RESEARCH-ARTICLE

Knowledge-Driven Large Language Models for Automating CVSS Score Prediction

SEYEDEH LEILI MIRTAHERI, University of Calabria, Rende, CS, Italy

AMIR HOSSEIN MAJD, University of Calabria, Rende, CS, Italy

REZA SHAHBAZIAN, University of Palermo, Palermo, PA, Italy

ANDREA PUGLIESE, University of Calabria, Rende, CS, Italy

Open Access Support provided by:

University of Calabria

University of Palermo

Published: 13 October 2025

Citation in BibTeX format

CheckMATE '25: Proceedings of the 2025 Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks
October 13 - 17, 2025
Taipei, Taiwan

Conference Sponsors:
SIGSAC

Knowledge-Driven Large Language Models for Automating CVSS Score Prediction

Seyedeh Leili Mirtaheri
University of Calabria
Rende, CS, Italy
leili.mirtaheri@dimes.unical.it

AmirHossein Majd
University of Calabria
Rende, Italy
amirhossein.majd@dimes.unical.it

Reza Shahbazian
University of Palermo
Palermo, Italy
reza.shahbazian@unipa.it

Andrea Pugliese
University of Calabria
Rende, Italy
andrea.pugliese@unical.it

Abstract

In response to the continuous growth of software vulnerabilities, this paper introduces a hybrid artificial intelligence (AI) approach that combines large language models (LLMs) with structured cybersecurity knowledge graphs. Our method guides LLMs through explicit, expert-driven rules to predict Common Vulnerability Scoring System (CVSS) Base metrics directly from vulnerability descriptions. Experiments on five state-of-the-art models, including GPT-4o and DeepSeek, demonstrate significant gains. Our proposed knowledge-infused prompts lead to accuracy improvement in GPT-4o from 59% to over 82%, while the accuracy in the predictions of DeepSeek rose from 48% to 78%. Moreover, we show that this integration boosts cost-efficiency for lighter models. These results highlight the practical value of merging symbolic knowledge with LLM reasoning to achieve faster, more consistent, and interpretable vulnerability assessments.

CCS Concepts

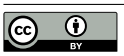
• **Computing methodologies** → **Machine learning**; *Natural language processing*; • **Security and privacy** → **Vulnerability management**; *Security services*.

Keywords

Large Language Models, Prompt Engineering, CVSS, Vulnerability Assessment, Machine Learning, Cybersecurity

ACM Reference Format:
Seyedeh Leili Mirtaheri, AmirHossein Majd, Reza Shahbazian, and Andrea Pugliese. 2025. Knowledge-Driven Large Language Models for Automating CVSS Score Prediction. In *Proceedings of the 2025 Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks (CheckMATE '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3733817.3762699

§Corresponding author: leili.mirtaheri@dimes.unical.it.



CheckMATE '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1906-6/25/10
https://doi.org/10.1145/3733817.3762699

1 Introduction

In today's digital world, software vulnerabilities can become powerful weapons for attackers. Such vulnerabilities don't just threaten lines of code, but they endanger confidentiality, integrity, and availability. Vulnerability management typically consists of three main stages: identifying new vulnerabilities (discovery), assessing their severity (importance), and ultimately taking action to mitigate their effects through proper control mechanisms [8]. During the assessment phase, standardized frameworks such as the Common Vulnerability Scoring System (CVSS) are employed to quantitatively describe the inherent characteristics and potential severity of each vulnerability. For instance, the National Vulnerability Database (NVD) has recorded more than 150,000 entries with corresponding Common Vulnerabilities and Exposures (CVEs) identifiers and associated CVSS scores. CVSS is an open and widely adopted framework designed to characterize and communicate the severity of software vulnerabilities [25].

CVSS consists of three main metrics: *Base Metrics*, *Temporal Metrics*, and *Environmental Metrics*. Among them, the *Base Metrics* capture the intrinsic and unchanging properties of a vulnerability, independent of time or environmental context. The Base Metrics are further divided into two subcategories of *Exploitability Metrics*, which focus on the mechanics of how an attack is carried out, and *Impact Metrics*, which describe the consequences of a successful exploit on the target system [1]. The most critical CVSS Base Metrics influencing the final severity score are Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR), User Interaction (UI), Scope (S), Confidentiality (C), Integrity (I), and Availability (A) [4]. More details on each metric are given in Appendix A. The values of each CVSS Base Metric are assembled into a CVSS vector, which is then processed through a standardized formula to compute the *Base Score*. This numerical score, ranging from 0 to 10, reflects the overall severity of a vulnerability and provides a consistent basis for comparison and prioritization. For example, consider the following CVSS version 3.1 [1] vector:

```
CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:C/C:L/I:L/A:H
```

This describes a vulnerability that is exploitable over the network (AV:N), requires high attack complexity (AC:H), and needs low privileges (PR:L) with user interaction (UI:R). The scope is changed (S:C), and it leads to low impacts on confidentiality and integrity (C:L, I:L), but a high impact on availability (A:H). When computed

using the CVSS formula, this vector might yield a score of, for instance, 9.0 out of 10, indicating a highly critical vulnerability.

Manually assigning these metrics to each vulnerability is not only slow but also leaves room for human errors. With the number of newly reported vulnerabilities growing so quickly, 38% from 2023 to 2024 by averaging over 135 new CVEs every day, security teams face an extreme workload. Analysts must sift through hundreds of vulnerability reports daily just to create the right CVSS vectors. Doing all of this by hand is simply not practical and often causes critical delays in assessing risks [2]. Also, some CVSS metrics are naturally subjective and depend a lot on expert judgment. This means two analysts might give different ratings for the same vulnerability. For example, they could disagree on how much a flaw affects confidentiality or integrity, leading to inconsistent scores. In reality, big organizations have to handle thousands of these assessments every month, which is far more than people can reasonably process on their own. For this reason, automating vulnerability scoring with artificial intelligence (AI) and machine learning isn't just helpful, but it is essential. Automation speeds up the process and potentially makes the results more consistent and reliable [16, 22]. To automate vulnerability scoring, some researchers have used machine learning models to predict CVSS metrics. These models can be quite accurate for certain tasks, but they often have trouble with the more qualitative parts, such as predicting the impact on confidentiality, integrity, and availability. Traditional ML and rule-based systems usually struggle to grasp the full context in vulnerability descriptions. This often leads to uneven performance across different metrics [14]. With the emergence of large language models (LLMs), new opportunities have been presented for natural language understanding and knowledge extraction from unstructured text [14]. Models such as GPT-3, GPT-4, BERT, and their successors are trained on massive general and domain-specific texts and have demonstrated capabilities in grasping semantic meaning.In the cybersecurity domain, LLMs have already been applied to various tasks such as malware detection, code analysis, threat reporting, and information extraction from unstructured sources [24].

