

# Large Language Models for Explainable Threat Intelligence

Tiago Dinis<sup>1</sup> Roger Tavares<sup>2</sup> Miguel Correia<sup>1</sup>

<sup>1</sup>INESC-ID, Instituto Superior Técnico, Universidade de Lisboa - Lisboa, Portugal

<sup>2</sup>UON - Lisboa, Portugal

**Abstract**—As cyber threats continue to grow in complexity, traditional security mechanisms struggle to keep up. Large language models (LLMs) offer significant potential in cybersecurity due to their advanced capabilities in text processing and generation. This paper explores the use of LLMs with retrieval-augmented generation (RAG) to obtain threat intelligence by combining real-time information retrieval with domain-specific data. The proposed system, RAGRecon, uses a LLM with RAG to answer questions about cybersecurity threats. Moreover, it makes this form of Artificial Intelligence (AI) explainable by generating and visually presenting to the user a knowledge graph for every reply. This increases the transparency and interpretability of the reasoning of the model, allowing analysts to better understand the connections made by the system based on the context recovered by the RAG system. We evaluated RAGRecon experimentally with two datasets and seven different LLMs and the responses matched the reference responses more than 91% of the time for the best combinations.

**Index Terms**—Large Language Models, Threat Intelligence, Retrieval-Augmented Generation, Natural Language Processing, Blockchain, Explainable AI

## I. INTRODUCTION

The volume and complexity of cyber threats continue to expand, driving organizations to rely on new technologies to provide more robust Cyber Threat Intelligence (CTI). Conventional security measures are often unable to handle and keep pace with the new range of cyber-attacks since the attackers adapt their methods. Large Language Models (LLMs) are highly effective in operating on textual data because of their ability to handle and process large unstructured semantic information. They demonstrate this by generating new semantic content and by analyzing massive volumes of previously collected data [1] [2]. Nevertheless, their capabilities can be further enhanced with a technique known as Retrieval-Augmented Generation (RAG), which combines real-time information retrieval with the generative capabilities of LLMs [3]. Generative AI, as in LLMs, refers to AI systems designed to create original content, such as text, leveraging their training on vast datasets to generate coherent and contextually appropriate outputs [4].

LLMs, such as the Generative Pre-trained Transformer (GPT) [5], show impressive generative capabilities and are powerful tools for analyzing and interpreting large amounts of data. However, they use data processed during a slow and expensive training phase, so they cannot handle real-time information and specialized domains. RAG is a solution to this limitation. RAG connects LLMs with a retrieval mechanism

that fetches documents from domain-specific datasets or live sources. This combination helps to ensure more accurate responses, a requirement in our fast-moving cybersecurity field.

Utilizing RAG, organizations can deploy and benefit from pre-trained LLMs more effectively as they do not need to build or train models themselves, which is an unreasonable expense and requires a significant amount of computational resources. Instead, models can be quickly applied to cybersecurity problems by simply integrating them with security-specific data and reports and as a result extending the parametric memory of the model with non-parametric memory, assisting security analysts and CTI teams to assess threats with similar speed, create responses and produce real-time CTI.

This paper presents *RAGRecon*, a system aimed at providing CTI to cybersecurity personnel, e.g., to Security Operations Center (SOC) operators. RAGRecon uses a LLM with RAG to answer questions about cybersecurity threats. Moreover, it makes this form of AI explainable by generating and visually presenting to the user a Knowledge Graph (KG) for every reply. More in detail, RAGRecon retrieves information, called context, for a question asked by the user. The context is extracted from user-supplied CTI documents (e.g., PDF security reports) using RAG. RAGRecon then prompts the LLM to identify relationships within that context and parses the information into a specific format. Using this, it creates a KG and answers the question based on the provided context. RAGRecon is designed to enable analysts to process and visualize connections in CTI data, facilitating the detection of vulnerabilities, the identification of anomalies, and the automation of responses to cyber threats, which is also applicable to the blockchain domain.

This project will leverage a wide spectrum of CTI reports, covering a far-reaching scope of cyber threats, from malware and phishing to advanced persistent threats. Moreover, the project will place particular emphasis on the domain of blockchain technology. As blockchain technology continues to acquire recognition across industries, it has become an increasingly attractive target for cybercriminals [6]. By focusing on strengthening security within the blockchain domain, this project strives to enhance our abilities to identify and respond to particular threats to this area, such as flaws within smart contracts, hacks of cryptocurrency wallets, and exploits affecting decentralized financial applications [7].

The motivation for RAGRecon comes from the need to

address the complexities of analyzing and responding to cyber threats in an increasingly data-intensive landscape. Employing LLMs allows the system to efficiently parse and process natural language CTI with accuracy and adaptability. The integration of RAG facilitates the retrieval of the most relevant and contextually accurate information, addressing the challenges of information overload while reducing the occurrence of hallucinations in answers. RAGRecon includes an explainable AI component that synthesizes relevant context to answer user questions. This benefits users by simplifying complex information while also highlighting which parts of the context the language model found most relevant to generate its response. The use of explainable AI ensures that the system's reasoning is transparent, fostering trust and enabling users to understand the rationale behind its answers. Finally, the incorporation of a KG provides an intuitive visual representation of the relevant interconnected CTI data within the user's documents necessary to answer the question.

The experimental evaluation, conducted on two custom-built datasets<sup>1</sup>, one for conventional CTI (i.e., CTI about common security threats) based on 24 PDF reports, and one for Blockchain CTI based on 28 PDF reports, each comprising 50 questions. The system achieved high Faithfulness scores, consistently exceeding 0.8 out of 1.0, indicating a low rate of factual hallucinations. Furthermore, it exhibited efficient Context Relevance, using approximately 8% of the retrieved context on average to generate answers. To validate the LLM self-evaluation methodology, a manual analysis of 2,050 automated decisions was performed, confirming the high accuracy of the approach with correct decision rates generally ranging from 90% to 97% with 7 LLMs. This verification also substantiated that minor performance variations resulted from occasional errors by both the generation model and the self-evaluation model. A slight performance advantage was observed with the blockchain dataset, although it was considered likely to be attributable to the limited dataset size rather than to a statistically significant trend.

