



POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING (DAUIN)

Master Degree in Computer Engineering

Master Degree Thesis

Evaluation and Optimization of Automated 5G Vulnerabilities Classification

Author: Pierpaolo BENE

Advisor: Nicolás MAUNERO

Co-Advisor(s): Andrea BERNARDINI, Leonardo SAGRATELLA

October, 2025

Abstract

As 5G networks continue to expand into critical sectors such as healthcare, energy, and transportation, the need for robust security measures is becoming crucial. At the same time, the volume of reported vulnerabilities has grown rapidly in the last few years, increasing from around 25,000 in 2022 to projections of up to 50,000 new CVEs (Common Vulnerabilities and Exposures) in 2025. In this evolving landscape, it is essential to rapidly identify the vulnerabilities that affect the 5G infrastructure. However, traditional methods such as keyword filtering and manual review are slow and error prone, making it increasingly difficult to cope with the continuous influx of newly reported vulnerabilities.

To address this problem, the study proposes a methodology to automate the classification of CVEs affecting the 5G infrastructure, making it capable of keeping up with the growing volume of vulnerabilities while preserving reliability.

Experiments are conducted on a manually annotated dataset to evaluate the performances of both state-of-the-art open-source LLMs, including Qwen, Gemma, and Llama, and transformer-based models combined with traditional machine learning classifiers such as logistic regression, SVM, and XGBoost.

Preliminary evidence shows that SVM achieves strong accuracy while being highly efficient in terms of computational costs. However, the findings suggest that LLM-based approaches generalize better to previously unseen CVEs and also provide explicit reasoning that supports the manual validation process. In particular, lightweight local LLMs (around 4 billion parameters) reach accuracy levels comparable to much larger models, and can be further improved with prompt engineering and fine-tuning, surpassing more resource-intensive models.

Notably, the fine-tuned LLMs emerge as the most effective configuration, allowing local execution of the pipeline on limited-resource machines while preserving data privacy, maintaining high accuracy, and offering detailed reasoning that supports manual validation.

The proposed approach is directly applicable to real-world security operations, where timely and reliable vulnerability assessment is crucial. Over time, the process could be refined to reduce the need for human validation, and can be easily extended to other critical sectors, broadening its impact beyond the 5G-specific use case.

Contents

List of Tables	5
List of Figures	6
1 Introduction	8
1.1 Thesis Structure	9
2 Background	11
2.1 Evolution of Mobile Networks	11
2.1.1 5G Stakeholders and Architecture	12
2.1.2 Security of the 5G Network	13
2.2 Common Vulnerability and Exposures (CVE) and The National Vulnerability Database (NVD)	14
2.2.1 Main Challenges in 5G security	15
2.3 Foundations of Machine Learning	16
2.3.1 History of AI	17
2.3.2 Language Models	17
2.3.3 Embedding Vectors	18
2.3.4 Practical Example of Text Embeddings: Word2Vec	18
2.3.5 Seq2Seq as example of Attention Model	21
2.3.6 The Transformer Architecture	22
2.3.7 From Transformer to BERT and GPT	24
2.3.8 Pre-training of Large Language Models	24
2.3.9 Post-training: Supervised Finetuning and Reinforcement Learning	25
2.3.10 Limitations of LLMs	25
3 Related Works	27
3.1 Vulnerability Classification	27
3.1.1 Motivation	28
4 Methodology	29
4.1 Dataset	29
4.2 Evaluation Metrics	32
4.3 Tools	33
4.3.1 Hardware	33
4.3.2 Software	34

4.4	Embedding-Based Approaches	38
4.4.1	Fixed Train-Test Pipeline	38
4.4.2	Cross-Validation Strategy	39
4.4.3	Diagnostic Functions: Avoiding Overfitting and Underfitting	39
4.5	LLM-Based Approaches	41
4.5.1	Prompt Engineering	43
4.5.2	Baseline Prompt	43
4.5.3	Few-Shot	47
4.5.4	Web-Context	50
4.5.5	Prompt Chaining	52
4.5.6	Chain-of-Thought	53
4.5.7	Tree-of-Thought (ToT)	54
4.5.8	Prompt Engineering Techniques Summary	55
4.5.9	Fine Tuning	55
4.6	Prototype Development	59
5	Experimental Results	63
5.1	Experiments on the Public 5G CVE Dataset	63
5.1.1	Baseline	63
5.1.2	Prompt Engineering Techniques	64
5.1.3	Web Context LLM Analysis	67
5.2	Experiments on the Extended Dataset	70
5.2.1	Baseline	70
5.2.2	Embedding-based approaches evaluation	71
5.2.3	Fine-tuning Evaluation	73
6	Conclusions and Future Work	77
6.1	Limitations	79
6.2	Future Work	79
	Bibliography	81

List of Tables

4.1	Comparison Between available Configurations [30, 31, 29].	34
4.2	Summary of the LLM-based web context summarization prompts.	51
4.3	Summary of the Prompt Engineering Strategies explored in this work. . . .	56
4.4	Fine-tuning configuration parameters and explored ranges.	59
5.1	Comparison between default and deterministic configurations ($T=0$) for baseline LLMs on the public dataset.	64
5.2	Comparison of MCC baseline and relative improvements (%) for each prompt- engineering strategy on the public 5G CVE dataset.	66
5.3	Detailed classification results for different Web Prompt configurations on Llama-3.1 (8B) and Qwen-3 (8B).	68
5.4	Baseline performance on the extended dataset, with Few-Shot (FS) relative MCC variation.	70
5.5	Performance comparison of embedding-based classifiers and Qwen3-32B on the extended dataset (80/20 split).	71
5.6	Generalization test of SVM and Qwen3-32B on 133 unseen 5G-related CVEs. .	73
5.7	Performance of fine-tuned models on the extended dataset compared to their base versions.	74
5.8	Comparison between fine-tuned and base Qwen3 models on real case scenario.	75

List of Figures

2.1	General architecture of a 5G cellular network [1].	13
2.2	Number of CVEs prediction for [68]	16
2.3	Visualization of word embeddings in three dimensions [11]	19
2.4	Example of the Sliding Window Mechanism [7].	19
2.5	Continuous Bag-of-Words vs Skip-Gram [10].	20
2.6	Sequence-to-sequence architecture [46].	22
2.7	Transformer architecture: composed of an encoder stack and a decoder stack connected by attention layers [6].	23
2.8	The internals of decoder and encoder components [6].	23
4.1	General overview of the 5G CVE classification methodology.	30
4.2	General overview of the embedding-based classification architecture.	38
4.3	Pipeline of embedding-based classification.	39
4.4	Pipeline of cross-validation embedding-based classification.	40
4.5	Comparison of underfitting, right fit, and overfitting using learning curves [42].	41
4.6	SVM validation curve over parameter gamma [21].	42
4.7	Baseline LLM-based classification pipeline.	42
4.8	Few-shot LLM-based classification pipeline.	47
4.9	Web-context LLM-based classification pipeline.	50
4.10	*	51
4.11	*	51
4.12	Comparison between the two web context summarization strategies: (a) using a Large Language Model as summarizer, and (b) using embedding similarity with cosine distance.	51
4.13	Overview of the Prompt Chaining approach.	53
4.14	Comparison between prompting strategies [65].	54
4.15	Overview of the fine-tuning pipeline.	59
4.16	Prototype architecture: from NVD to classification, logging, and alerting.	60
4.17	Screenshot of the Prototype's Interface.	61
5.1	Metrics Comparison between Default Configuration and $T = 0$	65
5.2	Comparison between MCC, for different Models and Prompt Engineering Techniques	66
5.3	Comparison between MCC (%) With Default Configuration and $T = 0$	67
5.4	*	72
5.5	*	72

5.6	Validation curves fro SVM (RBF) used to determine C and γ	72
5.7	Learning curve (accuracy vs. training set size) for SVM (RBF) with $C = 2$, $\gamma = 1$).	72

Chapter 1

Introduction

The fifth generation of cellular network technology, known as 5G, represents a cornerstone of modern telecommunications, enabling unprecedented levels of connectivity, high bandwidth, and ultra-low latency. This technological leap not only enhances consumer communication but also supports mission-critical applications such as autonomous vehicles, remote healthcare, and large-scale Internet of Things (IoT) deployments. In this sense, 5G should not be regarded as a mere incremental evolution of previous generations, but rather as a disruptive technology that drives the digital transformation of entire sectors.

However, the expansion of its application domains also results in a considerably larger attack surface. 5G networks rely extensively on software-defined networking (SDN), virtualization, and cloud-native components, which, while providing flexibility and scalability, also introduce new avenues for cyberattacks. As 5G becomes deeply embedded in critical services and infrastructures, ensuring its security is no longer optional but an essential requirement to guarantee safe, reliable, and trustworthy operations.

To properly address these security challenges, it is essential to identify and analyze the vulnerabilities that may affect 5G infrastructures. In the cybersecurity domain, vulnerabilities are typically documented and tracked through the Common Vulnerabilities and Exposures (CVE) system, which provides a standardized and publicly accessible catalog of known security flaws. Each CVE entry describes a specific weakness, along with identifiers and references that allow researchers, vendors, and practitioners to assess its potential impact.

The National Institute of Standards and Technology (NIST) plays a central role in this process through the National Vulnerability Database (NVD), which enriches CVE entries with additional metadata, such as severity scores, affected products, and links to mitigation strategies. This structured and authoritative information makes the CVE/NVD ecosystem a fundamental reference for vulnerability management in both academia and industry.

In the context of 5G, leveraging CVE data is particularly relevant: it enables the identification of weaknesses that directly or indirectly threaten the components, protocols, and software-defined architectures on which next-generation networks rely. Consequently, CVE descriptions constitute a valuable source of knowledge for understanding and classifying the risks associated with 5G infrastructures.

Nevertheless, the practical use of CVE data in the 5G domain presents significant challenges. In 2024 alone, more than 40,000 new CVEs were published, representing a volume

that far exceeds the capacity of manual analysis. While some vulnerabilities explicitly affect 5G-related products—such as the open-source core implementation Open5GS—the majority concern technologies, protocols, or software components whose connection to 5G is indirect and often difficult to determine. This heterogeneity underscores the need for advanced methodologies that can automatically distinguish which vulnerabilities are relevant to 5G, thereby facilitating the adoption of large language models (LLMs) to support the classification process.

The objective of this Thesis is to define and implement a methodology that leverages large language models (LLMs) to classify and filter CVE descriptions from the NIST database, with the specific aim of identifying those vulnerabilities that are relevant to 5G technologies. The proposed approach is designed to address both the scale of the problem, given the continuously growing number of published CVEs, and the ambiguity in determining their connection to 5G infrastructures. By developing, implementing, and validating this methodology, the Thesis seeks to provide a tool that not only facilitates more efficient vulnerability assessment in the 5G domain but also contributes to the broader effort of improving the resilience and security of next-generation communication networks.

To achieve this goal, a local pipeline has been developed to evaluate the performance of LLMs with respect to traditional embedding classifiers. Two manually annotated datasets, balanced to contain the same amount of 5G-related and non-5G CVEs, were employed to evaluate both families of models under reproducible conditions. The experiments revealed that, while traditional classifiers provide strong accuracy and computational efficiency, LLM-based approaches demonstrate superior generalization capabilities and also offer interpretable reasoning that supports later manual validation. Moreover, the study evaluates various optimization strategies, including prompt engineering and fine-tuning, which enable lightweight local LLMs to achieve performance comparable to much larger models. A proof-of-concept prototype was also implemented to demonstrate the applicability of the proposed methodology in real-world security operations.

1.1 Thesis Structure

The remainder of the document is organized as follows:

- *Background* 2: introduces the key concepts relevant to this research, including an overview of the 5G architecture and its stakeholders, the Common Vulnerabilities and Exposures (CVE) framework, and the role of the National Vulnerability Database (NVD). It also outlines the fundamental notions of Natural Language Processing (NLP).
- *Related Works* 3: reviews the existing research on automated vulnerability classification, with particular attention to studies that apply NLP and machine learning techniques to CVE analysis. It also identifies the research gap that this Thesis aims to address within the 5G security domain.
- *Methodology* 4: details the experimental methodology used to evaluate both LLM-based and embedding-based classification strategies. This chapter also describes the datasets and the evaluation metrics used to assess the performance of the models in the context of 5G vulnerability classification.

- *Experimental Results* 5: presents and discusses the experimental results obtained throughout this study. It first establishes the baseline performance of the evaluated models and then compares it with the proposed optimization mechanisms, assessing their effectiveness in a realistic operational context.
- *Conclusions and Future Work* 6: summarizes the key findings of the research, discusses the limitations of the proposed methodology, and outlines possible directions for future work.

Chapter 2

Background

This chapter provides the theoretical background necessary to understand the problem of automated classification of 5G vulnerabilities. It is articulated in three main sections:

- *Evolution of Mobile Networks* 2.1: outlines the 5G story, the architecture, and how difficult it is to monitor such a broad infrastructure.
- *Common Vulnerability and Exposures (CVE) and The National Vulnerability Database (NVD)* 2.2: present the CVE/NVD ecosystem, data fields, and trends that motivate systematic, scalable analysis.
- *Foundations of Machine Learning* 2.3: reviews core ML/NLP concepts, such as embeddings, attention, Transformers, and LLMs, forming the basis of the classification methodology adopted in this work.

2.1 Evolution of Mobile Networks

During the last decades, with the spread and rapid progress of mobile devices, cellular networks have evolved through several stages. Each new generation brought about improvements in technology, speed, and the offered services. The first generation (1G) only supported analogue voice communication. The second generation (2G), launched at the end of the 1980s, introduced digital transmission and the short message service (SMS), which extended the possible use of mobile phones. With 3G, higher data rates made access to the internet possible through mobile devices, opening the road for web browsing and multimedia content. The arrival of 4G pushed this evolution further with the introduction of IP-based communication, aligning mobile networks with Internet protocols and providing broadband-like performance. This development created the basis for today's mobile ecosystem, including real-time navigation, social media content, and video streaming [26].

The introduction of 5G can be considered a paradigm shift in mobile networks. In fact, with ultra-low latency, the possibility of connecting a high number of devices, and gigabit-level data rates, 5G expands the scope of mobile networks from consumer services to critical infrastructures such as healthcare, transport, manufacturing, and energy suppliers, increasing concerns over its security, since any vulnerability could have cascading effects on essential services and public safety.

2.1.1 5G Stakeholders and Architecture

The 5G ecosystem is wide and complex, not only because of the technologies underpinning it, but also because of the many actors that participate in its development, deployment, and maintenance. The broader scope of this mobile network consequently increases the number and type of stakeholders involved compared to older mobile generations. Each of these stakeholders has a different role and responsibility, especially when it comes to the security and reliability of the system.

According to a report from the European Union Agency for Cybersecurity (ENISA) on 5G security [23], the main stakeholders of the network are: *service customers*, who rely on 5G for both consumer and industrial applications, *service providers*, who offer connectivity and value-added services, *mobile network operators (MNOs)*, managing the core infrastructures, *virtualization infrastructure providers* enabling the deployment of virtualized network functions, and *data center providers* ensuring storage and computational capacity.

In addition, other important actors are *network infrastructure vendors*, who design and supply the hardware and software components, *national regulators* that are responsible for policy implementation and spectrum allocations, *cybersecurity agencies* tasked with monitoring and responding to threats, and *certification authorities* that guarantee compliance with security standards. At the European level, institutions as ENISA itself, BEREC (Body of European Regulators for Electronic Communications), or the European Commission have a coordinating role.

Over time, the role of these stakeholders can change. Nowadays, international standardization bodies, such as the **3rd Generation Partnership Project (3GPP)**, are responsible for defining open and traceable 5G standards. At the same time, accreditation organizations ensure the quality of test labs and auditors. There are also professional associations creating guidelines for secure deployment, research centers contributing to innovation by filling gaps in standards, and open source communities providing tools and frameworks for implementing 5G functions. Each of them contributes differently to the trust and resilience of the ecosystem [23].

From the technical point of view, 5G architecture can be seen as the result of this multi-stakeholder environment, separated into two main "planes": Control and User Plane. This separation takes the name of **Control and User Plane Separation (CUPS)**. The first focuses mainly on managing the signal and resource allocation by using routing protocols and algorithms. The second is the part of the network dedicated to forwarding the actual data to the end-user. This separation gives more flexibility because the two parts can be scaled or moved independently and managed by software instead of fixed hardware, which is harder to maintain and replace [1].

The structure of the network can be observed in Figure 2.1. Starting from the user's device, called **User Equipment (UE)**, the communication first reaches the **Radio Access Network (RAN)**, which represents the entry point to the mobile infrastructure. The connection of a huge number of nearby UEs is possible through the deployment of **small cells** and **Multiple Input Multiple Output (MIMO)** antennas, which handle multiple signal paths simultaneously, to increase the network coverage. The communication is then transmitted to the transport network and directed to the **Core network**, where the main operations, such as routing and service management, take place.

In the Core Network, most functions are no longer tied to dedicated hardware but are

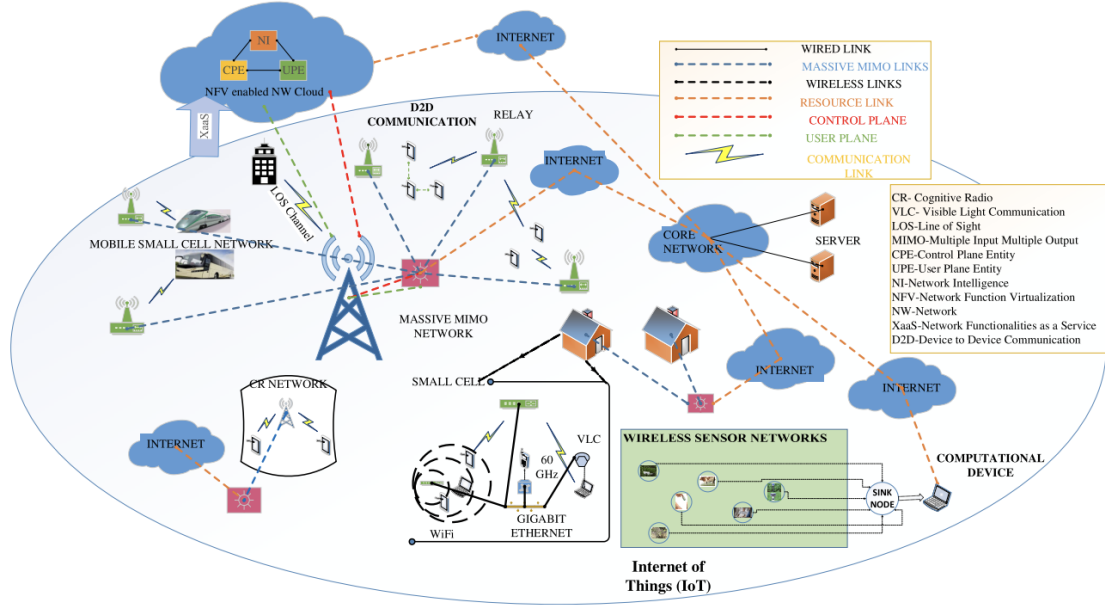


Figure 2.1. General architecture of a 5G cellular network [1].

instead virtualized and executed in software. This paradigm shift is enabled by **Network Function Virtualization (NFV)**, which decouples network functions from the underlying physical infrastructure, and by **Software Defined Networking (SDN)**, which introduces centralized and programmable control over data flows, as depicted in Figure [1]. Together, these technologies allow higher flexibility: operators can dynamically start, migrate, or scale network processes according to real-time demand, optimizing resource allocation and ensuring continuity of service even under fluctuating workloads. As a result, the core network evolves into a software-driven cloud platform capable of supporting applications requiring low latency.

Another important feature introduced with 5G is **Network Slicing**, which allows the creation of multiple logical and independent connections over the same physical channel. Each slice can be configured with different levels of latency or throughput, enabling the coexistence of heterogeneous services within a unified architecture [1]. For instance, one slice may be dedicated to high-bandwidth applications such as video streaming. In contrast, another slice can be deployed to support low-latency communications, such as those needed for autonomous driving, remote surgery, or industrial automation.

2.1.2 Security of the 5G Network

It is evident that with such a large number of diverse components, technologies, protocols, and stakeholders, the potential surface for failures, whether triggered by a malicious actor or by accidental causes, becomes extremely vast. The strong reliance on software, cloud infrastructures, and virtualization technologies significantly broadens the attack surface

of modern mobile networks, making elements such as hypervisors, orchestration frameworks, and supply chains particularly exposed to vulnerabilities. Even misconfigurations or delayed security updates can lead to cascading effects and potentially compromise entire network functions running within the core network [23].

Although the strong dependency on software already opens the door to numerous attack vectors, *Network slicing* brings additional risks. If the mechanisms ensuring isolation between slices fail, or if Application Public Interfaces (APIs) are inadequately protected, attackers may exploit these weaknesses to move laterally across different slices, compromising multiple services at once. At the radio access level, the massive deployment of *small cells* and the use of *millimeter-wave frequencies* increase the susceptibility of equipment to threats such as jamming, spoofing, or physical tampering.

Beyond the purely technical dimension, it is also important to secure **processes and supply chains**. In its analysis of standardization requirements, ENISA [22] highlights a major gap: the lack of standardized security requirements covering the full life cycle of 5G-specific cloud-native and edge deployments. This includes aspects such as centralized certificate management, interoperable automation and orchestration, and support for serverless environments. This gap underlines the urgency of adopting a fast and systematic way to analyze and classify vulnerabilities. In this regard, standards such as *Common Vulnerabilities and Exposures (CVE)* play a fundamental role, as they provide a standardized language to report security vulnerabilities, creating the backbone of vulnerability management practices in complex environments, such as the 5G ecosystem.

2.2 Common Vulnerability and Exposures (CVE) and The National Vulnerability Database (NVD)

The CVE program, maintained by MITRE Corporation, has over the years become the de facto standard for cataloging publicly known cybersecurity vulnerabilities. According to MITRE [41], the basic format of a CVE record includes, at least, the following elements:

- **CVE-ID**: a unique identifier composed of the prefix "CVE", the year, and an arbitrary digit. It is important to note that the year refers not to the discovery of the vulnerability, but to the year in which it was publicly disclosed (or, in some cases, privately).
- **Summary**: a concise description of the vulnerability.
- **Common Product Enumeration (CPE)**: the standardized label identifying the affected product(s) and version(s).
- **References**: one or more URLs linking to external resources that provide further details about the vulnerability.

The **National Vulnerability Database (NVD)** [44] is the official U.S. government repository for vulnerability information. It not only publishes CVE entries but also enriches them with additional metadata, such as the **Common Vulnerability Scoring System (CVSS)**, which provides a way to assess the severity and exploitability of vulnerabilities,

and the **Common Weakness Enumeration (CWE)**, which classifies vulnerabilities according to the underlying software/hardware weaknesses.

This enrichment, maintained by NIST, transforms the NVD from a simple catalog of vulnerabilities into a structured knowledge base. It supports organizations in performing risk assessments, developing effective mitigation strategies, and raising the overall level of security in their products and services.

As an illustrative example, consider **CVE-2023-43239**:

- **CVE-ID:** *CVE-2023-43239*.
- **Summary:** D-Link DIR-816 A2 v1.10CNB05 was discovered to contain a stack overflow via parameter `flag_5G` in `showMACfilterMAC`.
- **Common Platform Enumeration (CPE):** `cpe:2.3:h:dlink:dir-816_a2`.
- **References:** contains two external URLs.
- **CVSS v3.1**, Base Score **9.8 (Critical)**.

Each field of the CVE records provides intuitive information. The identifier contains the year of its public disclosure (2023). The summary specifies the affected product *D-Link DIR 816 A2*, and the vulnerability type in this case is a stack overflow. The CPE indicates with the segment *h* that the vulnerability affects a hardware device. Additionally, NIST enriched the vulnerability with the CVSS score, which allows one to instantly understand that the record is about a Critical vulnerability.

