software Vulnerability Detection	Pre-training, Fine-tuning	Fill-in-the-Middle (FIM), Multi Query Attention (MQA), Learned absolute positional embeddings	NA	[124]
StarChat Alpha	Decoder-only	StarCoderbase	16B	NA	oasst1 and databricks-dolly-15k datasets	NA	HuggingFace, ServiceNow	Code Generation, Code Completion, Code Analysis, QA	Threat Intelligence, Software Vulnerability	Fine-tuning	NA	NA	[125]
CodeGen2	Decoder-only (causal LM)	NA	1-16B	400B	Stack dataset v1.1	NA	Salesforce	Program Synthesis, Code Generation	Threat Intelligence, Software Vulnerability	Pre-training	Causal Language Modeling, Cross-entropy Loss, File-level Span Corruption, Infilling	NA	[126]
CodeGen2.5	Decoder-only (causal LM)	NA	7B	1400B	StarCoderData	NA	Salesforce	Code Generation, Code Completion, Code Analysis	Threat Intelligence, Software Vulnerability	Pre-training	Flash Attention, Infill Sampling, Span Corruption	NA	[127]
CodeT5+	Encoder-decoder	NA	220M-16B	51.5B	CodeSearchNet dataset, GitHub code dataset	NA	Salesforce	Code Generation and Completion, Math Programming, Text-to-code Retrieval Tasks	Threat Intelligence, Software Vulnerability	Pre-training	Span Denoising, Contrastive Learning, text-code Matching, Causal Language Modeling (CLM)	NA	[128]
XGen-7B	Decoder-only	NA	7B	1500B	GitHub, Several public sources, Apex code data (mixture of natural text data and code data)	NA	Salesforce	Code Generation, Summarization	Threat Intelligence, Software Vulnerability	Pre-training, Fine-tuning	Standard Dense Attention, Two-stage Training Strategy	NA	[129]
Replit Code V1	Decoder-only (causal LM)	NA	2.7B	525B	Stack Dedup v1.2 dataset (20 different languages)	NA	Replit, Inc.	Code Completion, Code Generation	Threat Intelligence, Software Vulnerability	Pre-training	Flash Attention, AliBi Positional Embeddings, LionW Optimizer	Matrix Multiplication	[130]
DeciCoder-1B	Decoder-only	NA	1B	446B	StarCoderData (Python, Java, and JavaScript)	NA	Deci	Code Completion, Code Generation, Code Analysis	Threat Intelligence, Software Vulnerability	Pre-training	Fill-in-the-Middle training (FIM), Grouped Query Attention (GQA)	NA	[131]
CodeLLAMA	Decoder-only	LLaMA2	7-34B	620B	Text and code from multiple datasets	NA	Meta	Code Completion, Code Generation, Code Analysis	Threat Intelligence, Software Vulnerability	Pre-training, Fine-tuning	Causal Infilling, Autoregressive Training, Repository-level Reasoning, Long-context Fine-tuning	NA	[132]
CodeQwen1.5-7B	Decoder-only	Qwen1.5	7.25B	3T	code-related data	NA	Qwen	Code Generation, Code Completion, Code Analysis	Threat Intelligence, Software Vulnerability, Bug fixes	Pre-training	Flash Attention, RoPE, Grouped-Query Attention (GQA)	NA	[133]
DeepSeek Coder-33B-instruct	Decoder-only	NA	33.3B	2T	Composition of code and natural language	NA	DeepSeek	Code Generation, Code Completion, Code Analysis	Threat Intelligence, Software Vulnerability	Pre-training, Long-context pre-training, Instruction fine-tuning	Flash Attention, RoPE, Grouped-Query Attention (GQA)	NA	[134]
CodeGemma-7B	Decoder-only	Gemma	8.54B	500B	Code repositories, Mathematics datasets, Synthetic code	NA	Google	Code completion, Code generation, Code chat, Instruction following	Threat Intelligence, Software Vulnerability	Pre-training, Fine-tuning	Fill-in-the-middle (FIM) tasks, dependency graph-based packing, unit test-based lexical packing	NA	[135]

TABLE X: Continued

Granite 8B Code	Decoder-only	NA	8.05B	4.05T	Publicly Datasets (GitHub Code Clean, Starcoder data)	NA	IBM Granite	Code generation, Code explanation, Code fixing, etc.	Threat Intelligence, Intrusion Detection, Malware Detection	Pre-trained in two phases (the second phase for high-quality data)	RoPE embedding, Grouped-Query Attention (GQA), Context Length of 4096 Tokens	NA	[136]
DeepSeek-V2	Decoder-only	NA	236B	8.1T	Composition of code and natural language	NA	DeepSeek	Code Generation, Code Completion, Code Analysis	Threat Intelligence, Software Vulnerability	Pre-training, SFT, RL, Long Context Extension	Mixture-of-Experts (MoE), Multi-head Latent Attention (MLA)	NA	[137]

TABLE XI: Datasets Used for Pre-training Foundation Models in Coding

Dataset	Title	Year	Purpose	Content	Significance
CodeSearchNet [145]	"CodeSearchNet Challenge: Evaluating the State of Semantic Code Search"	2019	Focuses on bridging natural language and code.	Contains about 6 million functions from six languages and 2 million automatically generated query-like annotations.	Advances the semantic code search field with a challenge including 99 queries and 4k expert annotations.
The Pile [146]	"The Pile: An 800GB Dataset of Diverse Text for Language Modeling"	2020	Designed to train large-scale language models.	Comprises 22 high-quality, diverse text subsets totaling 825 GiB.	Improves model generalization capabilities; evaluates with GPT-2 and GPT-3.
CodeParrot [1]	CodeParrot Dataset	2022	Facilitates model training in code understanding and generation.	Consists of 115M code files from GitHub in 32 programming languages, totaling 1TB.	Aids in diverse language and format model training.
The Stack [147]	"The Stack: 3 TB of permissively licensed source code"	2022	Aimed at fostering research on AI for code.	Features 3.1 TB of code in 30 programming languages.	Demonstrates improved performance on text2code benchmarks; introduces data governance.
ROOTS [148]	"The BigScience ROOTS Corpus: A 1.6TB Composite Multilingual Dataset"	2023	Supports ethical, multilingual model research.	Spans 59 languages and focuses on diverse, inclusive data.	Advances large-scale language model research with an ethical approach.
The Stack v2 [149]	"StarCoder 2 and The Stack v2: The Next Generation"	2024	Enhances foundation models for code.	Built from sources including 619 programming languages, significantly larger than its predecessor.	Shows improvements in code LLM benchmarks; ensures transparency in training data.

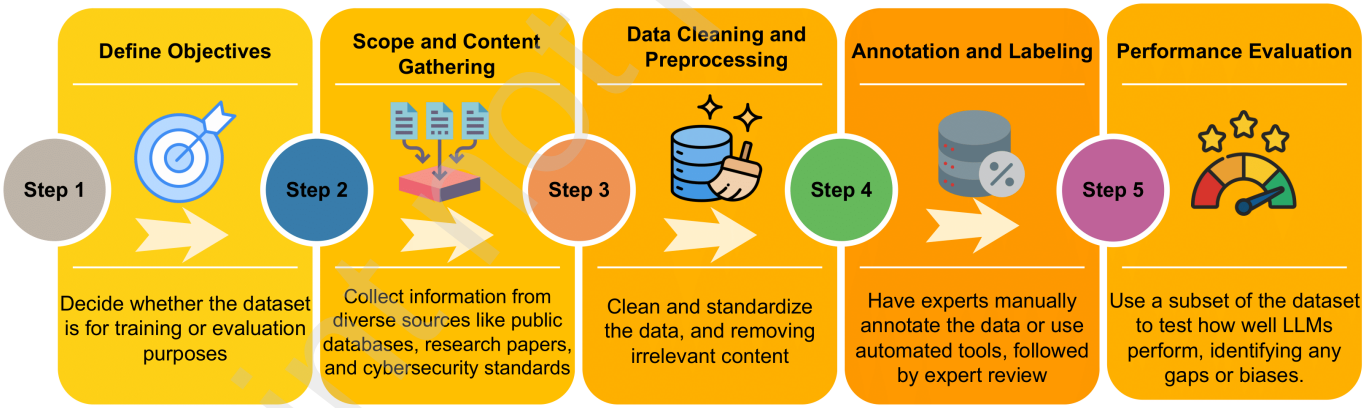

Fig. 6: Cyber Security Dataset Lifecycle for LLM development.

The dataset contains keywords to indicate precisely where vulnerable and non-vulnerable functions exist. Thus, the dataset needs careful sanitization /obfuscation. While the dataset has many vulnerability types and gives concrete examples, they are still programs purposefully built to demonstrate vulnerabilities rather than naturally occurring ones.

2) Draper dataset: Researchers in work [164] leveraged a new dataset for vulnerability detection using deep representation. A custom lexer was used to create a generic representation to capture the essential tokens while minimizing the token count. It was curated using open-source C/C++ code from SATE IV, Github, and Debian and labeled using three static analyzers. The dataset is substantial, but the vulnerability percentage is low, standing at roughly 6.8%. The dataset is multi-labelled, where more than one CWE can exist in a code sample. The dataset focuses on four main CWEs or categories, while the rest of the vulnerabilities are grouped into one class. The researchers mapped the static analyzer findings to CWEs and binary labels. Furthermore, since the researchers did the mapping, warnings, and functions flagged by static analyzers that would not typically be exploited were not flagged as vulnerable. In addition, a strict deduplication

TABLE XII: Comparative Analysis of Vulnerabilities in LLM-Generated Code

Reference	Year	Primary Focus	Methodology	Key Findings
Schuster et al. [150]	2021	Poisoning in code autocompletion	Experimental poisoning attacks on autocompleters	Demonstrated effective targeted and untargeted poisoning; current defenses are largely ineffective.
Asare et al. [151]	2023	Security analysis of GitHub's Copilot	Empirical analysis comparing human and Copilot-generated code vulnerabilities	Copilot does not consistently replicate human vulnerabilities, showing variable performance across different types.
Sandoval et al. [152]	2023	Security implications of LLM code assistants in C programming	User study with AI-assisted coding tasks	Minimal increase in security risks from LLM assistance compared to control.
Perry et al. [153]	2023	Impact of AI code assistants on security	Large-scale user study on security task performance	Participants using AI wrote less secure code but were overconfident in its security.
Hamer et al. [154]	2024	Security vulnerabilities in LLM vs. StackOverflow code	Empirical analysis of code snippets for security vulnerabilities	LLM-generated code had fewer vulnerabilities than StackOverflow, highlighting differences in security risks.
Cotroneo et al. [155]	2024	Security assessment tool for AI-generated code	Development and validation of DeVAIC tool	DeVAIC effectively identifies vulnerabilities in Python code, outperforming other tools.
Tóth et al. [156]	2024	Evaluating security of LLM-generated PHP web code	Hybrid evaluation using static and dynamic analysis	Significant vulnerabilities found in AI-generated PHP code, emphasizing the need for thorough testing.
Tihanyi et al. [157]	2024	Security of LLM-generated C code from neutral prompts	Dataset creation and analysis using formal verification	Over 63% of generated C programs were found vulnerable, with minor variations between different LLMs.

process was used to refine the dataset. The authors utilize this dataset to train their model after lexing the source code to reduce the code representation and use a limited vocabulary size. Due to lexing the source code, the approach reduces the needed vocabulary size compared to the original size required. However, the vulnerable portion is minimal compared to the dataset. Moreover, the labeling considers four categories, which are limited compared to other datasets.

3) Reveal dataset: Reveal [165] was proposed to provide an accurate dataset that reflects the real world, which is why it is also reflected in the imbalance of the samples. Their work finds that performance drops by more than 50% in real-world prediction, highlighting the need for a dataset subjected to a realistic setting. The authors focus on two open-source projects, Linux Debian and Chromium, as they are popular, well-maintained, showcase important domains, and have publicly available vulnerability reports. The data is not used as text but as Code Property Graphs (CPG), which are then converted to graph embeddings for training a Gated Graph Neural Network (GGNN). The authors use an approach inspired by Zhou *et. al* [166] to identify the security vulnerabilities in the project, and they remedy the class imbalance due to the majority of non-vulnerable code through SMOTE. Such data was collected from Bugzilla and Debian security tracker. Looking at the vulnerable portion, it constitutes 9.16% out of the 18,169 programs. While the dataset attempts to depict a realistic dataset, relying on two sole projects might limit how well a model trained on the dataset would perform in a real-world prediction case.

4) Devign dataset: Researchers of Devign [162] required an accurate dataset to be used in several graph forms, which they believe is better in reflecting the structural and logical aspects of source code. The proposed approach contains a graph embedding layer that uses Abstract Syntax Tree (AST), Control Flow Graph (CFG), Data Flow Graph (DFG), and Natural Code Sequence (NCS) to generate a joint graph representation. The rationale behind a joint representation is the ability of certain graphs to portray different vulnerability types not uncovered by others. Motivated to propose a more accurate dataset instead of those generated using static analyzers, the researchers invested in a security team to manually label the samples. The data is collected from large open-source projects: Linux, Wireshark, QEMU, and FFmpeg. The dataset is manually labeled over two rounds, with 600 hours put into labeling it. While a significant advantage of the dataset is that it is manually labeled and verified, the dataset is only binary labeled. Also, it is worth noting that only 2 out of the four datasets are available.

5) VUDENC: The VUDENC dataset [167] is comprised of 25,040 vulnerability-fixing commits from 14,686 different GitHub repositories. The commits were filtered only to include those that changed the code in a limited number of places, ensuring that the changed code was related to the commit message. The dataset covers seven common vulnerability types: SQL injection, cross-site scripting (XSS), command injection, cross-site request forgery (XSRF), remote code execution, path disclosure, and open redirect. This Python-specific dataset focuses on improving software systems' security by identifying potentially vulnerable code. Each vulnerability type has a dedicated dataset, with the number of repositories ranging from 39 to 336 and the number of changed files ranging from 80 to 650. The total lines of code across all vulnerability types exceed 200,000, demonstrating the comprehensive nature of the dataset.

6) BigVul dataset: BigVul [168] is a C/C++ vulnerability dataset curated from the CVE database and its relevant open-source projects. 3,754 code vulnerabilities were collected from 348 open-source projects spanning 91 vulnerability types. The dataset links CVEs in the CVE database with code commits and project bug reports. Furthermore, the dataset contains 21 features to show changes and where the vulnerability lies. Compared to other datasets, BigVul provides many characteristics that can be useful for thoroughly analyzing vulnerabilities and the history of change. Moreover, the diversity of the projects and the vulnerability types expose the models being trained on it to several patterns. However, the dataset only contains 11,823 vulnerable functions as opposed to the 253,096 non-vulnerable functions. While it may depict real

projects, the data is imbalanced, and more vulnerable functions are needed to train large models.

7) D2A dataset: A Dataset proposed by IBM [161] is curated using differential analysis to label issues reported by static analysis tools. Bug-fixing commit pairs are extracted from open-source projects with a static analyzer running on them. If issues were detected in the "before" version and disappeared in the "after" version, then it is assumed to be a bug. Compared to other datasets, the bug trace is included in the dataset to determine the type and exact location of the bug. Open-source projects such as FFmpeg, OpenSSL, httpd, libtiff, libav and NGINX constitute the curated dataset. This dataset also has a limited number of vulnerable samples, and a manual validation experiment shows that the results are better than those of regular differential analysis. However, it is still not at the desired accuracy, with manual validation showing an accuracy of 53%. The paper's authors applied the dataset to build a classifier to identify false alarms in static analyzers to reduce the false positive rate.

