Evaluating LLM Use In Identifying OSS Packages Linked to Cybersecurity Vulnerabilities for HAVOSS

Gerald Rigdon
Fellow, Software Engineering
Boston Scientific Corporation
St. Paul, USA
gerald.rigdon@bsci.com

***Abstract*—Building on the more general Capability Maturity Model (CMM) for software, the Handling Vulnerabilities In OSS (HAVOSS) model has emerged as an effective complement with a more detailed focus on handling Common Vulnerabilities and Exposures (CVEs) in third-party software packages used in embedded systems. As such, this HAVOSS model includes six capability areas and twenty-one practices. Of particular interest is the Identification and Monitoring of Sources area which presents significant challenges given the frequency of new vulnerability discoveries and disclosures. To address this HAVOSS area, this paper explores the use of Large Language Models (LLMs) within an agentic operational framework that employs multiple Retrieval Augmented Generation (RAG) methods to improve the accuracy of retrieved data based on a search input. Consequently, it explores first, whether a CVE identification query is sufficient to retrieve the correct record with software package identification; and second, whether a software package information based query is sufficient to retrieve the correct record with the CVE identifier. Hence, a security professional armed with a CVE identifier should be able to reliably extract the affected software packages and likewise armed with a list of software package names be able to connect to the correct CVEs. Using a custom developed CyberVulModelEval software application, this research presents the outcomes of several experiments that provide further insight into whether foundation LLMs can accurately link vulnerability descriptive elements with deployed open source software packages.**

***Index Terms*—CVE, Agentic LLM, Self-RAG, HAVOSS**

I. Introduction

Common Vulnerabilities and Exposures (CVEs) tracked in the National Vulnerability Database (NVD) [1] are an integral part of cybersecurity threat intelligence and vulnerability management. CVEs are a dictionary or a glossary of vulnerabilities that are compiled and maintained in a structured format to identify, describe, and provide references for use by cybersecurity professionals. CVEs are published to publicly notify industry participants of ways in which their software may be vulnerable to cybersecurity threats and exploits. To capture the scale of this effort, a recent Exploit Prediction Scoring System (EPSS) Data and Performance Study [2] stated that the total number of CVEs is nearing a quarter million with over 30,000 reported vulnerabilities in 2024. Moreover, the possible one to many mapping of CVEs to affected software packages results in a large repository of issues and concerns that have to be sifted through to enable cybersecurity professionals to identify vulnerabilities that exist in products under their supervision and respond accordingly.

Per Druckman [3] a 2024 Synopsis report showed that 96 percent of commercial code bases they sampled contained open source components. This corroborates the 2022 Linux Foundation study [4] that stated 70-90 percent of any given software code base contains open source components. Given the ubiquitous use of GitHub for managing open source code, entities such as the GitHub Advisory [5] was created to notify developers of vulnerabilities that are associated with affected software packages within the GitHub ecosystem. In terms of logistics, the workflow essentially consists of reported vulnerabilities that flow into the GitHub Advisory database from the NVD feed or from the GitHub internal npm (package manager) security team which are transformed into security advisories. As of 2022 [6], broader community participation and involvement has also been integrated into the process.

Given the importance of vulnerability disclosures and the linkage to software packages, especially open source, the security practitioner is confronted with the challenge of how to best consume the vast repository of data that is continually changing. To that end, the Handling Vulnerabilities In OSS (HAVOSS) model [7] includes the Identification and Monitoring of Sources as one of the key capability areas for evaluating cybersecurity process maturity and emphasizes the need for a well-defined and efficient process for monitoring the sources of vulnerability information. This includes determining the external sources to use for identifying new vulnerabilities and subsequently monitoring those sources for vulnerability identification and association with particular software of interest.

While conventional technology tools (i.e., web searches, database queries, commercial software applications, etc.) may be the most effective way to digest the GitHub Advisory database information at the present time, there have been other efforts exploring the use of Large Language Models (LLMs) [8] [9] to better link reported vulnerabilities with specific software package names and versions. However, those cited endeavors involve the use of fine tuning and training of

LLMs to actually assist in more accurate security advisory database entries. In contrast, the research presented in this paper focuses on the use of LLM core or base training in tandem with Retrieval Augmented Generation (RAG) strategies using the CyberVulModelEval custom software application to assess the viability of LLMs as compared to conventional tooling approaches. Thus, the central research questions to be answered are: (1) Given either CVE information or software package descriptions, can an LLM retrieve a correct GitHub advisory record that connects the missing counterpart? (2) Can a Self-RAG related approach overcome the limitations of Standard RAG? (3) What retrieval benefits can be achieved when executing tasks within an AI agentic framework?

The results of this research may help to further assess the state of RAG within a cybersecurity framework where a company may be contemplating the use of LLMs to help monitor CVE disclosures and their impact on software packages used in their deployed product(s). The primary advantage of this approach would be leveraging the power of LLMs to perform tasks that otherwise require additional technology tools, which often require long-term maintenance, personnel support, and expenditures.