Some studies use these capabilities of LLMs to analyze the text-based descriptions of vulnerability reports (e.g., CVE entries). These descriptions often provide valuable information on how a vulnerability can be exploited and what its impact might be. For example, if a report includes "remote attacker," that usually means it's a network attack (AV:N). Or if it includes "requires user interaction," it points to (UI:R). LLMs can match such descriptions to the right CVSS metrics. By learning from thousands of past examples, they can also identify deeper patterns. For example, when a report mentions "privilege escalation," it often means the attacker needs more than no privileges (PR $\neq$ N). State-of-the-art studies demonstrate that LLMs can generate CVSS scores quite well. However, there are still some challenges to face. For instance, general LLMs may not have enough built-in cybersecurity knowledge and might miss some of the fine details in these technical reports. That's why researchers are now looking at ways to feed LLMs more domain-specific knowledge to make them better at these specialized tasks [5]. One promising solution for enhancing LLM understanding of text-based descriptions is the integration of structured domain knowledge. Structured knowledge refers to the explicit representation of concepts and their relationships in machine-readable formats, such as knowledge graphs (see [9] as a recent survey).

The core idea of this paper is to integrate scattered domain knowledge from various documents and databases into a graph-based structure, where nodes represent entities (e.g., a specific vulnerability, software, or attack type), and edges denote relationships between them (e.g., "vulnerability A is an SQL injection" or "vulnerability B affects software X"). Such a security knowledge graph can act as an auxiliary memory for intelligent systems, enabling them to ground their reasoning on verified facts and connections. In the context of CVSS scoring, this structured knowledge can include formal definitions of metrics, canonical examples, and relationships between attack conditions and metric values. For instance, one part of the knowledge graph might encode decision trees that mirror the typical questions a human assessor would ask, such as:

> *"Can the attack be performed remotely?"* $\rightarrow$ If yes, then AV = N; otherwise, check whether the attack requires local or physical access.

Injecting structured security knowledge into LLMs can prevent misinterpretations and anchor the model's reasoning in factual context. In effect, an LLM guided by a knowledge graph behaves like a virtual expert, combining general linguistic capabilities with structured domain expertise [5].

1.1 Research Objective

The main objective of this paper is to design, implement, and evaluate an intelligent system that combines a domain-specific knowledge graph with a large language model to automatically predict the CVSS base vector of a vulnerability using only its textual description (e.g., from CVE). We evaluate if the proposed hybrid approach that integrates structured knowledge and LLMs can achieve accuracy levels comparable to human expert assessments in CVSS scoring.

1.2 Contributions

This paper makes several novel contributions, which can be summarized as follows:

- We propose a new hybrid framework for the automatic assessment of vulnerability severity by combining knowledge graphs with large language models.
- We construct a domain-specific security knowledge graph based on CVSS metrics and expert-defined relationships.
- We design a question-driven mechanism to incorporate knowledge graph insights into the LLM's reasoning process.
- We provide a comprehensive evaluation of the proposed framework using real-world vulnerability data.

This research represents a meaningful step toward automating vulnerability scoring and enhancing the overall management of software security risks. Subsequent sections will describe the related works, system implementation, and experimental results.

2 Related Works

Over the years, researchers have explored a range of techniques to automate vulnerability assessment and scoring. We divide them generally into four main categories. First, there are traditional machine

learning methods that rely on statistical patterns in data. Second, symbolic and rule-based systems that capture explicit expert logic. More recently, large language models have emerged as powerful tools for understanding and generating vulnerability assessments. Finally, hybrid approaches that aim to combine the strengths of both symbolic reasoning and machine learning or LLMs. In the following subsections, we briefly review each of these directions.

2.1 Machine Learning-Based Approaches

A significant body of research has explored the use of machine learning (ML) to predict vulnerability severity, specifically, CVSS scores. Early approaches employed classical ML models such as Support Vector Machines (SVMs) and Random Forests and even leveraged fuzzy logic to combine handcrafted rules. While these models achieved a reasonable degree of accuracy, they required manual feature extraction from the text (e.g., keywords related to attack vectors, complexity, privileges, etc.). With the advancement of deep learning, more recent methods have aimed to overcome the need for manual feature engineering [3, 15]. Modern deep learning architectures such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and more recently, Transformers have led to significant improvements in predictive accuracy. For instance, a BERT-based model trained solely on textual descriptions from the NVD was able to achieve high accuracy in predicting each parameter of the CVSS vector [27].

ML models are inherently dependent on labeled data [7, 10, 11, 20, 23] and more broadly, on the availability of high-volume, high-quality datasets for analytics in various domains [13]. Although the NVD provides CVSS scores and vectors for many CVEs, the quality of these labels is not always optimal. Some descriptions are ambiguous or incomplete, and severity labels may not be updated promptly. Another major limitation lies in generalization, and models trained on common patterns in historical data may fail to perform well on novel or differently phrased vulnerabilities [25].