2.2.1 Main Challenges in 5G security

The number of published vulnerabilities has been steadily increasing over the past years, and projections indicate that this trend will continue in the near future. According to the latest vulnerability forecast [68] by Leverett, a Cambridge researcher, the expected number of newly assigned CVE identifiers in 2025 varies between 41,000 and 50,000. As illustrated in 2.2, this represents a dramatic rise compared to the approximately 20,000 CVEs published in 2020, and forecasts suggest that the number could even triple by 2026.

While the rapid growth in the number of published vulnerabilities already represents a challenge to cybersecurity operations, the difficulty of correctly identifying which of these vulnerabilities affect 5G infrastructures introduces an additional and more critical layer of risk.

From a technical perspective, the automated classification of 5G-related CVEs is hindered by two main factors. First, the **ambiguity** in textual descriptions of CVE records, which is typically short and omits contextual details. As a result, truly relevant vulnerabilities may never mention 5G-related components or protocols. Second, **cross-layer dependencies**: 5G networks rely on a complex software stack that integrates cloud-native components, orchestration layers, and specific interfaces, as illustrated in Figure 2.1. A weakness in a seemingly generic component may become critical to the infrastructure if this component is deployed in the implementation of the core network. Third, the **ecosystem heterogeneity**: vendors and open-source projects adopt different naming conventions for functions, APIs, and interfaces, making purely keyword-based unreliable and error-prone.

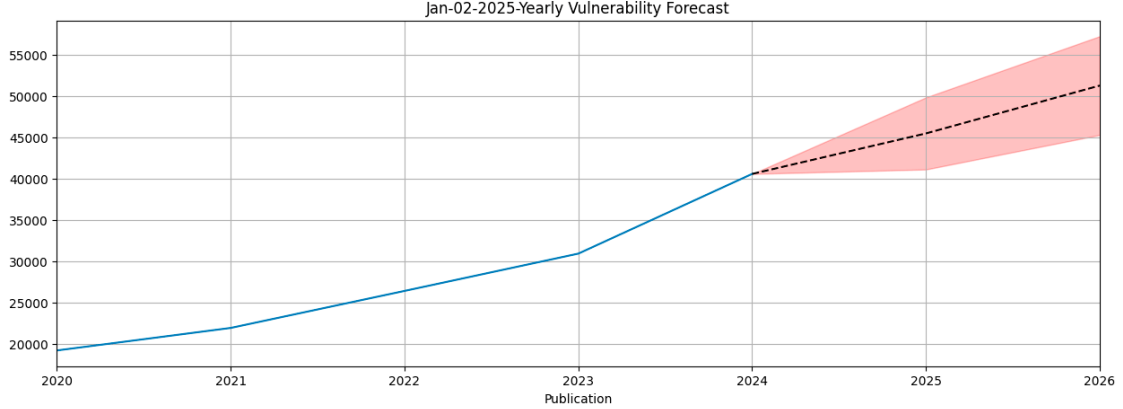


Figure 2.2. Number of CVEs prediction for [68]

An example of this phenomenon was analyzed by Trend Micro in their study on *ASN.1 vulnerabilities in 5G cores* [53]. The authors demonstrated that several vulnerabilities, already catalogued in the NVD, were initially regarded as minor implementation issues in generic ASN.1, a standard notation for describing data structures in various programming languages. These vulnerabilities could be exploited directly from the User Equipment, potentially crashing the User Plane Function. This means that by using a mobile phone and a few lines of Python code, a malicious actor can cause major issues in a core component of the network.

In this context, the efficiency of current vulnerability scanning tools is decreasing rapidly. They rely mainly on keyword matching and fixed taxonomies, which prevent them from capturing the semantic and contextual nuances of 5G complex infrastructure [12]. As a consequence, analysts must still perform extensive manual validation to interpret ambiguous cases and verify correlations between software components and 5G-specific functions. However, this approach is not feasible due to the continuous growth in the volume of CVEs published every year.

Overcoming these limitations requires the adoption of intelligent automated methods capable of learning patterns and reasoning over unstructured data.

2.3 Foundations of Machine Learning

To address the challenges introduced in SubSection 2.2.1, machine learning techniques, capable of processing unstructured text at scale, can play a pivotal role. Hence, this section aims to provide the theoretical background for the models used in this study.

2.3.1 History of AI

Artificial intelligence can be broadly defined as the abilities of a computational system that are typically associated with human intelligence, such as learning, reasoning, problem-solving, perception, and decision making [58].

The history of artificial intelligence is closely interlaced with the evolution of neural networks and, more recently, deep learning. Some of the fundamental concepts that made modern AI possible were developed centuries ago. For instance, backpropagation, the standard algorithm for training neural networks today, relies on the chain rule of calculus, introduced by Leibniz in the seventeenth century [51].

Over the following centuries, new mathematical and computational tools laid further foundations. In 1805, Gauss introduced the method of least squares, which became essential for optimization. Much later, in 1958, Rosenblatt proposed the perceptron, the first neural network model for pattern recognition. In 1965, Ivakhenko and Lapa developed algorithms for training multi-layer networks, marking one of the first practical steps toward neural networks. This was followed by the publication of the backpropagation algorithm during the 70s, which remains the core of modern neural architectures [51].

The subsequent decades brought additional innovations, such as convolutional neural networks (CNNs) and Long Short-Term Memory (LSTM) networks, which enabled models to capture spatial patterns and long-term temporal dependencies, respectively. Many of these breakthroughs were ahead of their time, only becoming widely adopted decades later when computational resources became more powerful.

Since the 2000s, the availability of faster GPUs combined with more efficient training methods has enabled deep networks to advance significantly in fields such as speech recognition, computer vision, and natural language processing (NLP) [51].

Since the problem of CVE classification belongs to the NLP domain, it is crucial to understand the building blocks of these models: how text can be converted into numerical representations, how classifiers can use these representations, and how modern neural architectures, such as large language models (LLMs), extend this semantic understanding to perform advanced tasks.

2.3.2 Language Models

A **language model** estimates the probability of a sequence of words. For example, given the phrase "The most famous city in the US is...", the model would assign a high probability to "New York" or "Los Angeles", and a very low probability to unrelated words, such as "paper" or "telephone". Early approaches to language modeling relied on storing word sequences in the form of *n-gram tables* [37], whereas modern language models are based on neural networks, which are capable of capturing long-range dependencies and richer semantic relationships.

However, language models cannot directly operate on raw text since neural networks process numerical inputs. Therefore, text must first be transformed into a numerical representation that preserves its semantic properties. This operation is performed by **embedding models**.

2.3.3 Embedding Vectors

The concept of **Embedding** refers to the representation of complex data, such as text, images, or audio, into numerical vectors that can be later processed by machine learning models [10]. For the specific task of CVE classification, the focus is on *text embeddings*, particularly on sentence transformers, which can encode the meaning of a whole vulnerability description in a compact vector form.

Embeddings can be intuitively understood as a coordinate system. Just as nearby latitude and longitude values correspond to nearby physical locations, words, sentences, or documents are mapped into a low-dimensional vector space where semantic similarity is preserved. Two pieces of text with similar meaning will appear close together in this space, while unrelated texts will be far apart. The smallest meaningful unit processed in this mapping is called **token**, and can correspond to a phrase, a single word, or even a part of a word [10].

A common real-world example of embeddings in action is provided by search engines such as Google [10]. When a query is issued, results are retrieved from a massive search space through the following steps:

1. Pre-computing the embeddings for Billions of items in the search space;
2. Mapping the user's query into the same embedding space;
3. Efficiently retrieving the items whose embeddings are closest to that of the query;

The dimensionality (i.e., the number of elements of a single vector) of the Embedding can vary depending on the mode, and this choice directly influences the quality of the representation. Higher-dimensional Embedding can capture more complex semantic patterns, but also requires greater computational and storage resources. Moreover, since these spaces usually have hundreds of dimensions, they cannot be directly visualized and represented in a classical Cartesian graphic. However, similarity between two vectors can still be evaluated with mathematical techniques such as cosine similarity, which operate regardless of the number of dimensions [10].

Although the full representation is not directly human-interpretable, compressed projections into two or three dimensions can still provide valuable insights. For instance, in Figure 2.3, male-related words such as "man", "king", and "father" are clearly separated from female-related ones as "woman", "mother", and "queen" along the gender axis.

2.3.4 Practical Example of Text Embeddings: Word2Vec

One of the most well-known models for generating word embeddings is **Word2Vec**, introduced by Mikolov et al. at Google in 2013 [39]. The core idea of Word2Vec is to learn numerical representations of words from large text corpora, so that words occurring in similar contexts are located close to one another in the vector space. Compared to previous approaches, Word2Vec was much more efficient thanks to its training mechanism, and made it possible to train embeddings on billions of tokens.

A Word2Vec model is *trained* on a large collection of raw text using a sliding window mechanism. For each position of the window, the model generates "training pairs" that

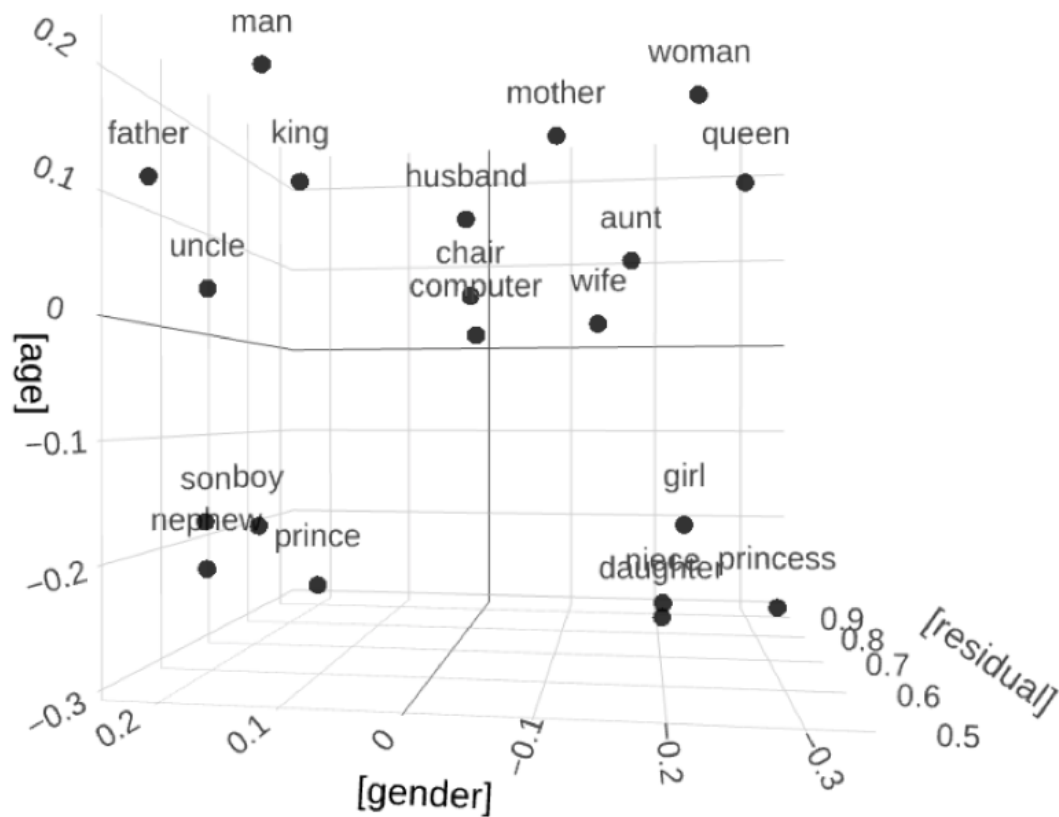


Figure 2.3. Visualization of word embeddings in three dimensions [11]

associate a target word with its surrounding context. This process produces millions of input-output pairs without requiring manual annotation [7].

Figure 2.4 illustrates how the sliding window generates training samples from a sentence. When the window is centered on "shalt not make", one word is taken as the output and the other two are labeled as input.

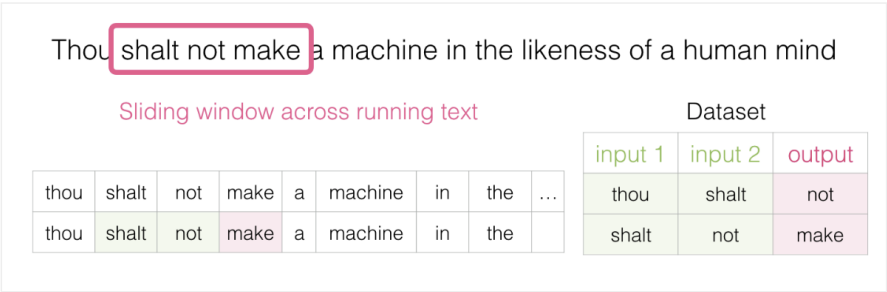


Figure 2.4. Example of the Sliding Window Mechanism [7].

Two alternative architectures are used in Word2Vec (see Figure 2.5):

- **Continuous Bag-of-Words (CBOW)**: the model receives the context words as input and tries to predict the central target word. For example, given the sequence "the _is barking", the model should predict "dog".
- **Skip-gram**: the model does the opposite. It takes a single word as input and attempts to predict the surrounding context words. For example, given the input "dog", it should predict "the" and "barking".

Both architectures share the same principle, but CBOW is faster and more effective on large datasets, while Skip-gram performs better with rare words and captures richer semantic relations [40].

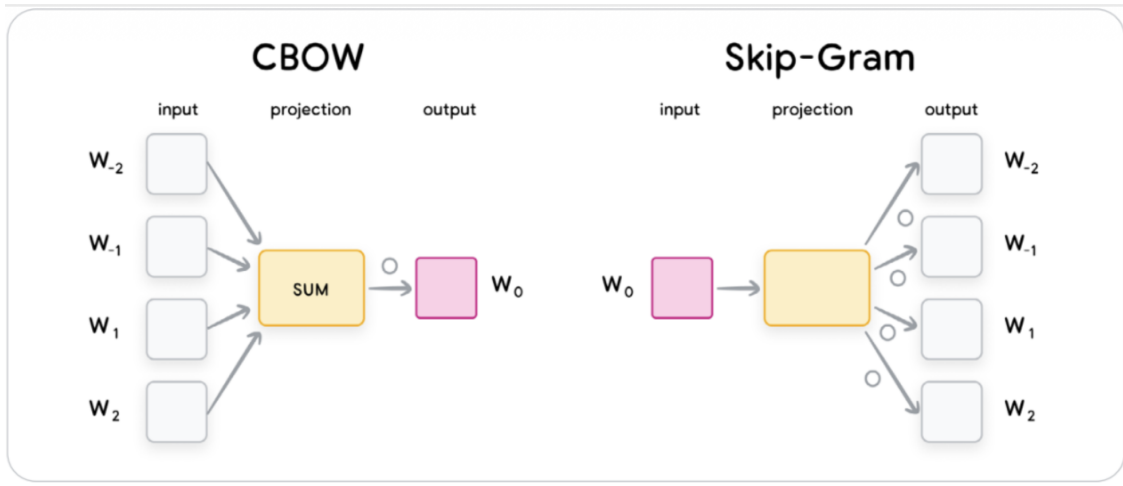


Figure 2.5. Continuous Bag-of-Words vs Skip-Gram [10].

One of the most surprising properties of Word2Vec embeddings is their ability to preserve semantic meaning even under simple algebraic operations. For example:

$$\text{vec}(\text{King}) - \text{vec}(\text{Man}) + \text{vec}(\text{Woman}) \approx \text{vec}(\text{Queen}) \quad (2.1)$$

This property demonstrates that the vectors capture both syntactic and semantic regularities, going far beyond simple keyword matching. As a result, Word2Vec has been widely applied to NLP tasks such as information retrieval, question answering, and machine translation [40].

However, Word2Vec and other similar embedding models have a fundamental limitation: they represent words in a **static** way. For instance, the vector for "bank" will always be the same, regardless of whether it appears in the meaning of "bank of a river", or "bank as a financial institution" [25].

This limitation was addressed with the introduction of the **attention mechanism** first proposed by Bahdanau et al. in 2014, for neural machine translation, and later generalized

in the Transformer Architecture [59]. Attention dynamically computes the relationships between words, taking into account the entire context of the input sequence rather than a small local window. This allows the model to capture long-range dependencies and complex linguistic patterns [56].

2.3.5 Seq2Seq as example of Attention Model

Sequence to sequence (Seq2Seq) models are a class of deep learning architectures that have obtained success in machine translation and text summarization. For instance, Google adopted them in 2016 as the foundation of Google Translate [8]. A sequence-to-sequence model is designed to take a sequence of elements (e.g., words, characters, or even image features) as input and to generate another sequence as output.

Seq2Seq models are built upon **Recurrent Neural Networks (RNNs)**, a family of neural architectures specifically designed to handle sequential data, where the order of the elements is crucial. RNNs employ a mechanism called **recurrent connections**: where the output at each time step (i.e. the output of each neuron) is fed back as input for the next steps. This mechanism allows them to maintain a **hidden state**, which acts as a form of memory and is updated at every step based on both the current input and the previous hidden states [62].

A Seq2Seq architecture typically consists of two components: an **encoder** and a **decoder** [8]. As described in Figure 2.6, the encoder processes the input sequence one element at a time and compresses it into a fixed-length numerical representation **context vector**. This vector is then passed to the decoder, which generates the output sequence token by token. For example, in machine translation, the encoder reads an English sentence, and the decoder produces its Italian translation based on the encoded representation. A similar process applies to text summarization.

Despite their effectiveness, Seq2Seq models face a key limitation: the fixed-length context vector often struggles to represent long input sequences, leading to a loss of information. This issue was solved with the introduction of **attention**, which allows the decoder to focus on the most relevant parts of the input sequence while generating each output token [8, 56].

In practice, attention enhances Seq2Seq models in two ways. First, instead of passing only the last hidden state, the encoder provides the decoder with the entire sequence of hidden states, each associated with a specific input token. Second, at each decoding step, the decoder computes a set of **attention weights**, which assign a relevance score to each hidden state. These scores are later normalized using the **softmax** function, and used to compute a weighted sum of the encoder states, amplifying the important ones while suppressing the less useful ones [8].

This process is repeated for every output token, enabling the model to dynamically align input and output sequences [8]. The attention mechanism rapidly became a central component in neural machine learning models and other NLP tasks. Building on this concept, Vaswani et al. introduced in 2017 the **Transformer**, a new architecture that relies entirely on attention to model dependencies between tokens [63]. The Transformer eliminated the sequential bottleneck of RNN-based models, enabling highly parallelized training and setting the foundation for today’s large-scale generative models.

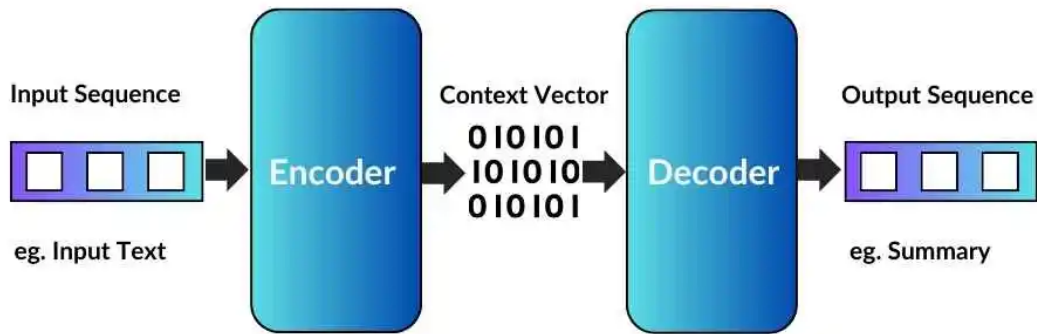


Figure 2.6. Sequence-to-sequence architecture [46].

2.3.6 The Transformer Architecture

The **Transformer** is a neural network architecture entirely built on the attention mechanism, introduced in the paper *Attention is All You Need* [56]. The key novelty of this model is that it removes recurrence and convolution, relying instead on a mechanism called **multi-head attention** to capture dependencies between tokens. This design allows the model to significantly reduce training time compared to earlier architectures such as LSTMs, while improving performance on tasks such as translation [6, 63].

At a high level, the Transformer takes a sequence of words in one language and gives the corresponding outputs depending on the task, just like a classical Seq2Seq model [6]. However, on the inside, the Transformer is composed by a stack of encoder and decoder, as depicted in Figure 2.7 connected through the so called 'attention layers'. In the original architecture, both encoder and decoder are composed of six layers, but the number can be adjusted depending on the task [56].

Each encoder has the same internal structure, but different weights. On the inside, encoders are composed of two elements (see Figure 2.8):

- **Self Attention:** the component where the relationships between words in the same sentence are calculated [56].
- **Feed Forward Neural Network:** where the calculated attention is multiplied by the weights to obtain the output [60].

The decoder follows a similar design but includes an additional **Encoder-Decoder Attention** layer between the self-attention and the feed-forward network (see Figure 2.8).

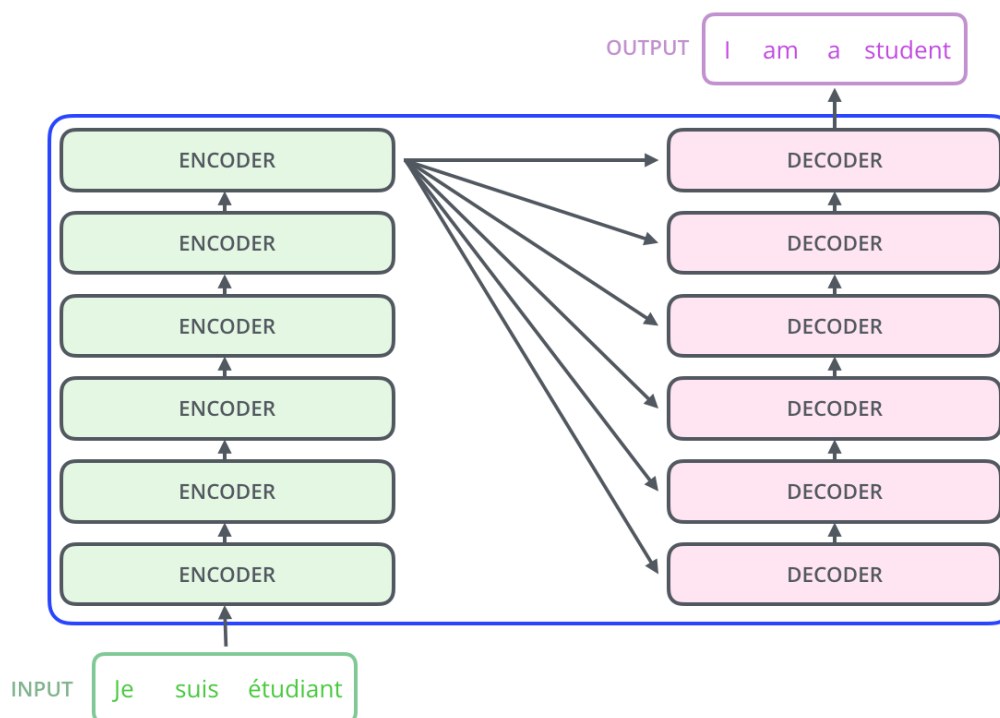


Figure 2.7. Transformer architecture: composed of an encoder stack and a decoder stack connected by attention layers [6].

This extra layer is used by the decoder to focus only on the relevant inputs, ignoring the remaining tokens.