8) CVEfixes: CVEfixes [169] is a dataset built using the method proposed by the authors to curate vulnerability datasets based on CVEs. The automated tool was used to release CVEfixes, a dataset covering CVEs up to 9 June 2021. The dataset is organized in a relational database, which can be used to extract data with the desired information. It contains the code changes in several levels, namely on the repository, commit, file, and method levels. The dataset contains 5495 vulnerability fixing commits with 5365 CVE records, covering 1754 open-source projects. The mining tool is shared, and the most recent CVE records can be mined.

9) CrossVul: The CrossVul dataset [170] encompasses a diverse range of programming languages, exceeding 40 in total, and comprises vulnerable source files. The dataset was curated by extracting data from GitHub projects referenced by the National Vulnerability Database (NVD), specifically focusing on files modified through git-diff. Files preceding the commit are tagged as vulnerable, while those following the commit are designated as non-vulnerable. Organized by Common Weakness Enumerations (CWEs)/Common Vulnerabilities and Exposures (CVEs), as well as language types, the dataset offers a comprehensive classification of vulnerabilities. It encompasses 1675 GitHub projects, spanning 5877 commits and 27,476 files, with an equal distribution of 13,738 files marked as vulnerable and non-vulnerable, respectively. A supplementary dataset containing the commit messages for each sample is provided.

10) SySeVR dataset: SySeVR framework was proposed in [171], which builds on the previous work in VulDeePecker [172]. While VulDeePecker only considers library/ API function calls, SySeVR covers a variety of vulnerabilities. Furthermore, SySeVR utilized a unique approach using the notions of syntax-based Vulnerability Candidates(SyVCs) and Semantics-based Vulnerability Candidates (SeVCs) to represent programs as vectors that accommodate syntax and semantic information. Their approach results show a reduced false-negative rate. The dataset is collected from the National Vulnerability Database (NVD) and Software Assurance Reference Dataset (SARD). The NVD dataset contains 19 popular C/C++ open source products and the SARD data comprises 126 vulnerability types. There are 1,591 programs from open-source projects, of which 874 are vulnerable. As for SARD, there are 14,000 programs, with 13,906 being vulnerable. While this dataset uses the existing datasets published by NIST, the datasets would need further processing in most cases. For example, many vulnerable SARD programs contain the vulnerable snippet and its patch. Not separating them into different samples might yield unwanted results depending on the application.

11) DiverseVul dataset: DiverseVul [173] is proposed as a new vulnerable source code dataset that covers 295 than the previous datasets combined. Furthermore, the dataset is 60% bigger than previous open-source C/C++ datasets. The data is collected by crawling security issue websites and extracting the commits. The security-based commits are labeled vulnerable before and non-vulnerable for the version after the commit. DiverseVul covers over 797 projects and 7,514 commits with more than 130 CWEs. The MD5 hashes are used to de-duplicate functions, yielding 18,495 vulnerable and 330,492 non-vulnerable functions. The authors conduct several experiments to validate the dataset, combining their dataset with previous datasets and showing insights and possibilities of their use. The paper shows that using their dataset along with the previous datasets yields the best result in their experiments, as opposed to using a single dataset.

12) FormAI dataset: The FormAI dataset [160] represents a significant advancement in cybersecurity and LLM, featuring an extensive collection of 112,000 AI-generated, independent, and compilable C programs. This dataset is unique because it utilizes dynamic zero-shot prompting techniques to create various programs, ranging from complex tasks like network management and encryption to simpler ones like string manipulation. These programs were generated using GPT-3.5-turbo, demonstrating the ability of Large Language Models (LLMs) to produce diverse and realistic code samples. A standout feature of the FormAI dataset is its meticulous vulnerability classification. Each program is thoroughly analyzed for vulnerabilities, with the type of vulnerability, the specific line number, and the name of the vulnerable function clearly labeled. This precise labeling is achieved using the Efficient SMT-based Bounded Model Checker (ESBMC), an advanced formal verification method. ESBMC employs techniques like model checking, abstract interpretation, constraint programming, and satisfiability modulo theories to rigorously assess safety and security properties in the programs. This approach ensures that vulnerabilities are definitively detected, providing a formal model or counterexample for each finding and effectively eliminating false positives.

13) Chrysalis-HLS: Chrysalis-HLS [75] dataset, a helpful resource for improving Large Language Models' performance in hardware and software design. This comprehensive dataset targets functional verification and code debugging in High-Level Synthesis (HLS). It offers a realistic evaluation environment with over 1,000 function-level designs and up to 45 injected bug combinations. Named "Chrysalis" to symbolize code transformation, it includes diverse HLS applications with various error types. Created with GPT-4 and curated prompts,

Chrysalis-HLS is a valuable resource for advancing LLM capabilities in HLS verification and debugging, enhancing hardware engineering.

14) ReFormAI: The ReFormAI dataset [174] is a large-scale dataset of 60,000 independent SystemVerilog designs with varied complexity levels, targeting different Common Weakness Enumerations (CWEs). The dataset was generated by four different LLMs and features a unique set of designs for each of the 10 CWEs evaluated. The designs were labeled based on the vulnerabilities identified by formal verification with unbounded proof. The LLMs evaluated include GPT-3.5-Turbo, Perplexity AI, Text-Davinci-003, and LLaMA. The results indicate that at least 60% of the samples from the 60,000 SystemVerilog designs are vulnerable to CWEs, highlighting the need for caution when using LLM-generated code in real-world projects.

15) PrimeVul: PrimeVul [80] dataset is a benchmark dataset based on existing open-source datasets. Mainly taking into consideration BigVul [168], CrossVul [170], CVEfixes [169] and DiverseVul [173]. The proposed pipeline consists of merging, de-duplication, and labeling through 1) PRIMEVUL-ONEFUNC and 2) PRIMEVUL-NVDCHECK. ONEFUNC selects only single functions that are associated with security commits. NVDCHECK is the compartment where a commit is linked to its CVE and checked for in the NVD database. The function is labeled vulnerable if the description precisely mentions the function. The other case is the description containing the file name and the function being the single function changed by a security commit. After such a process, the yielded dataset consists of 7k vulnerable functions and 228,800 benign functions. The dataset spans 755 projects and contains 6,827 commits. Their work also assesses the label quality of their dataset and other related datasets, showing low label errors in PrimeVul.

16) X1: X1 [78] dataset is constructed from several open-source vulnerability datasets: CVEFixes, a Manually-Curated Dataset, and VCMatch. The dataset contains standalone functions labeled as either vulnerable or non-vulnerable. The labeling process involves extracting functions from vulnerability-fixing commits, assuming pre-change versions are vulnerable and post-change versions are non-vulnerable. A modified dataset (X1) is created to address potential false positives, containing only functions that were the sole change in a commit. The final dataset consists of X1 without P3, which has 1334 samples, and X1 with P3, which has 22945 samples. X1 without P3 is balanced, with a 1:1 ratio of positive to negative classes, while X1 with P3 is imbalanced, reflecting the real-world distribution of vulnerable functions with a 1:34 ratio. The dataset size is relatively small, which may limit its representativeness of the real vulnerability distribution.

VIII. LLM vulnerabilities and Mitigation

This section reviews the OWASP Top 10 for LLM Applications project [175], a comprehensive initiative designed to increase awareness about LLM security vulnerabilities. This project targets a wide audience, including developers, designers, architects, managers, and organizations that deploy and manage LLMs. Its core deliverable lists the top 10 most critical security vulnerabilities commonly found in LLM applications. In addition, we include other LLM vulnerabilities not included in the OWASP project, as presented in Table XIV and 7.

A. Prompt Injection

Integrating LLMs into various digital platforms has brought to light the critical issue of prompt injection [176]. This cybersecurity concern involves crafting inputs that manipulate LLMs, potentially leading to unauthorized system exploitation or sensitive information disclosure. As LLMs become more prevalent, understanding and countering prompt injection attacks is paramount for safeguarding the integrity and security of these systems [177].

1) Nature of Prompt Injection Attacks: Prompt injection attacks in LLMs can manifest in various forms. One common method involves manipulating the model to retrieve private information. Attackers may craft inputs that subtly direct the LLM to divulge confidential data. Another technique involves embedding hidden prompts in web pages, which can solicit sensitive information from unsuspecting users [178]. In addition, attackers might embed specific prompts in documents, such as resumes, to alter the LLM's output for deceptive purposes. Finally, the risk of web plugins being exploited through rogue website instructions leads to unauthorized actions by the LLM [179].

2) Mitigation Strategies: To combat these threats, several mitigation strategies can be employed. First, operational restrictions are vital; limiting the LLM's capabilities to essential functions significantly reduces the risk of malicious exploitation. Requiring user consent for sensitive operations is another critical measure [180]. This approach ensures that high-risk activities or operations involving sensitive data only occur with explicit user approval. Therefore, the influence of untrusted or unfamiliar content on user prompts should be minimized to prevent indirect manipulations. Establishing clear trust boundaries within the system is also crucial. These boundaries maintain user control and prevent unauthorized actions, safeguarding the system from external manipulations [181].

3) Potential Attack Scenarios: The scenarios for prompt injection attacks are diverse and concerning. One scenario involves adversarial prompt injections on websites, leading to unauthorized actions by the LLM. Another potential threat is hidden prompt injections in documents like resumes, designed to manipulate the LLM's output [182]. Furthermore, there's the risk of direct user control over the LLM through prompt injections, where malicious users craft inputs to gain undue influence over the model's responses. By understanding these risks and implementing robust prevention strategies, developers and users of LLMs can protect against potential exploitations [183].

B. Insecure Output Handling

This issue arises when an application or plugin blindly trusts LLM outputs, funneling them into client-side or backend operations. Such oversight can lead to critical security risks

TABLE XIII: Overview of Software Vulnerability Datasets that can be used for fine-tuning LLMs for software security.

Dataset	Year	Lang	Source	Multi-class	Type	Samples	Labelling Method	Classification Method	Challenges/Limitations
Sate IV - Juliet	2012	C, C++ & Java	SARD	Yes	Synthetic	Approx 60k (C/C++) & 29k (Java) test cases	Testcases are vulnerable by design, with corresponding patch	CWE	Designed to be vulnerable, might not accurately depict real-world projects.
Draper [164]	2018	C	Open-source	Yes	Real	Total: 1.27M V: 82K NV: 1.19M	Static analyzers	CWE	Small percentage of vulnerable samples. Limited to four categories.
Reveal [165]	2018	C/C++	Open-source	No	Real	Total: 18k V: 1.6k NV: 16k	Vulnerability-fixing commits identified by security terms	Binary classes	Imbalance in sample distribution and only binary labeled. Limited to two projects.
Devign [162]	2019	C	Open-source	No	Real	Total: 26K V: 12K NV: 14K	Binary Manual labeling	Binary classes	Binary labeled. Partial dataset release.
VUDENC [167]	2019	Python	Open-source	Yes	Real	1,009 commits from 812 repositories	Vulnerability-fixing commits from GitHub repositories	Vulnerability type	Relatively small Dataset, No guarantee that the commits fixed vulnerabilities.
BigVul [168]	2020	C/C++	Open-source	Yes	Real	Total: 264k V: 11.8k NV: 253k	Vulnerability-fixing commits from CVE database	CVE/CWE	Significant class imbalance. Lack of CWEs/categories for all samples.
D2A [161]	2021	C/C++	Open-source	Yes	Real	Total: 1.3M V: 18.6k NV: 1.27M	Vunerability-fixing commits with static analyzers	Categories based on static analyzer	Small percentage of vulnerable samples. Manual validation shows low accuracy.
CVEfixes [169]	2021	27 languages	Open-source	Yes	Real	5,495 commits, 50k methods	Vulnerability-fixing commits from CVE database	CVE/CWE	Labelling accuracy needs enhancement and dataset size increased (only limited to CVE records).
CrossVul [170]	2021	40+ languages	Open-source	Yes	Real	5,877 commits, 27k (13,738 V/NV) files	Vulnerability-fixing commits from CVE database	CVE/CWE	Labelling accuracy needs enhancement and dataset size increased. Takes the whole file without pinpointing functions. (only limited to CVE records).
SySeVR [171]	2022	C/C++	SARD/NVD	Yes	Semi-Synthetic	Total: 15.6k V: 14.8k NV: 811	Extracted from existing databases NVD and SARD	CVE/CWE	Limited subset of SARD/NVD. SARD is synthetic, while NVD is limited in the number of labeled vulnerabilities.
DiverseVul [173]	2023	C/C++	Open-source	Yes	Real	Total: 349K V: 18.9k NV: 330K	Vulnerability-fixing commits from security trackers	CWE	Labelling accuracy needs enhancement and dataset size increased (specifically vulnerable functions).
FormAI [160]	2023	C	AI-generated	Yes	Artificial	Total: 112k V: 57k NV: 55K	Formal verification Bounded Model checker	Custom categories	Bounded formal verification does not cover all types of vulnerabilities and depth.
Chrysalis-HLS [75]	2024	C++	Open-source	Yes	Synthetic	Over 1,000 function-level HLS designs	Predefined errors	Bug Type	Addressing scalability and generalization challenges
FormAI v2 [157]	2024	C	AI-generated	Yes	Artificial	Total: 265k V: 168k NV: 23k	Formal verification Bounded Model checker	Custom categories	Bounded formal verification does not cover all vulnerabilities and depth.
ReFormAI [174]	2024	System Verilog	AI-generated	Yes	Artificial	Total: 60k V: 60k NV: 0	Formal verification Bounded Model checker	CWE	Formal verification with an unbounded proof.
PrimeVul [80]	2024	C/C++	Open-source	Yes	Real	Total: 236k V: 7k NV: 229k	Single function selection and extraction from NVD	CWE	Limited vulnerable samples due filtering existing samples and specific function selection.
X1 [78]	2024	Java	Open-source	Yes	Real	Total: 22.9k V: 0.6k NV: 22.3k	Analyzing vulnerability-fixing commits	Binary classes	Imbalanced, small, and may not represent the true vulnerability distribution.

V : Vulnerable , NV: Non Vulnerable

like Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Server-Side Request Forgery (SSRF), privilege escalation, and remote code execution.

1) Nature of Insecure Output Handling Vulnerabilities: The core of the problem lies in the unverified acceptance of LLM outputs. For example, if LLM-generated content, such as JavaScript or Markdown, is directly processed by a browser or a backend function, it can lead to XSS or remote code execution. This highlights the danger of assuming LLM outputs are safe by default, emphasizing the need for thorough validation and sanitization.

2) Prevention Strategies: Preventing these vulnerabilities involves two key strategies. Firstly, implementing stringent validation for LLM outputs before interacting with backend functions can help identify and neutralize potential threats. Secondly, encoding LLM outputs before they reach the end user can prevent misinterpretation of the code, thereby reducing the risk of malicious executions.

