Deep Learning and Web Applications Vulnerabilities Detection: An Approach Based on Large Language Models

Sidwendluian Romaric Nana, Didier Bassolé, Désiré Guel, Oumarou Sié
Laboratoire de Mathématiques et d'Informatique, Université Joseph KI-ZERBO,
Ouagadougou, Burkina Faso

***Abstract*—Web applications are part of the daily life of Internet users, who find services in all sectors of activity. Web applications have become the target of malicious users. They exploit web application vulnerabilities to gain access to unauthorized resources and sensitive data, with consequences for users and businesses alike. The growing complexity of web techniques makes traditional web vulnerability detection methods less effective. These methods tend to generate false positives, and their implementation requires cybersecurity expertise. As for Machine Learning/Deep Learning-based web vulnerability detection techniques, they require large datasets for model training. Unfortunately, the lack of data and its obsolescence make these models inoperable. The emergence of large language models and their success in natural language processing offers new prospects for web vulnerability detection. Large language models can be fine-tuned with little data to perform specific tasks. In this paper, we propose an approach based on large language models for web application vulnerability detection.**

Keywords*—*Deep learning; web application; vulnerability; detection; large language model

I. Introduction

Nowadays Information and Communication Technologies have facilitated business practices in all sectors of activity (finance, insurance, health, education, energy, etc.). Business processes are automated through the creation of software to improve the productivity of companies and administrations. In February 2024, the number of Internet users worldwide was estimated at over 5.4 billion[1]. Internet has become the "crossroads" where all types of data are exchanged. This ever-increasing accessibility to web resources by internet users has led to the growth of online services via web or mobile applications. All of which increases the attack surface for malicious users exploiting web applications vulnerabilities. From January to December 2022, more than 60 million attacks were observed daily against web applications[2].

Web applications are designed on a client-server architecture. The server side generally includes an application server and a database server. The client (a computer with a web browser) sends an HTTP request to the application server, which queries the database server with an SQL query. The database server sends a response to the application server, which returns an HTTP response to the client. Fig. 1 gives an overview of web application architecture [1].

Different parts of this architecture may be subject to vulnerabilities: web application programming, interaction between client and server, server configuration, etc. In the literature, several approaches have been developed to detect vulnerabilities in applications and prevent web attacks:

- integrating a secure code approach into application development;
- manual code review;
- vulnerability testing (white box, black box and hybrid method);
- use of intrusion detection systems.

These approaches generally use a list of pre-written rules and vulnerability databases. They require cybersecurity expertise, are time-consuming and have a high False Positive Rate (FPR). In a recent study on the detection of malicious URLs [2], we showed the importance of using FPR as an evaluation metric. Our approach enabled us to build models with an FPR of 1.13%, compared with similar works that have a FPR of between 8.15% and 12.03%.

Recent advances in Machine Learning (ML) and Deep Learning (DL) especially offer interesting prospects for detecting vulnerabilities in web applications. Where resources are limited, the use of Large Language Models (LLMs) could be an excellent alternative for obtaining better results with little data. The main objective of this work is to present a review of the different DL approaches used in the literature to detect vulnerabilities in web applications, the difficulties encountered by researchers and how LLMs can contribute to better results.

The rest of this paper is organized as follows: Section II presents the background of study. In Section III, we define some concepts used in the study. Section IV deals with related works. In Section V, we present our approach for detecting web applications vulnerabilities using LLMs. We conclude this work in Section VI.

II. Background Study

A vulnerability is a flaw or weakness in an application's design or implementation. A vulnerability exploited by an attacker has consequences for the application, its owner and the

[1]https://www.wpbeginner.com/fr/research/internet-usage-statistics-and-latest-trends/
[2]https://www.akamai.com/fr/resources/state-of-the-internet/slipping-through-the-security-gaps-the-rise-of-application-and-api-attacks

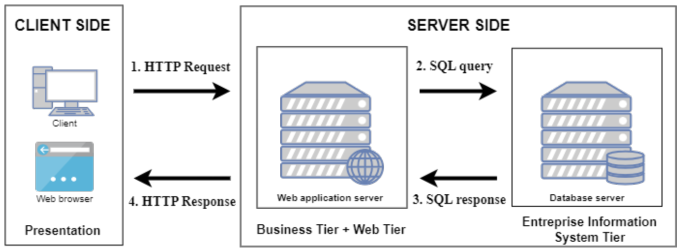

Fig. 1. Overview of web architecture.

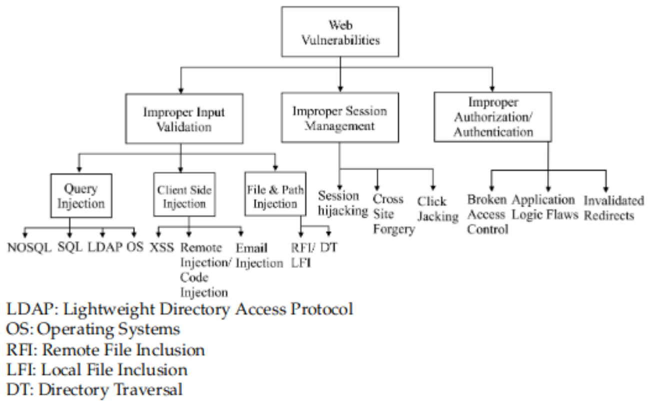

Fig. 2. Type of web vulnerabilities.

application's users[3]. In this section we present web application vulnerabilities, some countermeasures and DL approach in web vulnerabilities detection.

A. Vulnerabilities in Web Applications

OWASP (Open Web Application Security Project) regularly publishes the Top 10 web application security risks. The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications. The OWASP Top 10 for 2021[4] highlights the consolidation of certain vulnerabilities, such asInjection, Security Misconfiguration, and the emergence of others, such as Broken Access Control, Insecure Design, Vulnerable or Outdated Components.