2.2 Symbolic and Rule-Based Systems

Before the rise of deep learning and statistical methods, vulnerability severity assessment primarily relied on symbolic expert systems. These systems encoded expert knowledge in the form of if-then rules or ontologies, enabling the automatic derivation of CVSS scores from textual descriptions [21]. The main strength of this approach lies in its close alignment with the logical reasoning process of human analysts. For instance, a rule like:

> *"Mentions of remote code execution + no authentication required → Attack Vector = Network; Privileges Required = None"*

closely mirrors how a security analyst would manually work through a CVSS assessment. The authors in [21] translated CVE descriptions into a rule base, enabling an expert system to immediately compute scores and generate alerts for high-risk cases. Other studies have tried to make rule systems smarter by adding security ontologies like STIX, CAPEC, ATT&CK, or graphs that link CWEs to software, bringing structured knowledge to better handle real-world attacks [5]. Researchers have also used fuzzy logic in these systems to deal with uncertainty in vague phrases like "may result in information disclosure." This lets the rules move beyond simple yes-or-no decisions and handle more nuanced, in-between cases [19]. The primary advantages of symbolic approaches are transparency and alignment with formal guidelines. These systems explicitly show which rules lead to which decisions and can directly implement CVSS scoring manuals, OWASP guides, or CWE software graphs. Furthermore, outputs remain consistent under the same conditions, offering policy uniformity to organizations. On the downside, maintaining and updating hundreds of rules is costly and slow. Natural language coverage is limited, and managing exceptions increases complexity. As a result, these systems often underperform in comparison to data-driven models—especially when handling short or novel vulnerability descriptions. Recent research trends have thus shifted toward hybrid models, combining symbolic reasoning with deep learning, or progressively replacing static rule sets with advanced ML techniques.

2.3 LLM-Based Approaches

The emergence of large language models such as BERT and GPT-2/3/4 has introduced new opportunities for predicting vulnerability severity [14]. In the fine-tuning approach, LLMs are trained on thousands of labeled CVE descriptions. A notable example is BERT-CNN, which attaches eight dedicated classifiers (one for each base CVSS metric) to the output of a fine-tuned BERT model [15]. An alternative strategy uses zero-shot or few-shot prompting, where models like ChatGPT are applied without dedicated training. For example, ChatGPT has been shown to correctly predict the full CVSS-V2 vector in 44% of cases and the overall CVSS-V3 severity level in 65% of cases [22]. LLMs offer distinct advantages in deep language understanding and implicit knowledge transfer. For instance, a phrase like "allows remote attackers to execute arbitrary code" is interpreted as indicating high severity, even if the word "critical" is not explicitly mentioned. This kind of inference draws upon the model's broad training across general and domain-specific corpora. Moreover, LLMs can adapt rapidly to new CVSS versions or emerging threat scenarios with just a few updated examples or carefully designed prompts. Importantly, some LLMs can also provide natural language explanations alongside predictions, thereby improving transparency and trust in AI-based decision support systems [12]. Nonetheless, LLM outputs can be unpredictable or inconsistent, occasionally hallucinating values or producing CVSS vectors that violate formal constraints, requiring post-processing. Computational costs are also high, and real-time deployment at scale remains expensive—though compact versions such as DistilBERT have partially addressed this issue. Additionally, performance is sensitive to prompt engineering and parameter tuning, and keeping models up-to-date with emerging threats demands continuous retraining. Privacy concerns also arise when sending sensitive data to commercial cloud-based LLMs [26].

2.4 Hybrid Approaches

A growing research direction involves hybrid models that combine symbolic AI with machine learning, aiming to merge the explicit logic of human experts with the statistical flexibility of deep learning models [17]. A common pattern is to inject structured knowledge, such as a decision tree or CVSS–CWE relationship graph, into an LLM-based pipeline. In this setup, the LLM no longer starts

from a "blank slate," but is guided by a predefined security rulebook. Alternatively, the LLM's output may be post-processed using the knowledge graph to correct inconsistencies [5]. Studies integrating fuzzy logic with neural networks or connecting LLM predictions to rules extracted from the CVE database have shown that adding this symbolic layer improves both accuracy and explainability. The hybrid approach offers several important advantages. First, it enforces validity by keeping outputs within the proper CVSS ranges and resolving contradictions between vector components using established domain rules. Second, it allows for knowledge injection, where symbolic knowledge fills in insights that might not be directly stated in the text. This means the model isn't just relying on simple keyword matches. Finally, it supports domain adaptation: new rules, such as those tailored for ICS or IoT systems, can be added without retraining the model, making it easy to quickly specialize for new scenarios [5].

However, merging the flexibility of LLMs and the precision of knowledge graphs is nontrivial. Designing an architecture that balances text understanding with constraint enforcement requires careful engineering. Additionally, maintaining a comprehensive and up-to-date knowledge graph is labor-intensive and time-consuming. In the next section, we present our proposed method, which addresses these challenges by combining the strengths of both approaches in a unified framework.

3 Proposed Method

The proposed methodology involves several sequential stages designed to transform domain-specific security knowledge into a format usable by large language models. This pipeline begins with the collection of metric definitions, proceeds through the construction of a knowledge graph, prompt engineering, model training, and result analysis, and concludes with iterative improvements based on real-world NVD data and error analysis. The following stages outline the proposed framework for building an automated CVSS severity prediction system.

3.1 Collection of Metric Definitions

As the initial step in the proposed framework, we systematically extract formal definitions for all CVSS-V3.x Base Metrics, including Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR), User Interaction (UI), Scope (S), Confidentiality (C), Integrity (I), and Availability (A). To ensure comprehensive and accurate coverage, this work draws on three main categories of sources. It uses official documentation published by the international CVSS authority, FIRST[1]; incorporates the National Vulnerability Database (NVD) along with its supporting guidelines; and also relies on strategic resources and best-practice documents from respected security organizations such as OWASP and SANS. From these sources, we extract precise descriptions of the terms and label values used in CVSS metrics. For example, the meaning of the "Network" label in the Attack Vector metric or the "High" label in Attack Complexity is formally defined. This step results in a robust and trustworthy knowledge base, which serves as the analytical foundation in the subsequent stages [6].