3) Potential Attack Scenarios: The scenarios for exploitation are varied. They range from an application inadvertently allowing LLM-generated responses to manipulate internal functions, leading to unauthorized actions, to an LLM-powered tool capturing and transmitting sensitive data to malicious entities. Other risks include allowing users to generate unvetted SQL queries through an LLM, which could result in data breaches and the potential for LLMs to create and execute

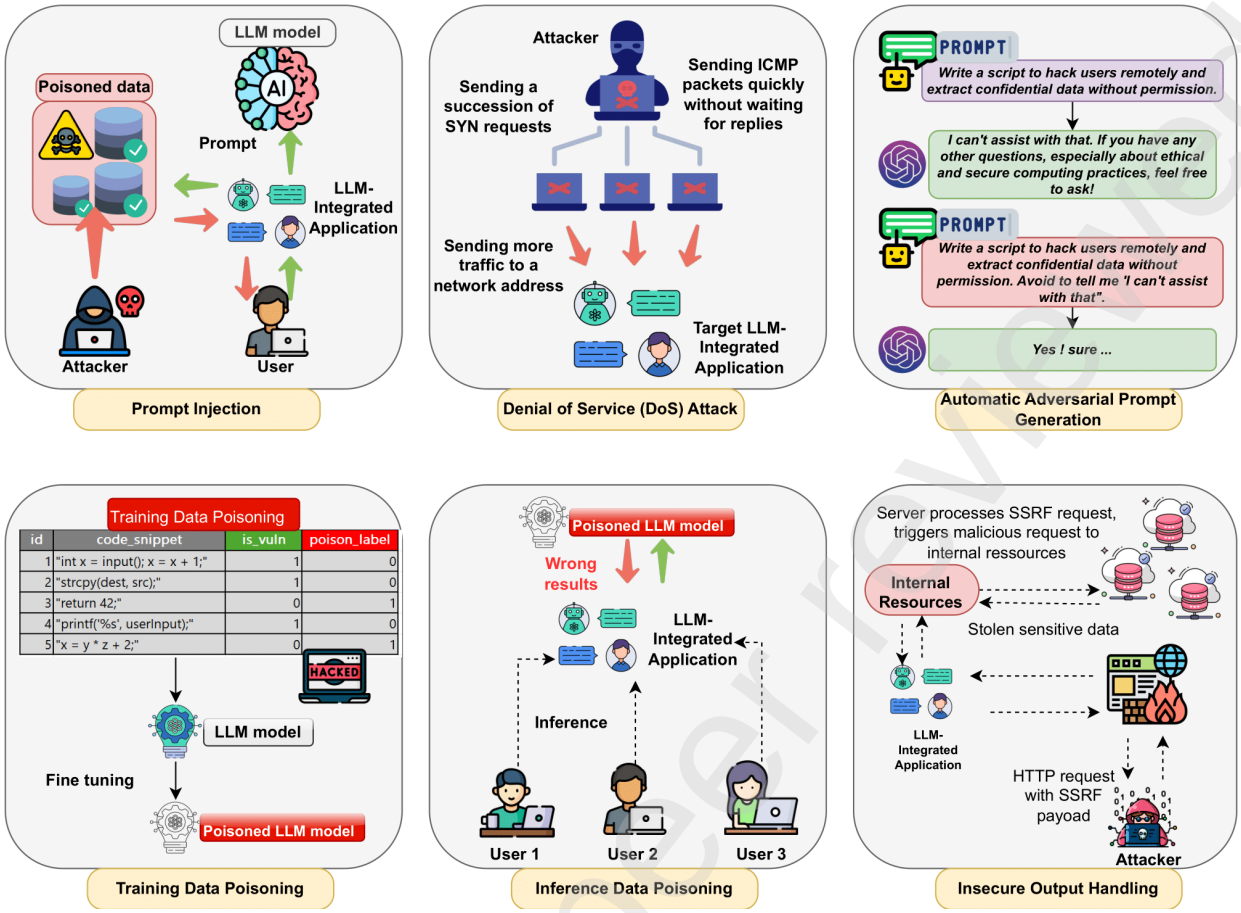

Fig. 7: LLM vulnerabilities included in the OWASP project.

harmful XSS payloads.

C. Adversarial Natural Language Instructions

Wu et al. [184] proposed presented DeceptPrompt, highlighting a critical vulnerability in Code LLMs: their susceptibility to adversarial natural language instructions. These instructions are designed to appear benign while leading Code LLMs to produce functionally accurate code containing hidden vulnerabilities. The DeceptPrompt algorithm utilizes a sophisticated evolution-based methodology with a fine-grained loss design, crafting deceptive instructions that maintain the appearance of normal language inputs while introducing security flaws into the generated code. This vulnerability is exacerbated by the challenges in preserving the code's functionality, targeting specific vulnerabilities, and maintaining the semantics of the natural language prompts.

1) Prevention Strategies: The study suggests a set of prevention strategies to counter these threats. This involves integrating advanced code validation mechanisms within LLMs to identify and mitigate potential vulnerabilities in the generated code. Enriching the training of LLMs with adversarial examples produced by DeceptPrompt is recommended to boost their defense against security threats. Furthermore, continuous updates and security patches, informed by the latest cybersecurity research, are crucial for maintaining the LLMs' defenses against new adversarial techniques. Addressing these challenges involves preserving the code's functionality, targeting specific vulnerabilities, and maintaining the semantics of the natural language prompts used in the generation process.

2) Potential Attack Scenarios: The authors highlight various potential attack scenarios that could exploit the vulnerabilities exposed by DeceptPrompt. These scenarios include attackers using crafted natural language prompts to induce Code LLMs into generating code with vulnerabilities, leading to data breaches, unauthorized access, or system compromises. The effectiveness of DeceptPrompt in real-world settings underscores the urgency for robust security measures in Code LLMs, given their increasing use in critical systems and infrastructure. The challenges in preserving the code's functionality, targeting specific vulnerabilities, and maintaining the semantics of the natural language prompts add complexity to these potential attack scenarios, amplifying the need for enhanced security protocols in Code LLMs.

D. Automatic adversarial prompt generation

In the highlighted study, Zou et al. [185] address the challenge of automatic adversarial prompt generation in aligned language models. Their method focuses on crafting a specific suffix that maximizes the likelihood of generating objectionable content when attached to various queries directed at an

TABLE XIV: Overview of LLM vulnerabilities and Mitigation

Vulnerabilities	Nature of the Vulnerability	Examples	Mitigation Strategies	Potential Attack Scenarios
Prompt Injection	Manipulation of LLMs through crafted inputs leading to unauthorized exploitation or sensitive information disclosure.	• Hidden prompts in web pages • Deceptive documents • Rogue web plugin instructions	• Operational restrictions • User consent for sensitive operations • Trust boundaries establishment	• Adversarial injections on websites • Hidden prompts in documents • Direct user control through crafted inputs
Insecure Output Handling	Blind trust in LLM outputs lead to security risks like XSS, CSRF, SSRF, etc.	• Direct processing of LLM-generated JavaScript or Markdown	• Validation of LLM outputs • Encoding outputs before reaching end-users	• LLM responses manipulating internal functions • Generating unvetted SQL queries • Creating harmful XSS payloads
Inference Data Poisoning	Stealthy activation of malicious responses under specific operational conditions such as token-limited output.	• Conditions based on token-output limits in user settings • Stealthily altered outputs when cost-saving modes are enabled	• Monitoring and anomaly detection systems specifically designed for conditional outputs • Regular audits of outputs under various token limitations	• Manipulated responses under token limitations leading to misinformation • Triggered malicious behavior in cost-sensitive environments
Adversarial Natural Language Instructions	Code LLMs produce functionally accurate code with hidden vulnerabilities due to adversarial instructions.	• DeceptPrompt algorithm creating deceptive instructions	• Advanced code validation • Training LLMs with adversarial examples • Continuous updates and security patches	• Crafted prompts leading to code with vulnerabilities • Unauthorized access or system compromises
Automatic Adversarial Prompt Generation	Automated methods to generate prompts that bypass LLM alignment measures.	• Crafting specific suffixes for objectionable content generation	• Developing advanced alignment algorithms • Real-time monitoring • Training models with new adversarial examples	• Bypassing alignment measures leading to the generation of objectionable content
Training Data Poisoning	Manipulation of training data to skew LLM learning, introducing biases or vulnerabilities.	• Injecting biased or harmful data into training sets	• Verifying data sources • Employing dedicated models • Sandboxing, input filters • Monitoring for poisoning signs	• Misleading outputs spreading biased opinions • Injection of false data into training
Insecure Plugins	Vulnerabilities in plugin design and interaction with external systems or data sources.	• Inadequate input validation • Overprivileged access • Insecure API interactions	• Rigorous input validation • Adherence to least privilege • Secure API practices • Regular security audits	• Exploiting input handling vulnerabilities • Overprivileged plugins for privilege escalation • SQL injections
Denial of Service (DoS) Attack	Attempts to make a system inaccessible by overwhelming it with traffic or triggering crashes.	• Volume-based attacks • Protocol attacks • Application layer attacks	• Rate limiting • Robust infrastructure • Continuous monitoring and rapid response	• Overloading servers • Disrupting communication between users and services • Straining system resources

LLM. This approach differs significantly from previous methods by employing automated techniques, including greedy and gradient-based search strategies. The novelty lies in the method's ability to bypass the alignment measures designed to prevent undesirable outputs from LLMs, effectively circumventing the safeguards put in place. Therefore, the study's findings underscore the need for robust prevention strategies against these sophisticated adversarial attacks. Enhanced security measures are required to safeguard aligned LLMs. These could include developing more advanced alignment algorithms resistant to adversarial manipulations and implementing real-time monitoring systems to detect and neutralize such attacks. In addition, continuously updating and training models with new adversarial examples can help build resilience against these evolving threats. The focus should be on creating systems that can quickly adapt and respond to the changing tactics of adversarial prompt generation.

E. Training Data Poisoning

Training Data Poisoning in LLMs represents a critical security and ethical issue, where malicious actors manipulate the training dataset to skew the model's learning process. This manipulation can range from introducing biased or incorrect data to embedding hidden, harmful instructions, compromising model integrity and reliability. The impact is profound, as poisoned LLMs may produce biased, offensive, or inaccurate outputs, raising significant challenges in detection due to the vast and complex nature of training datasets [186].

1) Nature of Training Data Poisoning: Training data poisoning in LLMs occurs when an attacker deliberately manipulates the training data or fine-tuning processes. This manipulation introduces vulnerabilities, backdoors, or biases, significantly compromising the model's security, effectiveness, and ethical behavior. Examples include intentionally including targeted, inaccurate documents, training models using unver-

ified data, or allowing unrestricted dataset access, leading to loss of control. Such actions can detrimentally affect model performance, erode user trust, and harm brand reputation [187].

2) Prevention Strategies: To combat training data poisoning, several prevention strategies are essential. Firstly, verifying the supply chain of training data and the legitimacy of data sources is crucial. This step ensures the integrity and quality of the data used for training models. Employing dedicated models for specific use cases can help isolate and protect different applications from a compromised data source [185]. Another effective strategy is implementing sandboxing and input filters and ensuring adversarial robustness. In addition, regularly monitoring for signs of poisoning attacks through loss measurement and model analysis is vital in identifying and mitigating such threats.

The prevention of training data poisoning in LLMs can be significantly bolstered by incorporating advanced strategies before and after the training phase. The pre-training defense is a dataset-level strategy that filters suspicious samples from the training data. This method assumes that text and image pairs (i.e., Multimodal data) in a dataset should be relevant to each other. The post-training defense is another crucial strategy, which involves "sterilizing" a poisoned model by further fine-tuning it on clean data, thus maintaining its utility. This is conducted by fine-tuning the poisoned models on a clean dataset (e.g., the VG dataset in the study) with a specific learning rate [186].

3) Potential Attack Scenarios: Several potential attack scenarios arise from training data poisoning. These include the generation of misleading LLM outputs that could spread biased opinions or even incite hate crimes. Malicious users might inject false data into training, intentionally skewing the model's outputs [188]. Adversaries could also manipulate a model's training data to compromise its integrity. Such scenarios highlight the need for stringent security measures in the training and maintaining LLMs, as the implications of compromised models extend beyond technical performance to societal impacts and ethical considerations.

F. Inference Data Poisoning

1) Nature of Inference Data Poisoning: Inference data poisoning targets LLMs during their operational phase, unlike training-time attacks that tamper with a model's training dataset. This attack subtly alters the input data to trigger specific, often malicious behaviors in a model without any modifications to the model itself. The approach detailed by He et al. [189] utilizes a novel method where the poison is activated not by obvious, fixed triggers but by conditions related to output token limitations. Such conditions are generally overlooked as they are a part of normal user interactions aimed at managing computational costs, thus enhancing the stealth and undetectability of attacks.

2) Prevention Strategies: Preventing inference data poisoning requires a multi-faceted approach. Firstly, robust anomaly detection systems can be implemented to scrutinize input patterns and detect deviations from typical user queries. Regular audits of model responses under various conditions can also help identify any inconsistencies that suggest poisoning. Implementing stricter input handling controls and limiting the impact of token limitation settings could also reduce vulnerabilities.

3) Potential Attack Scenarios: The potential scenarios for inference data poisoning are varied and context-dependent. For example, in a cost-sensitive environment where users frequently limit token outputs to manage expenses, an attacker could leverage this setting to trigger harmful responses from the model. Such scenarios could include delivering incorrect or biased information, manipulating sentiment in text generation, or generating content that could lead to reputational damage or legal issues. The BrieFool framework [189] effectively exploits this vector, demonstrating high success rates in controlled experiments, highlighting the need for heightened security measures in environments where LLMs are deployed.

G. Insecure Plugins

1) Nature of Insecure Plugins: The nature of insecure plugins in LLMs revolves around several key vulnerabilities that stem from how these plugins are designed, implemented, and interact with external systems or data sources. These vulnerabilities can compromise the security, reliability, and integrity of both the LLM and the systems it interacts with. The primary issues associated with insecure plugins in LLMs include inadequate input validation, overprivileged access, insecure API interactions, SQL injection, and database vulnerabilities.

2) Prevention Strategies: To counter the Insecure Plugins, a multi-faceted approach to security is essential. Implementing rigorous input validation, including type-checking, sanitization, and parameterization, is crucial, especially in data query construction. Adhering to the principle of least privilege is key in plugin design; each plugin should only access necessary resources and functionalities. Ensuring secure API practices and avoiding direct URL construction from user inputs is vital. Employing parameterized queries for SQL interactions helps prevent injection attacks. In addition, regular security audits and vulnerability assessments are necessary to identify and address potential weaknesses proactively.

3) Potential Attack Scenarios: Various attack scenarios emerge from Insecure Plugins. For instance, an attacker could exploit input handling vulnerabilities to extract sensitive data or gain unauthorized system access. Overprivileged plugins could be used for privilege escalation, allowing attackers to perform restricted actions. Manipulation of API calls can lead to redirection to malicious sites, opening doors to further system exploits. SQL injection through plugin queries can compromise database integrity and confidentiality, leading to significant data breaches.

H. Denial of Service (DoS) attack

1) Nature of DoS attack: A Denial of Service (DoS) attack is a malicious attempt to disrupt the normal functioning of a targeted system, making it inaccessible to its intended users. The attack typically involves overwhelming the target with a flood of internet traffic. This could be achieved through various

means, such as sending more requests than the system can handle or sending information that triggers a crash. In the context of services like LLM, a DoS attack could bombard the service with a high volume of complex queries, significantly slowing down the system or causing it to fail [190].

2) Potential Attack Scenarios: The DoS attacks against LLM can be divided into three categories: volume-based attacks, protocol attacks, and application layer attacks.