II. LITERATURE REVIEW

In [10], a systematic review of LLM use in cybersecurity includes the consideration of over 300 papers encompassing 25 different LLMs and categorizes them as follows:

- Threat Intelligence
- Vulnerability Detection
- Malware Detection
- Anomaly Detection
- Fuzz
- Program Repair
- LLM-Assisted Attacks
- (In)secure Code Generation
- Others

While this body of work is comprehensive, a close consideration of the categories reveals that much of the focus is on the evaluation of the software itself, namely, detection, repair, and concern over LLM generated code. Such categories align more with the Vulnerability Evaluation capability area of HAVOSS. Threat Intelligence stands out as a differentiated category where the research centers around ways to organize and analyze information from a large number of intelligence related documents. This category aligns more with the Identification and Monitoring of Sources of HAVOSS capability which is the focus of the research presented in this paper.

Of particular relevance is the research presented in [8] which addresses the task of identifying software packages affected by vulnerabilities with the use of LLMs by introducing the VulLibGen framework which employs Supervised Fine Tuning (SFT) in tandem with RAG and local search. In contrast to rank/retrieval methods, VulLibGen generates the names of affected software packages. The success of VulLibGen was captured in the research findings which demonstrated VulLibGen had an average accuracy of 0.806 for identifying

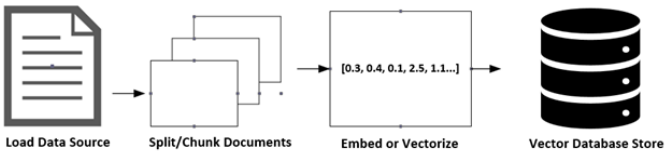


Fig. 1. Standard RAG - Storage

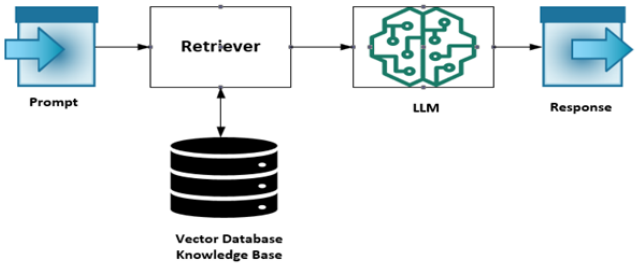


Fig. 2. Standard RAG - Retrieval

vulnerable packages in the four most popular ecosystems in GitHub Advisory (Java, Java Script, Python, Go) while the best average accuracy in previous work was 0.721. Although RAG was complementary to the primary SFT approach it also employed a re-ranking strategy using a BERT base model that was also fine-tuned on the same training data used for the core SFT VulLibGen component.

III. RESEARCH METHODOLOGY APPROACH

Rather than incurring the cost and overhead associated with SFT methods, the proposed approach in this paper revolves primarily around the evaluation of RAG using ideas from the latest research on ways to improve it. At its core, standard RAG as depicted in Fig. 1 involves taking a data source that exists outside the LLM base training and splitting or breaking it into several chunks, vectorizing it in the form of embeddings (a compressed semantic representation understood by LLMs) and storing the transformed data in a vector database.

When queried, as shown in Fig. 2, information is pulled from this additional knowledge base using vector mathematics to retrieve data that are semantically close the query. Next, the retrieved documents are subsequently presented to the LLM for additional context in answering the query. These queries or inputs are often supplied by the user prompt interface but can be carefully crafted when the questions to be answered are known in advance.

However, there are several known limitations of standard RAG. The Deakin University research in [11] presents seven failure points drawn from three case studies and an experiment involving 15,000 documents and 1,000 question and answer pairs. The study results demonstrate how standard RAG can fail and it provides advice for practitioners. Consequently, several papers have emerged that attempt to address inadequacies of standard RAG such as Context Retrieval [12]. This method, recommended by Anthropic, emphasizes the importance of re-

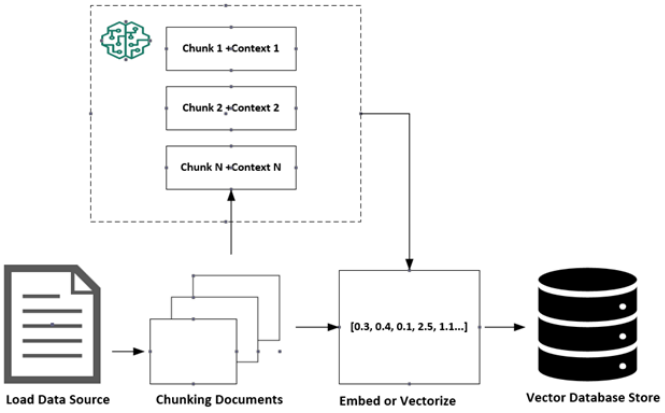


Fig. 3. Contextual RAG

ranking retrieved documents and involves adding an additional layer of information during the vectorization or embedding process. As shown in Fig. 3, additional context or a summary provided by an LLM is added to each chunk before storage in the vector database.

Moreover, additional agentic focused strategies have been forwarded to address standard RAG issues as well. ReAct [13] is a proposal to synergize LLM reasoning for general task solving situations using helpers or agents. Self-RAG related approaches such as Reflection [14], and Fusion [15], as complementary to ReAct, are techniques involving grading LLM responses with an LLM, re-writing queries with an LLM, and re-ranking retrieved documents using a cross-encoder. Even further, Correction [16] adds an additional sequence where the original query is rewritten especially for web search with the help of an LLM and the internet provides even further context when other methods have achieved sub-optimal results. Using a hybrid-mix of all these previously mentioned strategies, the RAG-Blend framework used, and research conducted in this paper, is captured in Fig. 4.