[1]https://www.first.org/

3.2 Design of the Knowledge Graph

In the second stage, after consolidating the formal definitions of CVSS Base Metrics, we transform this descriptive knowledge into a structured model in the form of a knowledge graph. This knowledge graph is designed as a directed, decision-oriented structure, which we treat as a self-contained and integrated system within our framework [18]. Each decision tree within the graph begins with a key question for the target metric (e.g., "Can the attack be performed remotely?"). From this root question, the graph branches into progressively detailed concepts and associated conditions, forming a logically coherent decision path. Nodes contain binary or multiple-choice questions, and each stopping point within the graph corresponds to a well-defined decision point that determines the label for that particular metric. In building the knowledge graph, we focused on practical and operational meanings behind each formal CVSS definition. For each metric, we closely compared official definitions from FIRST with real-world CVE examples from the NVD to extract the most relevant semantic features. Based on these insights, we designed a chain of targeted question–answer pairs to guide metric assignment in a step-by-step manner. From an applied perspective, the resulting decision tree serves as a transparent expert-assistance tool. Meaning that a human analyst can follow its branches by answering specific questions, ultimately reaching a final leaf node that represents the corresponding metric label. Additionally, the visual structure of the tree enhances interpretability by illustrating conceptual relationships clearly for end-users. To evaluate and refine the decision tree's performance, we applied it to a set of real-world CVEs from the NVD. During this process, we adjusted node content and branch sequences to ensure that the tree remained intuitive, accessible, and logically consistent throughout the decision-making process. The output of this phase is a comprehensive, well-documented, and updatable knowledge graph, an example of which is presented in Figure 1.

3.3 Evaluation of the Knowledge Graph

In the evaluation phase, the constructed knowledge tree was reviewed by a group of information security experts to assess its accuracy and effectiveness. Each node in the tree had to reflect a concept that is clearly identifiable in real-world security scenarios, and each leaf node had to unambiguously correspond to one of the official metric values. During this review, several aspects were evaluated, including the clarity of the questions, the logical flow of the branches, and whether all possible cases were thoroughly covered. They then tested the tree using real vulnerability samples, applying the decision paths to evaluate how closely the output aligned with the original CVSS labels for each metric. Facing the inconsistencies, the source of the issue, whether in the wording of a question or the structure of a branch, was identified and corrected accordingly. Expert knowledge was directly injected into the tree, reinforcing the logic and improving its reliability. The outcome of this stage is a validated decision tree, a structure that can be confidently used in automated scoring tools as well as training modules for human analysts.

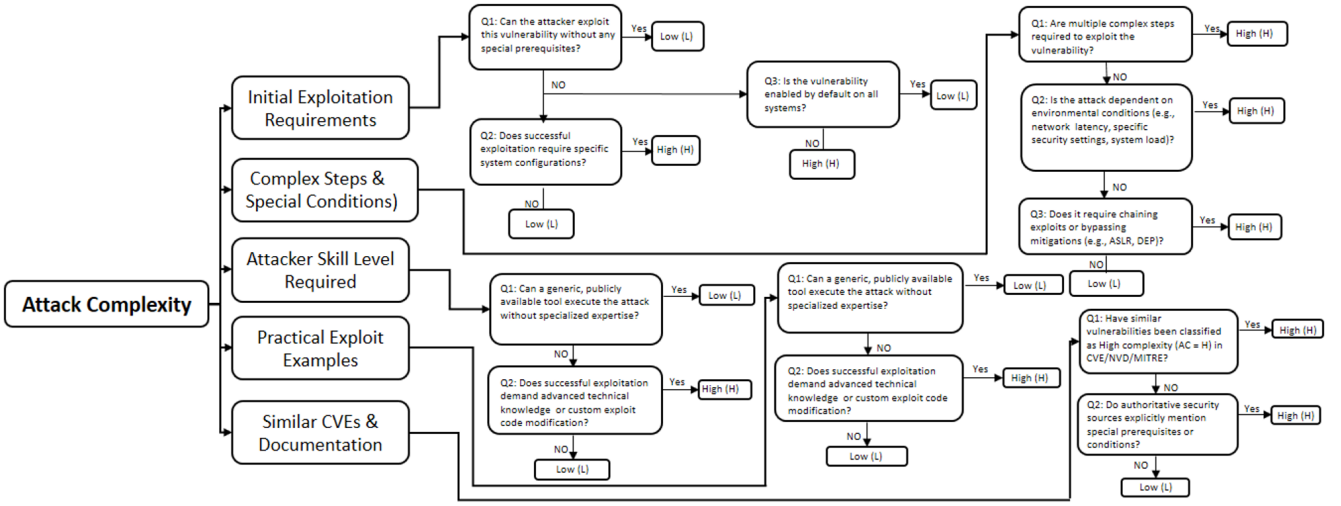


Figure 1: Example decision flow capturing expert reasoning for the Attack Complexity (AC) metric. The graph illustrates how different factors like initial exploitation requirements, attacker skill level, practical exploit tools, and known similar cases are combined to determine whether the complexity is classified as Low (L) or High (H).

3.4 Translating the Knowledge Tree into Prompts

This section introduces our five-step framework for transforming the structured knowledge tree into input prompts that are compatible with LLMs. As illustrated in Figure 2, the AV decision tree maps four official CVSS labels of Network (N), Adjacent (A), Local (L), and Physical (P) to a structured set of numbered questions. Each branch in the tree corresponds to a logical path through these questions, guiding the decision process clearly and in full alignment with the standard. By relying on this decision tree, we can systematically convert the reasoning path into a well-formed, unambiguous prompt that is easily understood and processed by an LLM. This translation step is essential, as it allows us to bridge the structured decision logic of symbolic systems with the natural language processing capabilities of modern LLMs, enabling the model to infer correct labels from complex CVE descriptions based on a guided reasoning template.

3.5 Prompt Conversion Framework

To transform the structured decision logic of the knowledge tree into a format usable by large language models (LLMs), we introduce a five-step pipeline. Each step systematically refines the decision rules into compact, interpretable, and LLM-compatible prompts. We illustrate this process using the Attack Vector (AV) parameter of the CVSS standard.