Reference [3] classifies vulnerabilities into three main categories such as:

- Improper input validation: relates to an incorrect validation and sanitization of user input. Some examples of web attacks caused by this category of vulnerability are SQL injection and Cross-Site Scripting (XSS).
- Improper session management: relates to when the web session is insecured. Web requests could not be identified as malicious or not until these are linked with a proper valid session identifier. Some examples of web attacks caused by this category of vulnerability are Cross-Site Request Forgery (CSRF) and session highjacking.
- Improper authorization and authentication: involves a logic flaw in the implementation of access control rules and authentication functions. If web application does not correctly manage authentication and authorization procedures, broken access control is one of the web attacks likely to occur.

Fig. 2 shows a summary of web vulnerability types [3].

B. Existing Countermeasures

Numerous approaches are proposed in the literature for detecting and preventing vulnerabilities in web applications. However, no single approach can detect all vulnerabilities present in web applications. Existing approaches are complementary and can be integrated into all phases of the web application development cycle. Referring to research works [1], [3], existing approaches include secure programming, static, dynamic and hybrid analysis, black box testing, Intrusion Detection Systems (IDS). ML and DL techniques can be integrated into the above-mentioned approaches.

1) Secure programming: A survey conducted in 2019 found that 82 percent of vulnerabilities were located in application code and one in five vulnerabilities was high severity[5]. Secure programming is a set of best practice rules to help programmers develop secure web applications. Secure programming protects coding practices by coding properly, checks input data, encode correctly the user input, its type further by setting the query's parameter, also by bringing stored procedures to work [3]. Secure programming makes programmers aware of the security risks involved in writing code and using libraries and components. In fact, in the OWASP Top 10 for 2021, Vulnerable and Outdated Components is ranked 6th, whereas this vulnerability was ranked 9th in the previous ranking (OWASP Top 10 for 2017). ASVS (Application Security Verification Standard)[6], ESAPI (Enterprise Security API), SAMM (OWASP Software Assurance Maturity Model)[7] are different standard proposed by OWASP project to allow developers to code secure web applications.

2) Static analysis: Static analysis can be carried out at the implementation phase of web application, where it looks for vulnerabilities in source codes and trying to flag them without executing applications [4]. In the literature, several research works have focused on the detection of web application vulnerabilities using static analysis [5], [6], [7], [8]. Overall, static analysis-based tools detect web vulnerabilities despite their trend to generate false positives. Time required to use these tools increases with the size of the code to be scanned [1].

3) Dynamic analysis: It is the opposite approach to static analysis. Its aim is to identify security violations during web application execution. It is a useful technique to prevent web vulnerabilities. This technique incurs no false positives but is less effective for large code coverage. Some existing studies using Dynamic analysis [4], [9], [10], [11]

[3]https://owasp.org/www-community/vulnerabilities/
[4]https://owasp.org/www-project-top-ten/
[5]https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/
[6]https://www.owasp.org/index.php/ASVS
[7]https://www.owasp.org/index.php/SAMM

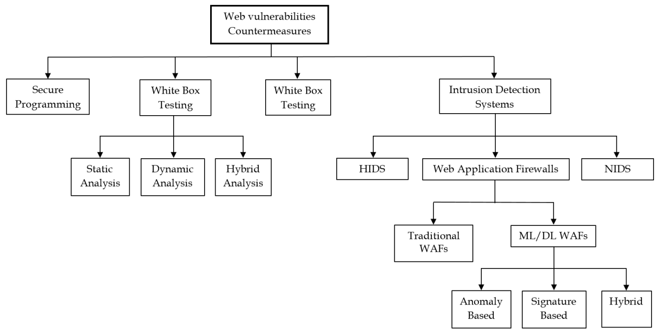

Fig. 3. Web vulnerabilities countermeasures.

4) Black Box testing: This method does not require source code information. It is done with no knowledge of the application's internals. This form of testing is carried out to evaluate the functionality, security, performance, and other aspects of an application. Details can be found in [12], [13]

5) Intrusion detection systems: An IDS is a mechanism designed to detect abnormal or suspicious activity on an analyzed target. It provides information on both successful and unsuccessful intrusion attempts. The target can be a host system (Host Intrusion Detection System (HIDS)) or Network communications (Network Intrusion Detection System (NIDS)) or Web Application (Web Application Firewall (WAF)). There are two main types of WAFs. Traditional WAFs and ML/DL based WAFs. Details can be found in [14], [15]

Fig. 3 shows a summary of web vulnerabilities countermeasures.

C. Deep Learning Approach in Web Vulnerabilities Detection

Alaoui et al. [1] conducted a Systematic Literature Review on the detection of vulnerabilities and attacks on web applications using DL techniques. The literature review covered 63 primary studies or articles on DL-based web application security published between 2010 and September 2021. The authors reviewed articles on various aspects and carried out a qualitative analysis of the results obtained. The qualitative analysis of the selected studies shows that the research area of DL-based web attack detection is yet to be properly explored, and that interest in web vulnerability detection using DL models is very recent. Since 2019, the number of papers published in this research area has increased significantly. Finally, authors noticed that CNN (Convolutional neural network), LSTM (Long short-term memory), and DFFN (Deep Feed Forward Network) are the most DL models used in the reviewed studies.

Dawadi et al. [15] focused on using DL techniques to improve performance of Web Application Firewall. Authors proposed a WAF layered architecture based on LSTM for DDoS (Distributed denial-of-service), SQL injection, and XSS detection in the web-based service system. The model achieved 97.57% accuracy for DDoS detection and 89.34% accuracy for XSS/SQL injection detection.