The paper has the following main contributions: 1) RAGRecon, a system that uses LLMs, RAG, and KGs to provide explainable CTI; 2) an experimental evaluation of RAGRecon with two CTI datasets and 7 different LLMs; 3) insights into performance variations between the two datasets and the different LLMs used.

## II. BACKGROUND AND RELATED WORK

This section explores three fundamental components central to our proposal: LLMs, RAG, and CTI.

### A. Large Language Models

LLMs are sophisticated AI systems designed to process and generate human language. Built on deep learning architectures like the transformer, they are trained on vast and diverse datasets, enabling them to perform tasks such as text completion, translation, and question-answering. The scale of these

models, often containing billions of parameters, allows them to capture complex linguistic patterns. However, LLMs have inherent limitations, including knowledge cutoffs, a tendency to "hallucinate" (generate plausible-sounding but factually incorrect information) and potential biases inherited from their training data [8]–[10].

Architecturally, LLMs can be distinguished by their core components. Encoder-only models like BERT (Bidirectional Encoder Representations from Transformers) [11] are designed to understand and represent input text, excelling at tasks like classification and entity recognition. Decoder-only models like GPT focus on generating text sequences [5]. Encoder-decoder models such as T5 (Text-to-Text Transfer Transformer) combine both functions to transform input sequences into new output sequences, making them suitable for summarization and translation [1], [12].

LLMs can be classified as public or private. Public models, such as Mistral 7B or those available through APIs like GPT-4o, democratize access to AI but raise concerns about misuse and security. Private models, such as Google's PaLM, offer greater control and data security for organizations, but require significant resources to develop and maintain.

To address the black-box nature of these models, the field of Explainable AI (XAI) aims to make their decision-making processes transparent and interpretable [13]. Graphs are a particularly effective tool in XAI, helping to visualize data flow and relationships within the model, which is crucial for building trust and ensuring accountability [14], [15].

### B. Retrieval-Augmented Generation

RAG is a framework that enhances LLMs by connecting them to external knowledge sources in real time. Introduced by Lewis et al. [3], RAG combines the parametric knowledge stored within an LLM's parameters with non-parametric knowledge retrieved from a vector store. This hybrid approach allows the model to access up-to-date, factual information, significantly reducing hallucinations and improving the accuracy of its responses. A key advantage of RAG is that the external knowledge base can be updated without retraining the entire model. The RAG process involves two main steps: retrieval and generation.

- 1) Retrieval: When a query is received, the system first retrieves relevant documents from a knowledge base. This can be done using two main methods: sparse retrieval, based on keyword matching, and dense retrieval, which uses embeddings to represent the semantic meaning of text. Dense retrieval converts both the query and the documents into vectors and finds the most relevant documents based on vector similarity. Dense retrieval is better at understanding context, but is more computationally expensive.
- 2) Generation: The retrieved information is then passed to the LLM along with the original query and the model generates a response based on this augmented context.

There are two primary RAG variants [3]:

<sup>1</sup> Available at: <https://github.com/Tiago-Din/RAGRecon-Datasets>

- RAG-Sequence: Retrieves a set of documents once and uses them to generate the entire response in a single pass. It is effective for tasks requiring a broad, coherent answer synthesized from multiple sources.
- RAG-Token: Retrieves information dynamically for each token being generated. This allows the model to adjust its focus in real-time based on the evolving context, making it suitable for more dynamic responses.

RAG is central to modern Question Answering (QA) systems. These systems can be extractive (pulling exact answers from text) or abstractive (generating new, summarized answers). They can also be closed domain (specialized in one topic) or open domain (covering a wide range of topics). RAG architecture is highly adaptable to these types of QA systems.

### C. Cyber Threat Intelligence

Threat intelligence is the process of gathering, analyzing and acting upon data about cyber threats to improve cybersecurity defenses. It helps organizations understand adversary tactics and make informed security decisions. CTI is often categorized into two types:

- Operational CTI: Focuses on immediate, actionable threats, providing technical details like Indicators of Compromise (IOCs) to help security teams detect and respond to ongoing attacks.
- Strategic CTI: Provides a high-level view of the threat landscape, helping decision makers prioritize resources and anticipate future challenges.

A significant challenge in CTI is the manual effort required to extract structured information from unstructured reports, a process that can be extremely time-consuming [16]. Automating this process is crucial for making CTI faster and more effective. Researchers have proposed systems to automate the classification and enrichment of threat data from various sources, including Open Source Intelligence (OSINT) from platforms such as Twitter [17], [18].

LLMs have shown promise in automating CTI tasks. Shafee et al. [19] found that LLMs perform well in binary classification of threat data, but struggle with more complex tasks like Named Entity Recognition (NER), which is essential to extract specific entities such as malware names or threat actor groups. This highlights a key area for improvement.

### D. Enhancing CTI with LLMs and RAG

The integration of RAG with LLMs offers a powerful solution to the challenges in CTI [20]. By providing LLMs with real-time access to specialized databases (e.g., threat reports, vulnerability databases), RAG enables the generation of high-quality, contextualized CTI [21].

This approach directly addresses the needs of SOC analysts, who often must manually contextualize global CTI for their specific organization. Mitra et al. [22] introduced LOCALINTEL, a RAG-based system that automates this process. LOCALINTEL retrieves information from global sources like the National Vulnerability Database (NVD) and combines it with an organization's internal knowledge base to produce

tailored, actionable intelligence. This system significantly reduces manual labor and improves the accuracy of local threat assessments.