3.5.1 Step 1: Decision-Path Extraction. The first step involves converting the branching logic of the knowledge tree into a linear set of decision paths. Each path begins at the root of the tree and traverses a unique chain of questions leading to a final leaf node (i.e., an AV label such as Network (N), Adjacent (A), Local (L), or Physical (P)). To do this, we enumerate all question nodes and label leaves and perform a full traversal of the tree—typically using depth-first search (DFS). Every time a leaf is reached, the path taken through the decision graph is saved as a sequence of "[Question ID] → [Answer]" pairs. For example:

Q1:Yes → Q2:Yes ⇒ Network (N)
Q1:Yes → Q2:No → Q3:Yes ⇒ Adjacent (A)

After collecting these raw paths, we refine them further. When different branches logically lead to the same outcome (duplicate paths), we merge them to eliminate redundancy. We also remove invalid paths that do not end in a valid label because of conflicting or inconclusive answers. Importantly, this step assumes that the decision tree is complete and consistent, meaning that every possible sequence of answers must terminate in one of the four AV labels, and no cycles or dead ends exist. The result is a final list of unique decision paths—in this case, four paths corresponding to the four valid AV values. These paths form the core logical rules that will be transformed into natural language in the next steps, ensuring the LLM only processes meaningful, label-determining logic.

3.5.2 Step 2: Template-Based Prompting. In this step, the previously extracted question–answer sequences are rewritten as clear, imperative natural language statements, avoiding the need for the LLM to parse or reconstruct the original tree structure.

Instruction Template Construction. Each decision path is translated into a concise, plain-language instruction. For example, Q1 = Yes → Q2 = Yes ⇒ Network (N) becomes: *"If the attacker can exploit the vulnerability remotely, without prior access, via a network service (such as HTTP, SSH, RDP, or FTP), assign AV as Network (N)."* Similarly, for Adjacent (A), *"If the attacker must be on an adjacent network (e.g., Wi-Fi, Bluetooth, or LAN), assign AV as Adjacent (A)."* For Local (L) and Physical (P), *"If exploitation requires login or code execution on the target system, assign AV as Local (L),"* and *"If exploitation requires physical access, such as via USB or JTAG, assign AV as Physical (P)."*

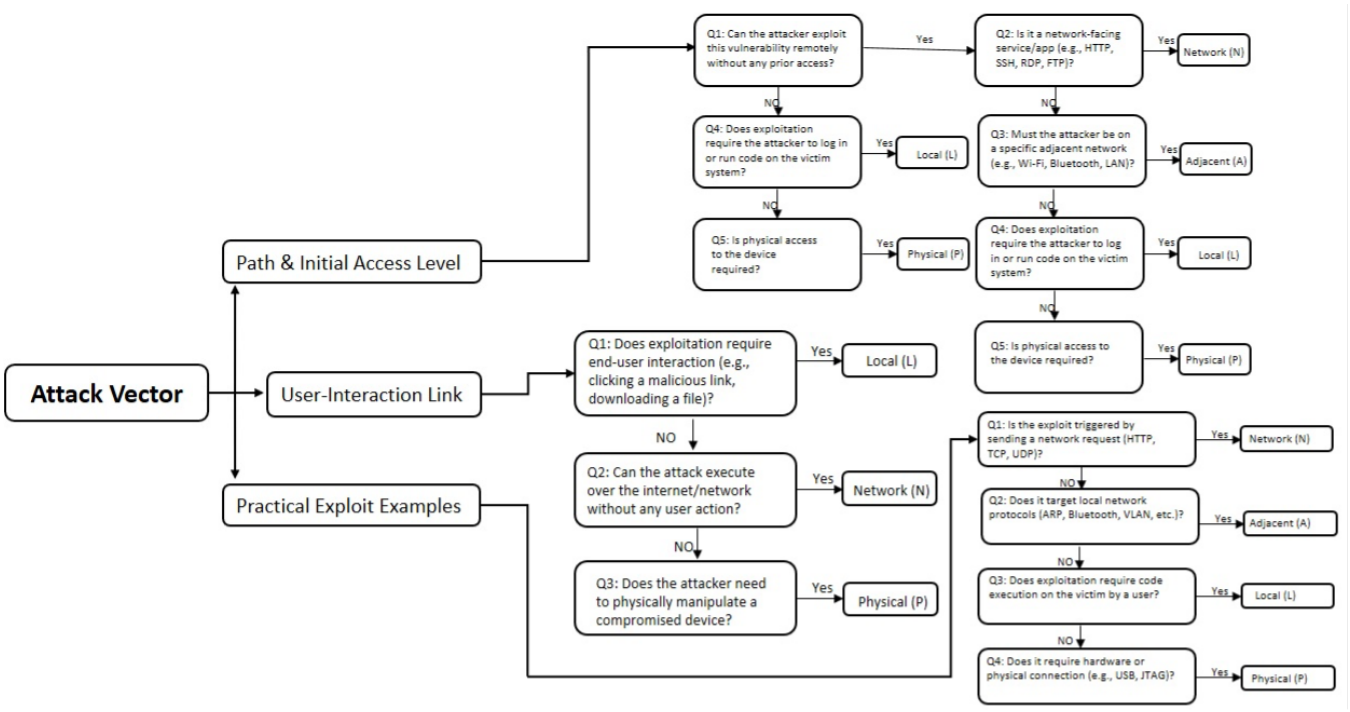


Figure 2: Example decision flow illustrating expert reasoning for determining the Attack Vector (AV) metric. The diagram shows how factors such as user interaction, network or physical access, and specific exploit scenarios are combined to classify the attack as Network (N), Adjacent (A), Local (L), or Physical (P).

Enrichment with Examples and Synonyms. Within each instruction, we add common keywords (such as protocols or technologies) in parentheses to help the model generalize better. For example, next to "HTTP," we might include "REST" or "HTTPS," while for "Physical," we could list "serial port" or "GPIO." This kind of low-cost token expansion greatly improves the model's ability to recognize a wider range of textual expressions.

Prompt Structure and Internal References. All instruction templates are organized into a structured block, each tagged with a unique identifier (for example, Rule-AV-N or Rule-AV-A). This setup allows the model—or a human analyst—to easily refer back to specific rules in explanations or debugging logs (e.g., "according to Rule-AV-L"). To establish context, we prepend a short header: *"You are a CVSS evaluator. Based on the following rules, determine the value of Attack Vector."* This primes the model to concentrate on the exact task and reduces the risk of drifting into unrelated CVSS metrics.