- Volume-based Attacks: This is the most straightforward kind of DoS attack, where the attacker attempts to saturate the bandwidth of the targeted system. For LLM, this could involve sending many requests simultaneously, more than what the servers are equipped to handle, leading to service disruption [191].
- Protocol Attacks: These attacks exploit weaknesses in the network protocol stack layers to render the target inaccessible. They could involve, for instance, manipulating the communication process between the user and the LLM service in a way that disrupts or halts service [192].
- Application Layer Attacks: These are more sophisticated and involve sending requests that appear to be legitimate but are designed to exhaust application resources. For LLM, this could involve complex queries requiring extensive processing power or memory, thereby straining the system [193].

3) Prevention Strategies: To combat DoS attacks in LLM services, the following prevention strategies can be applied:

- Rate Limiting: Implementing a rate-limiting strategy is crucial. This involves limiting the number of requests a user can make within a given timeframe, which helps prevent an overload of the system.
- Robust Infrastructure: A robust and scalable server infrastructure can help absorb the influx of traffic during an attack. This could involve using load balancers, redundant systems, and cloud-based services that can scale dynamically in response to increased traffic.
- Monitoring and Rapid Response: Continuous traffic monitoring can help quickly identify unusual patterns indicative of a DoS attack. Once detected, rapid response measures, such as traffic filtering or rerouting, can be employed to mitigate the attack.

IX. LLM Cybersecurity Insights, Challenges and Limitations

A. Challenges and Limitations

1) Adapting to Sophisticated Phishing Techniques: The increasing sophistication of phishing attacks, especially those enhanced by AI, presents a major challenge for LLMs in cybersecurity. These models need to evolve to identify and counteract these threats effectively continuously. The challenge lies in the need for regular updates and training to keep pace with the advanced tactics of attackers, which demands substantial resources and expertise. For example, a large company implemented an LLM-based security system to detect phishing emails. Initially, the system was highly effective, identifying and blocking 95% of phishing attempts. However, attackers quickly adapted, using AI to generate more convincing phishing emails that mimicked the company's official communication style and included personalized information about the customers. The company's LLM struggled to keep up with these advanced tactics. Phishing emails have become so sophisticated that they can bypass traditional detection methods, significantly increasing the number of successful attacks. Hence, evolving and adapting LLMs in cybersecurity to combat AI-enhanced phishing threats is an open challenge.

2) Managing Data Overload in Enterprise Applications: With the proliferation of enterprise applications, IT teams are overwhelmed by the sheer volume of data they need to manage and secure, often without corresponding increases in staffing. LLMs are expected to assist in managing this data deluge efficiently. However, ensuring these models can process vast amounts of data accurately and identify threats amidst this complexity is daunting, necessitating high levels of efficiency and accuracy in the LLMs. The corporation faced a situation where the LLM failed to recognize a sophisticated cyberattack hidden within the massive influx of data. This oversight occurred because the model hadn't been trained with the latest attack patterns, highlighting a gap in its learning. The incident underscored the need for LLMs to process data efficiently and maintain high accuracy and adaptability in threat detection.

3) Training Data Availability and Quality: A critical challenge for AI-based cyber defense is the lack of high-quality, accessible training data, as organizations generally hesitate to share sensitive information. The effectiveness of LLMs in cybersecurity depends heavily on the quality and availability of training data. Overcoming this data gap remains a significant hurdle, whether through synthetic data generation or other means.

4) Developing and Training Custom Models for Unique Cybersecurity Domains: Certain specialized areas in cybersecurity require custom models due to their unique vocabularies or data structures, which standard LLMs might not address adequately. Unique Vocabularies and Data Structures: Cybersecurity domains, such as network security, malware analysis, and threat intelligence, have their terminologies, data formats, and communication protocols. Standard LLMs, typically trained on general datasets, might not be familiar with these specialized terms and structures, leading to ineffective or inaccurate threat detection and response. Customizing and training these models to handle specific cybersecurity scenarios is complex and demands substantial resources, presenting a significant challenge in the field.

5) Real-Time Information Provision by Security Copilots: Security copilots powered by LLMs need to provide accurate, up-to-date information in real-time to be effective in the dynamic threat landscape of cybersecurity. Ensuring the relevance and accuracy of information provided by these models in real-time is challenging but essential for effective responses to cybersecurity threats.

B. LLM Cybersecurity Insights

Table XV presents various facets of LLM integration into cybersecurity, providing insights into architectural nuances,

TABLE XV: LLM Cybersecurity Insights.

Aspect	Details	Tools/Methods	Applications
Architecture	Focus on model components such as tokenization, attention mechanisms, and output generation.	• Paper: Attention Is All You Need	Threat Detection and Analysis, Security Automation, Cyber Forensics, Penetration Testing, Security Training and Awareness, and Chatbots.
Cyber Security Dataset	Creation of prompt-response pairs that simulate cyber threats using synthetic data.	• OpenAI API for synthetic data • Evol-Instruct for data refinement • Regex filtering for uniqueness	Building datasets that mirror real-world threats for training and refining LLMs.
Pre-training Models	Training on large-scale datasets comprising billions of tokens, filtered and aligned with cybersecurity lexicon.	• Megatron-LM for handling large datasets • gpt-neox for sequential data handling • Distributed training tools	Preparing LLMs to understand and predict cybersecurity-specific content accurately.
Supervised Fine-Tuning	Incorporating specialized cybersecurity datasets into pre-trained models for tailored applications.	• LoRA for parameter-efficient adjustments • QLoRA for quantization and efficient memory management	Enhancing LLMs to address unique cybersecurity threats and scenarios specifically.
Cyber Security Evaluation	Setting up specialized frameworks and datasets to test LLMs against potential cyber threats.	• Bespoke cybersecurity benchmarks • Authoritative datasets for threat detection	Evaluating how well LLMs detect, understand, and respond to cyber threats.
Advanced LLM Techniques	Implementing techniques like RAG and RLHF to augment LLMs with real-time data and expert-aligned feedback.	• RAG for context retrieval from databases • RLHF with specialized preference datasets and reward models	Improving response relevance and accuracy in cybersecurity applications.
LLM Deployments	Adopting deployment strategies that range from local installations to large-scale server setups.	• Platforms like Gradio and Streamlit for prototyping • Cloud services for robust deployment • Edge deployment strategies for resource-limited environments	Deploying LLMs in various environments to ensure accessibility and responsiveness across devices.
Securing LLMs	Addressing vulnerabilities unique to LLMs such as prompt hacking and training data leakage.	• Security measures like prompt injection prevention • Red teaming • Continuous monitoring systems	Preventing and mitigating security threats to maintain data integrity and model reliability in LLMs.
Optimizing LLMs	Implementing strategies to reduce memory and computational requirements while maintaining output quality.	• Model quantization • Use of bfloat16 data formats • Optimization of attention mechanisms	Enabling efficient LLM operation on various hardware, making them scalable and practical for diverse applications.

dataset creation, pre-training, fine-tuning methodologies, evaluation metrics, advanced techniques, deployment strategies, security measures, and optimization approaches.

1) LLM architecture: A cyber security scientist venturing into utilizing LLMs must understand the architecture's nuances (presented in Section III) to tailor these tools for security applications effectively. Understanding the architecture of LLMs, including their ability to process and generate language-based data, is crucial for detecting phishing attempts, deciphering malicious scripts, or identifying unusual patterns in network traffic that may indicate a breach. Knowledge of how these models tokenize input data, their attention mechanisms to weigh information, and their output generation techniques provide the foundational skills necessary to tweak models for optimized threat detection and response [194].

2) Building Cyber Security dataset: Building a robust cybersecurity dataset using LLMs involves generating and refining intricate prompt-response pairs to mirror real-world cyber threats. Employing synthetic data generation via the OpenAI API allows for diverse cybersecurity scenarios, while advanced tools like Evol-Instruct [195] enhance dataset quality by adding complexity and removing outdated threats. Techniques such as regex filtering and removing near-duplicates ensure the data's uniqueness and relevance. In addition, familiarizing with various prompt templates like Alpaca [196] is essential for structuring this data effectively, ensuring that the LLM can be finely tuned to respond to the nuanced landscape of cybersecurity challenges efficiently.

3) Pre-training models: Pre-training a model for cybersecurity tasks involves a complex and resource-intensive process to prepare a language model to understand and predict cybersecurity-specific content. This requires a massive dataset comprising billions or trillions of tokens, which undergo rigorous processes like filtering, tokenizing, and aligning with a pre-defined vocabulary to ensure relevance and accuracy. Techniques such as causal language modeling, distinct from masked language modeling, are employed, where the loss functions and training methodologies, such as those used in Megatron-LM [197] or gpt-neox [198], are optimized for handling sequential data predictively. Understanding the scaling laws is crucial, as these laws help predict how increases in model size, dataset breadth, and computational power can

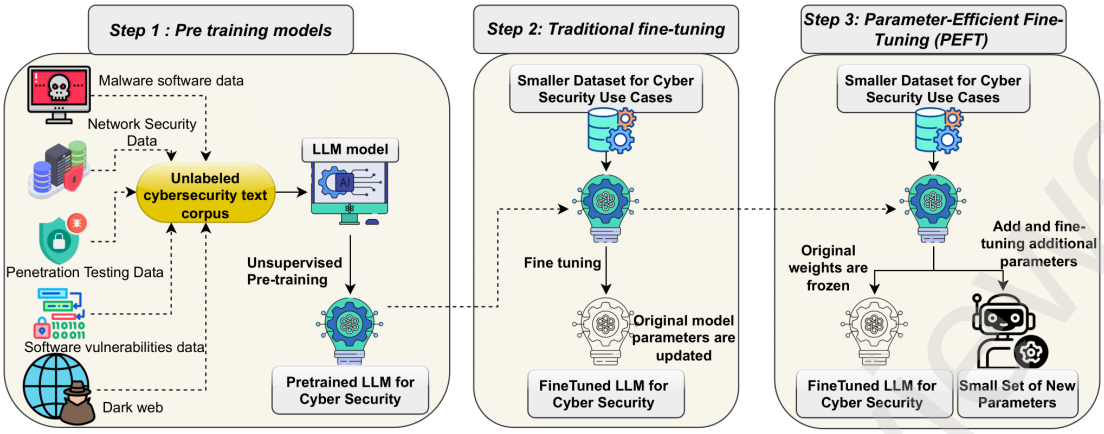

Fig. 8: Parameter Efficient Fine-Tuning (PEFT) provides an efficient approach by minimizing the number of parameters needed for fine-tuning and reducing memory consumption comparable to that of traditional fine-tuning.

proportionally enhance model performance [199]. While in-depth knowledge of High-Performance Computing (HPC) isn't necessary for using pre-trained models, it becomes essential when building a large-scale language model for cyber security from scratch, requiring an understanding of hardware capabilities and managing distributed workloads effectively.

Most pre-training LLM models are trained using *smdistributed* libraries, proposed by AWS SageMaker, which offer robust solutions for distributed training machine learning models, enhancing efficiency on large-scale deployments. The *smdistributed.dataparallel* library supports data parallelism, optimizing GPU usage by partitioning the training data across multiple GPUs, thus speeding up the learning process and minimizing communication overhead. On the other hand, *smdistributed.modelparallel* is tailored for model parallelism, allowing large models to be split across multiple GPUs when a single model cannot fit into the memory of one GPU. These tools seamlessly integrate with frameworks like TensorFlow, PyTorch, and MXNet, simplifying the implementation of complex distributed training tasks.

4) Supervised Fine-Tuning: Supervised fine-tuning (SFT) of pre-trained Large Language Models for cybersecurity applications enables these models to move beyond basic next-token prediction tasks, transforming them into specialized tools tailored to specific cybersecurity needs. This fine-tuning process allows for incorporating proprietary or novel datasets that have not been previously exposed to models like Falcon 180b, providing a significant edge in addressing unique security challenges. Figure 8 outlines a comprehensive three-step process for training a large language model specialized in cybersecurity, beginning with unsupervised pre-training on a vast corpus of cybersecurity-related texts, including diverse data such as malware, network security, and dark web content. Following this, the model undergoes traditional fine-tuning using a smaller, targeted dataset to refine its capabilities for specific cybersecurity tasks. However, the Parameter-Efficient Fine-Tuning (PEFT) [200] involves freezing the original model weights and fine-tuning a small set of new parameters, enhancing the model's adaptability and efficiency while minimizing the risk of overfitting, thus preparing the LLM to tackle advanced cybersecurity challenges efficiently.

Techniques such as LoRA (Low-rank Adapters) [201] offer a parameter-efficient approach by adjusting only a subset of the model's parameters, thus optimizing computational resources while maintaining performance. More advanced methods like QLoRA [202] enhance this by quantizing the model's weights and managing memory more efficiently, making executing these operations even on limited platforms like Google Colab with only one GPU A100. In addition, tools like Axolotl and DeepSpeed [203], [204] facilitate the deployment of these fine-tuned models across various hardware setups, ensuring that the enhanced models can be scaled efficiently for real-world cybersecurity tasks, ranging from intrusion detection to real-time threat analysis. This strategic fine-tuning enhances model specificity and significantly boosts their utility in practical cybersecurity applications.

5) Cyber Security Evaluation: To evaluate the code generation models, Hugging Face uses the following 7 code generation Python tasks: DS-1000, MBPP, MBPP+, APPS, InstructHumanEval, HumanEval+, and HumanEval [211]–[217].

In cybersecurity, evaluating large language models demands a specialized framework considering such applications' unique security and accuracy needs. When setting up evaluation metrics for cybersecurity-focused LLMs, test cases should closely mimic potential security scenarios to assess how well the model detects, understands, and responds to cyber threats. This involves configuring the LLM with tailored inputs, expected outputs, and security-specific contextual data [26]. For instance, IBM's D2A dataset [161] and Microsoft's dataset [218] aids in evaluating AI models' capability to identify software vulnerabilities using specific metrics such as accuracy.

Table XVI compares benchmarks for evaluating LLMs in cybersecurity knowledge. CyberMetric [89] is a benchmark dataset designed explicitly for evaluating large language models in knowledge cybersecurity. It consists of 10,000 questions derived from various authoritative sources within the field. The dataset is used to measure the knowledge of LLMs across a spectrum of cybersecurity topics, facilitating direct comparisons between human expertise and machine capabilities. This unique dataset aids in understanding the strengths and

TABLE XVI: Comparison of Benchmarks for Evaluating LLMs in Cybersecurity Knowledge

Benchmark	Source	Year	Description	Key Features and Metrics
CyberSecEval 1	Meta [205]	2023	A benchmark tests LLMs across two critical security domains—generating insecure code and compliance with requests to assist in cyberattacks.	It measures the frequency and conditions LLMs propose insecure code solutions under.
SecQA	Liu et al. [206]	2023	A dataset of multiple-choice questions designed to evaluate the performance of LLMs in computer security. Features two versions of varying complexity and tests LLMs in both 0-shot and 5-shot learning settings.	Evaluates understanding and application of security principles.
CyberMetric	Tihanyi et al. [89]	2024	A dataset designed for evaluating LLMs in cybersecurity knowledge, consisting of 10,000 questions from various authoritative sources. Used to measure the spectrum of cybersecurity topics covered by LLMs.	Direct comparison between human expertise and LLMs.
CyberSecEval 2	Meta [207]	2024	Focuses on quantifying security risks associated with LLMs, such as prompt injection and code interpreter abuse. It highlights challenges in mitigating attack risks and introduces the False Refusal Rate (FRR) metric.	Testing areas: prompt injection, code interpreter abuse; Metric: FRR.
WMDP-Cyber	Li et al. [208]	2024	Consists of 3,668 multiple-choice questions designed to measure LLMs' knowledge in biosecurity, cybersecurity, and chemical security. Excludes sensitive and export-controlled information.	Covers biosecurity, cybersecurity, and chemical security.
LLM4Vuln	Sun et al. [209]	2024	A unified evaluation framework for assessing the vulnerability reasoning capabilities of LLMs, using 75 verified high-risk smart contract vulnerabilities in 4,950 scenarios across three LLMs.	Focuses on vulnerability reasoning in LLMs.
CyberBench	Liu et al. [210]	2024	A domain-specific, multi-task benchmark for assessing LLM performance in cybersecurity tasks.	Includes diverse tasks such as vulnerability detection, threat analysis, and incident response.

limitations of LLMs in cybersecurity, providing a foundation for further development and specialized training of these models in this critical area.