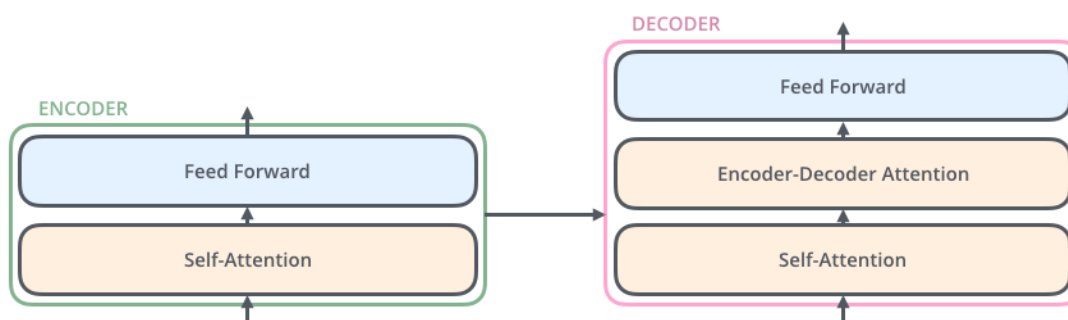


Figure 2.8. The internals of decoder and encoder components [6].

The central component of the Transformer is the **self-attention** mechanism, which allows each token in the sequence to incorporate information from all the other tokens. This is particularly useful to resolve ambiguities that depend on the context. For example, in the sentence *"The animal didn't cross the street because it was too tired"*, the word "it"

refers to "animal" and not to "street". While this is intuitive for a reader, it requires a model to connect the pronoun with the correct noun. This is exactly what self-attention is used for. When the model encodes the word "it", it can link it to the representation of "animal" and integrate this information into the encoding of the word [6]. This mechanism also enables parallelization: while RNNs capture context through a hidden state updated step by step, self-attention gives the model direct access to the entire sequence at once [63].

Several models were built using the Transformer architecture, including **BERT** and **GPT**, which represent the evolution of Transformers towards large-scale pre-trained language models that are now the state of the art in NLP applications [59].

2.3.7 From Transformer to BERT and GPT

Two of the most influential models derived from the Transformer architecture are:

- **Bidirectional Encoder Representations from Transformers (BERT)**: is based only on the encoder part of the Transformer architecture. Its main contribution is to provide a bidirectional representation of text, where each token is interpreted in the context of both its left and right neighbors. This design makes BERT particularly powerful for tasks that require a strong understanding of language, such as question answering. BERT was released in 2018 and quickly integrated into practical applications, like Google Search [4]
- **Generative Pre-trained Transformer (GPT)**: is based on the decoder side of the architecture. GPT models are trained in an 'auto-regressive way', predicting each word in a sequence based on the preceding ones. This approach makes this family of models well-suited for generative tasks. GPT was released in 2018, and later, with GPT-2 and GPT-3 up to GPT-5 in 2025, demonstrated that scaling up training data and parameters leads to huge improvements in fluency and coherence of generated text [5, 61].

Therefore, BERT and GPT show two complementary directions of the Transformer architecture. In particular, GPT marked the beginning of the current wave of large pre-trained language models, which today represent the foundation of most advanced NLP applications [59].

2.3.8 Pre-training of Large Language Models

As explained in the previous section, **Large Language Models (LLMs)** are tools that take a sentence as input and calculate the most probable next word. But, how is it possible to have models capable of solving more complex tasks like question answering or text summarization? LLMs receive two learning phases: pre-training and post-training. The first **pre-training** stage exposes the model to a huge amount of unlabeled textual data, in order to let it acquire general knowledge of the language and world. This stage consists of 3 main steps:

1. Data Collection and filtering: pre-training requires a vast corpus, usually collected by crawling the internet [45]. This raw data is then filtered to remove inappropriate

material (e.g. racism or sexual content), sensitive information, and to select one or more languages.

2. Tokenization: after collection, text must be converted into symbols that the neural network can process.
3. Neural Network Training: the core of pre-training is predicting the next token in a sequence, or filling a missing token (masked language modeling). A sliding window of tokens is fed to the Transformer, which outputs a probability distribution over the next token. Since the correct continuation is known, the difference between prediction and reality can be measured with a **loss function**. Then, gradient descent is used to update the network and reduce this loss. This process is repeated billions of times until the model parameters converge. For example, LLaMA 3, a model from Meta, was trained to reach hundreds of billions of parameters on trillions of tokens [32].

The output of pre-training is a so-called **base model**. It can be seen as a compressed statistical simulator of the dataset, generating continuations that look realistic. Post-training is required to make the model more reliable and aligned.

2.3.9 Post-training: Supervised Finetuning and Reinforcement Learning

To transform the base model into a more sophisticated model, the **Supervised Fine-tuning** is performed. This stage is much less computationally expensive than pre-training.

The key idea is to train the model on a dataset containing examples of human-assistant dialogues. The conversation can teach the model the length, the type, and the style of the answers. An essential part of fine-tuning is **Alignment** [36], where the model is taught not to respond to harmful or offensive requests.

In practice, these conversation data are transformed into token sequences, and special markers are used to indicate the beginning or the end of a text, denoting the role of the speaker and the boundaries of each turn. Then, the generated sequences are used for a shorter training phase, which occurs exactly like the first one, but with an increased network magnitude to enhance effectiveness. The result is a model that is aligned with the creator’s goals: it can answer questions, maintain a polite style, and refuse inappropriate requests.

2.3.10 Limitations of LLMs

Despite their performance, Large Language Models present several well-documented limitations that are important to consider when applying them in practical scenarios.

The most critical one is the phenomenon of **Hallucination**, where the model produces answers that are fluent and plausible, but factually incorrect. For instance, when asked about a non-existent person, the model may invent a detailed biography. Hallucination can be broadly divided into two categories [66]: *intrinsic*, where the output conflicts with the given input or source, and *extrinsic*, where the model introduces fabricated or unverifiable information. These errors arise from multiple factors, including the quality of training data,

knowledge gaps between pre-training and fine-tuning, and the non-deterministic nature of text generation.

In addition, LLMs operate within a fixed context window. When prompts exceed this window, the model must truncate or summarize, which can degrade performance on tasks that require long-range dependencies. Another related problem is that the longer the context window, the longer the computational costs. In fact, LLMs that demand thousands of GPU hours and huge budgets, with a non-negligible environmental impact, are called **Red AI** [34]. These trade-offs make it necessary to carefully evaluate their use in practical applications.

In this study, rather than relying on models with hundreds of billions of parameters, the focus was placed on **lightweight local LLMs** and transformer-based classifiers that can run on resource-constrained machines, such as those described in Section 4.3, while larger models were used only for comparison purposes. This methodological choice represents a trade-off between computational efficiency, data privacy, and model accuracy, while also reducing operational costs. Furthermore, by testing multiple prompt-engineering and fine-tuning strategies, this work aims to assess whether these smaller and resource-efficient models can achieve accuracy levels comparable to state-of-the-art large-scale LLMs.

Chapter 3

Related Works

After having introduced the background on NLP, on the 5G infrastructure, and on the CVE framework, this chapter reviews the existing studies that are most relevant to the present research. The goal is to analyze how the academic community has approached the problem of automatically processing and classifying vulnerabilities.

3.1 Vulnerability Classification

The task of vulnerability classification aims to identify, starting from a group of CVE records, those that can impact the 5G infrastructure. Since only a small number of these vulnerabilities are relevant for a specific infrastructure, it is essential to develop automated methods that can analyze CVE records and map them to a specific domain. Although the focus of this thesis is on the classification of 5G CVEs, it is useful to consider the broader field of CVE automated analysis to observe similar methodological trends and has a vision of how well models are capable of understanding CVEs.

A recent work by Marchiori et al. (2025) [38] provides a detailed evaluation of the task of CVE classification in CVSS vectors. The authors compare different open-source and proprietary LLMs, including LLama, Qwen, Gemma, Deepseek, and Mistral, with embedding-based approaches combined with traditional Classifiers such as Logistic Regression, Random Forest, and XGBoost. The results show that LLMs perform well, with Gemma 3 reaching an accuracy above 0.90, but the accuracy falls below 0.50 on more subjective components such as confidentiality, integrity, and availability. On the contrary, embedding-based methods, in particular all-MiniLM with XGBoost, achieve more consistent results with a mean accuracy of 0.81, showing the ability, once fine-tuned on the dataset, to capture more vague and subjective patterns.

Another notable work has been proposed by Ghosh et al. [28], who evaluated fine-tuned open-source models like Mistral-7B and LLama-7B. Their study focused on vulnerability evaluation in the context of medical devices, including classification tasks, and demonstrated that domain-adapted LLMs can be employed for this purpose.

A broader perspective is offered by Zhang et al. [67], who conducted a systematic review on the use of LLMs in cybersecurity by analyzing more than 300 works. This paper explores 10 different scenarios and, among these, the **vulnerability detection** is directly

related to the problem of CVE classification addressed in this thesis. The study highlights high potential for LLMs in this task, but also shows their limitation: in particular, the lack of datasets for specific domains.

3.1.1 Motivation

Given the literature reviewed above, it becomes evident that while many works address general CVE analysis or vulnerability detection, none focus on a complete evaluation of the vulnerability classification. In addition, to the best of my knowledge, no open-source study specifically considers the classification of CVEs relevant to 5G infrastructures.

This thesis aims to contribute to the development of reliable and efficient tools for local automated 5G vulnerability classification by evaluating the performance of modern open-source LLMs and comparing them to classifiers such as SVM and XGBoost. The evaluation is conducted on a publicly available annotated 5G CVEs dataset [14], containing 136 records, and over an expanded version of the dataset by the same authors, not yet publicly available. Therefore, this work represents, to the best of my knowledge, a first step towards a systematic evaluation of 5G-related CVEs automated classification, opening the way to the adoption of such tools in real-world 5G security operations.

Chapter 4

Methodology

This chapter presents the methodology adopted for evaluating and, eventually, optimizing the automated classification of 5G-related vulnerabilities. Figure 4.1 provides a high-level overview of the classification strategies explored in this work. The pipeline starts from the dataset and forks between the two complementary families of approaches investigated: Embedding-based and LLM-based. To ensure the fidelity of the evaluation, these two approaches are compared against each other, and against the applied optimization techniques, using well-defined evaluation metrics.

This chapter is organized as follows:

- *Dataset 4.1* introduces the dataset construction process and its structure.
- *Evaluation Metrics 4.2* describes the evaluation metrics used to assess performance across different models.
- *Tools 4.3* details the tools, hardware configurations, and software libraries adopted to implement the experimental pipeline.
- *Embedding-Based Approaches 4.4* presents the embedding-based classification approach and the corresponding evaluation strategy.
- *LLM-Based Approaches 4.5* discusses the LLM-based approaches, including prompt engineering techniques and fine-tuning strategies.
- *Prototype Development 4.6* illustrates the prototype application, developed as a proof of concept, that integrates the best-performing model into an operational classification system.

4.1 Dataset

The dataset was constructed at *Fondazione Ugo Bordoni (FUB)*, starting from the National Vulnerability Database(NVD), covering a time span ranging from 2014 to 2024. This dataset serves as the foundation for all subsequent evaluations.

To identify vulnerabilities related to the 5G domain, an initial **keyword-based filtering** step was performed. The filtering relied on a set of technical terms and acronyms

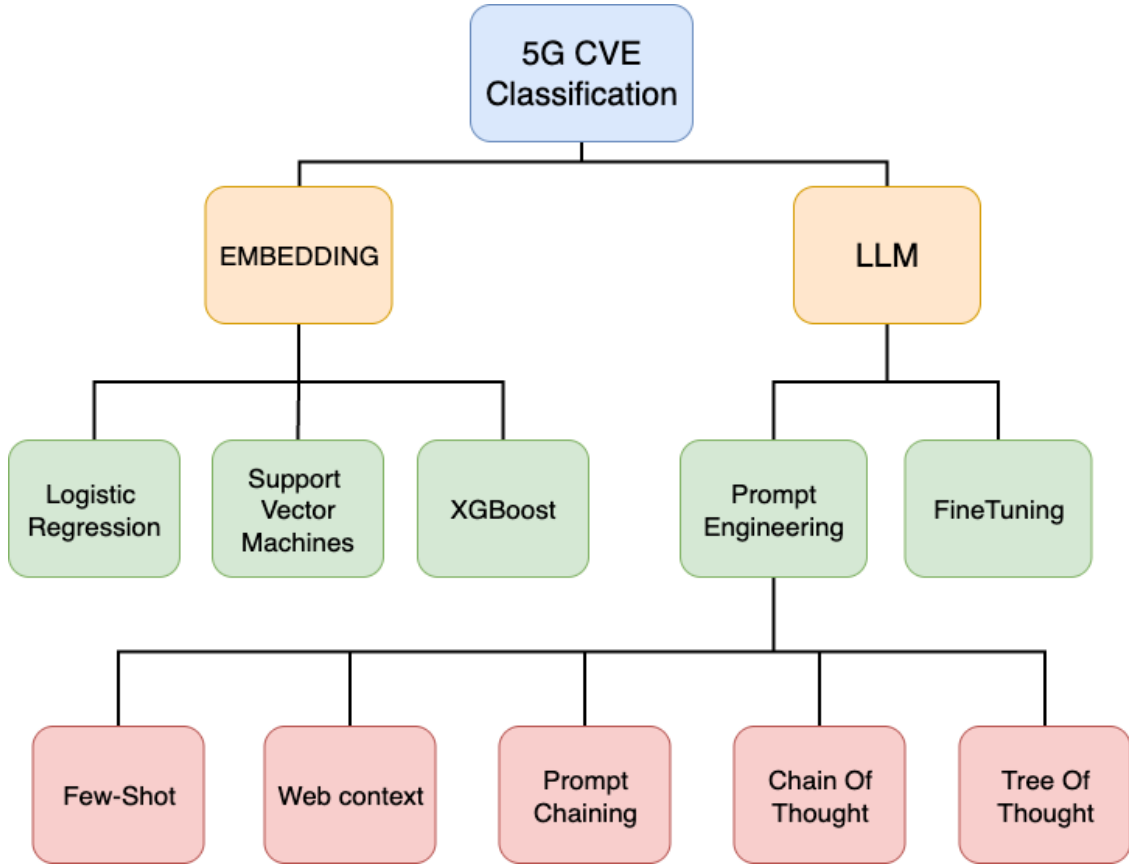


Figure 4.1. General overview of the 5G CVE classification methodology.

commonly associated with 5G networks, including but not limited to those seen in Sub-Section 2.1.1: *5G*, *5G-aka*, *ausf*, *amf*, *bsf*, *chf*, *core network*, *du*, *embb*, *f1ap*, *free5Gc*, *gnb*, *gtp*, *http2*, *imsi*, *mmtc*, *network slice*, *new radio*, *ngap*, *nrf*, *open5Gs*, *pcf*, *pcf*, *ran*, *sba*, *smf*, *srsran*, *udm*, *udr*, *ueransim*, *upf*, *urllc*, *xnap*.

The vulnerabilities obtained through this filtering process were then **manually** reviewed and annotated by FUB's researchers. Specifically, a CVE has been labeled as "5G" if:

- it targets core components of the 5G Standalone (SA) architecture, such as AMF, SMF, UPF, AUSF, or gNodeB (gNB);
- it affects 5G-specific components within user equipment (UE), such as the baseband processor or 5G chipset firmware;
- it exploits a vulnerability in 5G-specific protocols or interfaces defined by 3GPP, such as NAS security, N1/N2/N3 signaling, or Service-Based Architecture (SBA) protocols.

On the contrary, a CVE was considered "no5G" if:

- it refers to legacy or hybrid network components (e.g., LTE/EPC, NSA deployments, MME);
- it affects non-5G components within devices, such as operating systems, applications, GPUs, Wi-Fi chips, or processors unrelated to 5G connectivity;
- the term "5G" was used coincidentally (e.g., references to the Wi-Fi 5 GHz band, product names containing "5G");
- it impacts generic platforms (e.g., Linux, Windows, VMware, Docker) without directly targeting 5G-specific functions.

Finally, the dataset was **balanced** to contain an equal number of records for each class (i.e., "5G", "no5G") and a total of 136 CVE records, organized into four columns [14]:

- **CVE ID**

Type: String

Description: Unique identifier according to the CVE standard.

Format: CVE-YYYY-NNNN (e.g., CVE-2023-43239).

Purpose: Ensures traceability and provides a reference to the official CVE record.

- **CPE (Common Platform Enumeration)**

Type: String

Description: Standardized identifier of the vulnerable platform or product.

Format: cpe:2.3:part:vendor:product:version:....

Purpose: Provides precise identification of the affected system or component.

- **CVE Description**

Type: String

Description: Technical description of the vulnerability, including vulnerability type, affected components, and potential impact.

Language: English, using specialized technical terminology.

- **Label**

Type: String

Description: Final classification label.

Values: *5G* for vulnerabilities related to 5G networks and *no5G* for vulnerabilities not correlated to 5G.

Purpose: Defines the binary classification task used to train and evaluate machine learning models.

In this way, the dataset represents a reliable starting point for evaluating the performance of both embedding-based and LLM-based classification approaches. For this purpose, the records are converted into a JSON format as follows:

```
{
  "CveID": "CVE-2023-41305",
  "cpe": [
    "cpe:2.3:o:huawei:emui",
```

```

    "cpe:2.3:o:huawei:harmonyos"
  ],
  "description": "Vulnerability of 5G messages being sent without...",
  "label": "5G"
}

```

In later stages of this work, an extended dataset consisting of 476 CVE records, published between 2019 and July 2025, and written in the exact same format, was provided by FUB. Although not yet publicly available, this dataset enabled additional experiments and allowed for more precise evaluation.

4.2 Evaluation Metrics

To assess the performance of the proposed classification approaches, a set of standard evaluation metrics is used. These metrics are derived from the **confusion matrix**, which summarizes the prediction outcomes into four categories:

- True Positives (TP): records correctly predicted as *5G* (actual class is *5G*)
- False Positives (FP): records incorrectly predicted as *5G* (actual class is *no5G*)
- True Negatives (TN): records correctly predicted as *no5G* (actual class is *no5G*)
- False Negatives (FN): records incorrectly predicted as *no5G* (actual class is *5G*)

Based on these four quantities, the following evaluation metrics are defined:

- **Accuracy**

Measures the proportion of correctly classified records among the total number of samples:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**

Indicates the fraction of correctly predicted *5G* records among all the records predicted as *5G*:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**

Also called sensitivity, measures the fraction of actual *5G* records that were correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score**

Harmonic mean of precision and recall, providing a balanced metric between the two:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Matthews Correlation Coefficient (MCC)**

MCC takes into account all four categories of the confusion matrix:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

- **MCC improvement**

To measure the contribution of optimization with respect to the baseline, the MCC improvement is calculated as follows:

$$\Delta\text{MCC} = \text{MCC}_{\text{Approach}} - \text{MCC}_{\text{reference}}$$

The improvement can also be shown as a percentage:

$$\Delta\text{MCC}\% = \frac{\Delta\text{MCC}}{\text{MCC}_{\text{reference}}} \times 100$$

It is important to note that the dataset used in this work was explicitly balanced to contain the same number of *5G* and *no5G* records. As a result, accuracy provides a reliable overall measure of correctness. Nevertheless, complementary metrics such as precision, recall, and F1-score are included to capture different aspects of classification performance. In particular, a high precision score highlights the ability to avoid false positives, and recall measures the ability to identify all actual *5G* vulnerabilities, while the F1-score provides a balanced trade-off between these two perspectives.

The Matthews Correlation Coefficient (MCC) is also reported, as it remains one of the most informative metrics for binary classification, compressing in a single value a quantity that takes into account all four elements of the confusion matrix, and has been chosen over other metrics such as ROC-AUC, thanks to its high reliability [18]. In addition, to directly assess the benefits of more advanced strategies, the improvement in MCC with respect to the baseline configuration provides a straightforward interpretation of the effectiveness of each approach compared to the starting point.

4.3 Tools

The implementation of the methodology relies on a combination of hardware resources and software libraries. The experimental setup was designed to be flexible and reproducible, with all adopted libraries being cross-platform and compatible with the main operating systems, including Windows, macOS, and Linux, as well as both Arm and CUDA architectures. The only exception is the *Unslotch* library, used for fine-tuning, which is not yet compatible with Arm-based systems.

4.3.1 Hardware

The experiments were conducted on different hardware configurations, depending on the computational requirements of each model. When evaluating AI models, the most critical resources are typically the **GPU**, which determines the inference and training speed, and the **RAM**, which constrains the maximum model size that can be executed. In this work, four main configurations were employed 4.1:

- **m2_pro**: a MacBook Pro with Apple M2 Pro chip, used for development, preprocessing, and experiments conducted on lightweight models. *Note: In this system, the RAM and VRAM are shared and dynamically allocated from the os.
- **rtx_4090**: a high-performance workstation used for running large models.
- **colab**: a Google cloud computing platform which provides, in its free tier, a T4 GPU employed for finetuning smaller models with the Unsloth library.
- **external_api**: commercial APIs or platforms accessed remotely to benchmark proprietary LLMs and to support dataset preparation for fine-tuning.

Configuration	CPU	RAM	GPU	VRAM	TFLOPs (FP32)
m2_pro	M2 Pro	16 GB	Integrated	16 GB*	5.68
rtx_4090	Ryzen 9	64 GB	RTX 4090	24 GB	82.58
colab	Virtualized	12.7 GB	Tesla T4	15 GB	8.1

Table 4.1. Comparison Between available Configurations [30, 31, 29].

4.3.2 Software

All the experiments were implemented using **Python 3.11**, which served as the primary programming environment. Python was chosen for its wide adoption in the machine learning community, its rich ecosystem of libraries, and the availability of efficient implementations for both classical machine learning models and large language models (LLMs).

Embedding-based Classification Libraries For the embedding-based approaches, the following libraries were adopted:

- **Sentence Transformers** is a library that provides pre-trained transformer models specifically optimized for producing sentence-level embeddings [49]. Several models are available and, among these, the selected one was *all-MiniLM-L12-v2*, which is widely adopted in text classification tasks, as it represents the most powerful runnable text embedding model that could be reliably executed on the available hardware [43].
- **Scikit-learn** is one of the most widely used machine learning libraries in Python [47]. It was used in this work in multiple stages of the pipeline. First, it was used to train and evaluate traditional classifiers, including Logistic Regression, Support Vector Machines (SVM), and XGBoost, which are the core of the embedding-based approaches. **Logistic Regression** is a linear model that converts input into a probability value between 0 and 1, and based on that probability assigns a binary label to the CVE, based on that probability. A **Support Vector Machine (SVM)** is a supervised model that generates, during the training phase, the optimal hyperplane (i.e., a high-dimensional plane) which maximizes the margin between the two classes in the embedding space. **XGBoost** is an optimized implementation of gradient-boosting decision trees [16]. Unlike linear models, XGBoost builds an ensemble of

decision trees, starting from the training set, which allows it to capture non-linear relationships in the embedding space. In the case of binary classification, only a single regression tree is induced. Despite being more computationally demanding, it is widely used for its strong generalization ability. Second, scikit-learn provided the implementation of standard evaluation metrics 4.2 such as accuracy, precision, recall, F1-score, and the Matthews Correlation Coefficient (MCC), ensuring consistency in the validation process. Finally, it was used to represent and analyze a compressed version of the embedding space through dimensionality reduction techniques such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE), which were used to visualize the separability between the two classes in two and three-dimensional plots.

- **Matplotlib** is a standard Python library for data analysis and visualization [35]. In this work, it was used to generate learning curves and validation curves, helping to assess the behavior of the models during training and to detect possible underfitting or overfitting. These visualizations support both the analysis of classifiers' performance and the interpretability of the embedding space.

