POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Computer engineering - AI and data analytics

Master Degree Thesis

Design, Implementation and Evaluation of LLM-based Agents for Forensic Analysis

Supervisors
Prof. Danilo Giordano
Prof. Marco Mellia

Candidate
Stefano Fumero

Academic Year 2024-2025

Acknowledgements

Desidero esprimere la mia profonda gratitudine ai professori Danilo Giordano e Marco Mellia, relatori di questa tesi, per la loro guida instancabile e il supporto costante lungo tutto il percorso. La loro disponibilità e i preziosi consigli sono stati fondamentali per la realizzazione di un lavoro di cui vado particolarmente fiero.

Un sentito ringraziamento va anche a Matteo, che con la sua esperienza e i suoi suggerimenti è stato un punto di riferimento prezioso, non solo nella stesura della tesi, ma anche nelle scelte e negli orientamenti per il futuro.

Un grazie speciale va ai miei genitori: a mia mamma, per l'amore incondizionato e la premura che ogni giorno mi dimostra; a mio papà, che con il suo esempio mi ha insegnato il valore dell'impegno nel perseguire i miei obiettivi e a credere in me stesso. Siete stati per me una fonte inesauribile di affetto, sostegno morale e supporto economico, e vi sarò sempre profondamente grato.

Ringrazio di cuore le mie sorelle, Elisabetta e Francesca, che sin dalla mia infanzia sono state per me un esempio e una guida preziosa. Mi avete insegnato valori importanti come la determinazione e l'ironia con cui affrontare le sfide della vita. Crescere con voi, rischi dei primi anni a parte, è stato un grande privilegio e ha contribuito in modo significativo a rendermi la persona che sono oggi.

Un grazie speciale va anche alle piccole Adele e Giorgia, sempre pronte a regalarmi momenti di gioco e spensieratezza, preziosi per spezzare la monotonia delle lunghe giornate di studio.

Un ringraziamento particolare va a Elisa, per essere stata al mio fianco in ogni momento, tra studio e altri momenti di vita quotidiana. Il suo sostegno, sempre presente, sia morale che emotivo, ha reso questo percorso più leggero e significativo.

Desidero ringraziare tutti gli amici e le amiche che hanno colorato con la loro presenza questi cinque anni. Un grazie speciale a Manu, Marco, Mattia e Samu, compagni inseparabili di vacanze e protagonisti delle indimenticabili serate del mercoledì a base di birre e briscola.

Un ringraziamento sincero anche a Ste e Luca, presenti sin dagli albori di questo percorso e sempre grandi sostenitori. Grazie per aver sopportato le mie ansie e per le innumerevoli chiacchierate davanti a interminabili caffè.

Non posso dimenticare i ragazzi del gruppo Garage e le famose feste stagionali: forse mai esistite, ma sempre celebrate a dovere. Una menzione speciale va a Nico, Fra e Simo, per il supporto e la presenza costante.

Infine, un ringraziamento va a tutti i miei parenti che hanno sempre mostrato interesse nei confronti dei miei studi e mi hanno supportato quando necessario.

Abstract

Large Language Model (LLM) based agents are increasingly adopted for the automation of complex tasks. In this thesis, I systematically study their capabilities and limitations in cybersecurity forensics. Building upon a publicly available cybersecurity benchmark [7], I designed and evaluated a modular multi-agent system for forensic analysis. I first addressed two fundamental limitations of LLMs: the lack of long-term memory and the inability to access up-to-date knowledge. To overcome these challenges, I added a semantic memory module for storing and retrieving information and a web search tool (RAG) for external knowledge retrieval. Leveraging these solutions, I then enhanced the agent architecture through iterative refinements. The initial design relied on a single monolithic agent, while later versions added specialized components, including a PCAP Flow Reporter and a Log Reporter for traffic and log analysis. I evaluate the impact of design decisions on tool integration and architecture to provide guidance for practitioners. I benchmark four agent architectures and six LLM backends on 20 incident scenarios of increasing complexity. I also test 10 incidents from 2025, reaching 80% CVE identification accuracy with the best architecture. Finally, a human study with 22 experts rated the agent's reports as complete, useful, and coherent.

Contents

Chapter 1

Introduction

The thesis work focuses on the development of an AI Agent designed for Cybersecurity forensics. The objective of the agent is to automate the analysis of traffic and Docker log files to detect traces of malicious activity, evidence of potential attacks and support analysts in drawing meaningful conclusions. By combining Large Language Models (LLMs) with structured reasoning and memory, the agent aims to reduce the time and expertise required for complex forensic investigations.