Furthermore, RAG-enhanced LLMs can improve the understanding of adversary Tactics, Techniques and Procedures (TTPs), which are cataloged in frameworks like MITRE ATT&CK. Fayyazi et al. [23] found that while large decoder-only models like GPT-3.5 can struggle with precision when analyzing TTPs, integrating a RAG component significantly boosts their performance. This demonstrates the potential of combining LLMs with RAG to interpret complex cyberattack behaviors.

Our work contributes to the state-of-the-art by presenting a novel method for visualizing an LLM's reasoning using KGs. Our system synthesizes context retrieved via RAG, prioritizing information based on its relevance to the user's question. Then it generates a visual map of key concepts and their interconnections, providing users with clear visibility into the data points that support the answer.

## III. RAGRECON

This section presents RAGRecon. The section focuses on the approach; additional details are left for Section IV that presents its implementation.

### A. Architecture

The architecture of the system is shown in Figure 1. The central component, designated RAGRecon in the figure, essentially processes user queries.

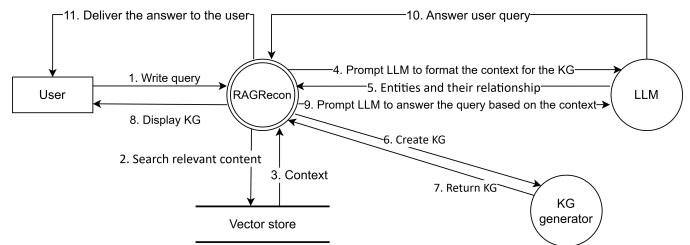


Fig. 1. Data flow diagram of the proposed system architecture

This architecture leverages RAG to extract and structure relevant information from the vector store, delivering accurate and context-aware responses to user queries. The process begins when the user submits a query via the command-line interface. Upon receiving the query, RAGRecon searches the vector store, which is preloaded with relevant data sources (in the form of embeddings, as explained later), to retrieve information that is contextually aligned with the user's input.

RAGRecon retrieves the context and processes it to construct the KG and guide the LLM's tasks. RAGRecon prompts the LLM to format this extracted context into a structured format, which is then used to build the KG. This graph visually and logically represents the domain knowledge context relevant to the query.

Once the KG is constructed, RAGRecon calls again the LLM to interpret the query within the previously retrieved

context. Using contextual information, the LLM generates a contextually rich answer to the user’s query. Finally, the KG and the generated answer are delivered to the user, completing the process.

### B. Data Ingestion and Vectorization

The preload of the documents into the system initiates a pipeline that processes and prepares a collection of documents for search and retrieval of similarity. Initially, the documents are loaded and then split into smaller, manageable chunks. This step is critical, particularly for handling long documents [24], as it ensures that each chunk is appropriately sized for embedding models, allowing for effective processing without overwhelming the system or losing important context.

After splitting the documents, RAGRecon proceeds to generate vector embeddings for each chunk using a pre-trained model. These embeddings are high-dimensional numerical representations that capture the semantic meaning of the text. Rather than relying on keyword matching, the model translates each chunk of text into a vector that encodes its contextual information, meaning that even if the exact phrasing does not match a query, similar content can still be retrieved based on its meaning.

Once the embeddings are generated, they are stored in a vector store, a specialized database able to store and index high-dimensional vectors. Through these embeddings, the system can identify and retrieve the most relevant document chunks based on their semantic similarity to the user’s query, rather than simply matching words or phrases.

### C. Interactive RAG

The system’s core functionality is a dual-output architecture that processes user queries to generate both a detailed textual response and a visual KG. Operating in an interactive loop similar to a conversational chatbot, the workflow enables a dynamic and exploratory user experience. This real-time iterative cycle allows users to ask follow-up questions, leveraging both textual and visual summaries to build a comprehensive understanding.

*1) Textual Response Generation:* Upon receiving a user query, the system’s primary objective is to generate an informed and coherent textual answer. This process is initiated by retrieving the relevant context from the vector store, which contains precomputed vector embeddings of document chunks. These high-dimensional vectors encode the semantic meaning of the text, enabling efficient and accurate information retrieval. The system performs a similarity search against the vector store to identify the top-K document chunks most relevant to the user’s query. Following the RAG-Sequence approach, these retrieved chunks are concatenated into a single block of text. This consolidated context is then integrated with the original user question into a RAG prompt. The prompt is engineered to instruct the LLM to synthesize the information and generate a user-friendly explanation while maintaining a natural conversational tone. Finally, the crafted prompt is passed to the LLM, which returns a detailed, contextually-grounded natural language response.

*2) Knowledge Graph Generation and Visualization:* To complement the textual response, the system generates and visualizes the contextual information as a Knowledge Graph (KG), providing users with a clear summary of the relationships between key concepts. Using the same retrieved context chunks, the system formulates a separate, graph-related prompt. This prompt specifically instructs the LLM to analyze the text and extract the primary entities (nodes) and the relationships (edges) that connect them. The LLM generates a structured textual description of these relationships. This output is then processed to construct the KG, where each entity is represented as a node, and each relationship is depicted as a connecting edge. This visualization provides an intuitive map of the information, helping users grasp complex connections at a glance.

## IV. IMPLEMENTATION OF RAGRECON

This section details the technical implementation of the RAGRecon system, represented in Figure 2. The system is built with a modular architecture, featuring a back-end for data processing and a choice of two distinct user interfaces: a script-based command line interface (CLI) and an interactive graphic user interface (GUI).

The implementation leverages the LangChain framework<sup>2</sup> to orchestrate the retrieval process, from data ingestion and vectorization to real-time query handling and context retrieval, ChromaDB<sup>3</sup> a DBMS for vector storage also targeted at LLMs, the Flask<sup>4</sup> framework for the web interface and various LLMs via aisuite<sup>5</sup>.

The foundational logic of RAGRecon is divided into two primary stages: a processing pipeline to create a queryable knowledge base and an interactive pipeline to process queries, retrieve context and generate responses. This core logic is common to the CLI and the GUI.