This RAG-Blend framework is actuated in a custom developed CyberVulModelEval software application and leverages the GPT4o LLM, an Open AI text-embedding-3-small embedding model [25], a Hugging Face paraphrase-multilingual-MiniLM-L12-v2 embedding model [26], a Hugging Face cross-encoder for document re-ranking [26], a Chroma vector database [27], the LangGraph agentic toolchain [28], Tavily web search [29], and the Ragas metrics library [30]. The addition of a cross-encoder [17] adds some retrieval diversity given the bi-encoder pattern is typically followed. In the cross-encoding paradigm, instead of executing a similarity calculation between two separate embedding inputs (a query and a retrieved document), both are combined and evaluated together as a single embedding where the output is generated in terms of a classification score.

IV. DATASET DESCRIPTION

For the VulLibGen research project described in [8], the GitHub Advisory [5] was used to create a dataset that provides

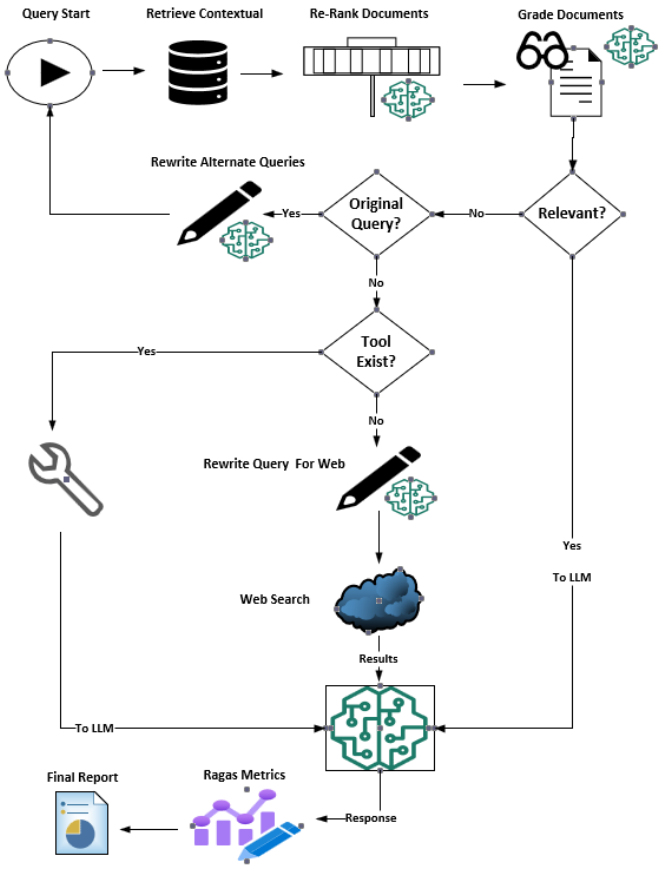


Fig. 4. Rag-Blend Framework - Agentic

records of vulnerabilities and the software packages they impact. This dataset was manually verified by security experts and comprised of vulnerability identifiers, textual descriptions detailing the nature, cause, and potential impact of the security issues, and the affected software package names. VulLibGen used this dataset to fine-tune the Vicuna-13B LLM through supervised learning to optimize its performance in generating accurate package names from the given vulnerability descriptions.

Using the publicly accessible VulLibGen [8] test JSON dataset [18] as an input for this paper, a custom Python script shown in Fig. 5 was developed to extract the information necessary to recreate a new text file dataset containing 565 records where each record is organized as three simple fields: CVE ID, CVE Desc, and CVE Software Package. A snippet, three records, of the transformed test dataset used in the research for this paper is shown in Fig. 6.

Creating a custom software search tool (see the tool flow referenced in Fig. 4 but not used during this research) to parse this test dataset is a common software engineering task that would likely yield accurate results given a specified CVE ID input that identically matches the content in the dataset. Partial CVE IDs, descriptions, and software package names would require further search tool sophistication, but the results would

```python
def parseAdvisoryDataSet(createfiles):
  record_counter = 0
  json_file = 'test.json'

  with open(json_file) as f:

   json_data = json.load(f)

   for item in json_data:

       cve_id = item.get('cve_id')
       cve_desc = item.get('desc')
       cve_label = item.get('labels')


       if (cve_label != []):
         if ("CVE-" in str(cve_id)):

           record_counter += 1
           print (record_counter)

           cve_id_str = f"CVE ID: {cve_id}\n"
           cve_desc_str = f"CVE Desc: {cve_desc}\n"
           cve_label_str = f"CVE Software Package: {cve_label}\n"

           print(cve_id_str)
           print(cve_desc_str)
           print(cve_label_str)

           print('-' * 80)

           if (createfiles):

             with open("goldenfull.txt", "a", encoding='utf-8') as goldenfilefull:
               goldenfilefull.write(cve_id_str)
               goldenfilefull.write(cve_desc_str)
               goldenfilefull.write(cve_label_str)

               goldenfilefull.write("\n")
```