1.1 Context and problem statement

When a Cybersecurity incident occurs, it almost always leaves behind digital traces. One of the most common sources of evidence is the network traffic, stored in technical files called PCAPs (packet capture files). A manual analysis of such files to find traces of malicious behavior is time-consuming and requires expertise. Therefore, this thesis addresses the problem of how to build an AI-based assistant that can help automating and streaming the analysis of forensic experts. The tool takes as input network traffic data and log files. It then starts a multi-step reasoning process, limited to a predefined number of iterations, to examine the data and find relevant insights. Finally, it has to provide a concise report summarizing key findings that can help humans to make further considerations and analysis. Also, I prompt the agent to specifically retrieve the affected service, the identified CVE (Common Vulnerabilities and Exposures) and to determine whether the attack was actually successful. This mirrors the typical workflow of a forensics expert, who must manually evaluate these same elements to understand whether a security incident has occurred and, in case the response is positive, answering with a proper incident management strategy.

To reduce the volume of data that needs to be analyzed, the network traffic provided to the agent is pre-filtered to include only information related to a specific, identified service of interest. As a result, the agent operates during the second phase of the forensic workflow. After an analyst has identified a potentially affected service based on preliminary evidence related to the incident, the relevant network data is filtered accordingly. This allows for a focused, in-depth analysis aimed at confirming the presence

of malicious activity.

1.2 Relevance of the problem

In recent years, increased digitalization and the widespread adoption of flexible work models have augmented the digital risk for both businesses and individuals. According to a report by the International Monetary Fund (IMF), the global cost of cybercrime is projected to reach $23 trillion by 2027, an increase of 175% compared to 2022 [38]. Moreover, according to a survey conducted by Gartner among executives [36], 50% of them believe that generative AI will increase the risk of cyber attacks. However, an interesting question arises: why not leveraging generative AI to also respond to cyber-attacks, adapting and detecting cyber-threats more effectively?

Such question is further supported by other data: the business is currently challenged by a skill gap in the cybersecurity sector: the growth of the global cyber-defense workforce of only 12.6% in 2024. This translates into a current shortage of approximately four million professionals worldwide, which could increase dramatically to eighty-five millions by 2030 without the introduction of innovative solutions [38].

The imbalance between rising threats and the availability of skilled professionals creates a significant bottleneck in the ability to detect, analyze and respond to security incidents in a timely manner. This challenge becomes even more critical in a world where virtually every process (economic, social, and institutional) relies on fragile and interconnected digital infrastructures. These elements clearly outline the need for intelligent and autonomous tools that can reduce the workload on human analysts and scale with the growing volume of digital evidence. To answer this call for automation, this thesis leverages the capabilities of Large Language Models (LLMs) to support cybersecurity forensics. I design the agent not only to reduce the burden on human analysts, but also to increase the consistency of the analysis. I hope that, in doing so, my contribution could help reducing the gap between the growing complexity of cyber threats and the limited availability of specialized human resources, paving the way for more scalable and intelligent security operations.

1.3 Main contribution of the thesis

The purpose of this research is to investigate the effectiveness of different AI agent architectures in the forensics analysis. Specifically, the main goal is to evaluate how architectural choices, such as memory management and tool integration, affect the agent's ability.

The project builds upon a previous work [7] that introduce a baseline forensic agent evaluated on a benchmark of near-real-world cybersecurity events. The objective of this thesis is to improve the results of the agent on the same benchmark by allowing it to overcome the limitations pointed out in that study. Specifically, the original work highlighted two main shortcomings:

1. **Lack of context window management**: some executions aborted because input tokens were higher of model's context length

2. **Inability to extract meaningful evidences:** in many cases, the agent failed to thoroughly explore the network traffic, resulting in incomplete analysis and inaccurate conclusions

To address the first issue, the refined agent proposed in this work adopts a more advanced prompting mechanism inspired by MemGPT [34]. This approach introduces a dynamic memory management strategy, where only the most relevant elements of the current reasoning process are preserved over time. In particular, the agent maintains a structured queue that is incrementally updated at each reasoning step, allowing it to carry latest information without exceeding context limitations. Additionally, in order not to lose relevant details early in the reasoning process, the agent has the possibility to store relevant details in a vector database. This information are retrieved at each step based on the context and the entire prompt is constructed to make it as much meaningful as possible, while being sure that the number of tokens is under a predefined limit.