Alghazzawi et al. [16] conducted a Systematic Literature Review on the detection SQL Injection using ML/DL techniques. The literature review covered 36 articles published between 2012 and 2021. A large proportion of the reviewed studies (83%) used datasets collected from public repositories and HTTP requests. The remaining 17% of the reviewed studies used synthetic datasets created by the authors using deep learning models that can be trained to learn the semantic features of SQL attacks in order to generate new test cases from user inputs.

Maurel et al. [17] explored static approaches to detect XSS vulnerabilities using neural networks. Authors compared two different code representations: *word2vec* based on NLP (Natural Language Processing) and *code2vec* based on PLP (Programming Language Processing); and generate models using different neural network architectures for static analysis detection in PHP and Node.js. Model performed better with PLP approach (Accuracy 95.38%). As programs to be analyzed were run on the server side, authors were faced with the problem of data availability, so they opted for generated datasets (source code). The code generator of the NIST SAMATE project[8]. Authors improved this generator by correcting shortcomings and increasing the number of datasets generated by taking into account the rules announced by OWASP.

Alaoui et al. [18] proposed an approach to detect XSS attack. This approach based on LSTM Encoder-Decoder and Word Embeddings. Authors experiment different free context word embeddings (*Word2vec, Glove, FastText*) to transform HTTP requests to numerical vectors that can be processed by the classification models. Authors implement LSTM Encoder-Decoder and CNN Encoder-Decoder models. Overall LSTM Encoder-Decoder achieves the best classification results regardless of the word embedding technique used: 99.08% accuracy, 99,09% precision, and 99,08% Recall.

As stated above, research interest in web vulnerability detection using DL techniques has been growing in recent years. There are some interesting results in the literature. However, they face a number of limits. Thanks to the various systematic literature reviews, we can see that the majority of articles are oriented towards binary classification. However, it is important to evaluate how well DL models specifically detect different types of web vulnerabilities. Added to this is the availability of data. Indeed, the performance of DL models can be improved by the availability of sufficient quality data. Training a deep learning model requires large volumes of data. Unfortunately, data relating to the security of enterprise applications is sensitive and not always accessible to the general public. Most of the datasets used in the literature are available publicly. They are outdated and do not take into account new vulnerabilities or recent attack techniques [1]. These datasets no longer reflect the complexity of modern web applications. To address this shortcoming, several researchers have experimented with the data generation approach to train or test their models. But data is still synthetic, not real. It is therefore important for companies to contribute to research by making their application code and WAF logs available to the public, even anonymously. Another challenge is code representation. How to represent software programs so that they can be used by a DL model? Li et al. [19] gives some guide principle with VulDeePecker and proposed a framework called SySeVR (Syntax-based, Semantics-based, and Vector

[8]https://www.nist.gov/itl/ssd/software-quality-group/samate

Representations) [20]. This framework focuses on obtaining program representations that can contains syntax and semantic relevant information to vulnerabilities. SySeVR has been evaluated with dataset of open source C/C++ programs from the NVD[9] and from the SARD[10]. NLP and PLP approaches have been explored in literature but it's still an open question.

With reference to the limitations outlined above, we propose an approach based on LLMs. This approach will consist in fine-tuning LLMs for web vulnerability detection. This can be performed on small dataset. Indeed, the recent emergence of LLMs offers interesting prospects for detecting vulnerabilities in web applications. LLMs have been pre-trained on large datasets and succeed in a variety of NLP tasks for which they have not been specially trained [21]. In addition, LLMs represent an excellent advance in natural language processing, with a good representation of textual data. The proposed approach therefore addresses some limitations mentioned by Alaoui et al. in [1].

III. Concepts

In this section, we define some important concepts in the field of cybersecurity and artificial intelligence. This will help us to have a good comprehension of our approach.

A. Vulnerability

A vulnerability is a weakness in a computer system that allows an attacker to undermine the integrity of the system: its normal operation, the confidentiality or integrity of the data it contains. Vulnerabilities are the result of weaknesses in the design, implementation or use of a hardware or software component of the system.

Vulnerabilities can be intentional (backdoors) or accidental, resulting from a lack of knowledge on the part of developers of good security practices or due to the ever-increasing complexity of modern technologies, which increasingly require the adoption of new design and development methods in order to limit the risk of adding vulnerabilities. These vulnerabilities are generally corrected as they are discovered.

Once discovered, vulnerabilities can be the subject of an identification called a CVE[11](Common Vulnerabilities and Exposures). These are published by the Massachusetts Institute of Technology Research Establishment (MITRE[12]) at the request of researchers. This organization can also delegate its identification powers to a company or research center. The latter then becomes a CNA (CVE Numbering Authority)[13].

The CVE lists are brief and do not include technical data or information about risks, effects and patches. These details are recorded in other databases, such as the US National Vulnerability Database (NVD) or the CERT/CC Vulnerability Notes Database[14], etc.

[9]https://nvd.nist.gov/

[10]https://samate.nist.gov/SARD/

[11]https://www.cve.org/

[12]https://www.cve.mitre.org/

[13]https://www.orangecyberdefense.com/fr/insights/blog/gestion-des-vulnerabilites/vulnerabilites-de-quoi-parle-t-on

[14]https://www.kb.cert.org/vuls/

B. Attack and Intrusion

An attack is any attempt to gain unauthorized access to a computer, computer system or computer network with the aim of causing damage. Computer attacks are aimed at disabling, disrupting, destroying or controlling computer systems or at modifying, blocking, deleting, manipulating or stealing data contained in these systems[15].

An intrusion is an internal malicious act, but of external origin, resulting from an attack that has succeeded in exploiting a vulnerability [22].