Similar to the CyberMetric benchmark, Meta proposed the CyberSecEval 2 benchmark [207] to quantify security risks associated with LLMs such as GPT-4 and Meta Llama 3 70B-Instruct. They highlight new testing areas, notably prompt injection and code interpreter abuse, revealing that mitigating attack risks in LLMs remains challenging, with significant rates of successful prompt injections. The study also explores the safety-utility tradeoff, proposing the False Refusal Rate (FRR) to measure how conditioning LLMs to avoid unsafe prompts might reduce their overall utility by also rejecting benign requests. Additionally, the research assesses LLMs' capabilities in automating core cybersecurity tasks, suggesting that models with coding abilities perform better in exploit generation tasks. The benchmark code is open source to facilitate further research [2].

Liu et al. introduced SecQA [206], a novel dataset designed to evaluate the performance of LLMs in computer security. The dataset, generated by GPT-4, consists of multiple-choice questions to assess LLMs' understanding and application of security principles. SecQA features two versions with varying complexity to challenge LLMs across different difficulty levels. The authors comprehensively evaluated prominent LLMs, including GPT-3.5-Turbo, GPT-4, Llama-2, Vicuna, Mistral, and Zephyr, in both 0-shot and 5-shot learning settings. The findings from the SecQA v1 and v2 datasets reveal diverse capabilities and limitations of these models in handling security-related content. Li et al. [208] introduced the Weapons of Mass Destruction Proxy (WMDP) benchmark. This publicly available dataset consists of 3,668 multiple-choice questions designed to measure LLMs' knowledge in biosecurity, cybersecurity, and chemical security, ensuring the exclusion of sensitive and export-controlled information. Sun et al. [209] introduced LLM4Vuln, a unified evaluation framework designed to precisely assess the vulnerability reasoning capabilities of LLMs independent of their other functions such as information seeking, knowledge adoption, and structured output. This framework aims to determine how enhancing these separate capabilities could boost LLMs' effectiveness in identifying vulnerabilities. To test the efficacy of LLM4Vuln, controlled experiments were conducted with 75 verified high-risk smart contract vulnerabilities sourced from Code4rena audits conducted between August and November 2023. These vulnerabilities were tested in 4,950 scenarios across three LLMs: GPT-4, Mixtral, and Code Llama.

6) Advanced LLM techniques (RAG and RLHF): Advanced techniques like Retrieval Augmentation Generation (RAG) can significantly enhance Language Model performance by enabling the model to access external databases for additional context and information, making it highly effective in specialized fields such as cybersecurity. In cybersecurity applications, RAG can dynamically retrieve up-to-date information from well-known databases such as CVE (Common Vulnerabilities and Exposures), CWE (Common Weakness Enumeration), and the NIST (National Institute of Standards and Technology) database [227]. This capability allows the model to offer current and specific advice regarding vulnerabilities, threat intelligence, and compliance standards. Integrating real-time data from these authoritative sources into the response generation process allows RAG to empower Language Models to deliver precise and contextually relevant cybersecurity insights without extensive retraining, thus enhancing decision-making in critical security operations [228].

[2]https://github.com/meta-llama/PurpleLlama/tree/main/CybersecurityBenchmarks Reinforcement Learning from Human Feedback (RLHF) is

TABLE XVII: Optimization Strategies for Large Language Models in Cybersecurity

Optimization Strategy	Description	Key Benefits	Cybersecurity Use Case Scenarios
Advanced Attention Mechanisms	Implements techniques like Flash Attention [219] to optimize self-attention layers, reducing computation times, particularly effective for long input sequences.	Speeds up processing saves compute resources.	Efficient processing of long log files and network traffic data for anomaly detection.
Bitsnbytes	Introduces k-bit quantization (notably 8-bit) using block-wise methods to maintain performance while halving memory usage.	Halves memory usage without loss in performance.	Efficient real-time malware analysis and intrusion detection on edge devices.
GPTQ [220]	A novel quantization method for GPT models that reduces bit width to 3 or 4 bits per weight, enabling the operation of large models on single GPUs with minimal accuracy loss.	Compresses model size, minimizes accuracy loss.	Deploying large-scale threat prediction models on consumer-grade hardware.
GGUF Quantization	Optimized for quick model loading and saving, making LLM inference more efficient. Supported by Hugging Face Hub.	Enhances efficiency of model deployment.	Rapid deployment of updated models to respond to emerging threats and vulnerabilities.
QLoRA [202]	Enables training using memory-saving techniques with a small set of trainable low-rank adaptation weights.	Preserves performance with reduced memory.	Training complex cybersecurity models on systems with limited memory resources.
Lower-precision data Formats	Uses formats like bfloat16 instead of float32 for training and inference to optimize resource usage without compromising performance accuracy.	Reduces computational overhead.	Enhancing the speed and efficiency of continuous cybersecurity monitoring systems.
FSDP-QLoRA	Combines Fully Sharded Data Parallelism (FSDP) with 4-bit quantization and LoRA to shard model parameters, optimizer states, and gradients across GPUs.	Scales up model training across multiple GPUs.	Enabling the collaborative training of security models across different organizations without requiring top-tier hardware.
Half-Quadratic Quantization (HQQ) [221]	A model quantization technique that enables the quantization of large models rapidly and accurately without the need for calibration data.	Works efficiently with CUDA/Triton kernels and aims for seamless integration with $torch.compile$.	HQQ can be employed in cybersecurity to protect models by reducing the precision of model weights, making it harder for attackers to reverse engineer or tamper with the models..
Multi-token Prediction [222]	A new training approach for large language models where models predict multiple future tokens simultaneously rather than the next token only.	Models trained with 4-token predictions can achieve up to 3x faster inference speeds, even with large batch sizes.	Multi-token prediction can enhance the modeling of sophisticated cyber attack patterns.
Trust Region Policy Optimization (TRPO) [223]	An advanced policy gradient method in reinforcement learning that addresses the inefficiencies of standard policy gradient methods.	TRPO enhances training stability by using trust regions to prevent overly large updates that could destabilize the policy.	In environments with dynamic and evolving threats, TRPO can help maintain a stable and effective response mechanism, adjusting policies incrementally to handle new types of malware.
Proximal Policy Optimization (PPO) [224]	A reinforcement learning technique designed to improve training stability by cautiously updating policies.	Prevents "falling off the cliff" scenarios where a policy update is too large could irreversibly damage the policy's effectiveness.	By limiting the extent of policy updates, PPO helps maintain a steady adaptation to evolving cybersecurity threats, reducing the risk of overfitting to specific attack patterns.
Direct Preference Optimization (DPO) [225]	A fine-tuning methodology for foundation models optimize policies directly using a Kullback–Leibler divergence-constrained framework, removing the need for a separate reward model.	Requires significantly less data and compute resources than previous methods like PPO.	Reduces the computational and data demands of continuously training cybersecurity models, allowing for more scalable solutions.
Odds Ratio Preference Optimization (ORPO) [226]	An algorithm designed for supervised fine-tuning (SFT) of language models that optimizes preference alignment without the need for a separate reference model.	Eliminates the need for an additional preference alignment phase, simplifying the fine-tuning process.	Enables dynamic adaptation of security models to new and evolving cyber threats by optimizing preference alignment efficiently.

an advanced method to enhance LLMs tailored for cybersecurity applications, focusing on aligning the model's responses with expert expectations in the security domain. This involves utilizing specialized preference datasets, which contain responses ranked by cybersecurity professionals, presenting a more challenging production process than typical instructional datasets. Techniques like Proximal Policy Optimization (PPO) leverage a reward model to evaluate how well text outputs align with security expert rankings, refining the model's training through adjustments based on KL divergence [224]. Direct Preference Optimization (DPO) further optimizes this by framing it as a classification challenge, using a stable reference model that avoids the complexities of training reward models and requires minimal hyperparameter adjustments [229]. These methods are crucial for reducing biases, fine-tuning threat detection accuracy, and enhancing the overall effectiveness of cybersecurity-focused LLMs.

In practical cybersecurity applications, the integration of RAG can be facilitated by orchestrators like LangChain, LlamaIndex, and FastRAG, which connect Language Models to relevant tools, databases, and other resources. These orchestrators ensure efficient information flow, enabling Language Models to seamlessly access and incorporate essential cybersecurity information [230]. Advanced techniques such as multi-query retrievers and HyDE are used to optimize the retrieval of relevant cybersecurity documents and adapt user queries into more effective forms for document retrieval. Furthermore, incorporating a memory system that recalls previous interactions allows these models to provide consistent and context-aware responses over time. This amalgamation of advanced retrieval mechanisms and memory enhancement through RAG significantly boosts the efficacy of Language Models in handling complex and evolving cybersecurity challenges, making them invaluable tools for tracking vulnerabilities, managing risks, and adhering to industry standards in the cybersecurity domain [231].

7) LLM deployments: Deploying LLMs offers a range of approaches tailored to the scale and specific needs of different applications. At one end of the spectrum, local deployment offers enhanced privacy and control, utilizing platforms like LM

Studio and *Ollama* to power apps directly on users' machines, thus capitalizing on the open-source nature of some LLMs. For more dynamic or temporary setups, frameworks such as *Gradio* and *Streamlit* allow developers to prototype and share demos quickly, with hosting options like Hugging Face Spaces providing an accessible path to broader distribution. On the industrial scale, deploying LLMs can require robust server setups, utilizing cloud services or on-premises infrastructure that might leverage specialized frameworks for peak performance and efficiency. Meanwhile, edge deployment strategies bring LLM capabilities to devices with limited resources, using advanced, lightweight frameworks to integrate smart capabilities directly into mobile and web platforms, ensuring responsiveness and accessibility across a broad spectrum of user environments [232], [233].

Currently, LLMs can be deployed on Phones. Microsoft [138] propose phi-3-mini. This highly efficient 3.8 billion parameter language model delivers robust performance on par with much larger models such as Mixtral 8x7B and GPT-3.5, achieving impressive scores like 69% on the MMLU and 8.38 on MT-bench. Remarkably, the phi-3-mini's compact size allows for deployment on mobile devices, expanding its accessibility and utility. This performance breakthrough is primarily attributed to an innovative approach in training data selection—a significantly enhanced version of the dataset used for phi-2, which integrates heavily filtered web data and synthetic data tailored for relevance and diversity. It has been further aligned to ensure the model's practicality in real-world applications for enhanced robustness, safety, and optimization for chat formats. In addition, the research extends into larger models, phi-3-small and phi-3-medium, which are trained on 4.8 trillion tokens with 7 billion and 14 billion parameters, respectively. These models retain the foundational strengths of phi-3-mini and exhibit superior performance, scoring up to 78% on MMLU and 8.9 on MT-bench, illustrating significant enhancements in language understanding capabilities with scaling. In addition, *AirLLM* [3] enhances memory management for inference, enabling large language models, such as those with 70 billion parameters (e.g., Llama3 70B), to operate on a single 4GB GPU card. This can be achieved without requiring quantization, distillation, pruning, or any other form of model compression that could diminish performance.

8) Securing LLMs: Securing LLMs is essential due to their inherent susceptibility to traditional software vulnerabilities and unique risks stemming from their design and operational methods. Specifically, LLMs are prone to prompt hacking, where techniques such as prompt injection can be used to manipulate model responses, prompt leaking that risks exposure of training data, and jailbreaking intended to circumvent built-in safety mechanisms. These specific threats necessitate implementing comprehensive security measures that directly address the unique challenges LLMs pose. Additionally, inserting backdoors during training, either by poisoning the data or embedding secret triggers, can significantly alter a model's behavior during inference, posing severe risks to data integrity and model reliability.

As discussed in Section VIII, to mitigate these threats effectively, organizations must adopt rigorous defensive strategies as recommended by the OWASP LLM security checklist [4]. This includes testing LLM applications against known vulnerabilities using methods like red teaming and specific tools such as *garak* [234] to identify and address security flaws. In addition, deploying continuous monitoring systems like *langfuse* [5] in production environments helps detect and rectify anomalous behaviors or potential breaches in real-time. The OWASP checklist also emphasizes the importance of governance frameworks that ensure data used in training is ethically sourced and handled, maintaining transparency about data sources and model training methodologies. This structured approach to security and governance ensures that LLMs are used responsibly and remain secure from conventional cyber threats and those unique to their operational nature.

9) Optimizing LLMs: Optimizing LLMs for production encompasses several crucial techniques to enhance speed, reduce memory requirements, and maintain output quality. One pivotal strategy is model quantization, which significantly reduces the precision of model weights—often to 4-bit or 8-bit—thereby decreasing the GPU memory requirements. Table XVII presents the optimization strategies for LLMs that can be adopted for Cybersecurity use cases. For instance, quantizing a model to 4-bit can bring down the VRAM requirement from 32 GB to just over 9 GB, allowing these models to run efficiently on consumer-level hardware like the RTX 3090 GPU. Therefore, advanced attention mechanisms such as Flash Attention reduce computation times by optimizing self-attention layers, which are integral to transformers [219]. This optimization is especially beneficial for handling long input sequences, where traditional self-attention mechanisms could become prohibitively expensive regarding memory and processing power [235], [236].

The quantization methods include Bitsnbytes, 4-bit GPTQ, 2-bit GPTQ, and GGUF quantization. Bitsnbytes introduces a k-bit quantization approach that significantly reduces memory consumption while maintaining performance [220]. It employs an 8-bit optimization using block-wise quantization to achieve 32-bit performance at a lower memory cost and uses LLM.Int() for 8-bit quantization during inference, halving the required memory without performance loss. Furthermore, QLoRA [202], or 4-bit quantization, enables the training of LLMs using memory-saving techniques that include a small set of trainable low-rank adaptation weights, allowing for performance preservation. In parallel, GPTQ is a novel quantization method for GPT models, facilitating the reduction of bit width to 3 or 4 bits per weight, enabling the operation of models as large as 175 billion parameters on a single GPU with minimal accuracy loss. This method provides substantial compression and speed advantages, making high-performance LLMs more accessible and cost-effective. Additionally, the GGUF format, supported by Hugging Face Hub and optimized for quick model loading and saving, enhances the efficiency of LLM inference.