### A. Data Ingestion and Vectorization

The initial phase of the system involves processing the source PDF documents and converting them into a queryable vector format. This is handled by a dedicated data preparation pipeline that performs a series of sequential steps.

*1) Document Loading and Pre-processing:* Source documents are placed in a designated data directory. The system utilizes the PyPDFDirectoryLoader from LangChain to load all PDF files from this location. This loader treats each page of every PDF as a distinct Document object, preserving its content and metadata, such as the source filename and page number.

*2) Text Chunking:* To ensure that the text segments are manageable in size for the embedding model and fit within the context windows of the LLM, the loaded documents are split into smaller chunks. This process uses LangChain’s

<sup>2</sup><https://www.langchain.com/langchain>

<sup>3</sup><https://docs.trychroma.com/docs/introduction>

<sup>4</sup><https://flask.palletsprojects.com/en/stable/>

<sup>5</sup><https://github.com/andrewyng/aisuite>

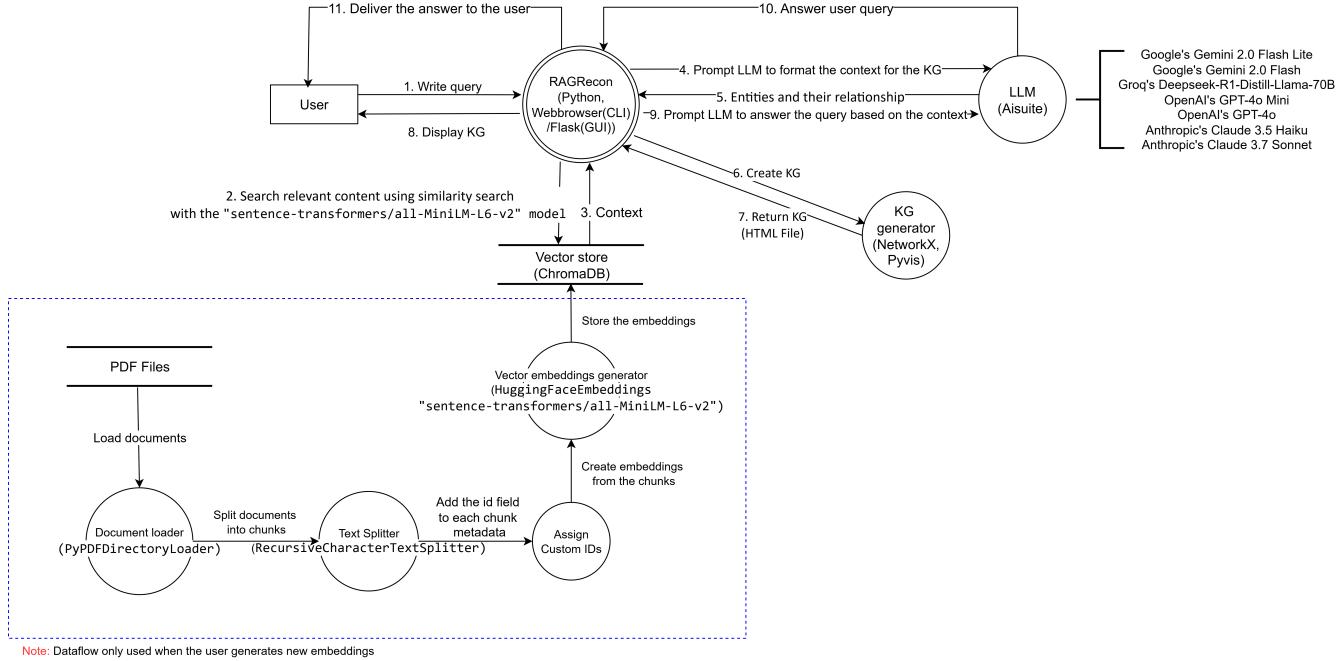


Fig. 2. Data flow diagram with the technologies used in the architecture

RecursiveCharacterTextSplitter configured with the following parameters:

- Chunk Size: 1000 characters.
- Chunk Overlap: 100 characters. This overlap helps maintain semantic context between adjacent chunks.
- Separators: A hierarchical list of separators is used to split text along logical boundaries, starting with paragraphs and progressively moving to smaller units:  
([ "\n\n", "\n", "(?<=\.\. )", " ", "" ])

3) *Custom Metadata and ID Generation:* A crucial step in this phase is the assignment of a unique identifier to each text chunk. After splitting, the script iterates through each chunk and enriches its metadata with a custom id. This ID has the format: filename\_p<page\_number>\_c<chunk\_index\_on\_that\_page>. For example, the third chunk from the fifth page (0-indexed page 4) of a file named report.pdf would receive the ID report.pdf\_p<4>\_c<2>. This identification scheme is essential for later retrieving and reconstructing the context of a document in its original order.

4) *Embedding Generation and Vector Storage:* The processed text chunks are then converted into numerical representations (embeddings). This is accomplished using LangChain's HuggingFaceEmbeddings wrapper, which loads the sentence-transformers/all-MiniLM-L6-v2 model. This model is chosen as it is free to use, efficient and effective in generating meaningful semantic vectors for sentences and paragraphs. The resulting embeddings, along with their corresponding text content and metadata (including the custom ID), are stored in a ChromaDB vector store. The database is configured to persist on disk in the ./chroma\_db directory. To ensure a fresh

build during each execution, the script automatically deletes any pre-existing database directory before creating a new one.

### B. Interactive RAG and Knowledge Graph Generation

The second part of the system is the interactive query engine, which provides an interface for users to query the indexed documents. This engine manages context retrieval, LLM interaction and visualization features.

1) *Query Processing and Context Retrieval:* When a user submits a query, the following steps occur:

- *Query Embedding:* The user's query is converted into an embedding using the same sentence-transformers/all-MiniLM-L6-v2 model.
- *Similarity Search:* The system performs a similarity search against the ChromaDB vector store. Retrieves the k=6 text chunks most semantically similar to the query embedding.
- *Context Assembly:* The content of these 6 chunks is concatenated to form the context that will be fed to the LLM.