C. Large Language Models

LLMs are recent advances in deep learning models to work on human languages. LLMs refer to large general-purpose language models that can be pre-trained and then fine-tuned for specific purposes. LLMs are trained to solve common language problems, such as text classification, question answering, document summarization, and text generation[16]. The models can then be adapted to solve specific problems in different fields using a relatively small size of field datasets via fine-tuning. LLMs rely on substantively large datasets to perform those functions. These datasets can include 100 million or more parameters, each of which represents a variable that the language model uses to infer new content [23]. Understanding the importance of LLMs requires background knowledge of Deep Neural Networks (DNNs), Transformers, Attention mechanisms, etc. Indeed [24]:

- LLM is based on transformer architecture
- Attention mechanism allows LLMs to capture long-range dependencies between words, hence the model can understand context
- LLM generates text autoregressively based on previously generated tokens

Large Language Models have evolved rapidly. From 2018 to early 2024, hundreds of models have been created[17]. These models can be differentiated into 4 generations as of now, mainly separating model complexity, but also aspects such as model parameters (embedding encoding, activation functions), quantity and quality of input data, and additional fine-tuning steps. Fig. 4 shows the evolutionary tree of modern LLMs [25].

There are many LLMs developed: GPT-3 and GPT-4 from OpenAI[18],BERT, PaLM 2 and T5 from Google[19], RoBERTa and LLaMA 2 from Meta[20], etc. These are models that can understand language and can generate text.

IV. Related Works

In this section, we review previous work on using Large Language Models to detect software vulnerabilities.

[15]https://www.cyberuniversity.com/post/attaque-informatique-en-quoi-ca-consiste

[16]https://guides.nyu.edu/data/llm

[17]https://admantium.com/blog/llm01_introduction_to_llms/

[18]https://openai.com/

[19]https://ai.google/

[20]https://www.ai.meta.com/

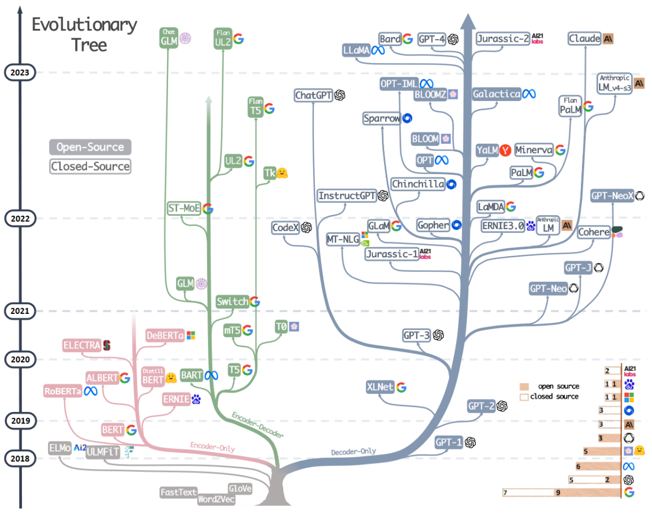

Fig. 4. The evolutionary tree of modern LLMs.

In 2021, Wu presented a literature review [26] on detection of software vulnerability using NLP technology. This paper that focused on static analysis, reviewed techniques to segment source code, extract features and modeling training. BERT [27], GPT [28] and their extended models have been presented as best models in NLP fields. Code being a kind of text, it is logic to think that these models can be used to detect software vulnerability.

Szabó and Bilicki [29] proposed a new approach to web application security using GPT language models for source code inspection. After showing the increasing use of AI in software development, authors formulated a categorization to determine the nature of the sensitive data and the application's vulnerability in a source code and then developed a method based on the GPT API. The targeted vulnerability in this study is CWE-653: Improper Isolation or Compartmentalization. The dataset used consisted of Angular projects collected from GitHub. The model trained on GPT-4 with an accuracy of 88.76% confirms the hypothesis that LLMs have the ability to analyse and interpret software source code. This study opens up prospects for research into the detection of other types of vulnerability using LLMs.

In 2021 Ranade et al. [30] developed CyBERT to represent textual data from the cyber security domain. CyBERT is a domain-specific BERT model. CyBERT is a BERT model fine-tuned with Masked Language Modeling (MLM) and an extended cybersecurity vocabulary. The model has been trained with a large corpus of text from Cyber Threat Intelligence. This model provides cyber security professionals the ability to perform tasks such as Named Entity Recognition (NER), multi-label classification of attacks based on a textual description of the vulnerability. CyBERT outperforms BERT-base model in these tasks. This demonstrates that fine-tuning has enabled the model to learn terms and concepts of cyber security, as well as the relationships between them.

Ameri et al. [31] also proposed CyBERT. In this paper, CyBERT stands for Cybersecurity Claim Classification by Fine-Tuning the BERT Language Model. This classification is based on sequences collected from the documentation of industrial control system devices. The experimental study carried out enabled the hyper-parameters to be optimised and led to a good choice of model architecture. This study led to the conclusion that fine-tuning a BERT model with 2 hidden dense layers and a classification layer achieves a greater accuracy. The resulting CyBERT model with an accuracy of 0.954 outperforms other language models (GPT-2, ULMFiT, ELMo+NN, ELMo+CNN, ELMo+BiLSTM, ELMo+LSTM) as well as other neural networks (CNN, LSTM, BiLSTM).

In 2022, Aghaei et al. [32] developed SecureBERT, a domain-specific language model for cybersecurity. SecureBERT is based on the architecture of RoBERTa (trained with RoBERTa-base) with weight adjustments of pre-trained model. SecureBERT has a good understanding of the semantics of words and phrases. In addition to the pre-trained tokenizer, authors have created a customized tokenizer specific to the cyber security domain, which preserves generic vocabulary while taking into account new tokens emerging from the cyber security domain. SecureBERT has been evaluated on several tasks such as cybersecurity masked word prediction, named entity recognition and sentiment analysis. Evaluation on this last task is proof that SecureBERT has a good understanding of generic language. SecureBERT outperforms others models (RoBERTa-base, RoBERTa-large, SciBERT).