Token Budget Control and Redundancy Reduction. Once the initial prompt is drafted, we check its token count to make sure it fits within the model's input limits. If needed, we shorten or move parenthetical examples to a separate list at the end of the prompt. We also review overlapping terms—like "Remote"—to prevent ambiguity across rules. The end result is a precise, compact set of instructions, with each sentence tied directly to a specific decision path for Attack Vector. This removes the need for the model to figure out structural logic on its own and lets it zero in on semantic cues in vulnerability descriptions with high accuracy.

3.5.3 Step 3: Contextual Enrichment. In this step, we strengthen the prompt by adding a clear role definition, specifying the scope, and including practical examples. These additions help reduce misunderstandings and make the model's predictions more consistent. For role specification, we use a straightforward instruction like: *"You are now acting as a CVSS evaluator. Your task is to determine only the Attack Vector (AV)."* This keeps the model focused on a single metric and prevents it from drifting into unrelated areas. We also clarify the scope by explaining that inputs are plain-text vulnerability descriptions, which may include technical cybersecurity terms such as "HTTP GET" or "Kernel Driver." This helps the model recognize relevant details and avoid confusion. By setting these boundaries, we create a mental frame within which the model can operate more reliably, reducing ambiguity and improving robustness—essential for any real-world automated scoring system.

3.5.4 Step 4: Final Prompt Assembly. Once all components are prepared, we combine them into a structured, executable prompt that fits within token limits and supports reliable interpretation. The prompt follows a layered format that includes context (the model's role and scope), rules (the instruction blocks for each AV label), the input vulnerability description, and the output format specification. For example, the expected output might be: line 1 contains one of the labels N, A, L, or P, and line 2 shows the corresponding Rule-ID (e.g., Rule-AV-N). This simple structure makes it easier to automate

evaluations and integrate with downstream tools. The order of rules also matters; placing broader categories like Network (N) and Local (L) before narrower ones such as Adjacent (A) or Physical (P) helps guide the model's reasoning from general to specific, much like a human expert would. If the prompt exceeds typical token limits (about 4k–8k), we adjust by shortening examples or splitting the RULES into two chained prompts: the first provides context and half the rules, the second completes the rule set and adds the input. In the end, this carefully assembled prompt balances clarity and domain precision, making it equally effective for guiding LLMs and assisting human analysts who require a consistent, interpretable process.

3.5.5 Step 5: Prompt Execution and Evaluation. In the final step, the complete prompt—including the context, rules, input, and output format—is submitted to the LLM through an API or command-line interface. The specific vulnerability description is placed in the INPUT section, and the model is instructed to return a single label (N, A, L, or P) along with the related rule identifier. For instance, given the input "A REST service is running on port 443 and no user interaction is required," the expected output would be simply: *N* on the first line and *Rule-AV-N* on the second, indicating that the model correctly followed the decision path Q1 → Yes → Q2 → Yes. Once the output is received, an evaluation layer compares the predicted label to the ground-truth label from the dataset. Metrics such as accuracy, confusion matrices, and F1-scores are then computed to quantify the model's performance in each test round.

This five-step pipeline illustrates how structured, graph-based expert reasoning can be transformed into a format that LLMs can process effectively. The Attack Vector example shows how this method preserves decision accuracy while enabling text-driven flexibility, making it straightforward to adapt the system to new CVSS versions or evolving threat scenarios.

4 Experimental Results

To explore whether injecting knowledge distilled from the CVSS concept graph—via a carefully designed prompt—can meaningfully improve large language models' accuracy when labeling the eight base CVSS metrics (AV, AC, PR, UI, S, C, I, A), we evaluated five models: GPT-4o, GPT-4o-Mini, GPT-4.1-Nano, GPT-4.1-Mini, and DeepSeek. For each metric, we prepared a balanced set of 100 CVE descriptions. Every description was supplied to each model twice: once with the engineered prompt and once with a simpler baseline prompt. The combined results across all metrics and models are summarized in Table 1.

Among these models, GPT-4o achieved the highest overall accuracy, benefiting from its unified multimodal core and large context window, though at the highest computational cost. GPT-4o-Mini maintained nearly the same performance while halving runtime and expense. The GPT-4.1 series expanded the context window dramatically and offered strong trade-offs between speed and precision. Meanwhile, DeepSeek provided competitive results on key metrics, despite being open-source with a smaller context size. Overall, these results highlight that structuring domain knowledge into prompts—through explicit rules and examples—consistently improves the models' ability to assign accurate CVSS metrics, especially for more challenging parameters like Scope, Confidentiality, and Integrity.

Table 1: Accuracy (%) for each CVSS metric comparing engineered vs. simple prompts, pairing models and showing overall averages

GPT-4o			GPT-4o-Mini		
Metric	Eng	Sim	Metric	Eng	Sim
AV	90	82	AV	87	64
AC	83	60	AC	70	52
PR	86	74	PR	87	52
UI	90	87	UI	78	56
S	70	42	S	58	48
C	76	39	C	78	62
I	74	37	I	64	43
A	92	54	A	93	74
GPT-4.1-Nano			GPT-4.1-Mini		
Metric	Eng	Sim	Metric	Eng	Sim
AV	82	39	AV	85	58
AC	87	53	AC	88	67
PR	78	45	PR	86	73
UI	74	65	UI	87	44
S	64	48	S	64	55
C	53	40	C	74	72
I	69	33	I	71	68
A	85	64	A	88	85
DeepSeek			Overall Average		
Metric	Eng	Sim	Metric	Eng	Sim
AV	87	54	AV	86	59
AC	87	49	AC	83	56
PR	79	40	PR	83	57
UI	90	62	UI	84	63
S	65	63	S	64	51
C	71	33	C	70	49
I	55	34	I	67	43
A	92	47	A	90	65

According to our initial observations, vanilla GPT-4o achieved an average accuracy of 59.4%, whereas the same model reached 82.6% when equipped with the engineered prompt—a striking improvement that highlights the value of knowledge support. GPT-4o-Mini, its compressed variant, increased from 56.4% to 76.9%. Notably, DeepSeek, which started with the lowest baseline accuracy (47.8%), jumped to 78.3% after receiving the knowledge-oriented prompt. This distribution of results confirms not only the overall effectiveness of prompt engineering but also reveals that lighter or more specialized models tend to benefit the most in relative terms.