2) *LLM Integration and Prompting:* The RAGRecon system is designed to be model-agnostic, supporting a variety of LLMs from providers like Google, OpenAI, Anthropic and Groq. A carefully crafted prompt is generated that combines the user's original query with the retrieved context. The prompt includes system instructions that guide the LLM to act as a helpful bot, answer in a conversational tone, and base its response on the provided context. The system also maintains a conversation history that is formatted in a model-agnostic way. This allows a user to switch between different LLMs in the same conversation while still enabling the active model to use previous turns for more coherent, multi-turn dialogues.

3) *Knowledge Graph Visualization*: A key feature of RAGRecon is its ability to generate and visualize knowledge graphs to represent relationships within the text. The process is as follows:

- **Graph Data Extraction**: A specialized prompt instructs the selected LLM to analyze the retrieved context and extract entities and their relationships. The LLM is required to return this information in a structured JSON format, like [“subject”: “A”, “relationship”: “B”, “object”: “C”], where the letters correspond to the pieces of relevant information identified by the LLM.
- **Visualization**: This is handled by a function that parses the LLM-generated JSON and builds a graph using NetworkX<sup>6</sup> and the Pyvis<sup>7</sup> Python libraries to create an interactive HTML file.

4) *Example*: To illustrate the system’s capabilities, this section provides a concrete example of how an input query is processed to generate a response and a corresponding knowledge graph.

a) *Input Query*: Let us assume a user submits the following query to the RAGRecon system using the anthropic provider with the claude-3-7-sonnet-20250219 model:

*“What is Broken Object Level Authorization (BOLA)?”*

b) *Context Retrieval*: The system converts this query into an embedding and performs a similarity search. The top k=6 retrieved chunks are concatenated to form the following context (a condensed version for clarity):

*“(Insecure Direct Object Reference). BOLA arises from APIs exposing object identifiers through their endpoints, posing significant Object Level Access Control concerns. This vulnerability allows attackers to manipulate or access API ...”*

c) *LLM-Generated Knowledge Graph Data*: The system then provides the retrieved context to the LLM with a specialized prompt, instructing it to extract entities and relationships in a structured JSON format. The LLM returns the following data:

```
[{"subject": "BOLA", "relationship": "arises from", "object": "APIs exposing object identifiers", "type": "Relationship"}, {"subject": "BOLA", "relationship": "poses", "object": "Object Level Access Control concerns", "type": "Relationship"}, {"subject": "BOLA", "relationship": "allows", "object": "attackers to manipulate or access API data/resources without proper authorization", "type": "Relationship"}, {"subject": "BOLA", "relationship": "leads to", "object": "severe consequences", "type": "Relationship"}, {"subject": "BOLA", "relationship": "can result in", "object": "unauthorized access, breaches, and misuse of critical functionalities", "type": "Relationship"}, {"subject": "BOLA", "relationship": "should implement", "object": "robust monitoring and logging mechanisms", "type": "Relationship"}, {"subject": "BOLA", "relationship": "is also known as", "object": "IDOR", "type": "Relationship"}, {"subject": "OWASP API Security Top 10", "relationship": "was updated in", "object": "2023", "type": "Relationship"}, {"subject": "Open Worldwide Application Security Project", "relationship": "is developed by", "object": "the list", "type": "Relationship"}]
```

d) *Resulting Knowledge Graph Visualization*: The JSON output is then parsed to build a graph using NetworkX, which

is then rendered as an interactive HTML file with Pyvis (Figure 3).

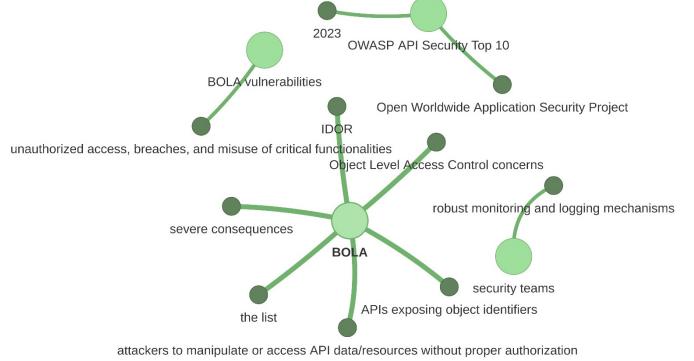


Fig. 3. Knowledge Graph Example

The KG is a tool to help users comprehend large and complex contexts passed to an LLM. When faced with a long document, a user often encounters a “wall of text” that leads to information overload. The linear nature of prose makes it difficult to quickly identify the main subjects and understand the relationships between concepts, often found pages apart, forcing the user to build a complex mental map from scratch in a cognitively demanding task.

The KG solves this problem by transforming unstructured text into a structured relational map that acts as a visual summary. It allows a user to grasp the main topics at a glance, as the most important concepts are represented by nodes. In Figure 3, the central subject is BOLA. However, the true power of this visualization is its ability to reveal the relationships between these concepts through its connecting lines. The user no longer needs to scan the entire document to understand how the ideas are linked. Observing the graph, one can instantly see these precise connections: the vulnerability BOLA arises from APIs exposing object identifiers and poses Object Level Access Control concerns. The graph clarifies that BOLA is also known as IDOR and shows that it allows attackers to manipulate or access API data/resources without proper authorization, which in turn leads to severe consequences. Furthermore, it connects other facts, visualizing that BOLA is top priority on “the list,” which it identifies as the OWASP API Security Top 10 that was updated in 2023.