Bokolo et al. [33] conducted a study on web attack detection using DistilBERT, RNN and LSTM. Using a dataset consisting of 33,000 http requests, several experiments were carried out: classification of attacks using URL, the content of the Body or the user data. RNN, with an accuracy of 94%, outperformed others models.

Gallus et al. [34] conducted an experimental penetration testing study on a web application. Thanks to its perfect understanding of web technologies and security principles, chatGPT was used as a penetration test guide. By following the procedures described by chatGPT, the authors were able to retrieve information about the targeted web application, such as the version of WordPress and the theme used. This information was used to discover vulnerabilities in the target application. Using chatGPT's instructions, the testers extracted the list of the application's user accounts as well as the administrator's account. This experiment shows that chatGPT can be used as a guide when testing vulnerabilities in web applications. However, malicious users, even with little technical knowledge, could reproduce chatGPT's instructions and perpetrate attacks on web applications.

Sakaoglu Sinan [35] presented KARTAL: Web Application Vulnerability Hunting Using Large Language Models; Novel method for detecting logical vulnerabilities in web applications with finetuned Large Language Models. The targeted vulnerability is Broken access control, more precisely:

- CWE-639 Authorization Bypass Through User-Controlled Key
- CWE-209: Generation of Error Message Containing Sensitive Information (Exposure of Sensitive Information)

GPT-3.5 was used to generate the dataset, followed by manual labelling for greater accuracy. A total of 1780 samples were annotated, containing at least 200 samples of each class (benign, CWE-639, CWE-209). The pre-trained models MP-Net, DistillRoBerTa and MiniLM were used for fine-tuning. The best model (all-mpnet-base-v2) obtained an accuracy of 87.19%, F1-score of 0.82 and MCC (Matthew's correlation coefficient) of 0.7.

Hanif et al. [36] presented VulBERTa, a deep learning approach to detect security vulnerabilities in C/C++ source code at function-level granularity. This approach pre-trains a RoBERTa model with a custom tokenisation pipline of source code collected from open-source C/C++ projects. The pre-trained model was fine-tuned for vulnerability detection with alternatively a Multi-Layer Perceptron (VulBERTa-MLP) and a Convolutional Neural Network (VulBERT-CNN). The model outperforms existing approaches on binary and multi-class vulnerability classification across different datasets (Vuldeepecker [19], Draper [37], REVEAL [38] and muVuldeepecker [39]) and benchmarks (CodeXGLUE [40] and D2A [41])

Kim et al. [42] developped VulDeBERT, a vulnerability detection model for C/C++ source code by fine-tuning BERT model. VulDeBERT analyses code, extracts well-represented abstract code fragments and generates code gadgets that will be embedded to feed BERT model. VulDeBERT focuses on security vulnerabilities related to system function calls. it outperforms VulDeePecker [19] in detecting two vulnerability types (CWE-119 and CWE-399).

Table I summarizes related works, with an analysis of strengths and weaknesses. Overall, the current state of the literature shows interest in the detection of web application vulnerabilities using LLMs. However, we note that there is more research on adapting LLMs to the cybersecurity field [30], [31], [32]. These work show that, thanks to fine-tuning, LLMs are able to understand the semantics of words in a text dealing with cybersecurity. LLMs have shown excellent performance in NLP downstream tasks: cybersecurity NER, English sentiment analysis, etc. In addition, we note that some studies have used LLMs or deep neural networks to detect software vulnerabilities in C/C++ source code, with difficulties of generalization to other programming languages[36], [42]. However, a few studies have focused on detecting vulnerabilities in web applications [29], [35]. Finally, we also note that BERT and GPT remain the most widely used language models for cybersecurity adaptation and vulnerability detection [43]. It will be more useful to experiment with other types of LLMs and conduct a comparative study based on their architecture in order to address the weaknesses of existing works.

V. Methodology

Detecting software vulnerabilities by using LLMs has produced interesting results, despite the difficulties of generalising the models to other programming languages, particularly web languages. In order to fully exploit the potential of LLMs for detecting vulnerabilities in web applications, we propose a three-stage approach:

- Exploration
- Experimentation
- Evaluation

The exploration consists of a literature review on the use of LLMs in the field of cyber security, more specifically the detection of vulnerabilities in web applications. The literature review (summarised in the previous section) carried out using scientific publications (articles, web articles, dissertations, etc.) enabled us to identify the most widely used and best performing LLMs, types of data used and sources of data collection. Although our literature review revealed a predominance of the BERT and GPT models and their variants, our approach defines criteria for choosing LLMs. These criteria are based on the following points:

- Type of licence : open-source or closed-source
- LLM's architecture: encoder-decoder, encoder-only, decoder-only
- Publication's year
- Performance of models evaluated on the same datasets[21]

The Table II shows a short list of LLMs selected on the basis of the above criteria.

Experimentation consists of defining the neural network architecture and the network learning strategy, training the model and optimising it to obtain better performance. The experimentation stage is designed to be iterative, with the hyper-parameters of models being adjusted and the performance of models being continuously evaluated. Although the performance of NLP tasks is improved by pre-training the basic models, it is important to note that the process requires enormous hardware resources (computing power) and large corpus of text. The fine-tuning process, on the other hand, can be carried out on small datasets and does not require a large corpus of text [44]. In a context where hardware resources and training data are limited, it is advisable to adopt a fine-tuning strategy.