Hyperparameter Configuration Each classifier was evaluated by adjusting a limited set of hyperparameters, selected to balance performance and computational efficiency while keeping the search space interpretable. For each model, the main tunable parameters were the following:

- **Logistic Regression:** the regularization strength parameter $C \in \{0.01, 0.1, 1, 10\}$ and the penalty type ($L1$ or $L2$). A higher value of C reduces regularization, allowing the model to fit the training data more closely. The $L1$ penalty applies a constraint on the sum of absolute values of the model weights, promoting sparsity by forcing some coefficients to zero, while the $L2$ penalty restricts the sum of squared weights, leading to smoother and more stable models.
- **Support Vector Machine (SVM):** the regularization parameter $C \in \{0.1, 1, 10\}$ and the kernel type (*linear*, *RBF*). The parameter *gamma* for the RBF kernel was explored in $\{10^{-3}, 10^{-2}, 10^{-1}\}$, controlling the influence of individual samples on the decision boundary.
- **XGBoost:** the maximum tree depth (`max_depth` $\in \{3, 5, 7\}$), learning rate (*eta* $\in \{0.01, 0.1, 0.3\}$), and number of boosting rounds (`n_estimators` $\in \{50, 100, 200\}$). The objective function was set to `binary:logistic` to match the binary nature of the classification task.

The selection of hyperparameter values was guided by grid search and by monitoring learning and validation curves, discussed in Section 4.4.2, to prevent overfitting and ensure stable convergence across models.

LLM-based Classification Libraries For the approaches that directly leverage Large Language Models, several libraries and frameworks were adopted:

- **Llama.cpp** is an open source software library [27], written in C++, which works on a wide variety of devices and architectures. It supports both Apple silicon and CUDA (NVIDIA) and makes it possible to experiment locally with open-source LLMs. In this work, Llama.cpp was the core engine to run local models and test different approaches.
- **Unsloth** is a library specifically designed for fine-tuning LLMs with limited memory resources [19]. It leverages the technique of **Low-Rank Adaptation (LoRA)**, which does not update all the parameters of the network during training. Instead, it learns a set of low-rank adapter matrices that approximate the difference between the original weights and the fine-tuned weights, which are then injected into specific layers of the model. This mechanism drastically reduces the number of trainable parameters and memory usage, making fine-tuning feasible even in low-resource environments, such as those offered by Google Colab. However, since Unsloth is not yet compatible with Apple Silicon architectures, Colab was employed for the fine-tuning experiments in this work.
- **Jina AI** provides neural search and retrieval-augmented generation capabilities [2]. It was integrated into the "web_context_llm" approach to enrich the prompts with additional context retrieved from external knowledge bases. This allowed LLM models to perform classification not only based on the information available from the NIST NVD page, but also supported by external information, retrieved from the web.
- **Sentence Transformers + Scikit-learn** are already explained in Paragraph 4.3.2. These libraries were additionally used in the "web_context_embedding" approach, where the cosine similarity method is used to extract the most relevant information from web pages, which can help the models with the classification.

LLM Inference Parameters The inference process in *Llama.cpp* allows control over several decoding hyperparameters that directly influence the diversity and determinism of the generated text. In this work, when available, the parameters suggested by the model developers were adopted as **default settings**, while additional tests were performed with a **temperature** set to 0 to obtain fully deterministic outputs. The main parameters used are:

- **Temperature:** controls the randomness of token sampling. Lower values (close to 0) make the output more deterministic, whereas higher values (e.g., 0.7–1.0) increase variability and creativity in the generated responses.
- **Top-p** (nucleus sampling): limits token selection to the smallest set whose cumulative probability exceeds p . For instance, with $top-p = 0.9$, only tokens contributing to 90% of the probability mass are considered.
- **Top-k:** restricts sampling to the k most probable tokens at each decoding step. Smaller values make the outputs more focused, while larger values increase diversity.
- **Repeat Penalty:** applies a multiplicative penalty to previously generated tokens to reduce repetition and encourage more coherent reasoning chains.

Hugging-Face **Hugging Face** [24] is an open-source platform that provides a unified ecosystem for developing, sharing, and deploying transformer-based models. It includes a large model hub that hosts thousands of pre-trained models for Natural Language Processing (NLP). The Hugging Face *Transformers* library standardizes the interface for model loading, tokenization, and inference across different model architectures such as BERT, GPT, Qwen, Llama, and Gemma. This library was used in the context of this study for downloading and managing various models. In particular, Hugging Face provides pre-converted models into the **GPT-Generated Unified Format (GGUF)**, which already includes chat templates necessary to execute the model locally with *Llama.cpp*.

Models The experiments involved multiple families of Large Language Models (LLMs), differing in architecture, parameter count, and training objectives. The models were selected based on three criteria: open-source availability and compatibility with local inference through *Llama.cpp*. The following models were adopted:

- **Llama 3.1**, Developed by Meta [**meta_llama**], optimized for efficiency and reasoning capabilities, used both in base and fine-tuned configurations.
- **Gemma 3**, is the state-of-the-art open-source Google’s model [50], designed for local execution.
- **Qwen 3**, a multilingual model by Alibaba [33], demonstrating strong performance in classification and analytical reasoning tasks.
- **Mistral** is the only European LLM in the list [3]. It features a lightweight architecture that emphasizes high throughput and efficiency.
- **Sec-GPT**, developed by Cloudfiter [64], is a model specialized in cybersecurity reasoning. All these models were loaded in quantized (Q4_K_M) **GGUF format** to enable local execution on limited-resource hardware.

Prototype Libraries To showcase the final system, a prototype application was developed. It mainly reuses the core libraries already described, such as *Llama.cpp*, *sentence-transformers*, and *scikit-learn*, which constitute the backbone of the classification pipeline. In addition, the following libraries and tools were integrated to provide external data access, a user interface, and alerting functionalities:

- **National Vulnerability Database (NVD) API**, provided by NIST, was used to retrieve up-to-date CVE records directly from the NVD, enabling the prototype to operate on real-world data, in real time.
- **Streamlit** used to design a lightweight graphical web interface to interact with the classification pipeline, enabling users to upload CVEs and view results in real time.
- **Telegram Bot APIs** were integrated to provide an alerting system when new 5G-related vulnerabilities are detected.

4.4 Embedding-Based Approaches

This family of methods is based on the use of an embedding model, combined with a supervised classifier. As illustrated in Figure 4.2, the process starts from the 5G CVE Dataset, described in Section 4.1, and goes across two main stages. First, the 5G CVE dataset is encoded through an **embedding model**. Second, a **classifier** receives the vector representations of the dataset and computes the binary classification (i.e., "5G" and "no5G"), used to compute the evaluation metrics defined in Section 4.2.

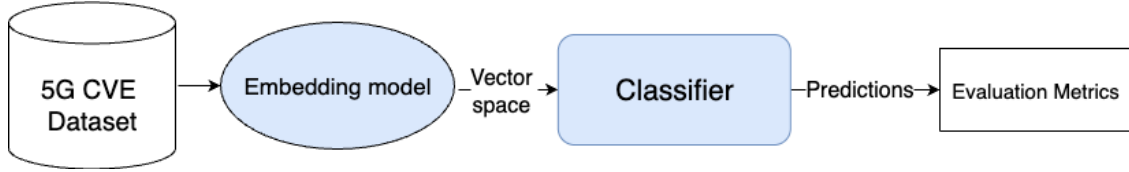


Figure 4.2. General overview of the embedding-based classification architecture.

Although the diagram in Figure 4.2 represents a simplified overview of the pipeline structure, it highlights the clear separation between the two functional blocks, which allows each component to be *independently replaced*. In this way, different combinations of embedding models and classifiers can be seamlessly tested within the same evaluation framework.

In this section, this architecture is described in greater detail, explaining how it has been implemented and which embedding models and classifiers have been employed for the experiments.

4.4.1 Fixed Train-Test Pipeline

As shown in Figure 4.3, the first evaluation pipeline developed relies on a fixed **train-test split**, following the standard 80%-20% ratio. Starting from the dataset, each CVE record is preprocessed and encoded into a **vector space** by the selected embedding model. The resulting vector space, along with the corresponding labels, is then divided into two subsets:

- **Training Set:** 80% of the dataset is used for the training phase, during which the classifier learns a decision boundary that separates the two classes;
- **Testing Set:** the remaining 20% of the dataset is used for the testing phase, where the classifier is evaluated on unseen records that were not included in the training.

The split is computed randomly, starting from a fixed seed to ensure reproducibility, and in a *stratified* way, meaning that both subsets preserve the same proportion between the two classes. After the testing phase, the predictions are compared with the ground-truth labels to compute the **Evaluation Metrics**.

This configuration provides a straightforward estimate of both the embedding model's ability to generate a high-fidelity vector representation of the dataset and the classifier's ability to interpret this representation by distinguishing between the two classes. However,

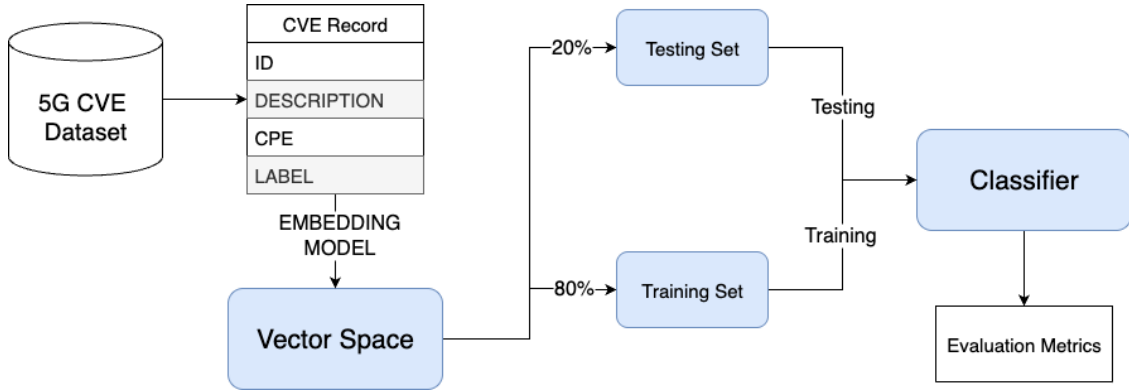


Figure 4.3. Pipeline of embedding-based classification.

the reliability of this fixed split may be affected by the randomness of the partition on which the classifier is tested. One possible solution would be to repeat the experiment multiple times with different random seeds, but this alone does not guarantee that the resulting partitions will be significantly different each time. Moreover, changing the seed does not provide deterministic control over which records are assigned to the training or testing subsets.

4.4.2 Cross-Validation Strategy

To overcome the limitations of the fixed train-test split, a **cross-validation** strategy was adopted. Cross-validation provides a more robust and statistically reliable estimate of model performance by ensuring that all records are used for both training and testing across different iterations. In this study, a **Stratified K-Fold** approach with $K = 5$ was applied, as illustrated in Figure 4.4. In this configuration, the dataset is divided into five equally sized *folds*, preserving the same proportion between the two classes within each fold.

At every iteration, four folds (i.e., 80% of the data) are used for training, and the remaining one (i.e., 20%) is used for testing. This process is repeated five times so that each fold serves as the test set exactly once, ensuring that every CVE record contributes to both training and evaluation, mitigating the effect of potential biases introduced by a single data partition. The final result is obtained by computing the mean and standard deviation of each metric across the five folds, providing a balanced and more reliable performance estimation across alternative embedding-classifier configurations.

4.4.3 Diagnostic Functions: Avoiding Overfitting and Underfitting

One of the key aspects when evaluating a classification model is to ensure that it has learned the general patterns of the data rather than memorizing the training samples. When the classifier fits "too closely" to the training data, it tends to perform exceptionally well on

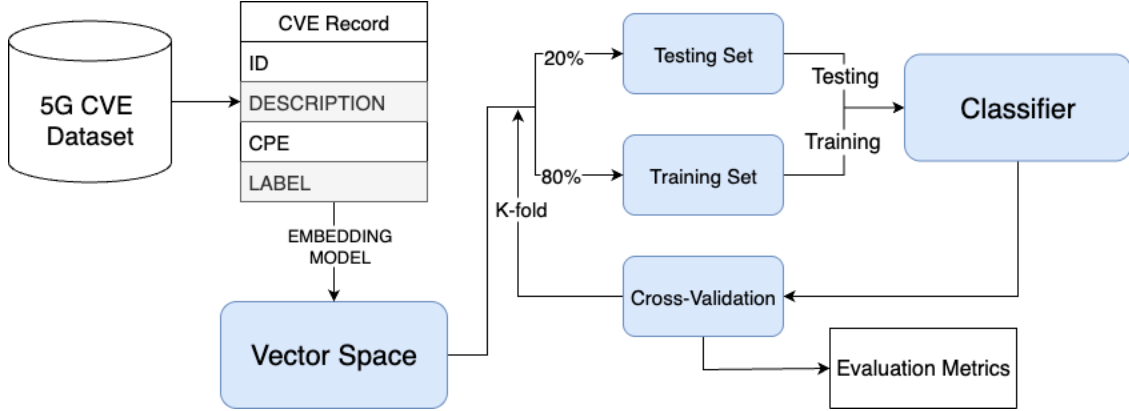


Figure 4.4. Pipeline of cross-validation embedding-based classification.

the training set but poorly on unseen data, a phenomenon known as **overfitting**. On the other hand, when the model is too simple to capture the relationship between the vectors and classes, it will result in higher error rates, a condition referred to as **underfitting**.

In the context of embedding-based classification, both problems may occur depending on the combination of embedding model, classifier, and the chosen hyperparameters (see Paragraph 4.3.2). To prevent these issues, the training process is accompanied by the monitoring of two plots: Learning Curves and Validation Curves.

Learning Curves illustrates how model performance evolves as the amount of training data increases. They are obtained by plotting a performance metric (such as accuracy, F1-score, loss, and so on) on the *y-axis*, against either the number of training samples or the number of training epochs on the *x-axis*. Learning curves provide an immediate visual understanding of how well the model generalizes as it is exposed to more data. Figure 4.5 shows a typical *loss curve*, but these three typical cases can be generalized to any type of learning curves:

- Underfitting (on the left), where both training and validation losses remain inert, indicating that the model is unable to capture patterns from the data, regardless of the amount of training.
- Overfitting (on the right), in which the training loss continues to decrease while the validation loss starts to increase after a certain point. The large gap between the two curves reveals that the model is memorizing the training data.
- In the center, the curves show a clear example of right fit, where there is a small gap between training and validation curves, indicating that the model has learned the relevant patterns and generalizes well.

In practice, the goal is to obtain a "right fit" configuration, where both curves converge smoothly to similar values without excessive divergence. This typically indicates that the model and its regularization parameters are well balanced with respect to the available data.

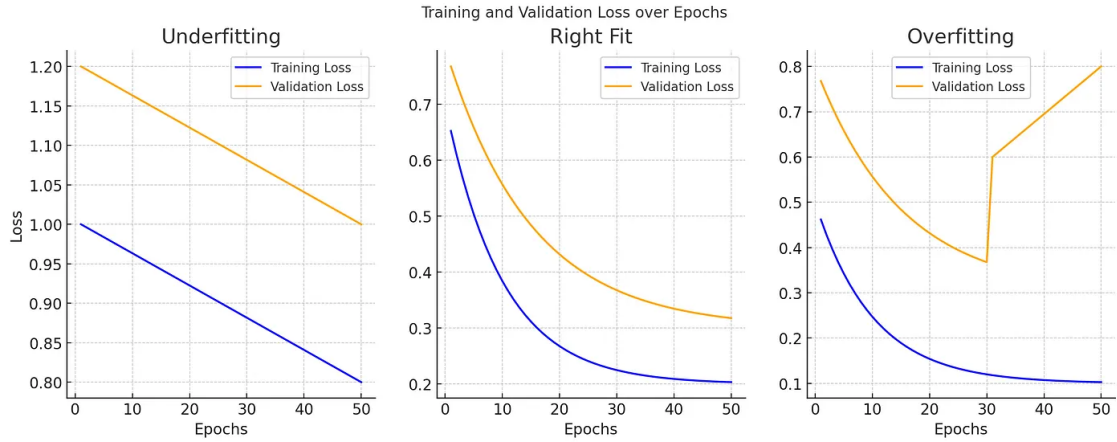


Figure 4.5. Comparison of underfitting, right fit, and overfitting using learning curves [42].

Validation Curves, on the other hand, illustrate how performance varies as a function of a model’s hyperparameter. They are obtained by plotting a chosen evaluation metric (see Section 4.2) (e.g., accuracy, F1-score, MCC, and so on) on the y-axis against a varying hyperparameter on the x-axis, typically in a logarithmic scale. This family of curves is primarily used to evaluate the impact of a specific hyperparameter on model performance, helping to determine the parameter value that provides the best results.

For instance, in Figure 4.6, the training and validation accuracies are plotted as functions of the regularization parameter γ in SVM. On the left side of the plot, both training and validation scores are low, indicating that the parameter is too restrictive. On the right side, the training score remains high while the validation score decreases, a symptom of overfitting. The optimal region is located near the peak of the validation curve, where the validation score is maximized and the gap between the two curves is minimal. This point represents the best trade-off between bias and variance, and it should be selected as the final parameter configuration.

4.5 LLM-Based Approaches

This section describes the second family of approaches explored in this study, which directly leverage **Large Language Models (LLMs)**. Unlike embedding models, LLMs are pre-trained on a vast textual corpus and are, out of the box, capable of capturing a broad range of domain-specific knowledge. This capability enables them to process the CVE description (optionally enriched with CPE information) and produce a structured output, containing both the binary classification and an explanation of the reasoning that led to that decision.

Figure 4.7 represents the base LLM pipeline deployed in this work. It starts from the 5G CVE dataset (see Section 4.1) and uses the CVE description and CPEs to build a **prompt**, which is provided as input to the Large Language Model. The LLM produces

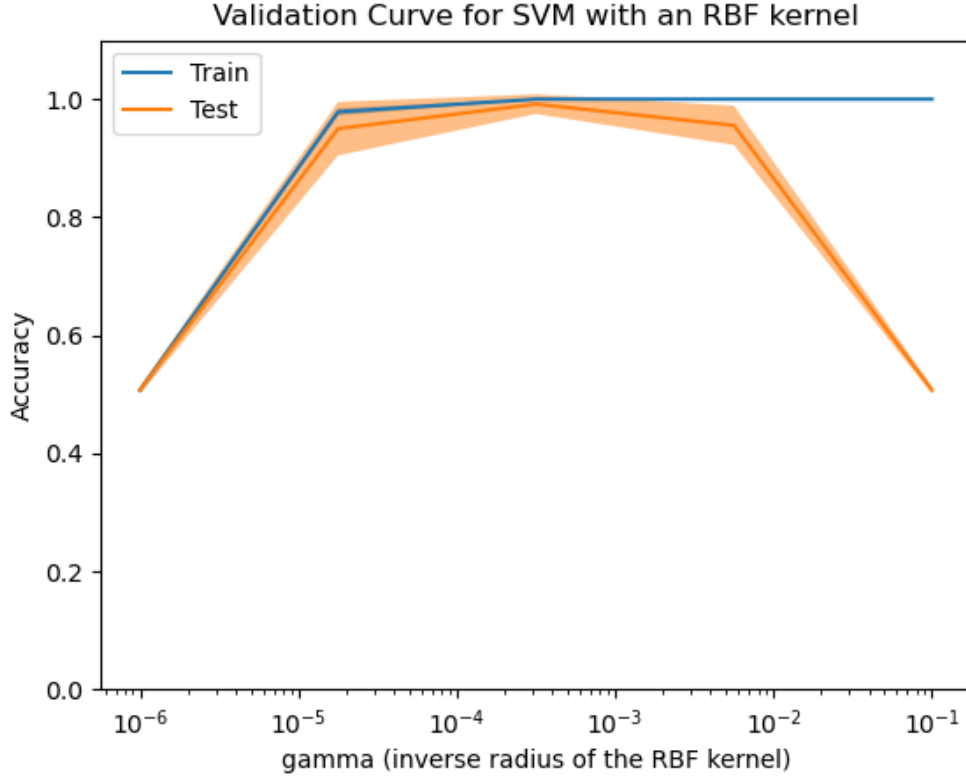


Figure 4.6. SVM validation curve over parameter gamma [21].

both a short *Reasoning* explaining the classification and the *Predicted Label* in the form of a structured *JSON object* for easy parsing. This process is repeated for every CVE record inside the database. In the end, the predicted labels are compared with the *ground truth labels* to compute the evaluation metrics described in Section 4.2.

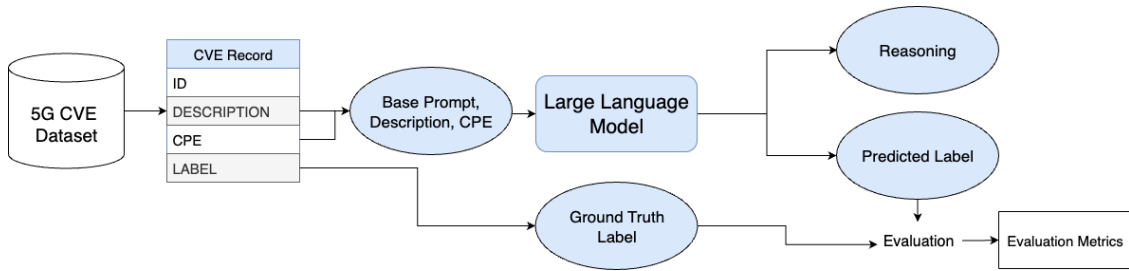


Figure 4.7. Baseline LLM-based classification pipeline.

In Figure 4.7, the baseline configuration represents the fundamental setup on which all subsequent optimization techniques are built and evaluated.

4.5.1 Prompt Engineering

Prompt Engineering refers to the process of designing and optimizing the input instructions given to a Large Language Model in order to guide its behavior toward a desired task or output format [9]. Since LLMs are pretrained on a wide variety of natural language data and not specifically on the task of 5G vulnerability classification, the way the prompt is formulated plays a crucial role in determining the quality, reliability, and consistency of the predictions.

In this context, prompt engineering aims to explicitly define the task, provide output constraints, and optionally include examples or external information that help the model interpret the CVE description correctly. Properly engineered prompts can significantly improve performance without requiring model retraining, making this technique a lightweight alternative to more resource-intensive alternatives, such as fine-tuning.

4.5.2 Baseline Prompt

The baseline prompt, reported in Listing 4.7, defines the standard instructions used to interact with the LLM in a **zero-shot** configuration (i.e., without giving examples). It is organized into multiple sections, each with a specific purpose, designed to ensure that the model receives complete and unambiguous instructions for the classification task.

The prompt, as seen in Listing 4.1, begins with a clear **Role Definition**. It is used to align the model's internal context with the domain of cybersecurity and telecommunications. In fact, grounding the model with a specific role improves the consistency and relevance of its reasoning, especially in specialized technical domains [52].

```
You are a telecommunications security analyst specializing in 5G
infrastructure.
Your task is to determine whether a CVE is relevant to 5G networks.
```

Listing 4.1. Role definition block.

The second section explicitly states the **task** to be performed (see Listing 4.2), ensuring that the model focuses on a **strict and binary** classification rather than open-ended summarization. The task description also clarifies the input format and describes the class separations between "5G" and "no5G" labels.

```
You are provided with a CVE description and, if available, a list of
CPEs identifying
the affected products.

Your classification must be **strict and binary**, using only:
- "5G" if the vulnerability directly or indirectly affects 5G
  networks, components, or protocols.
- "no5G" if it is unrelated to 5G, or belongs to legacy tech like
  4G, 3G, or unrelated fields
  (e.g., 5GHz Wi-Fi, IoT, industrial devices).
```

Listing 4.2. Task constraints.