To overcome the inability to extract meaningful evidences, the proposed solution adopts a multi-agent architecture that coordinates several specialized components to support forensic analysis. This includes dedicated agents and tools for network traffic inspection, Docker log exploration, and external knowledge retrieval. Each agent is responsible for a specific task and collaborates within a structured workflow, enabling a more thorough and modular investigation process.

As a result, the refined agent is not only capable of handling longer inputs but also more effective in extracting insights from complex forensic scenarios.

These improvements result in an agent that not only performs better on the given benchmark, but also opens new possibilities for future applications in real-world forensic workflows. The architectural choices made in this thesis, such as dynamic memory management and multi-agent coordination, represent a step toward building more autonomous and reliable AI systems for cybersecurity tasks.

1.4 Methodology

The methodology adopted in this thesis combines multiple approaches. First, a comprehensive literature review was conducted to understand the current landscape of LLM-based agent architectures and their application in cybersecurity and reasoning tasks. This includes reviewing recent advances in prompt engineering, memory management and multi-agent systems.

Subsequently, a first working version of the system was developed in order to reproduce the result of the baseline agent [7]. This served as a reference point for identifying performance gaps and architectural limitations. The implementation was based on an open-source framework called LangGraph, which provides a flexible environment for defining reasoning workflows, managing stateful agents and integrating external tools in a modular way.

The final version of the agent was then built incrementally, with architectural decisions driven by insights gained from an in-depth analysis of intermediate results. At each stage, the objective was to introduce new components, or change current ones, in order to address specific weaknesses observed during experimentation.

Each version of the agent was extensively evaluated against the benchmark, allowing for a clear comparison of approaches and a deeper understanding of their strengths and limitations. This iterative and experiment-driven approach ensures that each choice was driven by practical evidences rather than theoretical assumptions alone. By progressively refining the agent and assessing the impact of each architectural change, the research was able to converge toward a solution that balances performance and scalability. The entire path from the baseline to the final solution, with all their respective drawbacks and strengths, is discussed in detail in Chapter 4.

1.5 Thesis structure

The thesis is structured into six chapters, as follows:

- **Chapter 1 – Introduction**: Presents the context, motivation, main contributions and structure of the thesis.
- **Chapter 2 – Background**: Provides a literature overview for what concerns LLMs, agent and multi-agent architectures, prompting strategies, frameworks for agents development and evaluation.
- **Chapter 3 - Related works**: Reviews existing works that have addressed similar challenges in the field of AI-driven forensic analysis.
- **Chapter 4 – Method**: Describes in details the approach adopted in the design of the main architectures.
- **Chapter 5 – Experimental/numerical Evaluation**: Presents the results of all the experiments conducted and the insights gained from the in-depth analysis.
- **Chapter 6 – Conclusion and Future Work**: Summarizes the findings and outlines possible directions for extending the research.

Chapter 2

Background

AI agents are software entities that leverage Large Language Models (LLMs) to perform a task. Agents try to pursue a goal by iterating and taking decisions based on their internal knowledge, the tools they have been provided with and the visible outcome of previous decisions.

2.1 LLMs

Large Language Models (LLMs) are deep learning models trained on a vast amount of data for advanced language processing. Text generation, machine translation, summary writing, chat-bots or conversational AIs are all possible examples of application of LLMs. These models are typically pre-trained using self-supervised techniques, where the model learns to predict missing words or tokens in context. However, self-supervision is only the first step: LLMs are often further refined using supervised fine-tuning on curated datasets, and aligned with human preferences through Reinforcement Learning from Human Feedback (RLHF), enhancing their safety, coherence, and usefulness in real-world tasks.

2.1.1 Capabilities and limitations of LLMs

Through extensive training, LLMs such as GPT-4o have gained the ability to distill long and complex texts into concise summaries [5] [27]. They can contextually prioritize key information, making them highly valuable for professionals dealing with dense content such as legal documents, academic papers or technical reports [11].

Another notable strength is their versatility in creative writing tasks. LLMs can produce initial drafts for blogs, stories or articles, adapting to different tones and narrative styles [3] [21].

In the field of software development, LLMs assist with code generation, debugging and optimization. Thanks to their training on large codebases, they understand programming patterns across multiple languages and can offer relevant suggestions based on the developer's intent [51].