Fig. 5. Python Code - Test Set Conversion

```
CVE ID: CVE-2020-2307
CVE Desc: Jenkins Kubernetes Plugin 1.27.3 and earlier allows low-
privilege users to access possibly sensitive Jenkins controller
environment variables.
CVE Software Package: ['maven:org.csanchez.jenkins.plugins:kubernetes']

CVE ID: CVE-2007-5614
CVE Desc: Mortbay Jetty before 6.1.6rc1 does not properly handle "certain
quote sequences" in HTML cookie parameters, which allows remote attackers
to hijack browser sessions via unspecified vectors.
CVE Software Package: ['maven:org.mortbay.jetty:jetty']

CVE ID: CVE-2011-1088
CVE Desc: Apache Tomcat 7.x before 7.0.10 does not follow ServletSecurity
annotations, which allows remote attackers to bypass intended access
restrictions via HTTP requests to a web application.
CVE Software Package: ['maven:org.apache.tomcat.embed:tomcat-embed-core',
'maven:org.apache.tomcat:tomcat-catalina']
```

Fig. 6. Test Dataset Snippet

also likely result in acceptable tool performance considering the long history of effective software parsing applications. Therefore, with the demonstrated power of LLMs, one's intuition, having limited experience with RAG, could lead to the hypothesis that LLMs might rival custom-developed parsers and search tools. The results of investigating this hypothesis further in the context of the disclosed test dataset are presented in the next section.

V. RESULTS AND DISCUSSIONS

To assess the results of the RAG-Blend framework instantiated as the CyberVulModelEval software application, the complete experiment test set consists of 30 samples where each sample input pair comprises a query and the ground truth answer. The samples, which are random and derived from the test dataset, are diversified, such that 15 queries present a specified CVE ID in search of one or more software packages and 15 queries that specify a software package in search of one or more CVE IDs. Moreover, each 30 sample pair covers two experiments running with a common configuration totaling eight experiments and four distinct configurations. The Ragas metrics framework [19] [20] is used to evaluate the LLM responses. The key inputs to Ragas are 1) the queries; 2) the ground truth expected responses; 3) the retrieved context from the vector database; 4) the actual LLM responses.

Hence, when executed, the CyberVulModelEval application packages the results from the 30 sample LLM test execution and invokes Ragas with the necessary inputs. Ragas then generates output [20] with the following key metrics:

Context Precision – A value that ranges from 0 to 1 (less to more precise) and used to measure the proportion of relevant chunks in the retrieved context and calculated as the mean of Precision@k for each chunk in the retrieved context.

While Precision@k is calculated mathematically as:

$$\frac{TruePositives@k}{TruePositives@k + FalsePositives@k}$$

Where K is the total number of chunks in the retrieved contexts and $Vk \in \{0, 1\}$ is the relevance indicator at rank k.

Context Precision@K is calculated mathematically as:

$$\frac{\sum_{k=1}^{K} (Precision@k * Vk)}{TotalNumberRelevantItemsInTopKResults}$$

Faithfulness – A value that ranges from 0 to 1 (complete inconsistency to complete consistency) which measures the consistency of the LLM response with respect to the retrieved context. This is calculated mathematically as:

$$\frac{NumClaimsInLLMResponseFromContext}{TotalNumberClaimsInLLMResponse}$$

Answer Relevancy – A value that ranges from 0 to 1 (no relevance to perfect relevance) that computes a cosine similarity based on the query, the LLM response, and the retrieved context. This metric does not judge factuality based on the ground truth but penalizes incomplete or redundant LLM responses. Hence, an LLM response is considered more relevant if it directly and appropriately addresses the query. This is calculated mathematically as:

$$AnswerRelevancy = \frac{1}{N}\sum_{i=1}^{N} CosineSimilar(Egi, Eo)$$

$$AnswerRelevancy = \frac{1}{N}\sum_{i=1}^{N}\frac{Egi * Eo}{||Egi||Eo||}$$

Where:
Egi: Embedding of the ith generated answer.
Eo: Embedding of the user input.
N: Number of generated answers.

Context Recall – A value that ranges from 0 to 1 (no recall to total recall) that is computed using the retrieved context and the ground truth reference. This is calculated mathematically as:

$$\frac{NumberRelevantContextsRetrieved}{TotalNumGroundTruthReferenceContexts}$$

The tables that follow capture the actual output from the Ragas metrics framework [20] while running the CyberVulModelEval application for the test dataset. The number of web searches is displayed, along with a list of 'yes' or 'no' indications which are indexed to the query requiring web search. Each table result is associated with configuration settings that provide details on how each experiment was executed, where:

Configuration Settings:

- CVE ID – Retrieval of software packages associated with a CVE identifier input.
- CVE SP – Retrieval of CVE IDs associated with a specified software package input.
- GPT4o – Large Language Model.
- OAI EM – Open AI Embedding Model.
- HF EM – Hugging Face Embedding Model.
- CHRDB – Chroma Vector Database.
- MMR – A RAG algorithm based on Maximal Marginal Relevance [21] or the retrieved documents that are most similar to the input query.
- SST – A RAG algorithm based on cosine similarity that returns only retrieved documents that exceed a specified Similarity Score Threshold [22].
- CE – A RAG algorithm known as a Cross Encoder [17] that combines the query and context as a single embedding for re-ranking the most relevant retrieved documents.
- TWS – The Tavily Web Search. Per the RAG-Blend framework in Fig. 4, a web search is the last resort when all other retrieval mechanisms have failed.