The Listing 4.3 represents the core part of the prompt and reproduces, in natural language, the same labeling criteria used by human labelers during the dataset annotation

process (see Section 4.1). It enumerates the exact **criteria** under which a CVE should be considered "5G", including attacks on 5G Core, RAN, UE components, or 3GPP-defined interfaces and protocols, as seen in SubSection 2.1.1.

```

### Classification Criteria

A CVE is **5G-related** if:
It targets core components of the 5G Standalone (SA) architecture.
The vulnerability must affect a network function or component
unique to the 5G SA core (5GC) or the 5G New Radio (NR) access
network.
- 5G Core (5GC) Network Functions, such as the Access and
  Mobility Management Function (AMF), Session Management
  Function (SMF), User Plane Function (UPF), or Authentication
  Server Function (AUSF).
- 5G Radio Access Network (RAN) components, specifically the
  gNodeB (gNB).
It affects the 5G-specific components within User Equipment (UE).
For a vulnerability in a device (like a smartphone or IoT sensor)
to be in scope, it must specifically compromise the hardware or
firmware responsible for 5G communication.
- The 5G baseband processor or modem.
- Firmware managing the 5G chipset or its radio frequency
  components.
It exploits a vulnerability in a 5G-specific protocol or interface.
The CVE must be tied to protocols and interfaces defined within
the 3GPP standards for 5G, rather than a generic networking
protocol.
- Signaling interfaces like N1, N2, or N3 that connect the UE,
  RAN, and Core.
- Protocols within the 5G Core's Service-Based Architecture
  (SBA).
- 5G Non-Access Stratum (NAS) security protocols.

```

Listing 4.3. Criteria for classification.

The **Steps** section, shown in Listing 4.4, guides the model through a structured reasoning process: reading the CVE description, analyzing the CPEs, applying the classification criteria, and finally selecting one of the two allowed labels. Such procedural framing encourages the model to reason step-by-step, reducing the likelihood of hallucinations or inconsistent outputs.

```

### Steps
1. Read and interpret the CVE description carefully.
2. Examine the CPEs to identify the product type.
3. Apply the Classification Criteria to determine the final label.
4. Use only "5G" or "no5G" as the final classification label.

### Answer using the following JSON format
{
  "reasoning": "Explain clearly why this CVE is or isn't related to
    5G.",
  "classification": "5G | no5G"
}

```

```
}

```

Listing 4.4. Reasoning steps.

Finally, in Listing 4.5, the prompt enforces a strict output schema, requiring the model to produce a compact JSON object containing two fields. *"reasoning"*, which explains the decision, and *"classification"*, which contains the final label. This constraint has been used to automatically parse the output in order to produce the evaluation metrics, defined in Section 4.2. The explicit instruction "Answer using the following JSON format" was included to avoid additional text such as "I have analyzed the CVE and here is the result of the classification", which is often produced by this kind of model.

```
### Answer using the following JSON format
{
  "reasoning": "Explain clearly why this CVE is or isn't related to
               5G.",
  "classification": "5G | no5G"
}
```

Listing 4.5. Output JSON schema.

The last component of the baseline prompt, shown in Listing 4.6, defines the **input placeholders**. These placeholders are dynamically replaced at runtime with the actual content of each CVE record before being passed to the model. Specifically, the variable *{description}* is substituted with the text of the CVE description from the 5G CVE Dataset, while *{CPEs}* is replaced with the list of affected products in Common Platform Enumeration (CPE) format. In this way, the prompt preserves a uniform structure across all records while adapting its content to each specific CVE instance.

```
**CVE Description**:  
{description}  
**Vulnerable Products CPE**:  
{CPEs}
```

Listing 4.6. Input placeholders.

Finally, the complete baseline prompt, reported in Listing 4.7, is obtained by concatenating all the previously described blocks: (1) role definition, (2) task specification, (3) classification criteria, (4) reasoning steps, (5) output schema, and (6) input placeholders.

```
You are a telecommunications security analyst specializing in 5G
infrastructure.
Your task is to determine whether a CVE is relevant to 5G networks.

You are provided with a CVE description and, if available, a list of
CPEs identifying
the affected products.

Your classification must be strict and binary, using only:
- "5G" if the vulnerability directly or indirectly affects 5G
  networks, components, or protocols.
- "no5G" if it is unrelated to 5G, or belongs to legacy tech like
  4G, 3G, or unrelated fields
```

```

(e.g., 5GHz Wi-Fi, IoT, industrial devices).

### Classification Criteria

A CVE is **5G-related** if:
It targets core components of the 5G Standalone (SA) architecture.
The vulnerability must affect a network function or component
unique to the 5G SA core (5GC) or the 5G New Radio (NR) access
network.
- 5G Core (5GC) Network Functions, such as the Access and
  Mobility Management Function (AMF), Session Management
  Function (SMF), User Plane Function (UPF), or Authentication
  Server Function (AUSF).
- 5G Radio Access Network (RAN) components, specifically the
  gNodeB (gNB).
It affects the 5G-specific components within User Equipment (UE).
For a vulnerability in a device (like a smartphone or IoT sensor)
to be in scope, it must specifically compromise the hardware or
firmware responsible for 5G communication.
- The 5G baseband processor or modem.
- Firmware managing the 5G chipset or its radio frequency
  components.
It exploits a vulnerability in a 5G-specific protocol or interface.
The CVE must be tied to protocols and interfaces defined within
the 3GPP standards for 5G, rather than a generic networking
protocol.
- Signaling interfaces like N1, N2, or N3 that connect the UE,
  RAN, and Core.
- Protocols within the 5G Core's Service-Based Architecture
  (SBA).
- 5G Non-Access Stratum (NAS) security protocols.

### Steps
1. Read and interpret the CVE description carefully.
2. Examine the CPEs to identify the product type.
3. Apply the Classification Criteria to determine the final label.
4. Use only "5G" or "no5G" as the final classification label.

### Answer using the following JSON format
{
  "reasoning": "Explain clearly why this CVE is or isn't related to
    5G.",
  "classification": "5G | no5G"
}

**CVE Description**:
{description}
**Vulnerable Products CPE**:
{CPEs}

```

Listing 4.7. Full Baseline prompt for LLM classification.

4.5.3 Few-Shot

The few-shot strategy aims to provide the model with explicit reasoning patterns and decision boundaries, helping the model to generalize more reliably on new records, since this strategy can improve the performance of LLMs and sometimes also match the performance of fine-tuned models on the same task [15].

As illustrated in Figure 4.8, the 5G CVE record is processed as in the baseline setup, but the prompt is enriched with a few representative examples. In particular, four examples were written: one representing the *5G* class and three belonging to the negative *no5G* class. This choice has been made because the *no5G* class spans many unrelated domains (e.g., general OS issues, enterprise networking gear, web applications, or references to the "5 GHz" frequency band). Multiple negative examples help cover this variability, reducing spurious matches triggered by generic networking terms. Each example includes both the CVE Record input and the assistant output in JSON format.

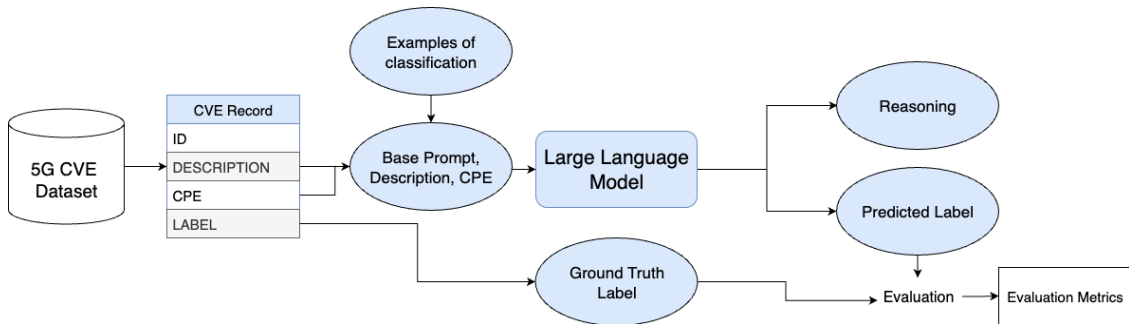


Figure 4.8. Few-shot LLM-based classification pipeline.

These examples are built starting from real vulnerabilities, not contained in the dataset, to cover the following typical cases:

- vulnerabilities in *5G Core* or *RAN* components ;
- edge cases, where possible misclassifications can be caused by unrelated references to "5G" (e.g., Wi-Fi 5GHz or product names);
- CVEs affecting general-purpose software or network devices not specific to 5G infrastructure.

Example 1 – Networking-related CVE

Listing 4.8 shows an example of a vulnerability representative of vulnerabilities that may initially appear relevant due to networking context, but are, in fact, unrelated to 5G technology or 3GPP-defined components. Including such examples helps the model learning to distinguish between generic network software and actual 5G infrastructure.

```

USER INPUT:
[baseline instruction truncated for brevity]
CVE description:
  
```

```

A vulnerability in the NX-API feature of Cisco NX-OS Software could
  allow an authenticated, remote attacker to execute arbitrary
  commands with root privileges.
CPE:
['cpe:2.3:o:cisco:nx-os']

ASSISTANT OUTPUT:
{
  "reasoning": "The vulnerability affects Cisco NX-OS Software,
    which is used in data center switches and not in 5G networks or
    5G-specific protocols.",
  "classification": "no5G"
}

```

Listing 4.8. Few-shot example – CVE not related to 5G (Cisco NX-OS).

Example 2 – RAN-related 5G CVE

Listing 4.9 presents a vulnerability of a component explicitly designed for 5G Radio Access Networks (RAN). This example represents the class of true positives, where the CVE clearly refers to software components involved in 5G communication.

```

USER INPUT:
[baseline instruction]
CVE description:
An issue was discovered in appmgr in O-RAN Near-RT RIC I-Release. An
  attacker could register an unintended RMR message type during
  xApp registration to disrupt other service components.

ASSISTANT OUTPUT:
{
  "reasoning": "The CVE affects the O-RAN Near-RT RIC I-Release, a
    component of the Open Radio Access Network (O-RAN) architecture
    used in 5G systems.",
  "classification": "5G"
}

```

Listing 4.9. Few-shot example – CVE related to 5G (O-RAN Near-RT RIC).

Example 3 – Non-5G CVE in a 5G vendor's product

Listing 4.10 includes a web application vulnerability in a ZTE firmware. Although ZTE is a telecommunications vendor, the affected component is unrelated to 5G networking. Such an example is useful to teach the model that vendor names alone are insufficient indicators of 5G relevance.

```

USER INPUT:
[baseline instruction]
CVE description:

```



```

The version V6.01.03.01 of the ZTE ZXCDN IAMWEB product is impacted
by a code injection vulnerability. An attacker could exploit the
vulnerability to inject malicious code into the management page,
resulting in user information leakage.
CPE:
['cpe:2.3:o:zte:zxcdn_iamweb_firmware']

ASSISTANT OUTPUT:
{
  "reasoning": "The vulnerability affects the ZTE ZXCDN IAMWEB
firmware, which is not a 5G network component. There is no
indication that this product is part of a 5G network
infrastructure or related to 5G-specific protocols. The issue
is a code injection in a management interface unrelated to 5G
communication layers.",
  "classification": "no5G"
}

```

Listing 4.10. Few-shot example – CVE not related to 5G (ZTE ZXCDN IAMWEB).

Example 4 – Non-5G network related CVE

Listing 4.11 shows another example of a non-5G vulnerability, involving a router. This device is a general-purpose router and not used specifically in 5G network. This example helps constrain the model's interpretation of "network-related vulnerabilities" to truly 5G-specific components.

```

USER INPUT:
[baseline instruction]
CVE description:
A Use After Free vulnerability in the Advanced Forwarding Toolkit
(AFT) manager process (aftmand) of Juniper Networks Junos OS
allows an unauthenticated remote attacker to cause a kernel crash
and Denial of Service (DoS). Continued gathering of interface
statistics will create a sustained DoS condition. This issue
affects Junos OS on MX Series platforms in versions between
20.1R1 and 21.2R2.
CPE:
['cpe:2.3:o:juniper:junos']

ASSISTANT OUTPUT:
{
  "reasoning": "The vulnerability affects Junos OS running on MX
Series routers, which are general-purpose routing platforms.
The CVE does not mention 5G network components, protocols, or
functions, and the product is not explicitly tied to 5G Core or
RAN. Therefore, it is not related to 5G networks.",
  "classification": "no5G"
}

```

Listing 4.11. Few-shot example – CVE not related to 5G (Juniper Junos OS).

4.5.4 Web-Context

With the techniques previously explained, LLM relies solely on the information contained in the CVE description and CPEs to make the classification. However, these fields are often short and may not be enough to understand the context of the vulnerability. To overcome this limitation, a **web-context prompting** strategy was introduced. In this approach, the model is enriched with external information retrieved from online sources, allowing it to reason based on both the CVE record and additional supporting evidence.

As illustrated in Figure 4.9, the web-context prompting pipeline extends the baseline approach by introducing an additional information retrieval phase. For each CVE record in the dataset, the corresponding URLs listed in the *References* section of the NVD entry (see Section 2.2) were collected. These links typically point to vendor advisories, technical blogs, or security bulletins that contain additional details about the vulnerability.

The retrieved web pages were parsed and summarized to extract concise and relevant text segments describing the affected components, exploitation vectors, or impacted systems. This summarized context, referred to as **web context**, was then appended directly to the input prompt along with the CVE description and CPEs.

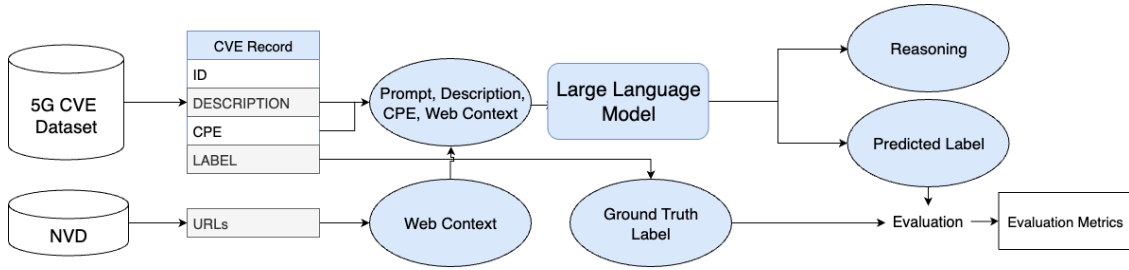


Figure 4.9. Web-context LLM-based classification pipeline.

To obtain the web context associated with each CVE, the URLs were processed using the **Jina AI** framework, in particular Jina AI Reader, introduced in Section 4.3, which parses the web page, removing all non-informative elements such as HTML tags or code fragments.

Since the raw web content can be long and contain mostly irrelevant information, the summarization step is fundamental to include in the prompt only relevant context. Two alternative summarization strategies were explored 4.12.

LLM Summarizer

In the first configuration, illustrated in Figure 4.10 and referred to as the **LLM summarizer**, the same language model used for classification was also adopted to summarize the retrieved web pages. To perform this task, the model was guided through specific summarization prompts.

To assess the impact of different prompt formulations on the quality of the generated summaries, four distinct prompt configurations were tested. They differ in two main aspects:

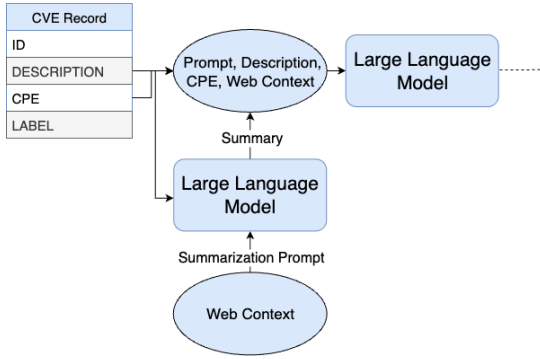


Figure 4.10. *

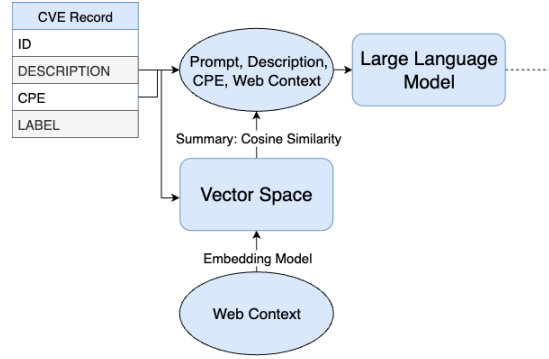


Figure 4.11. *

(a) LLM-based summarization.

(b) Embedding-based summarization.

Figure 4.12. Comparison between the two web context summarization strategies: (a) using a Large Language Model as summarizer, and (b) using embedding similarity with cosine distance.

- the **task definition**, i.e., whether the model is explicitly instructed to summarize the text for the purpose of 5G classification (*5G-specific summary*) or to produce a general summary (*generic summary*);
- the **presence of CVE-specific information** in the prompt, i.e., whether the CVE ID, CPEs, or CVE description are provided to help the model focus on the relevant product or vulnerability.

Table 4.2 summarizes the four configurations used for the summarization experiments.

Prompt ID	Task Definition	CVE Information Included
Web Prompt 0	5G-specific	Not Provided
Web Prompt 1	5G-specific	CVE ID, Description, CPEs
Web Prompt 2	Generic summary	Not Provided
Web Prompt 3	Generic summary	CVE ID, Description, CPEs

Table 4.2. Summary of the LLM-based web context summarization prompts.

To have an idea of what these prompts look like, Listing 4.12 shows the complete text of the first configuration (*Web Prompt 0*), which corresponds to the 5G-specific summarization task without providing CVE-specific information. The model is instructed to extract only the information relevant for determining whether a vulnerability is related to 5G or not, while discarding irrelevant or generic text.

```
You are a summarizer assistant for an LLM that will classify
vulnerabilities.
Below is the raw text extracted from a web page linked to a CVE.
The text may contain technical details, product descriptions,
general content, or unrelated material.
```

```

Your task is to analyze the text and extract only the information
that can help classify the CVE as 5g or non-5g related, without
refactoring it or modifying it in any way.
DO NOT add assumptions or unrelated info.

### Use the following JSON format for your classification:
{'{'
  "info": "relevant information for the classification",
  "useful": "yes | no"
{'}}

If useful is "yes", the 'info' field must contain relevant technical
information. If "no", leave it empty.{nothink_string}
### Text to analyze:
{web_text}

```

Listing 4.12. Example of LLM-based web summarization prompt (Web Prompt 0).

However, this configuration has a major drawback: it introduces a significant increase in computational cost.

Embedding Summarizer

To overcome this limitation, LLM-based summarization was compared with a **embedding-based summarizer**, illustrated in Figure 4.11. The retrieved web content was instead represented in a vector space using the *all-MiniLM-L12-v2* sentence transformer model. The first ten text segments with the highest cosine similarity to the CVE description were selected and concatenated to form the summary. The quantity of sentences to insert in the prompt was decided based on the maximum number of tokens that the context can contain using the available computational resources, described in Section 4.3.

This method is computationally lightweight and avoids redundant information, while still providing enough contextual cues for the LLM to reason effectively during inference.

Both summarization strategies were evaluated to understand the trade-offs between context quality, inference latency, and overall classification performance improvement.

4.5.5 Prompt Chaining

While the previous strategies rely on a single, monolithic prompt to guide the reasoning process, the **Prompt Chaining** approach decomposes the task into multiple consecutive steps. Each step produces an intermediate output that becomes the input for the following one, thus allowing the model to reason in a more structured and interpretable way. This approach aims to improve consistency during the analysis of the CVE, and it has been proven to improve accuracy in complex tasks [17].

As illustrated in Figure 4.13, the classification task is divided into three sequential stages.

The first step leverages the prompt presented in Listing 4.13 to extract from the CVE description and the CPEs the type of **Affected product** (e.g., protocol, network device, user equipment, or software product). This helps the model establish a clear context for the subsequent reasoning stage.



Figure 4.13. Overview of the Prompt Chaining approach.

The second step builds on the product description obtained in Step 1 and extends the baseline prompt described in Listing 4.3. Here, the model receives both the CVE information and the identified component, and must determine whether the vulnerability is *relevant to 5G infrastructure* and perform the classification.

Finally, in the last step, the model reformulates its decision in a compact JSON format containing both the reasoning and the classification, starting from the previous results, with a prompt that is a variation of the baseline presented in Listing 4.5.

```

Given the following CVE description and CPEs, identify what type of
component is affected by the vulnerability.

**CVE Description**:
{description}
**Vulnerable Products CPE**:
{CPEs}
Your goal is to describe what kind of product, technology, or
protocol is involved.

```

Listing 4.13. Prompt Chaining – Step 1: Component Identification.

This approach enhances the interpretability of the model’s decision process and provides more transparent intermediate reasoning. However, it also triples the inference cost, since the model must be queried multiple times for each CVE.

4.5.6 Chain-of-Thought

The **Chain Of Thought (Cot) prompting** technique encourages the model to reason explicitly by producing intermediate steps before reaching the final answer [57]. Unlike the Prompt Chaining approach, where reasoning is divided into multiple separate prompts, CoT operates within a single prompt, asking the model to *think step by step* to reach the final decision. This increases the number of generated tokens per response, yet it remains computationally efficient than Prompt Chaining because it still requires only one inference call.

In this configuration, the LLM receives the same inputs used in the baseline prompt (see Listing 4.7), but is explicitly instructed to verbalize its reasoning step by step (see Listing 4.14), rather than providing only a short justification in the JSON output. To preserve the ability to automatically parse the model’s responses, the reasoning section is enclosed in `<think>` tags, while the final JSON output is enclosed within `<answer>` tags.

```

Base prompt....
### Steps
....

```

```

5. Make sure to reason step by step before answering, enclose your
   reasoning in <think> tags, then answer using this JSON format and
   enclose it in <answer> tags.
....Rest of the base prompt....

```

Listing 4.14. Example of Chain of Thought prompting.

4.5.7 Tree-of-Thought (ToT)

The **Tree-of-Thought (ToT)** prompting technique extends the Chain-of-Thought paradigm by allowing the model to explore multiple reasoning paths instead of following a single linear one [65]. In contrast to CoT, where the reasoning is sequential, ToT introduces a branching exploration mechanism where alternative lines of reasoning are generated, evaluated, and progressively refined before converging on a final decision. This approach aims to emulate the human process of deliberation, exploring several hypotheses, comparing them, and reaching a consensus.

In this work, the Tree-of-Thought methodology was implemented following the Panel-GPT prompt [55], where "3 experts are discussing the question with a panel discussion, trying to solve it step by step, and make sure the result is correct and avoid penalty" [54]. As illustrated in Figure 4.14, each expert acts as an independent reasoning branch, expressing its initial interpretation, reacting to others' observations, and refining its stance over multiple turns. The goal is to achieve a structured form of collective reasoning that converges toward a single, well-justified classification outcome.

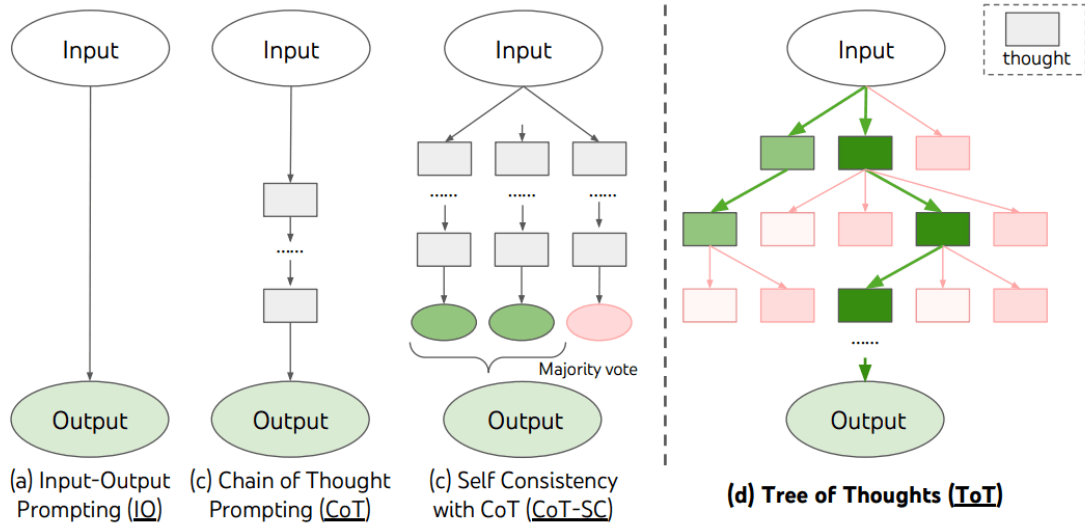


Figure 4.14. Comparison between prompting strategies [65].