In practice, this visualization acts as a factual map of the original context based on what the LLM found relevant to answer the question and as a help in formulating better questions. A user can review the graph to build an accurate mental model and then ask the LLM more specific questions based on the established relationships. This reduces cognitive load and aids the user to explore, verify, and question the LLM’s knowledge base with much greater precision and speed.

e) *Answer Generation*: The retrieved context and the original query are combined into a prompt. The LLM then generates the following conversational response based on this provided context:

<sup>6</sup><https://networkx.org/documentation/stable/index.html>

<sup>7</sup><https://pyvis.readthedocs.io/en/latest/index.html>

*“Broken Object Level Authorization (BOLA), also known as Insecure Direct Object Reference (IDOR), is a security vulnerability that occurs when APIs expose object identifiers through their endpoints. This creates significant Object Level Access Control issues, allowing attackers to manipulate or access API data and resources without proper authorization.*

*BOLA is the top priority on the OWASP API Security Top 10 list (updated in 2023). Although an average of 1.6 API endpoints per organization are at risk of BOLA abuse, the consequences can be severe, including unauthorized access, data breaches, and misuse of critical functionalities.*

*To mitigate BOLA vulnerabilities, security teams should implement robust monitoring and logging mechanisms to track API usage, detect anomalies, and identify potential unauthorized access.”*

### C. User Interfaces

RAGRecon provides two distinct interfaces that build on the core architecture, catering to different user preferences.

1) *Command-Line Interface:* The CLI provides a script-based way to interact with the RAGRecon system. It is designed for users who are comfortable working in a terminal environment. The RAGRecon script encapsulates the interactive logic in a while True loop that continuously prompts the user for input. The user can select their desired LLM at the beginning of the session and then begin asking questions. In addition to standard queries, the CLI supports special flag-based commands for managing the session, such as /quit to exit, /help to display options, /clean\_history to reset the conversation, and /change\_model to switch LLMs mid-session. Each query triggers the full RAG pipeline and the generated KG HTML file is automatically opened in the user’s default web browser.

2) *Graphical User Interface:* The GUI is web-based user interface built with the Flask web framework. It provides a more intuitive and interactive experience for managing documents, posing queries and visualizing results. The GUI’s backend is a Flask application that exposes several API endpoints (@app.route) that respond to HTTP requests from the user’s browser. This event-driven model replaces the CLI’s input loop. Key endpoints manage query submission, model selection, file uploads/deletions and the triggering of the embedding pipeline. The application state, such as the conversation history and the selected model, is managed via global variables within the Flask application.

## V. EXPERIMENTAL EVALUATION

The performance of the system was evaluated by comparing the answers it generated with predefined reference answers for a given set of questions. To automate this comparison, we used an LLM self-evaluation methodology introduced by Ren et al. in [25]. This involves using a separate LLM to assess the semantic similarity between the reference answer and the answer generated by the RAGRecon system. We not only used this methodology for all answers, but also manually assessed all of them.

### A. Dataset Construction

To facilitate the evaluation, two datasets were constructed:

- 1) Conventional CTI: The source material for this analysis consisted of 24 PDF reports on CTI, produced by various companies in the field.
- 2) Blockchain CTI: The process was repeated with 28 PDF reports specific to the blockchain domain.

Each dataset was built as follows:

- Question Generation: Our process involved prompting Gemini 2.5 Flash to generate a pool of questions for each document. We then manually selected 50 of these questions to form the datasets. For example, a question from the conventional CTI dataset asked “Which ransomware group targeted the UK’s Royal Mail in January 2023?” while a question from the blockchain dataset sought to “Explain the concept of a 51% attack in the context of a permissionless distributed ledger.”
- Reference Answer Generation: The same LLM was then used to generate a reference answer for each question, using the source document as context.
- Manual Review: Each question and reference answer pair underwent a manual review to ensure accuracy and eliminate any instances of hallucination.

### B. Benchmarking Framework

To benchmark performance, we evaluated seven distinct LLMs from four providers. The models included Google’s Gemini 2.0 Flash and Flash Lite (accessed via the free tier), OpenAI’s GPT-4o and GPT-4o Mini (paid access), Anthropic’s Claude 3.7 Sonnet and Claude 3.5 Haiku (paid access) and Groq’s freely available Deepseek-R1-Distill-Llama-70B. The performance of each LLM was assessed across multiple testing rounds on both the conventional security and the blockchain datasets. Visualizations, such as those exemplified in Figure 4, were used to analyze and compare their performance.

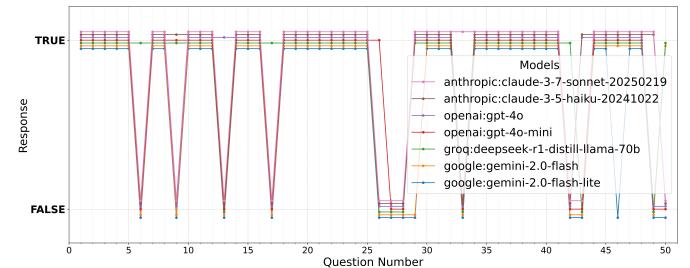


Fig. 4. Comparison different models Round 3 Conventional CTI Dataset

### C. Key Evaluation Metrics

Following the methodology outlined by Es et al. in [26], the system was evaluated using two primary metrics:

- Faithfulness: This metric measures the degree to which the generated answer is factually grounded in the provided context. High faithfulness ensures that the response

is accurate, avoids hallucination, and is justified by the retrieved information.

$$\text{Faithfulness} = \frac{\text{Number of Statements Supported by Context}}{\text{Total Number of Statements}}$$

How it is calculated:

- 1) An LLM breaks down the generated RAG answer into individual statements.
  - 2) A second LLM call verifies each statement against the context, producing a “Yes” or “No” verdict for each.
  - 3) Two more LLM calls are made to count the total number of “Yes” verdicts and “No” verdicts.
  - 4) The final score is the ratio of “Yes” verdicts to the total number of statements (“Yes” + “No”).
- Context Relevance: This metric checks how much useful information is in the data the system retrieves. It is crucial for efficiency, as it ensures that the context provided to the LLM contains minimal irrelevant information, thereby reducing the computational cost associated with processing long passages.

$$\text{Context Relevance} = \frac{\text{Number of Relevant Sentences}}{\text{Total Number of Sentences in Context}}$$

How it is calculated:

- 1) An LLM extracts sentences from the provided context that it deems relevant to the question.
- 2) A second LLM call counts the number of these relevant sentences.
- 3) A third LLM call counts the total number of sentences in the original context.
- 4) The final score is the ratio of relevant sentences to the total sentences.