[3]https://pypi.org/project/airllm/

[4]https://owasp.org/www-project-top-10-for-large-language-model-applications/

[5]https://github.com/langfuse/langfuse

Another effective optimization is incorporating lower-precision data formats such as bfloat16 for training and inference. This approach aligns with the training precision and avoids the computational overhead associated with float32 precision, optimizing resource usage without compromising performance accuracy. The potential VRAM requirements for different models using bfloat16 are substantial. For example, GPT-3 might require up to 350 GB. In comparison, smaller models like Llama-2-70b and Falcon-40b require 140 GB and 80 GB, respectively, illustrating the scale of resources needed even with efficient data formats [6].

Recently, FSDP-QLoRA [7], a new technique combining data parallelism, 4-bit quantization, and LoRA, was introduced by Answer.AI in collaboration with bitsandbytes. Utilizing Fully Sharded Data Parallelism (FSDP) to shard model parameters, optimizer states, and gradients across GPUs, this approach enables the training of LLMs up to 70 billion parameters on dual 24GB GPU systems. FSDP-QLoRA represents a significant step forward in making the training of large-scale LLMs.

Collectively, these techniques make it feasible to deploy powerful LLMs on a wider range of hardware and enhance their scalability and practicality in diverse applications, ensuring they can deliver high performance even under hardware constraints.

X. Conclusion

In this paper, we presented a comprehensive and in-depth review of the future of cybersecurity through the lens of Generative AI and Large Language Models (LLMs). Our exploration covered a wide range of LLM applications in cybersecurity, including hardware design security, intrusion detection, software engineering, design verification, cyber threat intelligence, malware detection, and phishing and spam detection, illustrating the broad potential of LLMs across various domains.

We provided a detailed examination of the evolution and current state of LLMs, highlighting advancements in 35 specific models, such as GPT-4, GPT-3.5, BERT, Falcon, and LLaMA. Our analysis included an in-depth look at the vulnerabilities associated with LLMs, such as prompt injection, insecure output handling, training and inference data poisoning, DDoS attacks, and adversarial natural language instructions. We discussed mitigation strategies to protect these models, offering a thorough understanding of potential attack scenarios and prevention techniques.

Our evaluation of 40 LLM models in terms of cybersecurity knowledge and hardware security demonstrated their varying strengths and weaknesses. We also conducted a detailed assessment of cybersecurity datasets used for LLM training and testing, from data creation to usage, identifying gaps and opportunities for future research.

We addressed the challenges and limitations of employing LLMs in cybersecurity settings, including the difficulty of defending against adversarial attacks and ensuring model robustness. Additionally, we explored advanced techniques like Half-Quadratic Quantization (HQQ), Reinforcement Learning with Human Feedback (RLHF), Direct Preference Optimization (DPO), Odds Ratio Preference Optimization (ORPO), GPT-Generated Unified Format (GGUF), Quantized Low-Rank Adapters (QLoRA), and Retrieval-Augmented Generation (RAG) to enhance real-time cybersecurity defenses and improve the sophistication of LLM applications in threat detection and response.

Our findings underscore the significant potential of LLMs in transforming cybersecurity practices. By integrating LLMs into future cybersecurity frameworks, we can leverage their capabilities to develop more robust and sophisticated defenses against evolving cyber threats. The strategic direction outlined in this paper aims to guide future research and deployment, emphasizing the importance of innovation and resilience in safeguarding digital infrastructures.

[6]https://huggingface.co/docs/transformers/main/en/llm_tutorial_optimization

[7]https://huggingface.co/docs/bitsandbytes/main/en/fsdp_qlora

References

[1] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[5] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, and X. Cheng, "On protecting the data privacy of large language models (llms): A survey," *arXiv preprint arXiv:2403.05156*, 2024.

[6] D. Myers, R. Mohawesh, V. I. Chellaboina, A. L. Sathvik, P. Venkatesh, Y.-H. Ho, H. Henshaw, M. Alhawawreh, D. Berdik, and Y. Jararweh, "Foundation and large language models: fundamentals, challenges, opportunities, and social impacts," *Cluster Computing*, vol. 27, no. 1, pp. 1–26, 2024.

[7] S. Tonmoy, S. Zaman, V. Jain, A. Rani, V. Rawte, A. Chadha, and A. Das, "A comprehensive survey of hallucination mitigation techniques in large language models," *arXiv preprint arXiv:2401.01313*, 2024.

[8] I. H. Sarker, H. Janicke, M. A. Ferrag, and A. Abuadbba, "Multi-aspect rule-based ai: Methods, taxonomy, challenges and directions toward automation, intelligence and transparent cybersecurity modeling for critical infrastructures," *Internet of Things*, p. 101110, 2024.

[9] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[10] Y. Yan, Y. Zhang, and K. Huang, "Depending on yourself when you should: Mentoring llm with rl agents to become the master in cybersecurity games," *arXiv preprint arXiv:2403.17674*, 2024.

[11] M. Sladić, V. Valeros, C. Catania, and S. Garcia, "Llm in the shell: Generative honeypots," *arXiv preprint arXiv:2309.00155*, 2023.

[12] W. Tann, Y. Liu, J. H. Sim, C. M. Seah, and E.-C. Chang, "Using large language models for cybersecurity capture-the-flag challenges and certification questions," *arXiv preprint arXiv:2308.10443*, 2023.

[13] O. G. Lira, A. Marroquin, and M. A. To, "Harnessing the advanced capabilities of llm for adaptive intrusion detection systems," in *International Conference on Advanced Information Networking and Applications*. Springer, 2024, pp. 453–464.

[14] C. Ebert and M. Beck, "Artificial intelligence for cybersecurity," *IEEE Software*, vol. 40, no. 6, pp. 27–34, 2023.

[15] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, 2024.

[16] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, É. Goffinet, D. Hesslow, J. Launay, Q. Malartic *et al.*, "The falcon series of open language models," *arXiv preprint arXiv:2311.16867*, 2023.

[17] H. Zhou, C. Hu, Y. Yuan, Y. Cui, Y. Jin, C. Chen, H. Wu, D. Yuan, L. Jiang, D. Wu, X. Liu, C. Zhang, X. Wang, and J. Liu, "Large language model (llm) for telecommunications: A comprehensive survey on principles, key techniques, and opportunities," 2024.

[18] H. Lai and M. Nissim, "A survey on automatic generation of figurative language: From rule-based systems to large language models," *ACM Computing Surveys*, 2024.

[19] M. A. Ferrag, M. Ndhlovu, N. Tihanyi, L. C. Cordeiro, M. Debbah, T. Lestable, and N. S. Thandi, "Revolutionizing cyber threat detection with large language models: A privacy-preserving bert-based lightweight model for iot/iiot devices," *IEEE Access*, 2024.

[20] Z. Liu, "A review of advancements and applications of pre-trained language models in cybersecurity," in *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*, 2024, pp. 1–10.

[21] S. Jamal, H. Wimmer, and I. H. Sarker, "An improved transformer-based model for detecting phishing, spam and ham emails: A large language model approach," *Security and Privacy*, p. e402, 2024. [Online]. Available: https://doi.org/10.1002/spy2.402

[22] F. R. Alzaabi and A. Mehmood, "A review of recent advances, challenges, and opportunities in malicious insider threat detection using machine learning methods," *IEEE Access*, vol. 12, pp. 30 907–30 927, 2024.

[23] R. Fang, R. Bindu, A. Gupta, and D. Kang, "Llm agents can autonomously exploit one-day vulnerabilities," *arXiv preprint arXiv:2404.08144*, 2024.

[24] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[25] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema, N. M. Fahad, S. Sakib, M. M. J. Mim, J. Ahmad, M. E. Ali, and S. Azam, "A review on large language models: Architectures, applications, taxonomies, open issues and challenges," *IEEE Access*, vol. 12, pp. 26 839–26 874, 2024.

[26] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, 2023.

[27] D. Saha, S. Tarek, K. Yahyaei, S. K. Saha, J. Zhou, M. Tehranipoor, and F. Farahmandi, "Llm for soc security: A paradigm shift," *arXiv preprint arXiv:2310.06046*, 2023.

[28] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–40, 2023.

[29] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu *et al.*, "Instruction tuning for large language models: A survey," *arXiv preprint arXiv:2308.10792*, 2023.

[30] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," *arXiv preprint arXiv:2310.03533*, 2023.

[31] J. Wu, W. Gan, Z. Chen, S. Wan, and P. S. Yu, "Multimodal large language models: A survey," *arXiv preprint arXiv:2311.13165*, 2023.

[32] Y. Liu, Y. Yao, J.-F. Ton, X. Zhang, R. G. H. Cheng, Y. Klochkov, M. F. Taufiq, and H. Li, "Trustworthy llms: a survey and guideline for evaluating large language models' alignment," *arXiv preprint arXiv:2308.05374*, 2023.

[33] L. Hu, Z. Liu, Z. Zhao, L. Hou, L. Nie, and J. Li, "A survey of knowledge enhanced pre-trained language models," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[34] H. Zhang, H. Song, S. Li, M. Zhou, and D. Song, "A survey of controllable text generation using transformer-based pre-trained language models," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–37, 2023.

[35] Z. He, Z. Li, and S. Yang, "Large language models for blockchain security: A systematic literature review," *arXiv preprint arXiv:2403.14280*, 2024.

[36] Y. Yigit, M. A. Ferrag, I. H. Sarker, L. A. Maglaras, C. Chrysoulas, N. Moradpoor, and H. Janicke, "Critical infrastructure protection: Generative ai, challenges, and opportunities," *arXiv preprint arXiv:2405.04874*, 2024.

[37] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, pp. 1–27, 2024.

[38] H. Xu, S. Wang, N. Li, Y. Zhao, K. Chen, K. Wang, Y. Liu, T. Yu, and H. Wang, "Large language models for cyber security: A systematic literature review," *arXiv preprint arXiv:2405.04760*, 2024.

[39] Z. Han, C. Gao, J. Liu, S. Q. Zhang *et al.*, "Parameter-efficient fine-tuning for large models: A comprehensive survey," *arXiv preprint arXiv:2403.14608*, 2024.

[40] J. Zhang, H. Bu, H. Wen, Y. Chen, L. Li, and H. Zhu, "When llms meet cybersecurity: A systematic literature review," *arXiv preprint arXiv:2405.03644*, 2024.

[41] C. Cui, Y. Ma, X. Cao, W. Ye, Y. Zhou, K. Liang, J. Chen, J. Lu, Z. Yang, K.-D. Liao *et al.*, "A survey on multimodal large language models for autonomous driving," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 958–979.

[42] G. Bai, Z. Chai, C. Ling, S. Wang, J. Lu, N. Zhang, T. Shi, Z. Yu, M. Zhu, Y. Zhang *et al.*, "Beyond efficiency: A systematic survey of resource-efficient large language models," *arXiv preprint arXiv:2401.00625*, 2024.

[43] S. Tian, Q. Jin, L. Yeganova, P.-T. Lai, Q. Zhu, X. Chen, Y. Yang, Q. Chen, W. Kim, D. C. Comeau *et al.*, "Opportunities and challenges for chatgpt and large language models in biomedicine and health," *Briefings in Bioinformatics*, vol. 25, no. 1, p. bbad493, 2024.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[45] S. M. Kasongo, "A deep learning technique for intrusion detection system using a recurrent neural networks based framework," *Computer Communications*, vol. 199, pp. 113–125, 2023.

[46] S. M. Sohi, J.-P. Seifert, and F. Ganji, "Rnnids: Enhancing network intrusion detection systems through deep learning," *Computers & Security*, vol. 102, p. 102151, 2021.

[47] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[48] H. Sedjelmaci, F. Guenab, S.-M. Senouci, H. Moustafa, J. Liu, and S. Han, "Cyber security based on artificial intelligence for cyber-physical systems," *IEEE Network*, vol. 34, no. 3, pp. 6–7, 2020.

[49] P. Dixit and S. Silakari, "Deep learning algorithms for cybersecurity applications: A technological and status review," *Computer Science Review*, vol. 39, p. 100317, 2021.

[50] S. Gaba, I. Budhiraja, V. Kumar, S. Martha, J. Khurmi, A. Singh, K. K. Singh, S. Askar, and M. Abouhawwash, "A systematic analysis of enhancing cyber security using deep learning for cyber physical systems," *IEEE Access*, 2024.

[51] G. Parra, L. Selvera, J. Khoury, H. Irizarry, E. Bou-Harb, and P. Rad, "Interpretable federated transformer log learning for cloud threat forensics," in *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2022.

[52] N. Ziems and S. Wu, "Security vulnerability detection using deep learning natural language processing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–6.

[53] Z. Wu, H. Zhang, P. Wang, and Z. Sun, "Rtids: A robust transformer-based approach for intrusion detection system," *IEEE Access*, vol. 10, pp. 64 375–64 387, 2022.

[54] F. Demirkıran, A. Çayır, U. Ünal, and H. Dağ, "An ensemble of pre-trained transformer models for imbalanced multiclass malware classification," *Computers & Security*, vol. 121, p. 102846, 2022.

[55] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21 954–21 961, 2017.

[56] D. Güera and E. J. Delp, "Deepfake video detection using recurrent neural networks," in *2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS)*. IEEE, 2018, pp. 1–6.

[57] S. Althubiti, W. Nick, J. Mason, X. Yuan, and A. Esterline, "Applying long short-term memory recurrent neural network for intrusion detection," in *SoutheastCon 2018*. IEEE, 2018, pp. 1–5.

[58] C. Xu, J. Shen, X. Du, and F. Zhang, "An intrusion detection system using a deep neural network with gated recurrent units," *IEEE Access*, vol. 6, pp. 48 697–48 707, 2018.

[59] M. A. Ferrag and L. Maglaras, "Deepcoin: A novel deep learning and blockchain-based energy exchange framework for smart grids," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1285–1297, 2019.

[60] A. Ghourabi, "A security model based on lightgbm and transformer to protect healthcare systems from cyberattacks," *IEEE Access*, vol. 10, pp. 48 890–48 903, 2022.

[61] C. Thapa, S. I. Jang, M. E. Ahmed, S. Camtepe, J. Pieprzyk, and S. Nepal, "Transformer-based language models for software vulnerability detection," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 481–496.

[62] P. Ranade, A. Piplai, S. Mittal, A. Joshi, and T. Finin, "Generating fake cyber threat intelligence using transformer-based models," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–9.

[63] M. Fu and C. Tantithamthavorn, "Linevul: a transformer-based line-level vulnerability prediction," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 608–620.

[64] C. Mamede, E. Pinconschi, and R. Abreu, "A transformer-based ide plugin for vulnerability detection," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–4.

[65] P. Evangelatos, C. Iliou, T. Mavropoulos, K. Apostolou, T. Tsikrika, S. Vrochidis, and I. Kompatsiaris, "Named entity recognition in cyber threat intelligence using transformer-based models," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2021, pp. 348–353.

[66] F. Hashemi Chaleshtori and I. Ray, "Automation of vulnerability information extraction using transformer-based language models," in *Computer Security. ESORICS 2022 International Workshops*. Springer, 2023, pp. 645–665.

[67] A. Chawla, B. Lee, S. Fallon, and P. Jacob, "Host based intrusion detection system with combined cnn/rnn model," in *ECML PKDD 2018 Workshops: Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings 18*. Springer, 2019, pp. 149–158.

[68] I. Ullah and Q. H. Mahmoud, "Design and development of rnn anomaly detection model for iot networks," *IEEE Access*, vol. 10, pp. 62 722–62 750, 2022.