In the prompt, the experts are explicitly instructed to discuss whether a CVE is related to 5G or not; also, the criteria from Listing 4.3 are included. Experts must alternate turns, consider each other's reasoning, and conclude the discussion with a JSON output format

that mirrors the baseline format. A simplified version of the implemented prompt is shown in Listing 4.15.

```
You are simulating a panel of 3 security experts discussing whether
a CVE is related to 5G or not.
Each expert must reason independently, then respond to others, and
reach a final consensus.

Discussion Process:
1. Each expert (Expert1, Expert2, Expert3) shares an initial
   assessment.
2. Experts read others' reasoning and refine their positions.
3. The discussion MUST end with at least one expert producing
   the final JSON output.

Use these tags:
, , to mark turns.
If an expert cannot apport any improvement to the discussion, they
must state it explicitly and may leave the discussion.
Even in full agreement, the final JSON output is mandatory.

Final output format:
{
  "reasoning": "Concise explanation of why this CVE is or isn't
    related to 5G networks.",
  "classification": "5g | no5g"
}
```

Listing 4.15. Simplified Tree-of-Thought (multi-expert) prompt.

The final classification corresponds to the most consistent and collectively endorsed reasoning path. However, as with other deliberative methods, ToT incurs a higher computational cost compared to single-pass prompting due to the highly increased length of the generated text.

4.5.8 Prompt Engineering Techniques Summary

To conclude the analysis of prompt engineering methods, Table 4.3 summarizes all the strategies explored in this work. Comparing how many distinct LLM calls are required to complete a single classification (*N. of. Inf.* column), and the computational effort they require to be executed (*Cost* column)

4.5.9 Fine Tuning

Large Language Models (LLMs) are pretrained through an unsupervised process on massive text corpora, allowing them to acquire both linguistic competence and a broad understanding of technical concepts (See SubSection 2.3.8). During this phase, the model learns general linguistic and technical knowledge across a wide range of domains. To perform a task, the model needs to be guided through a prompt, which provides all the necessary information and constraints on how the task should be completed.

Approach	Description	N. of Inf.	Cost
Baseline Prompt	Single zero-shot prompt defining the classification task and output format.	1	Low
Few-Shot Prompting	Extends the baseline prompt by including a small number of labeled examples to guide the model’s reasoning.	1	Low
Web-Context (LLM)	Enriches the prompt with contextual information retrieved from external web sources, summarized by an LLM.	2	High
Web-Context (Emb.)	Enriches the prompt with external information summarized using cosine similarity over sentence embeddings.	1	Medium
Prompt Chaining	Decomposes the task into multiple sequential steps: component identification, relevance evaluation, and output generation.	3	High
Chain of Thought (CoT)	Encourages explicit reasoning by instructing the model to verbalize intermediate steps within a single prompt.	1	Medium
Tree of Thoughts (ToT)	Simulates multiple reasoning paths through a multi-agent discussion that converges toward a consensus decision.	1	High

Table 4.3. Summary of the Prompt Engineering Strategies explored in this work.

In the field of 5G CVE classification, the base prompt (Listing 4.7) instructs the models with all the relevant network functions, protocols, and architectural components required to distinguish between the two classes. Extending this instruction with examples, as done in the few-shot configuration (see Listing 4.8), helps the model better understand the task. However, due to the limited context window of LLMs, these examples may still not provide enough coverage for the model to capture the complex linguistic and semantic patterns in the vulnerability Description that characterize 5G-related vulnerabilities.

To bridge this gap, a more effective approach consists of **Fine-Tuning**, where the model is trained directly on a dataset of conversations between a human user requesting the CVE classification and an assistant performing the task. This approach allows the neural network to autonomously learn the recurring linguistic and semantic patterns that differentiate 5G-related vulnerabilities from generic ones. The fine-tuning process slightly adjusts the model parameters to adapt to the specific task or domain, while preserving the general knowledge acquired during pretraining [48].

There exist multiple fine-tuning strategies, ranging from full-parameter training to *parameter-efficient fine-tuning* (PEFT) methods such as **LoRA** (Low-Rank Adaptation). These families of techniques drastically reduce memory usage and computational cost by updating only a small set of low-rank matrices instead of all model parameters [19]. In this work, the fine-tuning process was implemented using the **Unsloth** library (see SubSection 4.3.2), which supports LoRA-based training even on resource-constrained devices or cloud-hosted notebooks, such as the free tier on Google Colab used in this study (see SubSection 4.3.1).

Pipeline

LoRA introduces a set of small, trainable matrices into selected layers of the pretrained model. Instead of updating the full weight matrices, LoRA decomposes its update into the product of two lower-rank matrices, so that the original pretrained weights remain unchanged, drastically accelerating the fine-tuning process and reducing memory consumption, since not all the parameters need to be loaded in VRAM.

The fine-tuning process was performed on a **conversational dataset** derived from the 5G CVE dataset, described in Section 4.1. Each record was reformulated into a short dialogue, shown in Listing 4.16, where a human user requests the classification of a vulnerability, and the assistant provides both the reasoning and the final label in the JSON format defined in the baseline prompt (see Listing 4.5). This conversational structure, known as *chat template*, is detailed in Paragraph 4.3.2.

```
{
  "conversations": [
    {
      "role": "user",
      "content": {base_prompt},
    },
    {
      "role": "assistant",
      "content": { reasoning , classification},
    }
  ]
}
```

Listing 4.16. Structure of the Conversational format.

The conversational dataset was synthetically built from the test performed with larger models (up to 32B parameters). For each incorrect prediction produced by these models, the corresponding record was corrected by guiding the model toward the right label and requesting grounded and consistent reasoning, by extending the base prompt with two additional steps, and directly feeding the model with the true label, as shown in Listing 4.17.

```
You are provided with a CVE summary, if available, a list of CPEs
(Common Platform Enumerations) identifying the affected products,
and a label "5g" or "no5g", which is manually reviewed by experts.

The label means:
- "5G": the vulnerability directly or indirectly affects 5G
  networks, components, or protocols.
- "no5G": if it is unrelated to 5G, or belongs to legacy tech like
  4G, 3G, or unrelated fields (e.g., 5GHz Wi-Fi, IoT, industrial
  devices).

Your goal is looking at the summary to generate a reasoning that
describe why the product have the given label.

###Classification Criteria
```

```

.....

#Steps
.....
5. Describe why the product, components, or protocols are related or
   not related to 5G.
.....

**CVE Description**:
{description}

**Vulnerable Products CPE**:
{CPEs}

**True Label**:
{true_label}

```

Listing 4.17. Prompt to build syntetic Conversational dataset.

Given the computational constraints of the available hardware (see Subsection 4.3.1), the experiments focused on lightweight models ranging from 4 to 8 billion parameters, including **Llama 3.1**, **Qwen 3**, and **Gemma 2**. This configuration was chosen as a trade-off of the model provided by the Unsloth library, yet ensuring that the fine-tuning process could be executed within the limit of the Colab T4 GPU.

The main hyperparameters used during fine-tuning are summarized in Table 4.4. Each hyperparameter was selected to balance training stability, computational efficiency, and generalization ability, following the best practices reported in the Unsloth documentation [20].

- The **Learning rate** controls how much the model weights are updated during each optimization step. Lower values may lead to underfitting, while higher ones can destabilize the LoRA adapters and cause divergence.
- The **Batch Size** determines how many training examples are processed simultaneously. Smaller batches help regularize training and reduce GPU memory usage.
- The number of **Epochs** defines how many times the model iterates over the entire dataset. A range between 3 and 10 epochs was found to be sufficient to reach convergence, as longer training increases the risk of overfitting the model.
- The **Rank (r)** defines the dimensionality of the low-rank update matrices in the LoRA adapters. A lower rank reduces memory usage and training time, while higher ranks increase model capacity but also the number of trainable parameters.
- **Alpha α** parameter acts as a scaling factor for the LoRA updates, by controlling how strongly the learned deltas affect the original weights. It should be tuned accordingly with the chosen rank factor, to ensure stable gradient propagation.
- Mixed **Precision** training is possible, from 4-bit to 16-bit quantization, to reduce memory footprint and accelerate computation.

Parameter	Range
Learning rate	$1 \times 10^{-4} - 3 \times 10^{-4}$
Batch size	2 – 8
Epochs	3 – 10
LoRA rank (r)	8, 16, 32, 64, 128
LoRA alpha (α)	equals or double the rank (r or $2r$)
Precision	4, 16, or 32-bit

Table 4.4. Fine-tuning configuration parameters and explored ranges.

The complete fine-tuning workflow adopted in this study is summarized in Figure 4.15. The resulting **Fine-tuned LLM** is then evaluated, producing the evaluation metrics defined in Section 4.2. By comparing the obtained scores across different hyperparameter configurations, the best-performing model setup was selected as the final configuration for the final prototype.

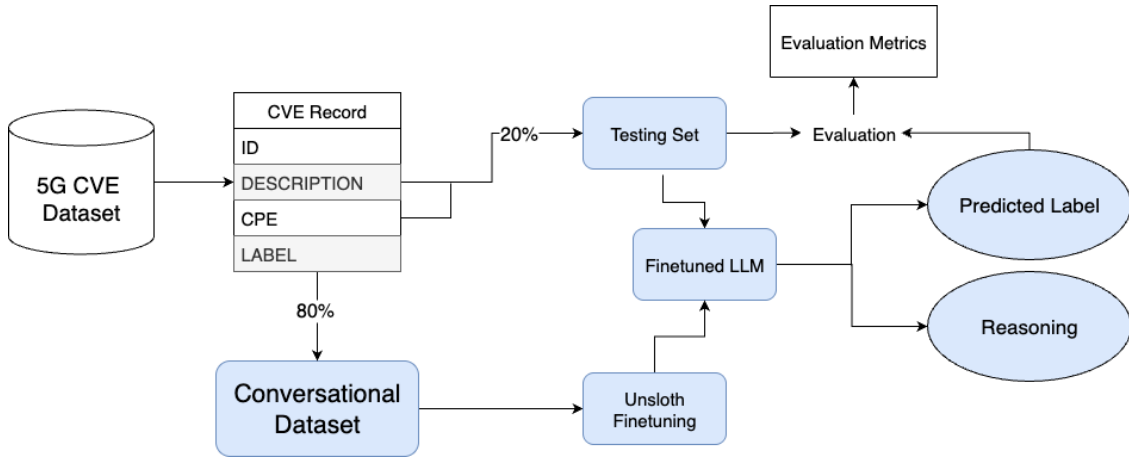


Figure 4.15. Overview of the fine-tuning pipeline.

4.6 Prototype Development

To demonstrate the applicability of the proposed classification pipeline in a real-world setting, a lightweight prototype system was developed. The goal of this prototype is to provide a proof-of-concept tool that can automatically classify newly published vulnerabilities and notify security analysts when a CVE is likely related to the 5G infrastructure.

The prototype integrates the best-performing model, chosen as a result of the comparison of the evaluation metrics. Figure 4.16 illustrates the architecture of the prototype that adopts the proposed pipeline on fresh CVE feeds. The system retrieves CVE records from the **NVD API**, extracts the fields used by the classifier, such as CVE description and

CPEs, and forwards them to the **Classification System**. This component is modular, so that the classification model can be updated, keeping up with the evolution of AI models.

At inference time, the classifier emits a **Predicted Label** together with a concise **Reasoning**. Both are stored in a **audit log** for subsequent evaluation. When the predicted label is *5G*, an **Alerting System** notifies subscribed stakeholders through a dedicated messaging channel. For the purpose of this prototype, a Telegram Bot has been implemented, but the modular design of the system allows this component to be easily replaced or extended in future versions.

The system performs classification periodically, looping over a configurable time interval that can be adjusted according to the user's needs. At each iteration, it retrieves newly published CVE records from the NVD API and restarts the classification pipeline, ensuring continuous monitoring of emerging vulnerabilities.

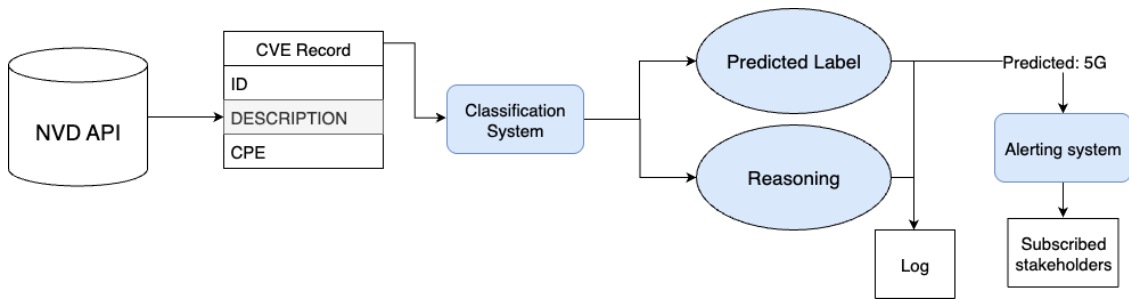


Figure 4.16. Prototype architecture: from NVD to classification, logging, and alerting.

The **NVD API** introduces some limitations. The maximum query range for a single API Key is limited to 120 days, meaning that the application must handle cases in which it is executed after a longer inactivity period by automatically performing multiple sequential queries. Moreover, to prevent denial-of-service attacks, the API enforces a rate limit of 5 requests with a rolling 30-second window, which can be increased to 50 requests by registering on the NIST website. So a short delay of 1 second between consecutive requests has been introduced to reduce server load.

The classification component of the prototype was implemented using the **Llama.cpp server**, which exposes a lightweight HTTP API for model inference. This choice allows the system to run the fine-tuned model locally or on any compatible GPU with minimal dependencies and without relying on external services. Thanks to the asynchronous request handling, the classifier can process multiple CVE records sequentially within each execution cycle, maintaining low latency even on limited hardware.

The User interface, pictured in 4.17, was developed using the **Streamlit** framework and provides a minimal yet responsive web dashboard where users can visualize and review the model's classifications.

The screenshot displays the 'CVE 5G Classifier' web application interface. On the left is a sidebar with 'app' and 'Classification History' links. The 'Settings' section includes a 'CALL_URL' field with the value 'http://127.0.0.1:8080/v1/chat', a 'Limit batch CVE' slider set to 50, a checked 'Mostra reasoning LLM' option, and a 'Start pipeline' button. The 'Single CVE test' section has a 'CVE ID' field with 'CVE-2024-12345' and a 'Summary' text area. The main content area features a 'CVE 5G Classifier' title, a pipeline status 'Pipeline: NVD Retriever → LLM Classifier → 5G', and a 'CVE Recovery from NVD' section showing '29 CVE recovered'. Below this is an 'LLM Classification' section with a progress bar and the message 'Classification Completed in 122.1s — 5 CVE 5G found'. The 'Results' section includes a '5G filter' dropdown set to 'Everyone', a 'Hide ERROR' checkbox, and a 'Search by CVE ID' input field. A table lists the results of the classification.

cve_id	summary	classification	is_5g	reasoning
CVE-2025-41442	A vulnerability exists in Advantech iView versions prior to 5.7.05 build 7057, which co	no5g	☐	The CVE summary mentions Advantech iView versions, w
CVE-2025-46358	Emerson ValveLink products do not use or incorrectly uses a protection mechanism t	no5g	☐	The CVE summary mentions Emerson ValveLink products
CVE-2025-46704	A vulnerability exists in Advantech iView in NetworkServlet.processImportRequest()	5g	☒	The CVE summary describes a directory traversal vulnera
CVE-2025-48496	Emerson ValveLink products use a fixed or controlled search path to find resources, t	no5g	☐	The CVE affects Qualcomm Snapdragon processors and tl
CVE-2025-48891	A vulnerability exists in Advantech iView that could allow for SQL injection through tl	no5g	☐	The CVE summary describes a SQL injection vulnerability
CVE-2025-50109	Emerson ValveLink Products store sensitive information in cleartext within a resource	no5g	☐	The CVE summary mentions Emerson ValveLink Products

Figure 4.17. Screenshot of the Prototype's Interface.

Chapter 5

Experimental Results

This chapter presents the experimental results obtained by applying the methodology described in Chapter 4.

The goal is to evaluate and compare the performance of different families of classifiers, including embedding-based and LLM-based, as well as the optimization strategies applied, such as prompt engineering and fine-tuning.

The evaluation follows a chronological structure reflecting the two experimental stages of the study (see Section 4.1):

- **Evaluation on the Public 5G CVE Dataset** (Section 5.1), consisting of 136 CVE records, used to define the baseline pipeline and to evaluate prompt-engineering techniques on different LLMs. On this dataset, both embedding-based classifiers and fine-tuning strategies were not feasible due to the restricted size of the CVE dataset.
- **Evaluation on the Extended Dataset** (Section 5.2), consisting of 476 CVEs, provided in a later phase, which allowed fine-tuning experiments and a statistically more robust assessment of embedding-based classifiers.

All models were executed on the hardware configurations introduced in SubSection 4.3.1. Lightweight LLMs (up to 8B parameters) were executed locally on the **m2_pro**, while Larger open-source models (up to 32B parameters) were executed on the **rtx_4090** workstation. A limited number of proprietary LLMs were evaluated through **external APIs** for reference.

5.1 Experiments on the Public 5G CVE Dataset

5.1.1 Baseline

The first set of experiments aimed to establish a baseline performance level, on the 136-record public 5G CVE dataset detailed in Section 4.1, for the LLMs detailed in Section 4.3.2. Each model was evaluated using the **Base prompt** described in Listing 4.7, using two different sets of hyperparameters **default temperature configuration** suggested by the model producers, which introduce randomness and reflects standard inference settings, and a more **reproducible configuration** with temperature $T = 0$, which ensures consistency across repeated runs.

As explained in Section 4.2, the **Matthews Correlation Coefficient (MCC)** is a representative reference metric for comparing the different prompt-engineering strategies, providing a single, balanced indicator that simultaneously accounts for true and false positives and negatives. Therefore, it is used as an indicator together with the **Accuracy** metric to create the baseline.

Table 5.1 shows a comparison of these metrics obtained by testing the models over the reproducible configuration ($T = 0$) compared to the default T configuration.

Table 5.1 shows that the $T = 0$ configuration consistently achieves slightly higher accuracy compared to the default temperature settings. This improvement is particularly evident for lightweight models such as **Gemma-3n (4B)** and **Llama-3.1(8B)**, while for larger models, including **Gemma-3 (27B)** and **Qwen-3 (32B)**, the results obtained with the two settings converge. For this reason, $T = 0$ can be considered reliable and subsequent tests, even on an additional model, were conducted with a fixed deterministic temperature ($T = 0$).

Model	Params (B)	Accuracy (Default T)	Accuracy ($T=0$)	MCC (Default / $T=0$)
Llama-3.2	3	0.74	0.76	0.56 / 0.57
Gemma-3n	4	0.74	0.84	0.55 / 0.71
Mistral	7	0.89	0.87	0.78 / 0.77
Qwen3 (8B)	8	0.90	0.91	0.79 / 0.84
Llama-3.1	8	0.89	0.93	0.78 / 0.87
Gemma-3 (12B)	12	0.93	0.95	0.86 / 0.92
SecGPT	14	0.88	0.92	0.77 / 0.85
Gemma-3 (27B)	27	0.98	0.98	0.96 / 0.96
Qwen-3	32	0.96	0.96	0.93 / 0.93

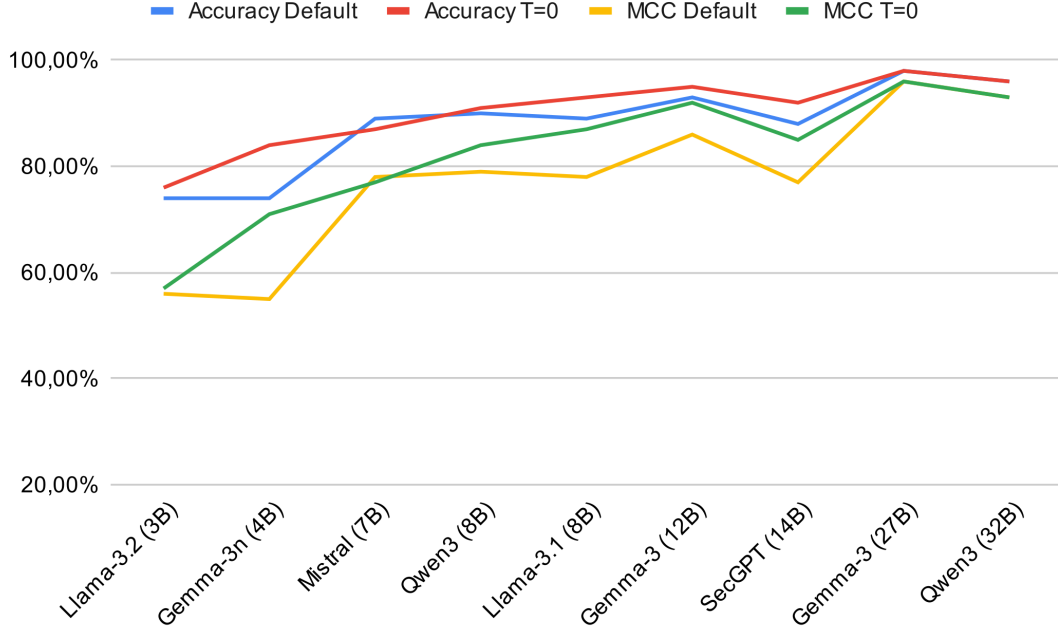
Table 5.1. Comparison between default and deterministic configurations ($T=0$) for baseline LLMs on the public dataset.

Additionally, from the plot in Figure 5.1, the overall performance trend can be observed. It is clear that performance increases almost monotonically with the number of parameters, from 0.74 for smaller 3–4B models, to values greater than 0.96 for models beyond 12B parameters. Notably, **Qwen3 (8B)** and **Gemma-3 (12B)** have performances comparable to larger models, achieving an accuracy of around 0.97 and 0.93, indicating that increasing the number of parameters over 12 billion yields only marginal performance gains.

5.1.2 Prompt Engineering Techniques

After establishing the baseline performance of the models, a series of experiments were conducted to evaluate the impact of different **prompt engineering techniques** on the 5G classification task. Compared to the baseline evaluation, this stage also includes the results for *Llama-3 (70B)*, which was tested only at this point due to its high computational cost and long inference time.

Table 5.2 summarizes the relative MCC % variations obtained by applying the different prompt-engineering techniques (see SubSection 4.5.1) to each model, compared against the

Figure 5.1. Metrics Comparison between Default Configuration and $T = 0$

baseline MCC value. For clarity, each prompt-engineering technique has been abbreviated in the table as follows: **Few-Shot (FS)**, **Web Context (WC)**, **Chain-of-Thought (CoT)**, **Prompt Chaining (PC)**, and **Tree-of-Thought (ToT)**. It is also important to note that the symbol “/” indicates cases where the output produced by the model was unusable, typically due to malformed JSON responses or ambiguous classifications that could not be automatically parsed. Similarly, the label “*unfeasible*” marks experiments that could not be executed due to excessive computational cost or memory requirements, which made those configurations impractical within the available hardware setup.

Figure 5.2 presents a comparative line plot in which the vertical axis represents the Matthews Correlation Coefficient (MCC), while the horizontal axis lists the evaluated models, ordered by increasing number of parameters. From this Figure, Prompt Chaining and Tree of Thought were excluded due to their poor results and the excessive GPU memory requirements. Overall, the graph shows that Prompt-engineering techniques appear to be much more effective on smaller models. As the model size increases, the difference between the baseline MCC and the optimized prompts generally narrows.

This trend can be explained by looking at the model scale: for bigger models, the internal representations may be sufficiently rich to generalize across the task without relying heavily on external guidance from the prompt, and so only smaller and mid-sized models benefit from optimized prompting, compensating for their limited capacity by providing more precise task framing and structured reasoning.