#### D. Results and Analysis

1) *System Pipeline and Determinism:* Log analysis provided key insights into the system’s internal workings. These logs document the automated tests conducted on the RAGRecon system. The tests evaluate how effectively the benchmarked models answer questions from two distinct datasets. The logs detail each critical stage of the process, including:

- Ingestion: Loading documents for embedding generation.
- Retrieval: Extracting relevant context from the vector store when RAGRecon is running.
- LLM self-evaluation: Running the RAGRecon system and comparing its answer with the Reference Answer.
- Evaluation: The key performance metrics discussed in the previous section.

The logs show the pipeline is deterministic from ingestion to retrieval. For a given dataset, the chunking process consistently produces the same number of chunks with identical splits and metadata. Consequently, a given query always retrieves the exact same context as it will find the exact same vectors.

While the answers from the RAGRecon system are semantically consistent, their phrasing may change from run to run due to the nature of LLMs.

2) *Analysis of Retrieval and Generation Metrics (at Top-K=6):* The evaluation highlighted strong and consistent performance in both context relevance and faithfulness.

**Context Relevance:** The system demonstrated efficient context utilization, using approximately 8% of the retrieved information on average to formulate an answer.

**Faithfulness:** The faithfulness of the generated answers was consistently high for both datasets, indicating a low rate of model hallucination. With only a single minor exception in one round, the average faithfulness score consistently exceeded 0.8 out of 1.0.

3) *LLM Self Evaluation Manual Verification:* The LLM self-evaluation showed high consistency, as evidenced by the “RAG answer match Reference Answers” histograms for both datasets (Figures 5 and 6). However, slight variations in these histograms across the three test rounds, despite identical retrieved context for each query, suggested some variability in the generation or self-evaluation step.

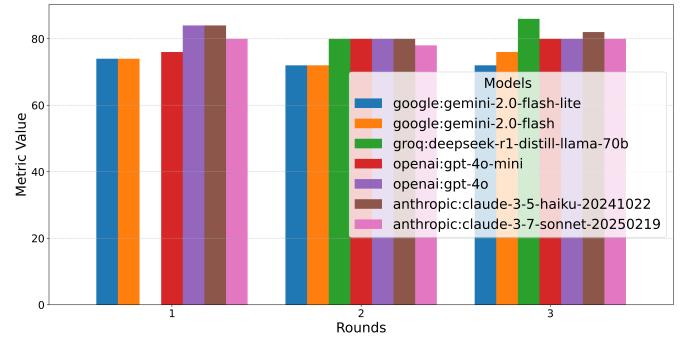


Fig. 5. RAG answer match reference answer Conventional CTI Dataset

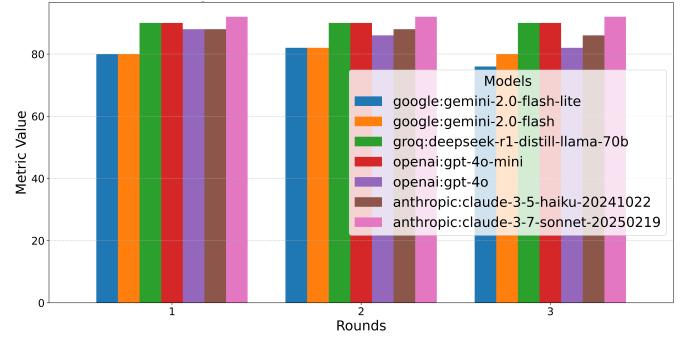


Fig. 6. RAG answer match reference answer Blockchain CTI Dataset

This led to two primary hypotheses:

- 1) The LLM self-evaluation occasionally made incorrect decisions about the semantic equivalence of an answer.
- 2) The generation model sometimes overlooked relevant information within the provided context when formulating its response.

To investigate this, a manual analysis of 2,050 LLM self-evaluation decisions was performed by reviewing testing logs and generated graphs. The verified correct answer percentages

are presented in Table I (Conventional CTI) and Table II (Blockchain CTI).

TABLE I  
LLM SELF EVALUATION PERFORMANCE METRICS CONVENTIONAL CTI DATASET

Name	Category	Correct Decision
Round 1	Round	93.00%
Round 2	Round	92.57%
Round 3	Round	93.43%
Google:gemini-2.0-flash-lite	Model	92.67%
Google:gemini-2.0-flash	Model	94.00%
Groq:deepseek-r1-distill-llama-70b	Model	92.00%
Openai:gpt-4o-mini	Model	93.33%
Openai:gpt-4o	Model	94.67%
Anthropic:claude-3-5-haiku-20241022	Model	90.67%
Anthropic:claude-3-7-sonnet-20250219	Model	93.33%

TABLE II  
LLM SELF EVALUATION PERFORMANCE METRICS BLOCKCHAIN CTI DATASET

Name	Category	Correct Decision
Round 1	Round	94.57%
Round 2	Round	94.57%
Round 3	Round	94.86%
Google:gemini-2.0-flash-lite	Model	97.33%
Google:gemini-2.0-flash	Model	96.00%
Groq:deepseek-r1-distill-llama-70b	Model	92.00%
Openai:gpt-4o-mini	Model	96.00%
Openai:gpt-4o	Model	95.33%
Anthropic:claude-3-5-haiku-20241022	Model	91.33%
Anthropic:claude-3-7-sonnet-20250219	Model	94.67%

Our analysis yielded two key findings. First, a comparison of the RAG answers with the reference answers and the respective decision confirmed both of our initial hypotheses. Second, the locally hosted Mistral 7B model exhibited high consistency in its output across multiple runs during the LLM self-evaluation process. A slight performance advantage was observed on the Blockchain CTI dataset compared to the Conventional CTI dataset. Although this could suggest that its training data was more heavily weighted toward blockchain topics, it is more likely a result of the limited size of our experimental datasets. More testing with larger datasets is required to determine whether this performance difference is statistically significant.