To quantify the benefit of injecting structured knowledge, we introduce the *Model Improvement Rate* (IR), a normalized metric that captures how much the engineered prompt boosts a model's accuracy relative to its baseline. This emphasizes relative gains, making it easier to compare models with different starting points. Formally, it is defined as:

$$\text{IR} = \frac{\Delta\text{Acc}}{\text{Acc}_{\text{simple}}} = \frac{\text{Acc}_{\text{eng}} - \text{Acc}_{\text{simple}}}{\text{Acc}_{\text{simple}}} \quad (1)$$

where Acc_{eng} and $\text{Acc}_{\text{simple}}$ denote the mean accuracies with the engineered and simple prompts, respectively.

To also factor in operational costs, we define a *Model Efficiency* (ME) score that normalizes the improvement rate by the cost of processing 1 million input tokens for each model:

$$\text{ME} = \frac{\text{IR}}{\text{Cost}} \quad (2)$$

where Cost is the relative cost of the model. This yields a scale-independent indicator of cost-effectiveness.

The combined results, shown in Table 2, reveal that while GPT-4o achieved notable absolute accuracy improvements, it posted the lowest efficiency. In contrast, GPT-4.1-Nano reached roughly 529 units, demonstrating the highest improvement relative to cost. DeepSeek achieved the highest raw improvement rate at nearly 64%, highlighting the substantial benefit of the engineered prompt even for smaller open-source models.

Table 2: Summary of model performance: mean accuracies, improvement rates, and cost-based efficiency

Model	Simple (%)	Engineered (%)	IR (%)	ME
GPT-4o	59.4	82.6	39.16	15.66
GPT-4o-Mini	56.4	76.9	36.36	242.42
GPT-4.1-Nano	53.0	81.1	52.97	529.72
GPT-4.1-Mini	64.5	79.5	23.18	57.95
DeepSeek	47.8	78.3	63.87	236.57

These findings demonstrate that combining symbolic knowledge with the reasoning capabilities of language models can produce an average accuracy improvement of about 23% without any parameter updates—a significant advantage, especially for organizations that lack the computational resources needed for fine-tuning. This represents an important step toward more practical, cost-effective deployment of such systems in real-world cybersecurity contexts.

5 Conclusion

This work explored how combining large language models with structured security knowledge can significantly improve the automatic assessment of software vulnerabilities. By guiding LLMs with explicit rules derived from CVSS concept graphs, we addressed major challenges such as the growing number of new vulnerabilities, the often ambiguous language in CVE descriptions, and the subjectivity of manual scoring.

Our experiments across multiple state-of-the-art models demonstrated clear benefits. For instance, GPT-4o's average accuracy rose from 59% to over 82%, while lighter models like DeepSeek saw relative gains exceeding 60%, all without any parameter fine-tuning. This shows that adding even lightweight, structured knowledge can dramatically boost performance.

Beyond accuracy, this hybrid approach also improves interpretability. By anchoring predictions in clear, rule-based reasoning, we move away from traditional black-box methods, making the decision process easier to trust and audit. This balance of high performance with transparency is especially valuable in real-world security contexts.

Overall, the proposed system offers a practical way to automate vulnerability scoring at scale. It helps security teams quickly and consistently prioritize risks, saving valuable analyst time and supporting faster, more informed decisions.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

References

[1] Artur Balsam, Maciej Nowak, Michał Walkowski, Jacek Oko, and Sławomir Sujecki. 2023. Analysis of CVSS vulnerability base scores in the context of exploits' availability. In *2023 23rd International Conference on Transparent Optical Networks (ICTON)*. IEEE, 1–4.

[2] Alicia Biju, Vishnupriya Ramesh, and Vijay K Madisetti. 2024. Security vulnerability analyses of large language models (llms) through extension of the common vulnerability scoring system (cvss) framework. *Journal of Software Engineering and Applications* 17, 5 (2024), 340–358.

[3] Joana Cabral Costa, Tiago Roxo, João BF Sequeiros, Hugo Proenca, and Pedro RM Inacio. 2022. Predicting CVSS metric via description interpretation. *IEEE Access* 10 (2022), 59125–59134.

[4] Lucas Senos Coutinho, Daniel Menasche, Lucas Miranda, Enrico Lovat, Srivastava Gaurav Kumar, Abhishek Ramchandran, Anton Kocheturov, and Tobias Limmer. 2024. How context impacts vulnerability severity: An analysis of product-specific CVSS scores. In *Proceedings of the 13th Latin-American Symposium on Dependable and Secure Computing*. 17–27.

[5] Xueying Du, Geng Zheng, Kaixin Wang, Jiayi Feng, Wentai Deng, Mingwei Liu, Bihuan Chen, Xin Peng, Tao Ma, and Yiling Lou. 2024. Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag. *arXiv preprint arXiv:2406.11147* (2024).

[6] BE FIRST. 2015. Common Vulnerability Scoring System v3. 1: Specification Document.

[7] Sergio Flesca, Domenico Mandaglio, Francesco Scala, and Andrea Tagarelli. 2024. A meta-active learning approach exploiting instance importance. *Expert Systems with Applications* 247 (2024), 123320. doi:10.1016/j.eswa.2024.123320

[8] Dimcho Georgiev. 2024. *Modern Vulnerability Management apart from CVSS*. Master's thesis. University of Applied Sciences Technikum Wien, Wien, Austria. Thesis submitted in fulfillment of the requirements for the degree of Master of Science in Engineering in IT-Security. Supervisors: Stefan Schubert, BSc., MSc., and Mag. David Warren. Student Number: 2210303040.