The Experiments

For the first three experiments, the queries led with the CVE ID.

Experiment 1 (15 queries)
Configuration: CVE ID, GPT4o, OAI EM, CHRDB, MMR, CE, TWS
The results of this experiment as captured in Fig. 7 reveal sub-optimal results. For some queries, Context Precision and

	context_precision	faithfulness	answer_relevancy	context_recall
0	0.000000	1.000000	0.875596	0.250000
1	1.000000	1.000000	0.868683	1.000000
2	1.000000	1.000000	0.820572	1.000000
3	0.000000	0.250000	0.864624	0.000000
4	0.000000	1.000000	0.851633	0.000000
5	1.000000	1.000000	0.866734	1.000000
6	0.000000	1.000000	0.877363	0.333333
7	0.000000	1.000000	0.894954	0.750000
8	0.500000	1.000000	0.904966	0.750000
9	0.916667	1.000000	0.868023	1.000000
10	1.000000	0.714286	0.885249	1.000000
11	1.000000	1.000000	0.861360	1.000000
12	0.000000	0.800000	0.873980	0.000000
13	0.000000	1.000000	0.878762	0.500000
14	0.000000	1.000000	0.904704	0.750000

Number of Web Searches: 7

['No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No']

Fig. 7. Experiment 1 Metrics Results

	context_precision	faithfulness	answer_relevancy	context_recall
0	0.000000	0.666667	0.902138	0.166667
1	0.333333	1.000000	0.881836	0.000000
2	1.000000	1.000000	0.878278	1.000000
3	0.000000	1.000000	0.895187	0.000000
4	1.000000	0.000000	0.884948	0.000000
5	0.000000	1.000000	0.898131	0.000000
6	0.000000	1.000000	0.850412	0.333333
7	0.000000	1.000000	0.873702	0.200000
8	0.000000	0.833333	0.857075	0.400000
9	0.000000	0.833333	0.883134	0.000000
10	0.333333	1.000000	0.885038	1.000000
11	1.000000	0.833333	0.868536	1.000000
12	0.000000	1.000000	0.891947	0.200000
13	0.000000	0.000000	0.896892	0.000000
14	0.000000	0.875000	0.889110	0.600000

Number of Web Searches: 4

['No', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No']

Fig. 8. Experiment 2 Metrics Results

Context Recall are zero. For most queries, Faithfulness scored high (the LLM did a respectable job at responding well to the context it was given), yet the Answer Relevancy fell short of .90 for most queries. Seven web searches were performed.

Experiment 2 (15 queries)
Configuration: CVE ID, GPT4o, OAI EM, CHRDB, SST, TWS
The results of this experiment as captured in Fig. 8 reveal sub-optimal results for all metrics, but with fewer web searches. The difference in this configuration was the replacement of MMR with SST and the absence of CE use.

Experiment 3 (15 queries)
Configuration: CVE ID, GPT4o, HF EM, CHRDB, SST, TWS
The results of this experiment as captured in Fig. 9 reveal sub-optimal results. In this case, web search was used for all queries. The only difference between this experiment and Experiment 2 was the replacement of the embedding model, using HF EM instead of OAI EM.

For the next three experiments, the queries led with the CVE SP but the configuration settings for each experiment are repeated from the first three experiments. As shown in Fig. 10-12, a similar trend of sub-optimal results was observed.

	context_precision	faithfulness	answer_relevancy	context_recall
0	0.0	1.000000	0.832577	0.750000
1	1.0	0.857143	0.772400	1.000000
2	1.0	1.000000	0.875490	1.000000
3	0.0	0.000000	0.000000	0.000000
4	0.0	0.500000	0.000000	0.000000
5	0.2	0.600000	0.884418	1.000000
6	0.0	0.818182	0.851103	0.333333
7	0.0	1.000000	0.845078	0.500000
8	0.5	1.000000	0.868835	1.000000
9	1.0	0.913043	0.851634	1.000000
10	0.0	1.000000	0.901768	0.000000
11	1.0	1.000000	0.862962	1.000000
12	0.0	0.571429	0.850734	0.000000
13	0.0	0.857143	0.880662	0.000000
14	0.0	1.000000	0.905582	1.000000

Number of Web Searches: 15

['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes']

Fig. 9. Experiment 3 Metrics Results

	context_precision	faithfulness	answer_relevancy	context_recall
0	0.0	0.800000	0.869646	0.166667
1	1.0	1.000000	0.886816	1.000000
2	1.0	1.000000	0.881593	1.000000
3	0.0	0.000000	0.882193	0.000000
4	0.0	0.750000	0.000000	0.000000
5	0.0	0.500000	0.852943	0.333333
6	0.0	1.000000	0.885241	0.333333
7	0.0	0.684211	0.848595	0.000000
8	0.0	1.000000	0.857075	0.400000
9	0.0	1.000000	0.872171	0.000000
10	1.0	0.000000	0.880333	1.000000
11	1.0	0.800000	0.868536	1.000000
12	0.0	1.000000	0.000000	0.000000
13	0.0	1.000000	0.858717	0.000000
14	0.0	1.000000	0.882790	0.400000