The detailed results presented in Table 5.2 provide a quantitative overview across the

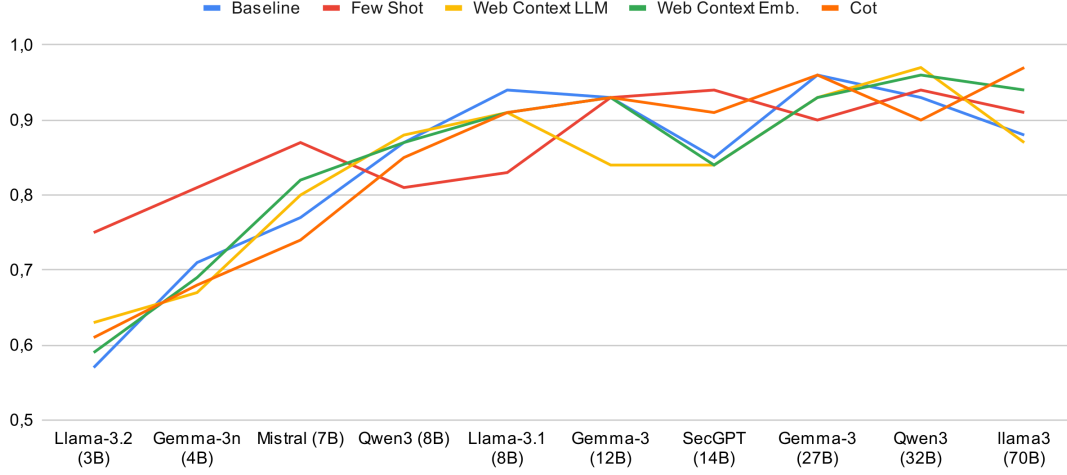


Figure 5.2. Comparison between MCC, for different Models and Prompt Engineering Techniques

different prompt engineering techniques with respect to the baseline. The *Few-shot* (FS) technique brings the highest mean improvement (+4.89%), suggesting that it is the best prompt engineering technique as a lightweight and low-cost optimization method. Conversely, more complex reasoning strategies such as *Prompt Chaining* (PC) and *Tree-of-Thought* (ToT) exhibit negative or unstable results, highlighting that additional inference steps and longer prompts do not necessarily translate into higher reliability.

Model	Params (B)	Base	FS	WC (LLM)	WC (Emb.)	CoT	PC	ToT
Llama-3.2	3	0,57	+31,58	+10,53	+3,51	+7,02	+24,56	-19,3
Gemma-3n	4	0,71	+14,08	-5,63	-2,82	-4,23	+4,23	+1,41
Mistral	7	0,77	+12,99	+3,90	+6,49	-3,90	+5,19	/
Llama-3.1	8	0,87	-6,90	+1,15	0,00	-2,30	-5,75	-13,79
Qwen-3 (8B)	8	0,84	-11,70	-3,19	-3,19	-3,19	0,00	-10,64
Gemma-3 (12B)	12	0,92	0,00	-9,68	0,00	0,00	+3,23	-17,20
SecGPT	14	0,85	+10,59	-1,18	-1,18	+7,06	-38,82	/
Gemma-3 (27B)	27	0,96	-6,25	-3,12	-3,12	0,00	/	/
Qwen-3 (32B)	32	0,93	+1,08	+4,30	+3,23	-3,23	+4,30	/
Llama3 (70B)	70	0,88	+3,41	-1,14	+6,82	+10,23	unfeasible	unfeasible
Mean Variation			+4,89	-0,41	+0,97	+0,75	-0,38	-11,9

Table 5.2. Comparison of MCC baseline and relative improvements (%) for each prompt-engineering strategy on the public 5G CVE dataset.

A complementary perspective is offered by Figure 5.3, which directly compares the MCC scores obtained across each configuration, excluding PC and ToT, which were unfeasible

for larger models. Here, the improvement introduced by Few-Shot prompting is visually evident for the smaller models (Llama-3.2 and Gemma-3n), while the differences among techniques become progressively less pronounced for larger architectures such as Gemma-3 (27B) and Qwen-3 (32B).

This visualization also confirms that Web-Context enrichment (both LLM and embedding-based) provides moderate but consistent improvements across small models and will be further analyzed in SubSection 5.1.3.

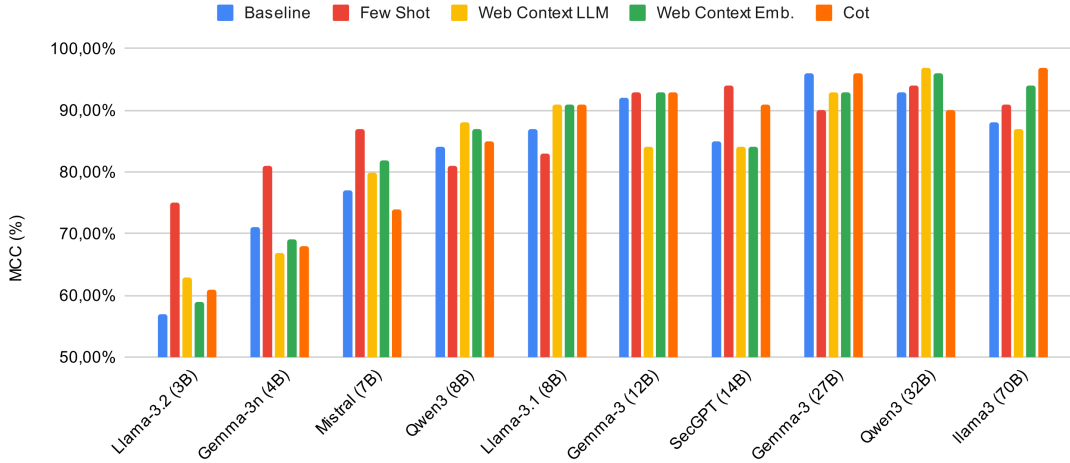


Figure 5.3. Comparison between MCC (%) With Default Configuration and $T = 0$

In conclusion, although some individual models exhibit notable improvements under specific prompting configurations, these results show a high degree of variability and cannot be directly generalized to an operational context. The **sensitivity** of the models to prompt formulation limits the practical applicability of the strategies tested beyond controlled experiments. Among all tested techniques, **Few-Shot prompting** emerges as the most balanced solution, providing consistent gains across multiple architectures without significantly increasing inference time or computational cost (see Table 4.3). However, it is essential to note that the examples used in this configuration were derived from the dataset used for evaluation. As a consequence, while the observed improvements demonstrate the potential of Few-Shot prompting in guiding model reasoning, these gains might not fully generalize to unseen vulnerabilities or to CVE records outside the curated dataset.

5.1.3 Web Context LLM Analysis

Despite the fact that the WEB-context approach substantially expands the input prompt by adding a large amount of external information, the overall MCC variations observed in Figure 5.3 appear unexpectedly limited compared to other approaches, such as Few-Shot.

This raises a fundamental question: how can long snippets of text produce, on average, such a small variation in the performance? To address this question, a dedicated analysis

was conducted on four distinct **Web Prompts**, designed to investigate how the formulation of the summarization influences the resulting classification. These prompts and the methodology used to conduct the experiments are described in detail in SubSection 4.5.4.

Due to hardware constraints, the experiments were conducted only on **Llama-3.1 (8B)** and **Qwen-3 (8B)** models, the two configurations that, as shown in Table 5.2, exhibited the smallest MCC variation in the WC (LLM) approach, among those testable on the *m2_pro* hardware configuration. The goal was to understand *how* the addition of external context leads the models to change their binary classification between "5G" and "no5G".

The results presented in Table 5.3 show that, across the four summarization prompts tested, plus the one that skips the summarization phase, the MCC values vary by approximately 10% for both models. At first glance, it might be observed that the best configuration of **Llama-3.1** achieves an MCC comparable to that of models nearly three times its size, as reported in Table 5.1. However, the high variability in MCC across the tests demonstrates that the binary classification strongly depends on the specific summary that the LLM receives.

Model	Approach	Acc.	Prec.	Rec.	F1	MCC	Δ MCC	Δ MCC %
Llama-3.1	Baseline	0.93	0.94	0.93	0.93	0.87	–	–
Llama-3.1	Skip summary step	0.91	0.89	0.93	0.91	0.83	-0.04	-4.60%
Llama-3.1	Web Prompt 0	0.94	0.94	0.94	0.94	0.88	+0.01	+1.15%
Llama-3.1	Web Prompt 1	0.91	0.90	0.93	0.91	0.82	-0.05	-5.75%
Llama-3.1	Web Prompt 2	0.95	0.98	0.91	0.94	0.90	+0.03	+3.45%
Llama-3.1	Web Prompt 3	0.96	1.00	0.91	0.95	0.92	+0.05	+5.75%
Qwen-3	Baseline	0.97	0.97	0.97	0.97	0.94	–	–
Qwen-3	Skip summary step	0.91	0.85	0.96	0.90	0.82	-0.12	-12.77%
Qwen-3	Web Prompt 0	0.93	0.89	0.97	0.93	0.86	-0.08	-8.51%
Qwen-3	Web Prompt 1	0.96	0.94	0.97	0.95	0.91	-0.03	-3.19%
Qwen-3	Web Prompt 2	0.93	0.88	0.99	0.93	0.86	-0.08	-8.51%
Qwen-3	Web Prompt 3	0.96	0.92	1.00	0.96	0.92	-0.02	-2.13%

Table 5.3. Detailed classification results for different Web Prompt configurations on Llama-3.1 (8B) and Qwen-3 (8B).

A closer inspection of the logs containing the individual predictions confirms the strong variability and high prompt sensitivity observed in the aggregated results. A notable example is the classification of the CVE record **CVE-2024-26793** (true label: *no5G*), performed by the *Llama 3.1 (8B)* model. Listings 5.1 and Listing 5.2 show the model’s responses to two prompts that differ only by the presence of empty *<webcontext>* tags. Nevertheless, the model changed its classification from *no5G* to *5G*, solely due to the structural difference introduced by the empty tags.

```
....Base Prompt....
### Answer using the following JSON format
{
  "reasoning": "Explain clearly why this CVE is or isn't related to
5G.",
  "classification": "5g | no5g"
```

```

}

Model Response:
{
  "reasoning": "The CVE affects the Linux kernel's GTP (GPRS
    Tunneling Protocol) implementation, which is used in various
    network protocols, including 5G. However, upon closer
    inspection, it appears that this specific vulnerability is
    related to a use-after-free and null-ptr-deref issue in the
    gtp_newlink() function. This is not directly related to any
    5G-specific protocol or component. The GTP protocol itself is
    used in both 4G and 5G networks, but the specific code path
    affected by this vulnerability does not seem to be unique to
    5G.",
  "classification": "no5G"
}

```

Listing 5.1. Model response to the base prompt (no web context).

```

....Base Prompt....
### Answer using the following JSON format
{
  "reasoning": "Explain clearly why this CVE is or isn't related to
    5G.",
  "classification": "5g | no5g"
}

<webcontext>

</webcontext>

Model Response:
{
  "reasoning": "This CVE is related to a Linux kernel vulnerability
    in the GTP (GPRS Tunneling Protocol) module, which is used by
    various protocols, including GTP-U and NAS. Although these
    protocols are used in 5G networks, this specific issue seems to
    be more related to general network functionality rather than
    being unique to 5G. However, it still affects the Linux kernel,
    which is used in many 5G infrastructure components.",
  "classification": "5G"
}

```

Listing 5.2. Model response to the same prompt by with the addition of empty tags.

This example clearly illustrates the problems of this approach and, in general, highlights how fragile prompt engineering can become when applied to lightweight models, where the addition of large and potentially noisy external content is likely to exacerbate this inherent prompt sensitivity. Additionally, looking at the bigger picture, Table 5.3 shows that in seven out of ten tests, the introduction of web-derived context led to a slight decrease in performance compared to the baseline configuration.

In light of these observations, the answer to the question posed at the beginning of this section becomes clearer: the issue is not the lack of information, but rather its quality

and structure. Small and mid-sized LLMs, such as those tested in this stage, exhibit a higher degree of prompt sensitivity and limited capacity to filter relevant details from long, noisy inputs. As a result, extending the prompt with unstructured web text often introduces ambiguity instead of clarity, leading to unstable reasoning paths and inconsistent classifications.

That being said, it is out of the scope of this Thesis to demonstrate whether the Web-Context approach could converge toward more stable results if tested with bigger models and on a substantially larger dataset.

5.2 Experiments on the Extended Dataset

Following the analysis on the public 5G CVE dataset, this section presents the second experimental phase of the study, conducted on a substantially larger and more representative dataset consisting of 476 manually annotated CVE records, described in Section 4.1.

This new dataset was provided in a later stage of the research and enabled techniques that weren't possible with the smaller dataset, such as embedding-based approaches and finetuning.

5.2.1 Baseline

Given the larger size of the extended dataset, it was not feasible to replicate the full prompt-engineering sweep performed on the public set. To verify the consistency of the LLMs' performances, the baseline was built using the two best models from Section 5.1 and using external APIs: **Gemma-3 (27B)** and **Qwen-3 (32B)**.

Table 5.4 reports the baseline results, with $T=0$, and the *relative* MCC variation with the Few-Shot (FS) approach, which showed the best results using the smaller dataset. The results are *consistent* with those observed on the public dataset: both models maintain high baseline performance. However, in this case, the Few-Shot approach leads to a small degradation of MCC in both cases.

This suggests that the examples previously crafted for the public dataset do not transfer to the extended one, and changing the set of technologies and products affected by the vulnerability can erase the benefits provided by this kind of approach. However, this consideration must be proven with broader tests and additional models, which are not feasible in this study.

Model	Params (B)	Accuracy (Base)	MCC (Base)	Δ MCC (FS) (% vs. Base)
Gemma-3	27	0.90	0.80	-2.50%
Qwen-3	32	0.90	0.80	-1.25%

Table 5.4. Baseline performance on the extended dataset, with Few-Shot (FS) relative MCC variation.

Overall, these findings justify the exploration of alternative strategies beyond prompt

engineering, namely embedding-based models and fine-tuning, which may offer more operational reliability.

5.2.2 Embedding-based approaches evaluation

The goal of this phase was to assess the capability of traditional machine learning classifiers, as detailed in SubSection 4.3.2.

For this purpose, the CVE descriptions, together with associated CPEs, were transformed into numerical vectors using the *all-miniLM-L12-v2* model. Three classifiers were trained and tested under the same 80/20 split: **Logistic Regression**, **SVM (RBF kernel)**, and **XGBoost**. Additionally, the performance of the best-performing LLM over this new dataset (**Qwen3-32B**) from the previous section was included as a reference for comparison.

From Table 5.5, the **SVM (RBF)** clearly emerges as the best-performing classifier, achieving an accuracy of 0.93 and an MCC of 0.85, outperforming Logistic Regression, XGBoost, and even Qwen3-32B.

Model	Accuracy	Precision	Recall	F1-Score	MCC	Pred. Correct
Logistic Regression	0.82	0.80	0.85	0.82	0.65	79 / 96
SVM (RBF)	0.93	0.91	0.93	0.92	0.85	89 / 96
XGBoost	0.88	0.83	0.91	0.87	0.75	84 / 96
Qwen3-32B (LLM)	0.92	0.89	0.93	0.91	0.83	88 / 96

Table 5.5. Performance comparison of embedding-based classifiers and Qwen3-32B on the extended dataset (80/20 split).

To decide the best parameter possible for the SVM configuration, a series of validation experiments was conducted varying the regularization parameter C and the kernel coefficient γ . The resulting validation curves, shown in Figure 5.4 and 5.5, illustrate how the model accuracy changes in relation to these hyperparameters.

The validation curve for C (Figure 5.4) shows that accuracy rapidly increases for low values of C and stabilizes beyond $C = 2$, after which the classifier performance converges.

Similarly, the validation curve for γ (Figure 5.5) reveals that excessively high values lead to overfitting: the training accuracy approaches one while validation accuracy decreases.

The optimal configuration was therefore determined as $C = 2$ and $\gamma = 1$, balancing bias and variance effectively.

To verify that the model has reached a good fit on the training set, the learning curve in Figure 5.7 illustrates how accuracy evolves as the training set size increases.

The validation accuracy steadily grows and converges toward the training curve, indicating that the SVM generalizes well and that additional training data would yield further improvements without overfitting.

To further evaluate the robustness of the SVM model, a real-world test was conducted on a set of *9,000 newly published CVE records* retrieved from the NVD. These records, not included in the extended dataset, were published between July 1st and September 13th, 2025. Among them, the SVM model identified 133 vulnerabilities as 5G-related.

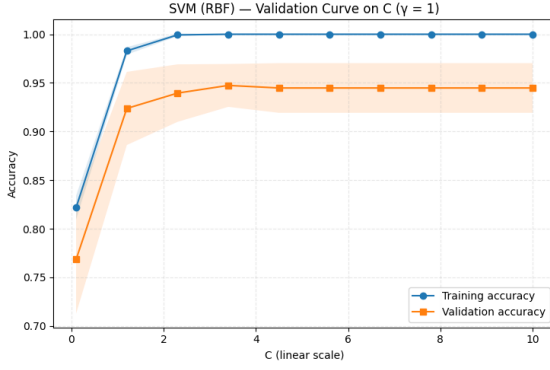


Figure 5.4. *

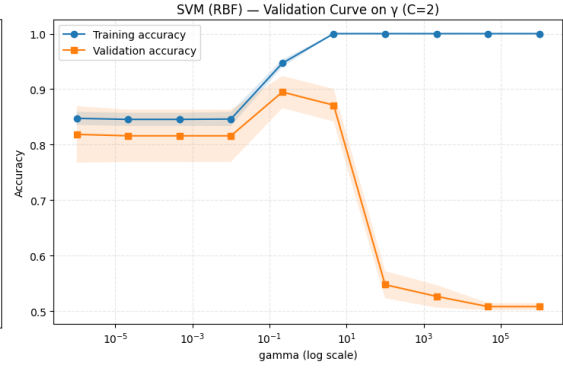


Figure 5.5. *

(a) Validation curve for SVM (RBF):
accuracy vs. C (fixed $\gamma = 1$).

(b) Validation curve for SVM (RBF):
accuracy vs. γ (fixed $C = 2$).

Figure 5.6. Validation curves from SVM (RBF) used to determine C and γ .

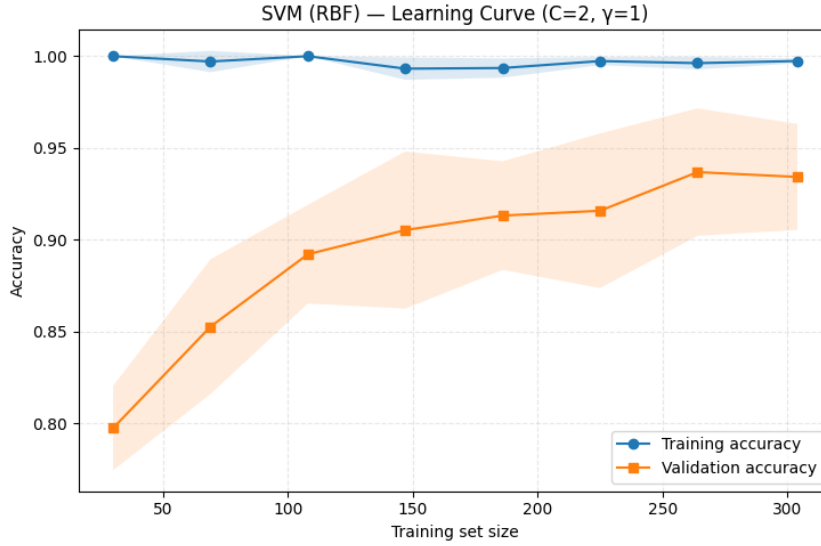


Figure 5.7. Learning curve (accuracy vs. training set size) for SVM (RBF) with $C = 2$, $\gamma = 1$).

This subset was then manually reviewed and annotated, and the same records were also classified using **Qwen3-32B**, enabling a direct comparison between the two approaches.

As shown in Table 5.6, the SVM model exhibits a severe drop in performance, achieving only a 0.23 accuracy and a negative Matthews correlation coefficient ($MCC = -0.11$). In contrast, Qwen3-32B model maintains performance with respect to the dataset, confirming its ability to generalize beyond the vocabulary and data distribution of the training set.

Model	Accuracy	Precision	Recall	F1-Score	MCC	Pred. Correct
SVM (RBF)	0.23	0.23	1.00	0.37	-0.11	31 / 133
Qwen3-32B (LLM)	0.98	0.94	0.97	0.95	0.94	130 / 133

Table 5.6. Generalization test of SVM and Qwen3-32B on 133 unseen 5G-related CVEs.

This result highlights an intrinsic limitation of the traditional SVM classifier: despite not suffering from overfitting on the training set, its decision boundaries are strongly influenced by keywords present in the training corpus. Consequently, when exposed to real-world CVE records containing unseen terminology or shifted contexts, SVMs fail to generalize reliably.

A notable example is the record **CVE-2025-53816**, describing a vulnerability in *7-Zip*, a software clearly unrelated to 5G technology:

“7-Zip is a file archiver with a high compression ratio. Zeroes written outside heap buffer in RAR5 handler may lead to memory corruption and denial of service in versions of 7-Zip prior to 25.0.0.”

The SVM incorrectly classified this vulnerability as **5G-related**, likely due to the presence of the token “*RAR5*”, which coincidentally resembles keywords found in training samples (i.e., RAN) associated with telecommunication protocols. When the same description was slightly modified by masking the term *RAR5*, the model correctly reclassified the CVE as **non-5G**, revealing a strong dependency on superficial lexical cues rather than on the semantic meaning of the text.

These findings confirm the limits on the generalization capabilities of embedding-based classifiers and highlight the more stable reasoning and contextual understanding provided by LLMs, making them a more promising choice for a 5G CVE classification system.

5.2.3 Fine-tuning Evaluation

The fine-tuning process followed the approach described in SubSection 4.5.9, leveraging the **Unsloth** library, which enables memory-efficient training through Low-Rank Adaptation (**LoRA**), as detailed in Section 4.3.

All-finetuning experiments were conducted in Google Colab environments using the free-tier GPU configuration, which imposed heavy constraints on model size and VRAM availability. As a result, only models up to **4 billion parameters** were fine-tuned, namely **Llama3.2-3B** and **Qwen3-4B**, both quantized to **Q4_K_M** format for optimization. The fine-tuning process, using Unsloth, was configured through two key parameters: *rank* (r) and *LoRA alpha* (α), which control the dimensionality and scaling of the adapter layers. Multiple combinations of these parameters were tested to identify the most effective configuration.

The training used the **extended 5G dataset** split 80/20 as in previous experiments. The 80% training set was converted into a *conversational format*, following the structure detailed in SubSection 4.5.9, while the remaining 20% was reserved for evaluation.

Table 5.7 reports the results obtained for all fine-tuned models, their corresponding base versions, and the results obtained on the same test subset for *Qwen3-32B*, which

was used as a reference to compute the $\Delta MCC(\%)$ values, with the formula reported in Section 4.2. It can be observed that all fine-tuned models outperform their corresponding base versions, proving the effectiveness of this strategy. For **Llama3.2-3B**, the MCC progressively increases with higher rank and LoRA_alpha values, reaching 0.69 at the configuration $r = 64$, $\alpha = 64$. A similar trend is observed for **Qwen3-4B**, where the configuration $r = 32$, $\alpha = 32$ achieves the best results, surpassing even the eight times larger **Qwen3-32B** model, with an MCC of 0.81.