Despite the aforementioned strengths, LLMs also exhibit significant limitations. One of the major concern is called hallucination [50], where models generate text that appears coherent with respect to the context and factually correct, but it is actually inaccurate or entirely fabricated. This stems from their design, which predicts likely tokens sequences without an internal fact checking system.

Moreover, LLMs often struggle with tasks requiring complex reasoning and causality understanding, such as multi-step logical problems or mathematical operations. Their outputs are based on pattern recognition rather than true understanding of rules and semantics [49] [10].

Security is another critical challenge in this context. LLMs are susceptible to attacks such as prompt injection [57], jailbreaks [14] and data poisoning [12]. These vulnerabilities can lead to harmful behavior, biased responses or the unintentional disclosure of sensitive information. Biases inherited from training data can also affect fairness in sensitive domains like recruitment or legal analysis.

One of the main concern of the growing popularity of LLMs is their energy consumption, not only for training, but also at inference time. Obviously this issue is strictly related to the carbon footprint of such technologies, raising environmental sustainability questions. Additionally, the high computational demands of large models translate into substantial operational costs, particularly when relying on API calls for inference.

To conclude, one of the most pressing challenges with LLMs is their lack of interpretability. As highlighted by Murdoch et al. [26], interpretation in machine learning refers to the *extraction of relevant knowledge from a model concerning relationships either contained in the data or learned by the model.* When dealing with LLMs with billions of parameters, they typically act as a black box, receiving an input and providing an output. Interpretability is more than a theoretical concept in high-stakes domains. For instance, in healthcare, a reliable interpretation can help understanding whether a medical diagnosis made by an LLM is trustworthy or not. Similarly, in cybersecurity, the stakes are high: a false negative may result in an undetected attack, compromising sensitive systems, whereas a false positive may lead to wasted resources, but is typically less harmful. In such scenarios, knowing why a model reached a certain conclusion is crucial to build trust, guide human decision-making and implement appropriate countermeasures.

2.2 AI agents

The potential of LLMs extends beyond writing summaries, essays and code, they can be framed as powerful and organized general-purpose problem solver called AI agents.

2.2.1 System overview

In a LLM-powered autonomous agent system, the language model functions as the agent's brain, supported by several key components:

- **Planning**

The agent is capable of decomposing complex tasks into smaller, more manageable subgoals, allowing it to operate in a structured and efficient manner. Furthermore, it can engage in self-reflection, analyzing the outcomes of previous decisions to refine its strategies and improve future performance.

- **Memory**
  Agents typically employ two types of memory:
  - *Short-term memory*, which includes mechanisms like prompt chaining, allowing the agent to keep track of recent interactions and adjust its behavior accordingly.
  - *Long-term memory*, which enables persistent knowledge retention across sessions by integrating external storage systems (e.g., vector databases) for fast and scalable information retrieval.
- **Tool Use**
  Agents can interact with external tools and APIs to access information beyond their internal training data. This includes querying databases, performing code execution and retrieving real-time knowledge on the web, extending the agent's functionality and adaptability.

2.2.2 Planning

A complicated task is typically faced by decomposing it into many steps, which requires the agent knows who is it and plan ahead. Many prompting techniques have been introduced to guide the model in decomposing a complex task:

- **Chain of Thoughs** [48]: The model is explicitly instructed to think step by step to transform big tasks into multiple manageable tasks.
- **Tree of Thoughts** [52]: It extends CoT by exploring multiple reasoning possibilities at each step. It generates multiple thoughts per step, creating a tree structure. The search process in the generated tree can be performed through BFS (breadth-first search) or DFS (depth-first search).

2.2.3 Self-reflection

Self-reflection is fundamental to make an agent improve its actions by iteratively refining past decisions and correcting previous mistakes. It plays a crucial role in real-world scenarios where trial and error are inevitable.
ReAct [53] integrates reasoning and act within LLMs. It allows the model to both generate natural language reasoning steps and perform concrete actions, such as calling external tools. By doing so, it can explain its thought process while interacting with the environment to gather new information, improving both transparency and effectiveness in solving complex tasks.

The ReAct prompt template incorporates explicit steps for the LLM to reason and act, and is typically structured as follows:

```
Thought: ...
Action: ...
Observation: ...
```

This cycle is repeated multiple times until a final answer or conclusion is reached.

2.2.4 Tool use

LLMs tool calling comprises mainly 3 stages:

1. **Task planning**: the LLM, acting as the brain of the system, parses the action into a tool call with related and required arguments. At this stage, the LLM is required to understand, based on the description of the tool and the required arguments, whether invoking it is beneficial for completing the task.