Number of Web Searches: 5

['No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No']

Fig. 10. Experiment 4 Metrics Results

Experiment 4 (15 queries)
Configuration: CVE SP, GPT4o, OAI EM, CHRDB, MMR, CE, TWS

Experiment 5 (15 queries)
Configuration: CVE SP, GPT4o, OAI EM, CHRDB, SST, TWS

Experiment 6 (15 queries)
Configuration: CVE SP, GPT4o, HF EM, CHRDB, SST, TWS

An important observation from these experiments is the

	context_precision	faithfulness	answer_relevancy	context_recall
0	0.0	0.875	0.878957	0.500000
1	0.0	1.000	0.856234	0.000000
2	1.0	1.000	0.869187	1.000000
3	0.0	0.750	0.892211	0.000000
4	0.0	0.750	0.000000	0.000000
5	0.0	1.000	0.886791	0.000000
6	0.0	1.000	0.881238	0.333333
7	0.0	0.500	0.891939	0.200000
8	0.0	1.000	0.857075	0.200000
9	0.0	0.000	0.894449	0.000000
10	0.0	0.875	0.883861	0.000000
11	1.0	1.000	0.870096	1.000000
12	0.0	0.500	0.000000	0.000000
13	0.0	1.000	0.879153	0.000000
14	0.0	1.000	0.894453	0.200000

Number of Web Searches: 6

['No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No']

Fig. 11. Experiment 5 Metrics Results

	context_precision	faithfulness	answer_relevancy	context_recall
0	1.0	0.916667	0.870924	0.500000
1	1.0	1.000000	0.881007	1.000000
2	1.0	1.000000	0.869427	1.000000
3	0.0	0.000000	0.866120	0.000000
4	0.0	0.333333	0.000000	0.000000
5	0.0	1.000000	0.876751	0.666667
6	0.0	0.888889	0.867963	0.333333
7	0.0	NaN	0.847941	0.200000
8	0.0	0.800000	0.907839	0.200000
9	0.0	1.000000	0.854472	0.000000
10	0.0	1.000000	0.883861	0.000000
11	1.0	1.000000	0.868536	1.000000
12	0.0	1.000000	0.871954	0.000000
13	0.0	1.000000	0.874169	0.000000
14	0.0	0.666667	0.909256	0.200000

Number of Web Searches: 6

['No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No']

Fig. 12. Experiment 6 Metrics Results

	context_precision	faithfulness	answer_relevancy	context_recall
0	1.0	0.0	0.790339	1.0
1	1.0	1.0	0.841420	1.0
2	1.0	1.0	0.823533	1.0
3	1.0	1.0	0.849387	1.0
4	1.0	1.0	0.842632	1.0
5	1.0	1.0	0.821148	1.0
6	0.0	0.0	0.856257	0.0
7	1.0	0.5	0.841015	1.0
8	1.0	1.0	0.837851	1.0
9	1.0	1.0	0.843300	1.0
10	1.0	1.0	0.831946	1.0
11	1.0	1.0	0.841688	0.0
12	1.0	1.0	0.841592	1.0
13	1.0	1.0	0.850231	1.0

Number of Web Searches: 0

['No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No']

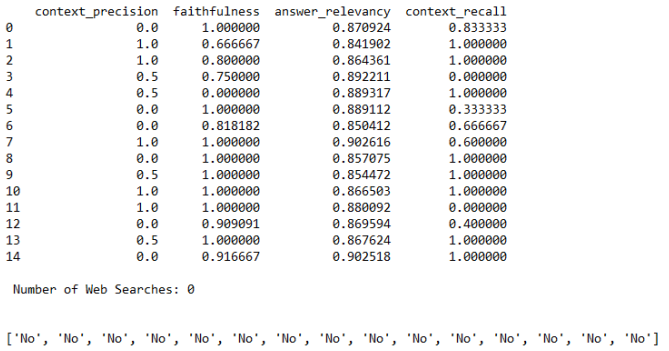
Fig. 13. Experiment 7 Metrics Results

inconsistent RAG results across all of the evaluation metrics. As previously mentioned, the inadequacies of standard RAG are well known as discussed in [11]. In order to address the common failure points in RAG architectures, a RAG-Blend approach, as presented in Fig. 4, was taken in these experiments. However, the results demonstrate that significant RAG issues remain.

Consequently, additional experiments were conducted using CyberVulModelEval but replacing the entire intermediate retrieval chain and pipeline with an undisclosed commercial product dubbed “Alternate RAG” to maintain confidentiality. The test dataset was presented to Alternate RAG which executed its own proprietary retrieval infrastructure including data chunking, embeddings generation, vector database storage, the retrieval algorithm, indexing, re-ranking, etc. In essence, only the agentic operational components within CyberVulModelEval, along with the use of GPT4o, were preserved. The results per Experiments 7-8 reveal significant improvement compared to the evaluation results from Experiments 1-6 (see Fig. 13-14), especially when the CVE ID was the key search term.