The evaluation stage enables a comparative analysis of the approach with the results of the state of the art; to identify the strengths and limitations of the proposed approach while studying the applicability of the model in a real environment.

Fig. 5 summarises the methodology described above.

VI. Conclusion and Future Works

As the number of Internet users increases, web applications are ubiquitous in all sectors of activity. Unfortunately, this proliferation of web platforms is accompanied by major security risks. Web applications are subject to numerous vulnerabilities reported in several vulnerability databases. Many approaches are proposed in the literature to detect these vulnerabilities. In this paper, after an overview of different approaches, we focus on DL techniques applicable to web vulnerability detection. We presented difficulties and challenges of these approaches to obtain better results and detect several types of vulnerabilities. Finally, we presented the potential of Large Language Models in the cybersecurity domain and proposed an approach for web vulnerabilities detection.

Application of this approach, subdivided into three stages (exploration, experimentation and evaluation), will enable researchers to carry out experimental studies, starting with the

[21] https://admantium.com/blog/llm02_gen1_overview/

TABLE I. Analysis of Strengths and Weaknesses of Related Work

Paper	Topic covered	Models	Strengths	Weaknesses
[26]	Literature review on vulnerability detection using NLP	BERT, GPT	Review of vulnerabilities detection using neural network; Good description of models	Segmentation of code source and feature extraction; No critical analysis of models
[29]	CWE-635 vulnerabilities detection in Angular applications	GPT-3.5, GPT-4 via API call	Classification of sensitive data; Determining protection levels	Depend on prompt's quality; No comparative study with other LLMs; Results inspected manually for evaluation
[30]	Cybersecurity domain adaptation	BERT + MLM Fine-tuning with text in cybersecurity domain	Cybersecurity NER; recognizing a vulnerability from a text description	May require high computational burden with the size of extended vocabulary
[31]	CyBERT: Cybersecurity Claim Classification by Fine-Tuning the BERT Language Model	CyBERT (fine-tuned BERT), GPT-2	Hyperparameters tuning; Comparaison with other DL models	Require high computing resources (GPUs memory); Not easy to reproduce
[32]	SecureBERT: A Domain-Specific Language Model for Cybersecurity	RoBERTa + Customized tokenization + Altering pre-trained weights + Fine-tuning with text in cybersecurity domain	Cybersecurity NER; English sentiment analysis	Require high computing resources (GPUs memory)
[35]	CWE-639 and CWE-209 vulnerabilities detection	all-mpnet-base-v2, all-distilroberta-v1, all-MiniLM-L12-v2	Effectiveness of fine-tuning LLMs with small size, Acceptable inference performance	Data depend on prompt's quality; May require high computational burden with the sequence length; Only open-source LLMs are used in this study
[36]	VulBERTa: Detection of software vulnerabilities in C/C++ source code	Pre-training (RoBERTa + Custom tokenization) + Fine-tuning	Well-described methodology	Require high computing resources (GPUs memory), High false positive rate
[42]	VulDeBERT: CWE-119 and CWE-399 vulnerabilities detection in C/C++ source code	BERT + Fine-tuning	Improving methods to extract code gadgets; Models outperform traditional DL models	Require high computing resources (GPUs memory); Specific on C/C++ programming languages; Focus on two vulnerabilities

TABLE II. List of Selected LLMs

	Open source	Closed source	Encoder-Decoder	Encoder-Only	Decoder-Only	Publication's year
BERT	x			x		2019
RoBERTa	x			x		2019
T5	x		x			2019
FLAN T5	x		x			2022
GPT-3.5		x			x	2022
XLNet	x				x	2020
BLOOM	x				x	2022
LLaMA	x				x	2023
PaLM		x			x	2022
MPNet	x			x		2020

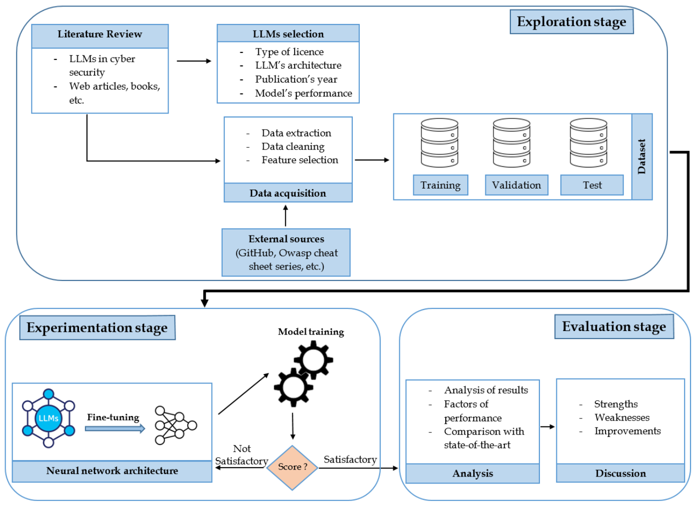

Fig. 5. Overview of methodological framework.

constitution of the dataset, the choice of model architecture and culminating in performance evaluation. In terms of research perspectives, we will:

- Experiment with our LLM-based approach to detecting web attacks from a public dataset.
- Implement a data collection strategy to get malicious URLs from Burkinabe cyberspace in order to build a local dataset.
- Use LLMs to detect malicious URLs from the local dataset.
- Analyze and discuss results in terms of quality of the dataset and the performance of LLMs models in web attacks detection.

References

[1] R. L. Alaoui and E. H. Nfaoui, 'Deep Learning for Vulnerability and Attack Detection on Web Applications: A Systematic Literature Review', Future Internet, vol. 14, no. 4, 2022, doi: 10.3390/fi14040118.