Model	Params (B)	Accuracy	MCC	ΔMCC (%)
Llama3.2-3B	3	0.78	0.58	-25.64
Llama3.2-3B_r16_la16	3	0.80	0.61	-21.79
Llama3.2-3B_r32_la32	3	0.81	0.63	-19.23
Llama3.2-3B_r64_la64	3	0.84	0.69	-11.54
Llama3.2-3B_r128_la128	3	0.83	0.67	-14.10
Qwen3-4B	4	0.85	0.71	-8.97
Qwen3-4B_r16_la16	4	0.86	0.74	-5.13
Qwen3-4B_r32_la32	4	0.91	0.81	+3.85
Qwen3-4B_r64_la64	4	0.85	0.71	-8.97
Qwen3-4B_r128_la128	4	0.85	0.72	-7.69
Qwen3-32B (reference)	32	0.89	0.78	0.00

Table 5.7. Performance of fine-tuned models on the extended dataset compared to their base versions.

Results in Table 5.7 demonstrate that fine-tuning enables lightweight LLMs to achieve accuracy levels comparable to larger models while maintaining significantly lower computational requirements.

To further assess whether the fine-tuned models preserve their generalization ability, the best configuration **Qwen3-4B**, with $r = 32$, $\alpha = 32$, was tested on the same 133 CVE records used in Subsection 5.2.2 for the SVM evaluation. This test allows us to ensure that the fine-tuned model maintains the generalization capability, and can serve as further evaluation to compare its performance with respect to *Qwen3-4B* base version, and the large reference model *Qwen3-32B*.

As shown in Table 5.8, fine-tuning brings a substantial improvement in both accuracy and MCC, with the finetuned version of **Qwen3-4B** surpassing the base version and almost matching the performance of the much larger 32B model. This demonstrates that the finetuned model maintains high performance even on CVE records outside the extended dataset.

These findings confirm the effectiveness of LoRA-based fine-tuning in improving the reasoning and domain adaptation of lightweight LLMs in the task of CVE 5G classification. Not only does it enhance performance, but it also preserves the ability to execute the model locally, in a computationally constrained environment, ensuring data privacy and reducing the dependence on external APIs [13]. Within the context of this Thesis, the fine-tuned **Qwen3-4B** model represents the most balanced configuration, achieving high accuracy, strong generalization, and low computational cost. For this reason, it was selected as the

Model	Params (B)	Accuracy	MCC
Qwen3-4B (base)	4	0.88	0.64
Qwen3-4B (fine-tuned)	4	0.96	0.91
Qwen3-32B (reference)	32	0.97	0.92

Table 5.8. Comparison between fine-tuned and base Qwen3 models on real case scenario.

core model for the prototype implementation discussed in Section [4.6](#).

Chapter 6

Conclusions and Future Work

This Thesis addressed the problem of automatically identifying and classifying vulnerabilities that affect 5G infrastructures, in response to the growing number of CVEs published each year and the increasing complexity of modern network architectures. Traditional approaches based on keyword filtering and manual review are no longer sufficient to keep up with the scale and ambiguity of vulnerability data, particularly when vulnerabilities indirectly impact 5G components through shared protocols, libraries, or virtualization layers.

To overcome this challenge, a methodology was developed that combines machine learning and large language model (LLM)-based techniques to automate the classification of 5G-related CVEs. Two complementary families of approaches were explored: embedding-based classifiers, which transform CVE descriptions into semantic vector representations and apply traditional machine learning algorithms such as SVM and logistic regression; and LLM-based methods, which leverage the contextual understanding of transformer models to reason directly over textual descriptions. The methodology was evaluated through multiple experiments, including baseline testing, prompt engineering, and parameter-efficient fine-tuning, to assess model performance, scalability, and generalization capability in realistic operational settings.

The initial experiments were conducted on a dataset of 136 CVE records, which allowed the construction of an LLM-based baseline and the exploration of various prompt-engineering strategies. The baseline showed a clear correlation between model size and classification accuracy: performance increased from approximately 74% for 3B-parameter models to around 90% for models between 8B and 12B parameters. Beyond this, results began to converge, and additional parameters provided only marginal gains, with accuracy stabilizing between 96% and 98% for models up to 32B parameters.

Regarding prompt engineering strategies, the results suggest that using prompting as an optimization technique is generally unreliable, especially for smaller models that are highly sensitive to minor variations in the prompt structure. This makes prompt-based optimization difficult to adopt in operational domains such as 5G vulnerability analysis, where reliability and stability are critical.

The only technique that consistently provided promising improvements was the **Few-Shot prompting** approach, which augments the input with examples to guide the model’s reasoning. This was particularly useful in ambiguous cases, for instance, when the CVE description referred to technologies whose names resemble 5G-related components (e.g.,

“5GHz modem”). On average, Few-Shot prompting increased accuracy by roughly 5% across tested models. However, these examples were inevitably built around the dataset used at that time and may not generalize well to unseen data. Once the model is applied outside the original dataset, the benefit of Few-Shot prompting may diminish due to potential bias in the selected examples.

In later stages of the research, the "Fondazione Ugo Bordonì" provided a larger, unpublished dataset of 476 CVE records—more than triple the size of the initial dataset. Given hardware constraints, this phase required the use of external APIs to test the models. Only **Qwen3-32B** and **Gemma3-27B** were evaluated as baselines on this extended dataset, both maintaining strong performance with about 90% accuracy. However, on this larger and more diverse dataset, the Few-Shot prompting strategy led to slightly worse results, supporting the hypothesis that the examples previously crafted were not representative of the broader and more heterogeneous 5G ecosystem.

The availability of a larger dataset also enabled, for the first time, experiments based on classical training-testing splits (80/20). In this phase, a vectorized representation of CVE descriptions was created using the **all-MiniLM-L12-v2** sentence transformer model, which maps each record into a semantic vector space. This embedding-based approach is extremely efficient and computationally lightweight compared to full LLM inference. Among the tested classifiers, the **SVM with RBF kernel** achieved the best results, reaching 93% accuracy on the testing split—slightly higher than the 92% obtained by **Qwen3-32B**.

To validate robustness, the models were then tested on a set of 133 newly published CVEs retrieved directly from the NVD, entirely external to the training data and without any keyword filtering. This scenario best approximates a real-world operational setting, where the classifier continuously processes new CVEs as they are published. In this test, the SVM performance dropped sharply to 23% accuracy, while **Qwen3-32B** maintained consistent performance above 90%. These results suggest that the SVM overfits lexical cues present in the training data, making it unable to generalize to new product names or unseen terminology. Conversely, the LLM maintained its contextual understanding and reasoning ability in all domains.

The final and most decisive technique evaluated was **parameter-efficient fine-tuning (PEFT)** on compact models of 3–4B parameters. Using the **Unsloth** library, fine-tuning was successfully executed within the free tier of Google Colab for both **Llama3.2-3B** and **Qwen3-4B**. The fine-tuning was performed on 80% of the extended dataset, with evaluation on the remaining 20%. Results showed consistent improvements over the respective base models, and notably, the fine-tuned **Qwen3-4B** achieved performance comparable to that of **Qwen3-32B** despite being eight times smaller.

An additional evaluation of the fine-tuned model on unseen CVEs confirmed its robustness: **Qwen3-4B (fine-tuned)** reached 96% accuracy compared to 97% for **Qwen3-32B**. This result demonstrates that fine-tuned lightweight LLMs retain the rich linguistic and contextual understanding developed during pretraining, while effectively specializing in the 5G vulnerability classification task through the fine-tuning process. Compared to traditional classifiers, fine-tuned LLMs exhibit far greater adaptability and semantic reasoning, enabling them to generalize across domains and remain reliable even in continuously evolving real-world conditions.

6.1 Limitations

Although the proposed methodology demonstrated promising results, several limitations must be acknowledged to frame the scope of this work correctly.

First, the experiments were strongly constrained by the available hardware resources. These constraints limited both the number and the scale of experiments that could be performed, preventing a more extensive and statistically robust evaluation of the models on the 5G classification task. For instance, the baseline evaluation of the extended dataset included only a reduced set of models. It should be expanded to obtain a more complete view of model behaviour across different architectures and parameter sizes. Similarly, the parameter-efficient fine-tuning was executed on quantized 4B-parameter models ($Q_4_K_M$) using the free tier of Google Colab. While this setup proved the feasibility of low-resource fine-tuning, it may be too restrictive, potentially affecting the performance of the original models.

Second, the evaluation was conducted on two datasets, both constructed through keyword-based filtering. Although this approach is a practical necessity for identifying 5G-related vulnerabilities within the National Vulnerability Database, it inevitably introduces selection bias. The rapidly increasing number of published CVEs each year brings a continuous variety of technologies, products, and naming conventions that the dataset may not accurately represent. This limitation was particularly evident in the embedding-based classifiers, which performed well during controlled validation but failed to generalize in real-world conditions. Such behavior does not stem from overfitting to the training data—as shown by stable learning and validation curves—but rather from a lack of representativeness of the dataset itself. While this issue can be partially mitigated by leveraging Large Language Models that have already undergone extensive pretraining, further testing on broader and more diverse real-world datasets is necessary to fully assess the generalization capabilities of the proposed approaches and to draw stronger conclusions.

6.2 Future Work

The results obtained in this Thesis open several directions for future research and development.

An immediate extension of this work involves the continuous expansion and refinement of the dataset. Using the same methodology proposed in this study as a supporting tool, newly published vulnerabilities can be automatically labeled and progressively added to the dataset. This would enable a continuous improvement loop in which the fine-tuning process is periodically repeated, leading to increasingly accurate models. In an operational environment, such a mechanism could facilitate the protection of the entire 5G infrastructure by automating the classification process and reducing the need for human supervision, which is essential given the accelerating rate at which new CVEs are disclosed.

In addition, the same methodology can be extended beyond the 5G domain and reused across other critical sectors, such as energy, transportation, healthcare, and public administration. Adapting the classification pipeline to these domains would not only contribute to improving their cybersecurity posture but would also demonstrate the broader potential of AI-based vulnerability classification. Such cross-domain applications could

significantly enhance current risk identification frameworks, providing a tangible improvement by supporting timely and data-driven decision-making in the protection of critical infrastructures.

Bibliography

- [1] Irfan Ahmed, Hedi Khammari, Adnan Shahid, Ahmed Musa, Kwang Soon Kim, Eli De Poorter, and Ingrid Moerman. «A Survey on Hybrid Beamforming Techniques in 5G: Architecture and System Model Perspectives». In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3060–3097. DOI: [10.1109/COMST.2018.2843719](https://doi.org/10.1109/COMST.2018.2843719).
- [2] Jina AI. *reader*. [Online; Accessed 2025, 5 October]. 2024. URL: <https://github.com/jina-ai/reader>.
- [3] Mistral AI. *Mistral AI*. <https://huggingface.co/mistralai>. [Online; Accessed 2025, 11 October]. 2025.
- [4] Jay Alammar. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. <https://jalammar.github.io/illustrated-bert/>. [Online; Accessed 2025, 30 September]. 2018.
- [5] Jay Alammar. *The Illustrated GPT-2 (Visualizing Transformer Language Models)*. <https://jalammar.github.io/illustrated-gpt2/>. [Online; Accessed 2025, 30 September]. 2018.
- [6] Jay Alammar. *The Illustrated Transformer*. <https://jalammar.github.io/illustrated-transformer/>. [Online; Accessed 2025, 30 September]. 2018.
- [7] Jay Alammar. *The Illustrated Word2vec*. <https://jalammar.github.io/illustrated-word2vec/>. [Online; Accessed 2025, 30 September]. 2018.
- [8] Jay Alammar. *Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)*. <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>. [Online; Accessed 2025, 30 September]. 2018.
- [9] Xavier Amatriain. «Prompt design and engineering: Introduction and advanced methods». In: *arXiv preprint arXiv:2401.14423* (2024).
- [10] Xiaoqi Ren Anant Nawalgaria and Charles Sugnet. *Embeddings & Vectors Store*. <https://www.kaggle.com/whitepaper-embeddings-and-vector-stores>. [Online; Accessed 2025, 27 September]. 2025.
- [11] Saptarashmi Bandyopadhyay, Jason Xu, Neel Pawar, and David Touretzky. «Interactive visualizations of word embeddings for k-12 students». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 11. 2022, pp. 12713–12720.
- [12] Bogdan Barchuk and Kyrylo Volkov. «Limitations of modern vulnerability scanners and CVE Systems». In: *World Journal of Advanced Engineering Technology and Sciences* (2024). DOI: [10.30574/wjaets.2024.12.2.0348](https://doi.org/10.30574/wjaets.2024.12.2.0348).

- [13] Pierpaolo Bene, Andrea Bernardini, Leonardo Sagratella, Nicolo Maunero, and Marina Settembre. *Optimizing Local LLM Deployment for 5G CVE Classification Avoiding External Data Exposure IEEE CNS 25 Poster*. Sept. 2025. DOI: [10.5281/zenodo.16736495](https://doi.org/10.5281/zenodo.16736495).
- [14] D’Alterio Francesco Bernardini Andrea Sagratella Leonardo. *5G CVE Dataset*. 2025. DOI: [10.5281/zenodo.16736494](https://doi.org/10.5281/zenodo.16736494).
- [15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*. 2020. arXiv: [2005.14165](https://arxiv.org/abs/2005.14165) [cs.CL]. URL: <https://arxiv.org/abs/2005.14165>.
- [16] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. «Xgboost: extreme gradient boosting». In: *R package version 0.4-2* 1.4 (2015), pp. 1–4.
- [17] Yu Cheng, Jieshan Chen, Qing Huang, Zhenchang Xing, Xiwei Xu, and Qinghua Lu. *Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains*. 2023. arXiv: [2306.12028](https://arxiv.org/abs/2306.12028) [cs.SE]. URL: <https://arxiv.org/abs/2306.12028>.
- [18] Davide Chicco and Giuseppe Jurman. «The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification». In: *BioData Mining* 16.1 (2023), p. 4.
- [19] Michael Han Daniel Han. *Unsloth Documentation*. <https://docs.unsloth.ai/>. [Online; Accessed 2025, 3 October]. 2025.
- [20] Michael Han Daniel Han. *Unsloth Documentation*. <https://docs.unsloth.ai/get-started/fine-tuning-llms-guide/loras-hyperparameters-guide>. [Online; Accessed 2025, 6 October]. 2025.
- [21] scikit-learn developers. https://scikit-learn.org/1.5/auto_examples/model_selection/plot_validation_curve.html. [Online; Accessed 2025, 4 October]. 2025.
- [22] ENISA. *5G CYBERSECURITY STANDARDS*. <https://www.enisa.europa.eu/publications/5g-cybersecurity-standards>. [Online; Accessed 2025, 2 October]. 2022.
- [23] ENISA. *ENISA THREAT LANDSCAPE FOR 5G NETWORKS*. <https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-for-5g-networks>. [Online; Accessed 2025, 18 September]. 2025.
- [24] Hugging Face. *Huggingface*. <https://huggingface.co/huggingface>. [Online; Accessed 2025, 11 October]. 2025.
- [25] Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. *Problems With Evaluation of Word Embeddings Using Word Similarity Tasks*. 2016. arXiv: [1605.02276](https://arxiv.org/abs/1605.02276) [cs.CL]. URL: <https://arxiv.org/abs/1605.02276>.

- [26] Anju Uttam Gawas. «An overview on evolution of mobile wireless communication networks: 1G-6G». In: *International journal on recent and innovation trends in computing and communication* 3.5 (2015), pp. 3130–3133.
- [27] Georgi Gerganov, Diego Devesa, Johannes Gäbler, Xuan-Son Nguyen, Daniel Bevenius, Jeff Bolz, Sigbjørn Skjæret, Jared Van Bortel, Kawrakow, Olivier Chafik, Ruben Ortlam, Pierrick Hymbert, compilade, ROCKSTAR, Radoslav Gerganov, Kerfuffle, Akarshan Biswas, lhez, Eve, Neo Zhang Jianyu, Aman Gupta, uvos, Stephan Walter, Brian, Aaron Teo, Eric Curtin, Jhen-Jie Hong, Chenguang Li, Pavol Rusnak, and Someone. *ggml-org/llama.cpp*. <https://github.com/ggml-org/llama.cpp>. [Online; Accessed 2025, 3 October]. 2025.
- [28] Rikhiya Ghosh, Hans-Martin von Stockhausen, Martin Schmitt, George Marica Vasile, Sanjeev Kumar Karn, and Oladimeji Farri. «Cve-llm: Ontology-assisted automatic vulnerability evaluation using large language models». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 28. 2025, pp. 28757–28765.
- [29] Google. *GPU Architecture*. https://colab.research.google.com/github/d2l-ai/d2l-tvm-colab/blob/master/chapter_gpu_schedules/arch.ipynb. [Online; Accessed 2025, 3 October]. 2023.
- [30] GPU-Monkey. *Apple M2 Pro 16-Core GPU*. https://www.gpu-monkey.com/en/gpu-apple_m2_pro_16_core_gpu. [Online; Accessed 2025, 3 October]. 2023.
- [31] GPU-Monkey. *NVIDIA GeForce RTX 4090 Founders Edition*. https://www.gpu-monkey.com/en/gpu-nvidia_geforce_rtx_4090_founders_edition. [Online; Accessed 2025, 3 October]. 2022.
- [32] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. 2024. arXiv: [2407.21783](https://arxiv.org/abs/2407.21783) [cs.AI]. URL: <https://arxiv.org/abs/2407.21783>.
- [33] Alibaba Group. *Welcome to Qwen*. <https://huggingface.co/Qwen>. [Online; Accessed 2025, 11 October]. 2025.
- [34] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. «Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects». In: *Authorea preprints* 1.3 (2023), pp. 1–26. DOI: [10.36227/techrxiv.23589741.v7](https://doi.org/10.36227/techrxiv.23589741.v7).
- [35] J. D. Hunter. «Matplotlib: A 2D graphics environment». In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: [10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55).
- [36] Jiaming Ji, Tianyi Qiu, Boyuan Chen, Borong Zhang, Hantao Lou, Kaile Wang, Yawen Duan, Zhonghao He, Jiayi Zhou, Zhaowei Zhang, et al. «Ai alignment: A comprehensive survey». In: *arXiv preprint arXiv:2310.19852* (2023).
- [37] Daniel Jurafsky and James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.
- [38] Francesco Marchiori, Denis Donadel, and Mauro Conti. *Can LLMs Classify CVEs? Investigating LLMs Capabilities in Computing CVSS Vectors*. 2025. arXiv: [2504.10713](https://arxiv.org/abs/2504.10713) [cs.CR]. URL: <https://arxiv.org/abs/2504.10713>.

- [39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. «Efficient estimation of word representations in vector space». In: *arXiv preprint arXiv:1301.3781* (2013).
- [40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. «Distributed representations of words and phrases and their compositionality». In: *Advances in neural information processing systems* 26 (2013).
- [41] MITRE. *CVE: Common Vulnerabilities and Exposures*. <https://www.cve.org/about/Process>. [Online; Accessed 2025, 22 September]. 2025.
- [42] Sohinee Mondal. *Overfitting and Underfitting: Identify and Resolve — Part 2*. <https://medium.com/@sohineetitin/overfitting-and-underfitting-identify-and-resolve-part-2-48cbab897024>. [Online; Accessed 2025, 4 October]. 2024.
- [43] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. «Mteb: Massive text embedding benchmark». In: *arXiv preprint arXiv:2210.07316* (2022).
- [44] NIST. *NVD - General Information*. <https://nvd.nist.gov/general>. [Online; Accessed 2025, 22 September]. 2022.
- [45] OpenAI. *How ChatGPT and our foundation models are developed*. <https://help.openai.com/en/articles/7842364-how-chatgpt-and-our-foundation-models-are-developed>. [Online; Accessed 2025, 30 September]. 2025.
- [46] Neri Van Otten. *Sequence-to-sequence architecture made easy & how to tutorial in Python*. <https://spotintelligence.com/2023/09/28/sequence-to-sequence/>. [Online; accessed 30-September-2025]. 2023.
- [47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [48] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. «Language models are unsupervised multitask learners». In: *OpenAI blog* 1.8 (2019), p. 9.
- [49] Nils Reimers and Iryna Gurevych. «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: <https://arxiv.org/abs/1908.10084>.
- [50] Google Research. *Organization Card*. <https://huggingface.co/google>. [Online; Accessed 2025, 11 October]. 2025.
- [51] Juergen Schmidhuber. *Annotated History of Modern AI and Deep Learning*. 2022. arXiv: [2212.11279](https://arxiv.org/abs/2212.11279) [cs.NE]. URL: <https://arxiv.org/abs/2212.11279>.
- [52] Murray Shanahan, Kyle McDonell, and Laria Reynolds. «Role play with large language models». In: *Nature* 623.7987 (2023), pp. 493–498.
- [53] Salim S.I. *Attacks on 5G Infrastructure From Users’ Devices*. https://www.trendmicro.com/en_us/research/23/i/attacks-on-5g-infrastructure-from-users-devices.html. [Online; Accessed 2025, 9 October]. 2023.
- [54] Hao Sun. *PanelGPT: Prompt Language Models with a Penal Discussion*. <https://github.com/holarissun/PanelGPT>. [Online; Accessed 2025, 4 October]. 2023.

- [55] Hao Sun. «Reinforcement Learning in the Era of LLMs: What is Essential? What is needed? An RL Perspective on RLHF, Prompting, and Beyond». In: *arXiv preprint arXiv:2310.06147* (2023).
- [56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention is all you need». In: *Advances in neural information processing systems* 30 (2017).
- [57] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: [2201.11903 \[cs.CL\]](https://arxiv.org/abs/2201.11903). URL: <https://arxiv.org/abs/2201.11903>.
- [58] Wikipedia. *Artificial intelligence* — *Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Artificial%20intelligence&oldid=1314568599>. [Online; accessed 02-October-2025]. 2025.
- [59] Wikipedia. *Attention Is All You Need* — *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Attention_Is_All_You_Need. [Online; accessed 30-September-2025]. 2025.
- [60] Wikipedia. *Feedforward neural network* — *Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Feedforward%20neural%20network&oldid=1313833670>. [Online; accessed 30-September-2025]. 2025.
- [61] Wikipedia. *Generative pre-trained transformer* — *Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Generative%20pre-trained%20transformer&oldid=1309626917>. [Online; accessed 30-September-2025]. 2025.
- [62] Wikipedia. *Recurrent neural network* — *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Recurrent_neural_network. [Online; accessed 30-September-2025]. 2025.
- [63] Wikipedia. *Transformer (deep learning architecture)* — *Wikipedia, The Free Encyclopedia*. [http://en.wikipedia.org/w/index.php?title=Transformer%20\(deep%20learning%20architecture\)&oldid=1313511942](http://en.wikipedia.org/w/index.php?title=Transformer%20(deep%20learning%20architecture)&oldid=1313511942). [Online; accessed 30-September-2025]. 2025.
- [64] Yunqi Wuyin. *Sec-GPT*. <https://huggingface.co/clouditera/secgpt>. [Online; Accessed 2025, 11 October]. 2023.
- [65] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. 2023. arXiv: [2305.10601 \[cs.CL\]](https://arxiv.org/abs/2305.10601). URL: <https://arxiv.org/abs/2305.10601>.
- [66] Hongbin Ye, Tong Liu, Aijia Zhang, Wei Hua, and Weiqiang Jia. *Cognitive Mirage: A Review of Hallucinations in Large Language Models*. 2023. arXiv: [2309.06794 \[cs.CL\]](https://arxiv.org/abs/2309.06794). URL: <https://arxiv.org/abs/2309.06794>.
- [67] Jie Zhang, Haoyu Bu, Hui Wen, Yongji Liu, Haiqiang Fei, Rongrong Xi, Lun Li, Yun Yang, Hongsong Zhu, and Dan Meng. «When llms meet cybersecurity: A systematic literature review». In: *Cybersecurity* 8.1 (2025), p. 55.

- [68] Éireann Leverett. *Vulnerability Forecast for 2025*. <https://www.first.org/blog/20250607-Vulnerability-Forecast-for-2025>. [Online; Accessed 2025, 23 September]. 2025.