Experiment 7 (15 queries)
Configuration: CVE ID, GPT4o, Alternate RAG, TWS

Experiment 8 (15 queries)
Configuration: CVE SP, GPT4o, Alternate RAG, TWS



	context_precision	faithfulness	answer_relevancy	context_recall
0	0.0	1.000000	0.870924	0.833333
1	1.0	0.666667	0.841902	1.000000
2	1.0	0.800000	0.864361	1.000000
3	0.5	0.750000	0.892211	0.000000
4	0.5	0.000000	0.889317	1.000000
5	0.0	1.000000	0.889112	0.333333
6	0.0	0.818182	0.850412	0.666667
7	1.0	1.000000	0.902616	0.600000
8	0.0	1.000000	0.857075	1.000000
9	0.5	1.000000	0.854472	1.000000
10	1.0	1.000000	0.866503	1.000000
11	1.0	1.000000	0.880092	0.000000
12	0.0	0.909091	0.869594	0.400000
13	0.5	1.000000	0.867624	1.000000
14	0.0	0.916667	0.902518	1.000000

Number of Web Searches: 0

['No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No']

Fig. 14. Experiment 8 Metrics Results

	H1	H2	H3
E1	-1	1	0
E2	1	-1	1
Scoring	0	0	1

Fig. 15. ACH Matrix

The key takeaways from these additional experiments are as follows:

- The retrieval chain components used affect RAG performance.
- While RAG can be improved, it can remain sub-optimal in many areas.
- The generic structure of the agentic RAG-Blend framework employed by the CyberVulModelEval application allows for RAG engine components to be easily swapped out which facilitates experimentation.

Hypothesis Analysis

Employing the multi-step process [23] as part of a simplistic Analysis of Competing Hypothesis (ACH) for this research consists of the following:

Identify the hypotheses

- H1: Effective RAG can be consistently demonstrated using the test dataset and eval metrics.
- H2: Effective RAG cannot be consistently demonstrated using the test dataset and eval metrics.
- H3: Effective RAG can sometimes be demonstrated using the test dataset with eval metrics.

Evidence Collection

- E1: Experiments 1-6 as part of a customized retrieval system reveal sub-optimal RAG.
- E2: Experiments 7-8 as part of an independent commercial retrieval system reveals effective RAG.

Matrix Mapping

- 1 = Evidence maps positively to the hypothesis
- 0 = Evidence maps neutrally to the hypothesis
- -1 = Evidence maps negatively to the hypothesis

Based on the ACH matrix results, as shown in Fig.15, H3 emerges as the most plausible due to the supporting evidence from the research experiments.

VI. RECOMMENDATION AND FUTURE DIRECTIONS

The landscape of possible further investigations is vast. Therefore, to bound this discussion, the following five areas of exploration are presented.

LLM Diversity. There are many LLMs available. This paper chose GPT4o given the focus of this research revolves around enhancing the retrieval component of RAG. If retrieval is poor, then the response generated by any LLM will likewise be sub-optimal since that provided context is the input and therefore the only other information the LLM has available outside of its base training.

Embedding Model Diversity – This paper experimented with only three different embedding models counting the use of Alternate RAG. Using different embedding models [24] has a direct impact on retrieval and experimenting with other embeddings may improve retrieval outcomes.

Chunking Strategies – The primary chunking strategy chosen for Experiments 1-6 was a fixed size using the recursive text splitter within the LangChain/LangGraph ecosystem [31]. However, there are many other strategies one could explore including semantic, token-based, document specific, overlapping, sliding window, character, etc.

Context Window – With the introduction of LLMs like Gemini 2.0 that have a massive context window [32], one could simply provide the entire test dataset from this research as a full length document instead of the conventional retrieval component or the "RA" in RAG and allow the LLM to simply execute the "G" or generate the response. While these process windows are getting larger and larger, there will always be a document repository that is larger. However, RAG may not be the best solution for every problem. Thus, running Context Window only experiments with this research test dataset may yield better outcomes and provide a better solution for this use case.

Alternate Search Tool – Revisiting Fig. 4, another option to explore would be the use of a custom search tool instead of deferring to web search as the last resort. LLMs can be a valuable aid in generating functional software based on a description of the requirements for a tool. Again, while this approach can be an effective component within an agentic framework, it was considered outside the scope of the research presented in this paper especially given a custom search tool would be used when RAG fails to produce relevant information. The primary goal of this paper is to evaluate the performance of RAG using conventional semantic search techniques in concert with LLM base training.

VII. CONCLUSION

This paper proposed a CyberVulModelEval software application to explore the viability of LLMs using only base training to identify software packages associated with CVEs.

To overcome the limitations of standard RAG [11], CyberVulModelEval was realized within a RAG-Blend agentic framework. The first six experiments conducted and documented in this paper revealed sub-optimal results. The final two experiments yielded notable improvement, demonstrating the effectiveness of RAG in some contexts. However, considering the gravity of identifying all of the software packages associated with CVE identifiers tracked by the NVD and documented in GitHub Advisories [5], LLMs in tandem with RAG semantic search should be employed with caution. Hence, simply using LLMs with any RAG pipeline and claiming compliance with the HAVOSS model capability area expectations, such as the Identification and Monitoring of Sources, would not be a sign of cybersecurity process maturity. While using LLMs in flexible agentic frameworks together with robust retrieval infrastructure components may offer some benefits, their overall effectiveness and reliability for cybersecurity scenarios involving the appropriate linking of CVEs with vulnerable software packages show potential but require further research.