[2] S. R. Nana, D. Bassolé, J. S. Dimitri Ouattara, and O. Sié, 'Characterization of Malicious URLs Using Machine Learning and Feature Engineering', in Innovations and Interdisciplinary Solutions for Underserved Areas, vol. 541, A. Seeam, V. Ramsurrun, S. Juddoo, and A. Phokeer, Eds., in Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 541. , Cham: Springer Nature Switzerland, 2024, pp. 15–32. doi: 10.1007/978-3-031-51849-2_2.

[3] M. Noman, M. Iqbal, and A. Manzoor, 'A Survey on Detection and Prevention of Web Vulnerabilities', International Journal of Advanced Computer Science and Applications (IJACSA), vol. 11, no. 6, 2020, doi: 10.14569/IJACSA.2020.0110665.

[4] F. Faisal Fadlalla and H. T. Elshoush, 'Input Validation Vulnerabilities in Web Applications: Systematic Review, Classification, and Analysis of the Current State-of-the-Art', IEEE Access, vol. 11, pp. 40128–40161, 2023, doi: 10.1109/ACCESS.2023.3266385.

[5] Z. Zhioua, S. Short, and Y. Roudier, 'Static Code Analysis for Software Security Verification: Problems and Approaches', in 2014 IEEE 38th International Computer Software and Applications Conference Workshops, Jul. 2014, pp. 102–109. doi: 10.1109/COMPSACW.2014.22.

[6] P. J. C. Nunes, J. Fonseca, and M. Vieira, 'phpSAFE: A Security Analysis Tool for OOP Web Application Plugins', in 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Jun. 2015, pp. 299–306. doi: 10.1109/DSN.2015.16.

[7] P. Nunes, I. Medeiros, J. Fonseca, N. Neves, M. Correia, and M. Vieira, 'On Combining Diverse Static Analysis Tools for Web Security: An Empirical Study', in 2017 13th European Dependable Computing Conference (EDCC), Geneva: IEEE, Sep. 2017, pp. 121–128. doi: 10.1109/EDCC.2017.16.

[8] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, 'Static Analysis-Based Approaches for Secure Software Development: First International ISCIS Security Workshop 2018, Euro-CYBERSEC 2018, London, UK, February 26-27, 2018, Revised Selected Papers', 2018, pp. 142–157. doi: 10.1007/978-3-319-95189-8_13.

[9] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow, 'jÄk: Using Dynamic Analysis to Crawl and Test Modern Web Applications', in Research in Attacks, Intrusions, and Defenses, H. Bos, F. Monrose, and G. Blanc, Eds., Cham: Springer International Publishing, 2015, pp. 295–316. doi: 10.1007/978-3-319-26362-5_14.

[10] A. Alhuzali, R. Gjomemo, B. Eshete, and V. N. Venkatakrishnan, 'NAVEX: Precise and Scalable Exploit Generation for Dynamic Web Applications', in the Proceedings of the 27th USENIX Security Symposium, August 15–17, 2018, Baltimore, MD, USA. ISBN 978-1-939133-04-5

[11] J. Park, Y. Choo, and J. Lee, 'A Hybrid Vulnerability Analysis Tool Using a Risk Evaluation Technique', Wireless Pers Commun, vol. 105, no. 2, pp. 443–459, Mar. 2019, doi: 10.1007/s11277-018-5959-z.

[12] G. Pellegrino and D. Balzarotti, 'Toward Black-Box Detection of Logic Flaws in Web Applications', in Proceedings 2014 Network and Distributed System Security Symposium, San Diego, CA: Internet Society, 2014. doi: 10.14722/ndss.2014.23021.

[13] M. S. Aliero, I. Ghani, K. N. Qureshi, and M. F. Rohani, 'An algorithm for detecting SQL injection vulnerability using black-box testing', J Ambient Intell Human Comput, vol. 11, no. 1, pp. 249–266, Jan. 2020, doi: 10.1007/s12652-019-01235-z.

[14] Sawadogo, L.M., Bassolé, D., Koala, G., Sié, O. (2021). Intrusions Detection and Classification Using Deep Learning Approach. In: Faye, Y., Gueye, A., Gueye, B., Diongue, D., Nguer, E.H.M., Ba, M. (eds) Research in Computer Science and Its Applications. CNRIA 2021. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 400. Springer, Cham. doi: 10.1007/978-3-030-90556-9_4.

[15] B. R. Dawadi, B. Adhikari, and D. K. Srivastava, 'Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks', Sensors, vol. 23, no. 4, Art. no. 4, Jan. 2023, doi: 10.3390/s23042073.

[16] M. Alghawazi, D. Alghazzawi, and S. Alarifi, 'Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review', Journal of Cybersecurity and Privacy, vol. 2, no. 4, Art. no. 4, Dec. 2022, doi: 10.3390/jcp2040039.

[17] H. Maurel, S. Vidal, and T. Rezk, 'Statically identifying XSS using deep learning', Science of Computer Programming, vol. 219, p. 102810, Jul. 2022, doi: 10.1016/j.scico.2022.102810.

[18] R. Lamrani Alaoui and E. H. Nfaoui, 'Cross Site Scripting Attack Detection Approach Based on LSTM Encoder-Decoder and Word Embeddings', International Journal of Intelligent Systems and Applications in Engineering(IJISAE), vol. 11, pp. 277–282, Feb. 2023.

[19] Z. Li et al., 'VulDeePecker: A Deep Learning-Based System for Vulnerability Detection', in Proceedings 2018 Network and Distributed System Security Symposium, San Diego, CA: Internet Society, 2018. doi: 10.14722/ndss.2018.23158.