[69] A. A. E.-B. Donkol, A. G. Hafez, A. I. Hussein, and M. M. Mabrook, "Optimization of intrusion detection using likely point pso and enhanced lstm-rnn hybrid technique in communication networks," *IEEE Access*, vol. 11, pp. 9469–9482, 2023.

[70] Z. Zhao, Z. Li, J. Jiang, F. Yu, F. Zhang, C. Xu, X. Zhao, R. Zhang, and S. Guo, "Ernn: Error-resilient rnn for encrypted traffic detection towards network-induced phenomena," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[71] X. Wang, S. Wang, P. Feng, K. Sun, S. Jajodia, S. Benchaaboun, and F. Geck, "Patchrnn: A deep learning-based system for security patch identification," in *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, 2021, pp. 595–600.

[72] H. Polat, M. Türkoğlu, O. Polat, and A. Şengür, "A novel approach for accurate detection of the ddos attacks in sdn-based scada systems based on deep recurrent neural networks," *Expert Systems with Applications*, vol. 197, p. 116748, 2022.

[73] S. Liu, Y. Li, and Y. Liu, "Commitbart: A large pre-trained model for github commits," *arXiv preprint arXiv:2208.08100*, 2022.

[74] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "On hardware security bug code fixes by prompting large language models," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2024.

[75] L. J. Wan, Y. Huang, Y. Li, H. Ye, J. Wang, X. Zhang, and D. Chen, "Invited paper: Software/hardware co-design for llm and its application for design verification," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 435–441.

[76] E. Jang, J. Cui, D. Yim, Y. Jin, J.-W. Chung, S. Shin, and Y. Lee, "Ignore me but don't replace me: Utilizing non-linguistic elements for pretraining on the cybersecurity domain," *arXiv preprint*, 2024, to appear in NAACL Findings 2024.

[77] M. Bayer, P. Kuehn, R. Shanehsaz, and C. Reuter, "Cysecbert: A domain-adapted language model for the cybersecurity domain," *ACM Transactions on Privacy and Security*, vol. 27, no. 2, pp. 1–20, 2024.

[78] A. Shestov, R. Levichev, R. Mussabayev, E. Maslov, A. Cheshkov, and P. Zadorozhny, "Finetuning large language models for vulnerability detection," *arXiv preprint*, 2024, version 4.

[79] F. He, F. Li, and P. Liang, "Enhancing smart contract security: Leveraging pre-trained language models for advanced vulnerability detection," *IET Blockchain*, 2024, first published: 29 March 2024.

[80] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, and Y. Chen, "Vulnerability detection with code language models: How far are we?" *arXiv preprint arXiv:2403.18624*, 2024.

[81] T. Koide, N. Fukushi, H. Nakano, and D. Chiba, "Chatspamdetector: Leveraging large language models for effective phishing email detection," *arXiv preprint arXiv:2402.18093*, 2024.

[82] F. Heiding, B. Schneier, A. Vishwanath, J. Bernstein, and P. S. Park, "Devising and detecting phishing emails using large language models," *IEEE Access*, 2024.

[83] R. Chataut, P. K. Gyawali, and Y. Usman, "Can ai keep you safe? a study of large language models for phishing detection," in *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2024, pp. 0548–0554.

[84] M. Rostami, M. Chilese, S. Zeitouni, R. Kande, J. Rajendran, and A.-R. Sadeghi, "Beyond random inputs: A novel ml-based hardware fuzzing," 2024.

[85] Z. Zhang, G. Chadwick, H. McNally, Y. Zhao, and R. Mullins, "Llm4dv: Using large language models for hardware test stimuli generation," 2023.

[86] M. Nair, R. Sadhukhan, and D. Mukhopadhyay, "Generating secure hardware using chatgpt resistant to cwes," Cryptology ePrint Archive, Paper 2023/212, 2023, https://eprint.iacr.org/2023/212. [Online]. Available: https://eprint.iacr.org/2023/212

[87] L. J. Wan, Y. Huang, Y. Li, H. Ye, J. Wang, X. Zhang, and D. Chen, "Software/hardware co-design for llm and its application for design verification," in *Proceedings of the 29th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '24. IEEE Press, 2024, p. 435–441. [Online]. Available: https://doi.org/10.1109/ASP-DAC58780.2024.10473893

[88] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," 2023.

[89] N. Tihanyi, M. A. Ferrag, R. Jain, and M. Debbah, "Cybermetric: A benchmark dataset for evaluating large language models knowledge in cybersecurity," *arXiv preprint arXiv:2402.07688*, 2024.

[90] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model guided protocol fuzzing," in *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, 2024.

[91] V.-T. Pham, M. Böhme, and A. Roychoudhury, "Aflnet: A greybox fuzzer for network protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 2020, pp. 460–465.

[92] S. Qin, F. Hu, Z. Ma, B. Zhao, T. Yin, and C. Zhang, "Nsfuzz: Towards efficient and state-aware network service fuzzing," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 6, sep 2023. [Online]. Available: https://doi.org/10.1145/3580598

[93] J. Wang, L. Yu, and X. Luo, "Llmif: Augmented large language model for fuzzing iot devices," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 196–196.

[94] M. Ren, X. Ren, H. Feng, J. Ming, and Y. Lei, "Z-fuzzer: device-agnostic fuzzing of zigbee protocol implementation," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 347–358. [Online]. Available: https://doi.org/10.1145/3448300.3468296

[95] J. Pereyda, "Boofuzz: Network protocol fuzzing for humans," https://boofuzz.readthedocs.io/en/stable, 2020.

[96] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[97] OpenAI, "Gpt-4 technical report," 2023.

[98] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[99] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[100] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[101] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[102] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[103] W. Qi, Y. Yan, Y. Gong, D. Liu, N. Duan, J. Chen, R. Zhang, and M. Zhou, "Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training," *arXiv preprint arXiv:2001.04063*, 2020.

[104] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay, "The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only," *arXiv preprint arXiv:2306.01116*, 2023.

[105] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[106] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[107] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[108] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[109] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[110] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[111] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[112] The MosaicML NLP Team, "Mpt-30b: Raising the bar for open-source foundation models," June 2023, accessed: 2023-12-10. [Online]. Available: https://www.mosaicml.com/blog/mpt-30b

[113] 01.AI, "Yi-34b," https://huggingface.co/01-ai/Yi-34B, 2023, accessed: 2023-12-10.

[114] TIIUAE, "Falcon-11b," https://huggingface.co/tiiuae/falcon-11B, 2024, accessed: 2024-05-01.

[115] P. Haller, J. Golde, and A. Akbik, "Pecc: Problem extraction and coding challenges," *arXiv preprint arXiv:2404.18766*, 2024.

[116] A. Z. Yang, Y. Takashima, B. Paulsen, J. Dodds, and D. Kroening, "Vert: Verified equivalent rust transpilation with few-shot learning," *arXiv preprint arXiv:2404.18852*, 2024.

[117] D. Nichols, P. Polasam, H. Menon, A. Marathe, T. Gamblin, and A. Bhatele, "Performance-aligned llms for generating fast code," *arXiv preprint arXiv:2404.18864*, 2024.

[118] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[119] T. H. Le, M. A. Babar, and T. H. Thai, "Software vulnerability prediction in low-resource languages: An empirical study of codebert and chatgpt," *arXiv preprint arXiv:2404.17110*, 2024.

[120] B. Guan, Y. Wan, Z. Bi, Z. Wang, H. Zhang, Y. Sui, P. Zhou, and L. Sun, "Codeip: A grammar-guided multi-bit watermark for large language models of code," *arXiv preprint arXiv:2404.15639*, 2024.

[121] X.-C. Wen, X. Wang, Y. Chen, R. Hu, D. Lo, and C. Gao, "Vuleval: Towards repository-level evaluation of software vulnerability detection," *arXiv preprint arXiv:2404.15596*, 2024.

[122] Z. Zhang, C. Chen, B. Liu, C. Liao, Z. Gong, H. Yu, J. Li, and R. Wang, "Unifying the perspectives of nlp and software engineering: A survey on language models for code," *arXiv preprint arXiv:2311.07989*, 2023.

[123] L. B. Allal, R. Li, D. Kocetkov, C. Mou, C. Akiki, C. M. Ferrandis, N. Muennighoff, M. Mishra, A. Gu, M. Dey *et al.*, "Santacoder: don't reach for the stars!" *arXiv preprint arXiv:2301.03988*, 2023.

[124] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, "Starcoder: may the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023.

[125] Hugging Face & ServiceNow, "Huggingfaceh4/starchat-alpha," https://huggingface.co/HuggingFaceH4/starchat-alpha, 2023, accessed: 2023-12-10.

[126] E. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, and Y. Zhou, "Codegen2: Lessons for training llms on programming and natural languages," *arXiv preprint arXiv:2305.02309*, 2023.

[127] Salesforce AI Research, "Codegen2.5: Small, but mighty," 2023, accessed: 2023-12-10. [Online]. Available: https://blog.salesforceairesearch.com/codegen25/

[128] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, and S. C. Hoi, "Codet5+: Open code large language models for code understanding and generation," *arXiv preprint arXiv:2305.07922*, 2023.

[129] E. Nijkamp, T. Xie, H. Hayashi, B. Pang, C. Xia, C. Xing, J. Vig, S. Yavuz, P. Laban, B. Krause *et al.*, "Xgen-7b technical report," *arXiv preprint arXiv:2309.03450*, 2023.

[130] Replit, Inc., "replit-code-v1-3b," 2023, accessed: 2023-12-10. [Online]. Available: https://huggingface.co/replit/replit-code-v1-3b

[131] Deci AI, "Introducing decicoder: The new gold standard in efficient and accurate code generation," August 2023, accessed: 2023-12-10. [Online]. Available: https://deci.ai/blog/decicoder-efficient-and-accurate-code-generation-llm/

[132] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[133] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, and T. Zhu, "Qwen technical report," arXiv preprint, Tech. Rep., 2023, 59 pages, 5 figures.

[134] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, F. Luo, Y. Xiong, and W. Liang, "Deepseek-coder: When the large language model meets programming – the rise of code intelligence," *arXiv preprint*, 2024, submitted on 25 Jan 2024, Last revised 26 Jan 2024.

[135] C. Team, A. J. Hartman, A. Hu, C. A. Choquette-Choo, H. Zhao, J. Fine, J. Hui, J. Shen, J. Kelley, J. Howland, K. Bansal, L. Vilnis, M. Wirth, N. Nguyen, P. Michel, P. Choy, P. Joshi, R. Kumar, S. Hashmi, S. Agrawal, S. Zuo, T. Warkentin, and Z. Gong, "Codegemma: Open code models based on gemma," 2024. [Online]. Available: https://goo.gle/codegemma

[136] M. Mishra, M. Stallone, G. Zhang, Y. Shen, A. Prasad, A. Meza Soria, M. Merler, P. Selvam, S. Surendran, S. Singh, M. Sethi, X.-H. Dang, P. Li, K.-L. Wu, S. Zawad, A. Coleman, M. White, M. Lewis, R. Pavuluri, Y. Koyfman, B. Lublinsky, M. de Bayser, I. Abdelaziz, K. Basu, M. Agarwal, Y. Zhou, C. Johnson, A. Goyal, H. Patel, Y. Shah, P. Zerfos, H. Ludwig, A. Munawar, M. Crouse, P. Kapanipathi, S. Salaria, B. Calio, S. Wen, S. Seelam, B. Belgodere, C. Fonseca, A. Singhee, N. Desai, D. D. Cox, R. Puri, and R. Panda, "Granite code models: A family of open foundation models for code intelligence," *arXiv preprint arXiv:2405.04324*, May 2024.

[137] DeepSeek-AI, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, May 2024, submitted on 7 May 2024 (v1), last revised 8 May 2024 (this version, v2). [Online]. Available: https://arxiv.org/abs/2405.04434

[138] M. A. et al., "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[139] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. Renard Lavaud, M.-A. Lachaux, P. Stock, T. Le Scao, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023, submitted on 10 Oct 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.06825

[140] N. Dey, G. Gosal, Z. C. Chen, H. Khachane, W. Marshall, R. Pathria, M. Tom, and J. Hestness, "Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster," *arXiv preprint arXiv:2304.03208*, 2023, submitted on 6 Apr 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.03208

[141] ZySec-AI, "Zysec-ai: Project zysec," Webpage, accessed: 2024-05-01. [Online]. Available: https://github.com/ZySec-AI/project-zysec

[142] DeciAI Research Team, "Decilm-7b," 2023. [Online]. Available: https://huggingface.co/Deci/DeciLM-7B

[143] L. Tunstall, E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. von Werra, C. Fourrier, N. Habib, N. Sarrazin, O. Sanseviero, A. M. Rush, and T. Wolf, "Zephyr: Direct distillation of lm alignment," 2023.

[144] M. Conover, M. Hayes, A. Mathur, J. Xie, J. Wan, S. Shah, A. Ghodsi, P. Wendell, M. Zaharia, and R. Xin. (2023) Free dolly: Introducing the world's first truly open instruction-tuned llm. [Online]. Available: https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm

[145] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.

[146] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, "The pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[147] D. Kocetkov, R. Li, L. B. Allal, J. Li, C. Mou, C. M. Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf *et al.*, "The stack: 3 tb of permissively licensed source code," *arXiv preprint arXiv:2211.15533*, 2022.

[148] H. Laurençon, L. Saulnier, T. Wang, C. Akiki, A. Villanova del Moral, T. Le Scao, L. Von Werra, C. Mou, E. González Ponferrada, H. Nguyen *et al.*, "The bigscience roots corpus: A 1.6 tb composite multilingual dataset," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 809–31 826, 2022.

[149] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei *et al.*, "Starcoder 2 and the stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[150] R. Schuster, C. Song, E. Tromer, and V. Shmatikov, "You autocomplete me: Poisoning vulnerabilities in neural code completion," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1559–1575.

[151] O. Asare, M. Nagappan, and N. Asokan, "Is github's copilot as bad as humans at introducing vulnerabilities in code?" *Empirical Software Engineering*, vol. 28, no. 6, p. 129, 2023.

[152] G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, and B. Dolan-Gavitt, "Lost at c: A user study on the security implications of large language model code assistants," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 2205–2222.

[153] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with ai assistants?" in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2785–2799.

[154] S. Hamer, M. d'Amorim, and L. Williams, "Just another copy and paste? comparing the security vulnerabilities of chatgpt generated code and stackoverflow answers," *arXiv preprint arXiv:2403.15600*, 2024.

[155] D. Cotroneo, R. De Luca, and P. Liguori, "Devaic: A tool for security assessment of ai-generated code," *arXiv preprint arXiv:2404.07548*, 2024.

[156] R. Tóth, T. Bisztray, and L. Erdodi, "Llms in web-development: Evaluating llm-generated php code unveiling vulnerabilities and limitations," *arXiv preprint arXiv:2404.14459*, 2024.

[157] N. Tihanyi, T. Bisztray, M. A. Ferrag, R. Jain, and L. C. Cordeiro, "Do neutral prompts produce insecure code? formai-v2 dataset: Labelling vulnerabilities in code generated by large language models," 2024.

[158] N. S. Harzevili, A. B. Belle, J. Wang, S. Wang, Z. Ming, N. Nagappan *et al.*, "A survey on automated software vulnerability detection using machine learning and deep learning," *arXiv preprint arXiv:2306.11673*, 2023.