## VI. CONCLUSION

This paper introduces RAGRecon, a system that integrates LLMs and RAG to efficiently process and visualize complex CTI.

To evaluate our system, we benchmarked its performance on conventional and blockchain CTI datasets and demonstrated that it is possible to generate accurate, context-aware answers with high faithfulness.

A key limitation we observed, particularly in models with up to 20 billion parameters, was their inconsistent reliability in processing data for the KG. Formatting errors were the primary cause of this problem.

Despite this, our approach transforms the analysis of unstructured reports by not only automating the process but also allowing users to visualize threat relationships and derive clear explainable insights.

*Acknowledgments.* This work was financially supported by Project Blockchain.PT - Decentralize Portugal with Blockchain Agenda (Project no 51), WP6: Digital Assets Management, Call no 02/C05-i01.01/2022, funded by the Portuguese Recovery and Resilience Program (PPR), The Portuguese Republic and The European Union (EU) under the framework of Next Generation EU Program. This work was also supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID).

## REFERENCES

- [1] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, “Large Language Models: A Survey,” *arXiv preprint arXiv:2402.06196*, 2024.
- [2] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, “A comprehensive overview of large language models,” *ACM Transactions on Intelligent Systems and Technology*, 2023.
- [3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.
- [4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.
- [5] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [6] L. Hornuf, P. P. Momtaz, R. J. Nam, and Y. Yuan, “Cybercrime on the Ethereum blockchain,” *CESifo Working Paper No. 10598*, 2023.
- [7] L. W. Cong, K. Grauer, D. Rabetti, and H. Updgrade, “Blockchain forensics and crypto-related cybercrimes,” *Available at SSRN 4358561*, 2023.
- [8] G. Roffo, “Exploring advanced large language models with LLMSuite,” *arXiv preprint arXiv:2407.12036*, 2024.
- [9] A. Pal, L. K. Umapathi, and M. Sankarasubbu, “Med-halt: Medical domain hallucination test for large language models,” *arXiv preprint arXiv:2307.15343*, 2023.
- [10] M. Dahl, V. Magesh, M. Suzgun, and D. E. Ho, “Large legal fictions: Profiling legal hallucinations in large language models,” *Journal of Legal Analysis*, vol. 16, no. 1, pp. 64–93, 2024.
- [11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.
- [12] Yenduri *et al.*, “GPT (generative pre-trained transformer)—a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions,” *IEEE Access*, 2024.
- [13] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins *et al.*, “Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai,” *Information fusion*, vol. 58, pp. 82–115, 2020.
- [14] J. Schneider, “Explainable Generative AI (GenXAI): A survey, conceptualization, and research agenda,” *Artificial Intelligence Review*, vol. 57, no. 11, p. 289, 2024.
- [15] E. Rajabi and S. Kafaei, “Knowledge Graphs and Explainable AI in Healthcare,” *Information*, vol. 13, no. 10, p. 459, 2022.
- [16] G. Siracusano, D. Sanvito, R. Gonzalez, M. Srinivasan, S. Kamatchi, W. Takahashi, M. Kawakita, T. Kakumaru, and R. Bifulco, “Time for action: Automated analysis of cyber threat intelligence in the wild,” *arXiv preprint arXiv:2307.10214*, 2023.
- [17] C. Martins and I. Medeiros, “Generating quality threat intelligence leveraging OSINT and a cyber threat unified taxonomy,” *ACM Transactions on Privacy and Security*, vol. 25, no. 3, pp. 1–39, 2022.

- [18] F. Alves, A. Bettini, P. M. Ferreira, and A. Bessani, “Processing tweets for cybersecurity threat awareness,” *Information Systems*, vol. 95, p. 101586, 2021.
- [19] S. Shafee, A. Bessani, and P. M. Ferreira, “Evaluation of LLM-based chatbots for OSINT-based cyber threat awareness,” *Expert Systems with Applications*, p. 125509, 2024.
- [20] J. Zhang, H. Bu, H. Wen, Y. Chen, L. Li, and H. Zhu, “When LLMs meet cybersecurity: A systematic literature review,” *arXiv preprint arXiv:2405.03644*, 2024.
- [21] S. Rajapaksha, R. Rani, and E. Karafili, “A RAG-based question-answering solution for cyber-attack investigation and attribution,” *arXiv preprint arXiv:2408.06272*, 2024.
- [22] S. Mitra, S. Neupane, T. Chakraborty, S. Mittal, A. Piplai, M. Gaur, and S. Rahimi, “Localintel: Generating organizational threat intelligence from global and local cyber knowledge,” *arXiv preprint arXiv:2401.10036*, 2024.
- [23] R. Fayyazi, R. Taghdimi, and S. J. Yang, “Advancing TTP analysis: Harnessing the power of encoder-only and decoder-only language models with retrieval augmented generation,” *arXiv preprint arXiv:2401.00280*, 2023.
- [24] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the middle: How language models use long contexts,” *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.
- [25] J. Ren, Y. Zhao, T. Vu, P. J. Liu, and B. Lakshminarayanan, “Self-evaluation improves selective generation in large language models,” in *Proceedings on “I Can’t Believe It’s Not Better: Failure Modes in the Age of Foundation Models” at NeurIPS 2023 Workshops*, 2023, pp. 49–64.
- [26] S. Es, J. James, L. E. Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 2024, pp. 150–158.