[20] 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 Trans. Dependable and Secure Comput., vol. 19, no. 4, pp. 2244–2258, Jul. 2022, doi: 10.1109/TDSC.2021.3051525.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, 'Language Models are Unsupervised Multitask Learners', OpenAI, 2019. [Online]. Available: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

[22] R. Akrout, E. Alata, M. Kaâniche, and V. Nicomette, 'Identification de vulnérabilités Web et génération de scénarios d'attaque', Institut National des Sciences Appliquées de Toulouse (INSA Toulouse), Thesis, 2012. [Online]. Available: https://theses.hal.science/tel-00782565

[23] M. Goyal, S. Varshney and E. Rozsa, 'What is generative AI, what are foundation models, and why do they matter? - IBM Blog'. Accessed: Apr. 22, 2024. [Online]. Available: https://www.ibm.com/blog/what-is-generative-ai-what-are-foundation-models-and-why-do-they-matter/

[24] A. Tam, 'What are Large Language Models', MachineLearningMastery.com. Accessed: Apr. 16, 2024. [Online]. Available: https://machinelearningmastery.com/what-are-large-language-models/

[25] J. Yang et al., 'Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond'. arXiv, Apr. 27, 2023. doi: 10.48550/arXiv.2304.13712. [Online]. Available: https://doi.org/10.1145/3649506

[26] J. Wu, 'Literature review on vulnerability detection using NLP technology'. arXiv, Apr. 22, 2021. [Online]. Available: https://arxiv.org/abs/2104.11230.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), J. Burstein, C. Doran, and T. Solorio, Eds., Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

[28] A. Radford and K. Narasimhan, 'Improving Language Understanding by Generative Pre-Training', 2018.[Online]. Available: https://www.semanticscholar.org/paper/Improving-Language-Understanding-by-Generative-Radford-Narasimhan/

[29] Z. Szabó and V. Bilicki, 'A New Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection', Future Internet, vol. 15, no. 10, Art. no. 10, Oct. 2023, doi: 10.3390/fi15100326.

[30] P. Ranade, A. Piplai, A. Joshi, and T. Finin, 'CyBERT: Contextualized Embeddings for the Cybersecurity Domain', in 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA: IEEE, Dec. 2021, pp. 3334–3342. doi: 10.1109/BigData52589.2021.9671824.

[31] K. Ameri, M. Hempel, H. Sharif, J. Lopez Jr., and K. Perumalla, 'CyBERT: Cybersecurity Claim Classification by Fine-Tuning the BERT Language Model', Journal of Cybersecurity and Privacy, vol. 1, no. 4, Art. no. 4, Dec. 2021, doi: 10.3390/jcp1040031.

[32] E. Aghaei, X. Niu, W. Shadid, and E. Al-Shaer, "Securebert: A domain-specific language model for cybersecurity," in Security and Privacy in Communication Networks, F. Li, K. Liang, Z. Lin, and S. K. Katsikas, Eds. Cham: Springer Nature Switzerland, 2023, pp. 39–56.

[33] B. G. Bokolo, L. Chen, and Q. Liu, 'Detection of Web-Attack using DistilBERT, RNN, and LSTM', in 2023 11th International Symposium on Digital Forensics and Security (ISDFS), May 2023, pp. 1–6. doi: 10.1109/ISDFS58141.2023.10131822.

[34] P. Gallus, M. Štěpánek, T. Ráčil, and P. Františ, 'Generative Neural Networks as a Tool for Web Applications Penetration Testing', in 2023 Communication and Information Technologies (KIT), Oct. 2023, pp. 1–5. doi: 10.1109/KIT59097.2023.10297109.

[35] S. Sakaoglu, “Kartal: Web application vulnerability hunting using large language models novel method for detecting logical vulnerabilities in web applications with finetuned large language models”, Master thesis, 2023. [Online]. Available: https://urn.fi/URN:NBN:fi:aalto-202308275121

[36] H. Hanif and S. Maffeis, ‘VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection’, in 2022 International Joint Conference on Neural Networks (IJCNN), Jul. 2022, pp. 1–8. doi: 10.1109/IJCNN55064.2022.9892280.

[37] R. L. Russell et al., ‘Automated Vulnerability Detection in Source Code Using Deep Representation Learning’. arXiv, Nov. 27, 2018. [Online]. Available: https://arxiv.org/pdf/1807.04320

[38] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, “Deep learning based vulnerability detection: Are we there yet?” IEEE Transactions on Software Engineering, vol. 48, no. 9, pp. 3280–3296, 2022. doi: 10.1109/TSE.2021.3087402

[39] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, ‘μVulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection’, IEEE Transactions on Dependable and Secure Computing, vol. PP, pp. 1–1, Sep. 2019, doi: 10.1109/TDSC.2019.2942930.

[40] S. Lu et al., ‘CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation’. arXiv, Mar. 16, 2021. doi: 10.48550/arXiv.2102.04664. [Online]. Available: https://doi.org/10.48550/arXiv.2102.04664

[41] 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). Los Alamitos, CA, USA, 2021, pp. 111–120. doi: 10.1109/ICSE-SEIP52600.2021.00020

[42] S. Kim, J. Choi, M. E. Ahmed, S. Nepal, and H. Kim, ‘VulDeBERT: A Vulnerability Detection System Using BERT’, in 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Oct. 2022, pp. 69–74. doi: 10.1109/ISSREW55968.2022.00042.

[43] F. N. Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, and C. Meinel, ‘Large Language Models in Cybersecurity: State-of-the-Art’. arXiv, Jan. 30, 2024. [Online]. Available: https://arxiv.org/abs/2402.00891

[44] C. Sun, X. Qiu, Y. Xu, and X. Huang, “How to fine-tune bert for text classification?” in Chinese Computational Linguistics, M. Sun, X. Huang, H. Ji, Z. Liu, and Y. Liu, Eds. Cham: Springer International Publishing, 2019, pp. 194–206. https://doi.org/10.1007/978-3-030-32381-3_16