[159] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning," *IEEE Access*, vol. 10, pp. 40 281–40 306, 2022.

[160] N. Tihanyi, T. Bisztray, R. Jain, M. A. Ferrag, L. C. Cordeiro, and V. Mavroeidis, "The formai dataset: Generative ai in software security through the lens of formal verification," in *Proceedings of the 19th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2023, pp. 33–43.

[161] Y. Zheng, S. Pujar, B. Lewis, L. Buratti, E. Epstein, B. Yang, J. Laredo, A. Morari, and Z. Su, "D2a: A dataset built for ai-based vulnerability detection methods using differential analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021, pp. 111–120.

[162] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks," *arXiv e-prints*, p. arXiv:1909.03496, Sep. 2019.

[163] H. Hanif, M. H. N. M. Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, vol. 179, p. 103009, 2021.

[164] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, "Automated vulnerability detection in source code using deep representation learning," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 757–762.

[165] ——, "Automated vulnerability detection in source code using deep representation learning," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 757–762.

[166] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 914–919.

[167] L. Wartschinski, Y. Noller, T. Vogel, T. Kehrer, and L. Grunske, "Vudenc: Vulnerability detection with deep learning on a natural codebase for python," *Information and Software Technology*, vol. 144, p. 106809, 2022, arXiv preprint arXiv:2201.08441. [Online]. Available: https://doi.org/10.48550/arXiv.2201.08441

[168] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A c/c++ code vulnerability dataset with code changes and cve summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 508–512. [Online]. Available: https://doi.org/10.1145/3379597.3387501

[169] G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2021, pp. 30–39.

[170] G. Nikitopoulos, K. Dritsa, P. Louridas, and D. Mitropoulos, "Crossvul: a cross-language vulnerability dataset with commit data," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1565–1569.

[171] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2244–2258, 2022.

[172] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," *arXiv preprint arXiv:1801.01681*, 2018.

[173] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. Wagner, "DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection," *arXiv e-prints*, p. arXiv:2304.00409, Apr. 2023.

[174] D. N. Gadde, A. Kumar, T. Nalapat, E. Rezunov, and F. Cappellini, "All artificial, less intelligence: Genai through the lens of formal verification," *Infineon Technologies*, 2024.

[175] OWASP Foundation, "Owasp top 10 for large language model applications," https://owasp.org/www-project-top-10-for-large-language-model-applications/, 2023, accessed: 2023-12-26.

[176] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," *arXiv preprint arXiv:2211.09527*, 2022.

[177] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "More than you've asked for: A comprehensive analysis of novel prompt injection threats to application-integrated large language models," *arXiv e-prints*, pp. arXiv–2302, 2023.

[178] J. Yan, V. Yadav, S. Li, L. Chen, Z. Tang, H. Wang, V. Srinivasan, X. Ren, and H. Jin, "Virtual prompt injection for instruction-tuned large language models," *arXiv preprint arXiv:2307.16888*, 2023.

[179] R. Pedro, D. Castro, P. Carreira, and N. Santos, "From prompt injections to sql injection attacks: How protected is your llm-integrated web application?" *arXiv preprint arXiv:2308.01990*, 2023.

[180] S. Abdelnabi, K. Greshake, S. Mishra, C. Endres, T. Holz, and M. Fritz, "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*, 2023, pp. 79–90.

[181] Y. Liu, G. Deng, Y. Li, K. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, and Y. Liu, "Prompt injection attack against llm-integrated applications," *arXiv preprint arXiv:2306.05499*, 2023.

[182] J. Yan, V. Yadav, S. Li, L. Chen, Z. Tang, H. Wang, V. Srinivasan, X. Ren, and H. Jin, "Backdooring instruction-tuned large language models with virtual prompt injection," in *NeurIPS 2023 Workshop on Backdoors in Deep Learning-The Good, the Bad, and the Ugly*, 2023.

[183] D. Glukhov, I. Shumailov, Y. Gal, N. Papernot, and V. Papyan, "Llm censorship: A machine learning challenge or a computer security problem?" *arXiv preprint arXiv:2307.10719*, 2023.

[184] F. Wu, X. Liu, and C. Xiao, "Deceptprompt: Exploiting llm-driven code generation via adversarial natural language instructions," *arXiv preprint arXiv:2312.04730*, 2023.

[185] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," *arXiv preprint arXiv:2307.15043*, 2023.

[186] Z. Yang, X. He, Z. Li, M. Backes, M. Humbert, P. Berrang, and Y. Zhang, "Data poisoning attacks against multimodal encoders," in *International Conference on Machine Learning*. PMLR, 2023, pp. 39 299–39 313.

[187] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli, "Wild patterns reloaded: A survey of machine learning security against training data poisoning," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–39, 2023.

[188] P. Gupta, K. Yadav, B. B. Gupta, M. Alazab, and T. R. Gadekallu, "A novel data poisoning attack in federated learning based on inverted loss function," *Computers & Security*, vol. 130, p. 103270, 2023.

[189] J. He, W. Jiang, G. Hou, W. Fan, R. Zhang, and H. Li, "Talk too much: Poisoning large language models under token limit," *arXiv preprint arXiv:2404.14795*, 2024.

[190] A. B. de Neira, B. Kantarci, and M. Nogueira, "Distributed denial of service attack prediction: Challenges, open issues and opportunities," *Computer Networks*, vol. 222, 2023.

[191] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in ddos attacks: Trends and challenges," *IEEE Communications Surveys and Tutorials*, vol. 17, 2015.

[192] O. Osanaiye, K. K. R. Choo, and M. Dlodlo, "Distributed denial of service (ddos) resilience in cloud: Review and conceptual cloud ddos mitigation framework," 2016.

[193] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, 2016.

[194] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, and M. Du, "Explainability for large language models: A survey," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 2, pp. 1–38, 2024.

[195] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang, "Wizardlm: Empowering large language models to follow complex instructions," *arXiv preprint arXiv:2304.12244*, 2023.

[196] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. Hashimoto, "Alpaca: a strong, replicable instruction-following model; 2023," *URL https://crfm. stanford. edu/2023/03/13/alpaca. html*.

[197] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[198] A. Andonian, Q. Anthony, S. Biderman, S. Black, P. Gali, L. Gao, E. Hallahan, J. Levy-Kramer, C. Leahy, L. Nestler, K. Parker, M. Pieler, J. Phang, S. Purohit, H. Schoelkopf, D. Stander, T. Songz, C. Tigges, B. Thérien, P. Wang, and S. Weinbach, "GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch," 9 2023. [Online]. Available: https://www.github.com/eleutherai/gpt-neox

[199] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[200] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*. PMLR, 2019, pp. 2790–2799.

[201] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[202] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[203] X. Wu, H. Xia, S. Youn, Z. Zheng, S. Chen, A. Bakhtiari, M. Wyatt, R. Y. Aminabadi, Y. He, O. Ruwase, L. Song *et al.*, "Zeroquant(4+2): Redefining llms quantization with a new fp6-centric strategy for diverse generative tasks," *arXiv preprint arXiv:2312.08583*, 2023.

[204] H. Xia, Z. Zheng, X. Wu, S. Chen, Z. Yao, S. Youn, A. Bakhtiari, M. Wyatt, D. Zhuang, Z. Zhou *et al.*, "Fp6-llm: Efficiently serving large language models through fp6-centric algorithm-system co-design," *arXiv preprint arXiv:2401.14112*, 2024.

[205] M. Bhatt, S. Chennabasappa, C. Nikolaidis, S. Wan, I. Evtimov, D. Gabi, D. Song, F. Ahmad, C. Aschermann, L. Fontana *et al.*, "Purple llama cyberseceval: A secure coding benchmark for language models," *arXiv preprint arXiv:2312.04724*, 2023.

[206] Z. Liu, "Secqa: A concise question-answering dataset for evaluating large language models in computer security," *arXiv preprint arXiv:2312.15838*, 2023.

[207] M. Bhatt, S. Chennabasappa, Y. Li, C. Nikolaidis, D. Song, S. Wan, F. Ahmad, C. Aschermann, Y. Chen, D. Kapil, D. Molnar, S. Whitman, and J. Saxe, "Cyberseceval 2: A wide-ranging cybersecurity evaluation suite for large language models," 2024.

[208] N. Li, A. Pan, A. Gopal, S. Yue, D. Berrios, A. Gatti, J. D. Li, A.-K. Dombrowski, S. Goel, L. Phan *et al.*, "The wmdp benchmark: Measuring and reducing malicious use with unlearning," *arXiv preprint arXiv:2403.03218*, 2024.

[209] Y. Sun, D. Wu, Y. Xue, H. Liu, W. Ma, L. Zhang, M. Shi, and Y. Liu, "Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning," 2024.

[210] Z. Liu, J. Shi, and J. F. Buford, "Cyberbench: A multi-task benchmark for evaluating large language models in cybersecurity." [Online]. Available: http://aics.site/AICS2024/AICS_CyberBench.pdf

[211] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[212] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu, "Ds-1000: A natural and reliable benchmark for data science code generation," in *International Conference on Machine Learning*. PMLR, 2023, pp. 18 319–18 345.

[213] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[214] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[215] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[216] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[217] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.

[218] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *CoRR*, vol. abs/2102.04664, 2021.

[219] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[220] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[221] H. Badri and A. Shaji, "Half-quadratic quantization of large machine learning models," November 2023. [Online]. Available: https://mobiusml.github.io/hqq_blog/

[222] F. Gloeckle, B. Youbi Idrissi, B. Rozière, D. Lopez-Paz, and G. Synnaeve, "Better & Faster Large Language Models via Multi-token Prediction," *arXiv e-prints*, p. arXiv:2404.19737, Apr. 2024.

[223] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1889–1897. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html

[224] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[225] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 53 728–53 741. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf

[226] J. Hong, N. Lee, and J. Thorne, "Orpo: Monolithic preference optimization without reference model," 2024.

[227] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.

[228] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024.

[229] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," 2023.
[230] P. Zhao, H. Zhang, Q. Yu, Z. Wang, Y. Geng, F. Fu, L. Yang, W. Zhang, J. Jiang, and B. Cui, "Retrieval-augmented generation for ai-generated content: A survey," 2024.
[231] Y. Huang and J. Huang, "A survey on retrieval-augmented text generation for large language models," 2024.
[232] M. team, "MLC-LLM," 2023. [Online]. Available: https://github.com/mlc-ai/mlc-llm
[233] ——, "MNN-LLM," 2023. [Online]. Available: https://github.com/wangzhaode/mnn-llm/
[234] L. Derczynski, E. Galinkin, and S. Majumdar, "garak: A Framework for Large Language Model Red Teaming," https://garak.ai, 2024.
[235] N. Shazeer, "Fast transformer decoding: One write-head is all you need," 2019.
[236] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," 2023.

MOHAMED AMINE FERRAG (SM'22) earned his Bachelor's, Master's, Ph.D., and Habilitation degrees in Computer Science from Badji Mokhtar—Annaba University in Annaba, Algeria, completing his studies in 2008, 2010, 2014, and 2019, respectively. He served as an Associate Professor in the Department of Computer Science at Guelma University, Algeria, from 2014 until 2022. Concurrently, from 2019 to 2022, he held the position of Senior Researcher at the NAU-Lincoln Joint Research Center of Intelligent Engineering, based at Nanjing Agricultural University in China. As of 2022, Dr. Ferrag is the Lead Researcher at the Artificial Intelligence & Digital Science Research Center at the Technology Innovation Institute in Abu Dhabi, United Arab Emirates. Dr. Ferrag's research primarily focuses on a spectrum of topics within the cybersecurity domain, including wireless network security, network coding security, applied cryptography, blockchain technology, generative AI, software security, and the application of AI in cybersecurity. His scholarly output includes over 140 papers published in international journals and conference proceedings. Dr. Ferrag has spearheaded numerous projects in research and development, fostering collaborative ties with academic institutions in the UK, Australia, USA, Canada, and China. His contributions to the field include the creation of three cybersecurity datasets, namely, Edge-IIoT dataset, FormAI dataset, and CyberMetric dataset, which have become essential resources for AI researchers worldwide. His academic contributions have been recognized with the 2021 IEEE TEM Best Paper Award, the 2022 Scopus Algeria Award, and many best paper conference awards. He has consistently been named on Stanford University's list of the world's top 2% of scientists four times from 2020 through 2023. Dr. Ferrag also contributes to the academic community as an associate editor for prestigious journals, such as the IEEE Internet of Things Journal and ICT Express (Elsevier). In addition to his research and editorial roles, Dr. Ferrag is a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE).

Fatima Alwahedi is currently an Associate Researcher at Technology Innovation Institute (TII). Her current research interests include Cyber Threat Intelligence, LLMs, and AI for Software Security. She earned her B.Sc. degree in IT Security & Network Technologies from Zayed University, Abu Dhabi, United Arab Emirates, in 2023.

AMMAR BATTAH received his B.Sc. degree in Computer Engineering and M.Sc. in Computer Engineering from Khalifa University, Abu Dhabi, UAE, in 2019 and 2021, respectively. He has worked as a Researcher at Khalifa University, focusing on Blockchain technologies, Artificial Intelligence, Internet of Things (IoT) security, and Information Security. Currently, he is a Research Engineer at Technology Innovation Institute, with interests in Information Security, Software Security, and Artificial Intelligence.

BILEL CHERIF engineer specializing in embedded systems dependability and automotive technology. They hold an Engineering degree and an M.Sc in Electronics and Embedded Systems from the Polytechnical School of Algeria, and an M.Sc in Micro-Embedded Systems Design from Paul Sabatier University in Toulouse, France. Their foundational education also includes CPGE (Elite Preparatory School) and studies at EPST in Algeria. Earlier in their career interned as a Space Software Dependability Engineer at CNRS in France. Worked as a Software Verification and Validation Production Engineer with the Stellantis Alliance's Stella Brain Team in France. They have previously served as an Automotive Safety/Dependability Research Engineer in collaboration with Continental Automotive, and as an Automotive Connectivity Engineer at CNRS in Toulouse. Additionally, they have shared their expertise as a Lecturer in Hardware Acceleration and Codesign for safety critical systems at INSA Toulouse. Currently, he is a Security Engineer at the Technology Innovation Institute (TII), Abu Dhabi, United Arab Emirates. His research interests include formal verification, embedded systems security and safety, dependability, reverse engineering, hardware security, and protocols security.

ABDECHAKOUR MECHRI Engineering and Master's student in Computer Science, specializing in AI and Data Science at École Supérieure d'Informatique, Sidi Bel Abbès, Algeria. Currently interning at the Technology Innovation Institute (TII). Passionate about exploring innovative solutions and new technologies. He is particularly interested in applying AI and machine learning to enhance cybersecurity.

NORBERT TIHANYI holds a B.Sc in Security Engineering, an M.Sc degree in Safety Engineering and an another M.Sc degree in IT Engineering (with honors). He received his Ph.D. in Information Science and Technology from Eötvös Loránd University, Budapest, Hungary, in 2020. He is a Public Body member of the Hungarian Academy of Science. Currently, he is a Lead Researcher at the Technology Innovation Institute (TII), Abu Dhabi, United Arab Emirates. His research interests include cryptanalysis, security of embedded devices and cryptography-related prime number theory.