REFERENCES

[1] NIST National Vulnerability Database (2025). CVEs and the NVD Process. nvd.nist.gov. https://nvd.nist.gov/general/cve-process

[2] Hughes, C. (2024, August). Vulnerability Exploitation in the Wild. resilientcyber.io. https://www.resilientcyber.io/p/vulnerability-exploitation-in-the

[3] Druckman, K. (n . d.). The Careful Consumption of Open Source Software. intel.com. https://www.intel.com/content/www/us/en/developer/articles/guide/the-careful-consumption-of-open-source-software.html

[4] Perlow, J. (2022, March). A Summary of Census II: Open Source Software Application Libraries the World Depends On. linuxfoundation.org. https://www.linuxfoundation.org/blog/blog/a-summary-of-census-ii-open-source-software-application-libraries-the-world-depends-on

[5] About the GitHub Advisory Database. (n. d). docs.github.com. https://docs.github.com/en/code-security/security-advisories/working-with-global-security-advisories-from-the-github-advisory-database/about-the-github-advisory-database

[6] Catlin, K. (2022, February). GitHub Advisory Now Open To Community Contributions. github.blog. https://github.blog/security/vulnerability-research/github-advisory-database-now-open-to-community-contributions/

[7] Bideh, P., Host, M., Hell, M. (2018, November). HAVOSS: A Maturity Model for Handling Vulnerabilities in Third Party OSS Components. researchgate.net. https://www.researchgate.net/publication/328689964

[8] Chen T., Li L., Zhu L., Li, Z., Liu X., Liang G., Wang Q., Xie, T. (2023). Vullibgen: Generating Names of Vulnerability Affected Packages via a Large Language Model. arxiv.org. https://arxiv.org/abs/2308.04662

[9] Chen, Y., Sharma, A, (2020, May). Automated Identification of Libraries from Vulnerability Data. mysmu.edu. http://www.mysmu.edu/faculty/davidlo/papers/icse20-vulnerability.pdf

[10] Zhang, J., Bu, H., Wen, H., Liu, Y., Fei, H., Xi, R., Li, L., Yang, Y., Zhu, H., Meng, D. (2025, February). When LLMs meet cybersecurity: a systematic literature review. cybersecurity.springeropen.com. https://cybersecurity.springeropen.com/articles/10.1186/s42400-025-00361-w

[11] Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., Abdelrazek, M. (2024, January). Seven Failure Points When Engineering a Retrieval Augmented Generation System. arvix.org. https://arxiv.org/html/2401.05856v1

[12] Anthropic. (2024. September). Introducing Contextual Retrieval. anthropic.com. https://www.anthropic.com/news/contextual-retrieval

[13] Yao, S., Zhao, J., Yu, D. Du, N. Shafran, I., Narasimhan, K., Cao, Y. (2023, March.) ReAct: Synergizing Reasoning And Acting In Language Models. arvix.org. https://arxiv.org/pdf/2210.03629

[14] Asai, A., Wu, Z., Wang, Y., Sil, A., Hajishirzi. H. (2023, October). Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. arvix.org. https://arxiv.org/abs/2310.11511

[15] Rackauckas, Z. (2024, February). RAG-Fusion: A New Take On Retrieval-Augmented Generation. arvix.org. https://arxiv.org/pdf/2402.03367

[16] Yan, S., Gu, J., Zhu, Y., Ling, Z. (2024, October). Corrective Retrieval Augmentation Generation. arvix.org. https://arxiv.org/pdf/2401.15884

[17] Sanseviero, O. (2024, January). HackerSentence Embeddings. Cross-encoders and re-ranking. osanseviero.github.io.https://osanseviero.github.io/hackerllama/blog/posts/sentence

[18] Vullibgen Test Dataset. https://github.com/anonymous4ACL24/submission1129

[19] Vectorhub. (2024, May). Evaluating Retrieval Augmented Generation Using RAGAS. superlinked.com. https://superlinked.com/vectorhub/articles/retrieval-augmented-generation-eval-qdrant-ragas

[20] Ragas. (2025, January). Metrics. docs.ragas.io. https://docs.ragas.io/en/stable/concepts/metrics/

[21] Rutecki, M. (2024). RAG: MMR Search in LangChain. kaggle.com. https://www.kaggle.com/code/marcinrutecki/rag-mmr-search-in-langchain

[22] LangChain (2025). How to add scores to retriever results. python-langchain.com. https://python.langchain.com/docs/

[23] Goss, A. (2024, April). Analysis of Competing Hypothesis. kravensecurity.com. https://kravensecurity.com/analysis-of-competing-hypotheses/

[24] Xu, S. (2024, March). A Guide to Open-Source Embedding Models. bentoml.com. https://www.bentoml.com/blog/a-guide-to-open-source-embedding-models

[25] https://platform.openai.com/docs/guides/embeddings

[26] https://huggingface.co/

[27] https://www.trychroma.com/

[28] https://www.langchain.com/langgraph

[29] https://tavily.com/

[30] https://docs.ragas.io/en/stable/

[31] https://python.langchain.com/v0.1/docs/modules/

[32] https://ai.google.dev/gemini-api/docs/long-context