[9] Nourhan Ibrahim, Samar Aboulela, Ahmed Ibrahim, and Rasha Kashef. 2024. A survey on augmenting knowledge graphs (KGs) with large language models (LLMs): models, evaluation metrics, benchmarks, and challenges. *Discover Artificial Intelligence* 4, 1 (2024), 76.

[10] Domenico Mandaglio, Andrea Tagarelli, and Francesco Gullo. 2020. In and Out: Optimizing Overall Interaction in Probabilistic Graphs under Clustering Constraints. In *Proc. KDD Conf.* 1371–1381. doi:10.1145/3394486.3403190

[11] Domenico Mandaglio, Andrea Tagarelli, and Francesco Gullo. 2021. Correlation Clustering with Global Weight Bounds. In *Proc. ECML PKDD Conf.*, Vol. 12976. 499–515. doi:10.1007/978-3-030-86520-7_31

[12] Francesco Marchiori, Denis Donadel, and Mauro Conti. 2025. Can LLMs Classify CVEs? Investigating LLMs Capabilities in Computing CVSS Vectors. *arXiv preprint arXiv:2504.10713* (2025).

[13] Ali Mirarab, Seyedeh Leili Mirtaheri, and Seyed Amir Asghari. 2019. Value creation with big data analytics for enterprises: a survey. *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 17, 6 (2019), 2790–2802.

[14] Seyedeh Leili Mirtaheri and Andrea Pugliese. 2024. Leveraging Generative AI to Enhance Automated Vulnerability Scoring. In *2024 IEEE Conference on Dependable, Autonomic and Secure Computing (DASC)*. IEEE, 57–64.

[15] Seyedeh Leili Mirtaheri, Andrea Pugliese, Narges Movahed, and Reza Shahbazian. 2025. A comparative analysis on using GPT and BERT for automated vulnerability scoring. *Intelligent Systems with Applications* 26 (2025), 200515.

[16] Seyedeh Leili Mirtaheri, Andrea Pugliese, Narges Movahedkor, and AmirHossein Majd. 2024. Advanced automated vulnerability scoring: Improving performance with a fine-tuned BERT-CNN model. In *2024 11th International Symposium on Telecommunications (IST)*. IEEE, 109–113.

[17] Abhishek Sharma, Sangeeta Sabharwal, and Sushama Nagpal. 2023. A hybrid scoring system for prioritization of software vulnerabilities. *Computers & Security* 129 (2023), 103256.
[18] Leslie F Sikos. 2023. Cybersecurity knowledge graphs. *Knowledge and Information Systems* 65, 9 (2023), 3511–3531.
[19] Romilla Syed. 2020. Cybersecurity vulnerability management: A conceptual ontology and cyber intelligence alert system. *Information & Management* 57, 6 (2020), 103334.
[20] Dinesh T Vasireddy, Dakota S Dale, and Qinghua Li. 2023. Cvss base score prediction using an optimized machine learning scheme. In *2023 Resilience Week (RWS)*. IEEE, 1–6.
[21] Ju An Wang and Minzhe Guo. 2009. OVM: an ontology for vulnerability management. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. 1–4.
[22] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* 4, 2 (2024), 100211.
[23] Lorenzo Zangari, Domenico Mandaglio, and Andrea Tagarelli. 2024. Link Prediction on Multilayer Networks through Learning of Within-Layer and Across-Layer Node-Pair Structural Features and Node Embedding Similarity. In *Proc. WWW Conf.* 924–935. doi:10.1145/3589334.3645646
[24] Jie Zhang, Haoyu Bu, Hui Wen, Yongji Liu, Haiqiang Fei, Rongrong Xi, Lun Li, Yun Yang, Hongsong Zhu, and Dan Meng. 2025. When llms meet cybersecurity: A systematic literature review. *Cybersecurity* 8, 1 (2025), 55.
[25] Siqi Zhang, Minjie Cai, Mengyuan Zhang, Lianying Zhao, and Xavier de Carné de Carnavalet. 2023. The flaw within: Identifying CVSS score discrepancies in the NVD. In *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 185–192.
[26] Ziyao Zhang, Chong Wang, Yanlin Wang, Ensheng Shi, Yuchi Ma, Wanjun Zhong, Jiachi Chen, Mingzhi Mao, and Zibin Zheng. 2025. Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 481–503.
[27] Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. 2024. Large language model for vulnerability detection and repair: literature review and the road ahead (2024). *arXiv preprint arXiv:2404.02525* (2024).

A Base Metrics of CVSS

Attack Vector (AV). Describes how close an attacker needs to be to exploit a vulnerability. The more remote the attack, the higher the risk.

- Network (N): Exploitable remotely over a network like the Internet.
- Adjacent (A): Requires access to the same local or adjacent network.
- Local (L): Needs local system access, such as executing code or logging in.
- Physical (P): Requires physical interaction with the device.

Attack Complexity (AC). Captures conditions outside the attacker's control needed for the exploit to succeed.

- Low (L): No special conditions; easy to exploit.
- High (H): Requires uncommon or challenging circumstances.

Privileges Required (PR). Indicates the level of access an attacker must already have.

- None (N): No prior access needed.
- Low (L): Needs user-level privileges.
- High (H): Requires administrative or root-level access.

User Interaction (UI). Shows whether the exploit depends on another user's actions.

- None (N): No user involvement needed.
- Required (R): Needs someone to click a link, open a file, etc.

Scope (S). Indicates if the impact goes beyond the vulnerable component's security boundary.

- Unchanged (U): Effects stay within the same scope.
- Changed (C): Can affect other components or systems.

Confidentiality (C). Measures potential unauthorized data exposure.

- None (N): No data disclosure.
- Low (L): Some limited information leaks.
- High (H): Complete or serious compromise of sensitive data.

Integrity (I). Assesses how much data or system trustworthiness could be compromised.

- None (N): No impact on integrity.
- Low (L): Minor data modifications.
- High (H): Significant data corruption or loss.

Availability (A). Captures the extent of disruption to system services or resources.

- None (N): No effect on availability.
- Low (L): Minor or temporary disruptions.
- High (H): Severe or complete service outages.