2. **Task execution**: the tool called executes its function and returns the corresponding results, including an appropriate description of errors, if any

3. **Result integration**: the LLM receives the output of the tool, appends it to the prompt of the following step. it is used to decide whether to continue the reasoning process or conclude the task.

Despite their impressive capabilities, LLM-powered agents can be computationally expensive, especially when multiple reasoning steps, external tool calls or memory lookups are involved. Indeed, the aforementioned result integration step often requires to input many words (that translate to tokens) in the model. This not only increases operational costs and energy consumption, but also introduces latency and can push the system toward the limits of the model's context window, potentially degrading performance or truncating important information.

Such challenges highlight the need for efficient memory management strategies, prompt engineering techniques and careful architectural design to ensure scalability and sustainable deployment of AI agents in real-world applications.

2.2.5 Autonomy and safety trade-offs

Recent research increasingly focuses on implementing fully autonomous AI agents that do not require continuous supervision. Thus, they are provided with a task, a limit on the amount of iterations in their reasoning process and have to provide an answer. Such agents would be particularly powerful in terms of task automation, especially to improve performance of humans. However, increasing the autonomy of AI agents in dynamic environments introduces risks in terms of misalignment, unexpected behaviors and unsafe tool usage.

A common mitigation strategy involves Human-in-the-Loop (HITL) configurations or adjustable autonomy levels, where humans retain decision authority over critical actions or final outputs. Additionally, it often happens that an LLM gets stuck in a loop without the ability of understanding the next step. This may happen because it lacks the concept of causality or because it is not able to highlight relevant details. In such scenarios, it is particularly useful to introduce a human-in-the-loop that is able to guide the agent in the correct direction.

2.3 Multi-Agent paradigm

In a real-world scenario, multiple decision-makers often need to interact in order to solve a complex task. A Multi-Agent System (MAS) provides an accurate and effective abstraction of this approach.

In a LLM-MAS application, several generative agents can collaborate and communicate with one another or being coordinated by an orchestrator (which is itself an agent communicating with the others), which delegates sub-tasks to the most suitable sub-agent.
As stated by Chen et al. [6], the communication of agents has two different purposes:

1. Achieve collaboration, obtaining performance that goes beyond the single agent
2. Achieve consensus, enabling faster convergence

The type of communication content changes: it could be natural language or custom content such as a vector or a discrete signal that does not allow for interpretability as it can be understood only by the generative agents in the system.

Multi-agent systems can be broadly categorized based on their coordination strategy:

- **Centralized coordination**
  A single orchestrator (possibly a generative agent) oversees the planning, task delegation and integration of outputs from sub-agents. This design simplifies global reasoning but may become a bottleneck or single point of failure.
- **Decentralized coordination**
  Agents communicate peer-to-peer and negotiate sub-tasks without a central controller. This architecture is more robust and scalable in dynamic environments but poses challenges for convergence, alignment and conflict resolution.

In case the multi-agent system makes use of an orchestrator, sub-agents can be conceptualized as advanced tools or services that are invoked by the orchestrator to handle specific sub-tasks. These sub-agents may be either generative or non-generative, like tools for retrieval, classification or logic-based execution. Similarly, the orchestrator itself can be generative or not: it may be implemented as a language model capable of planning and prompting, or as a static controller using predefined rules or graphs. Regardless of its nature, the orchestrator is typically responsible for coordinating agent

interactions and integrating their outputs to achieve coherent global behavior.

There are two main problems related to LLM-MAS:

- **Efficiency explosion**
  LLMs are inherently slow at inference time because of their autoregressive nature. In generative multi-agent systems, this inefficiency is amplified because each agent may need to query the LLM multiple times per action. As the number of agents increase, this overhead grows significantly

- **Accumulative error effect**
  In LLM-MAS, the output of each round influences the input of the following one. Early errors propagate and compound over time, leading to degraded performance in later stages

Moreover, evaluating the behavior of a group introduces another challenge in Multi-Agent systems. In complex settings, it is important to assess both the quality of inter-agent communication and the effectiveness of the result they return with respect to the assigned sub-task. This often requires inspecting long execution logs to determine whether each sub-agent correctly interpreted its role when invoked and to verify if its response aligns with the original task requirements.
Multi-agent systems are being actively explored for real-world applications such as collaborative scientific discovery, distributed customer support, complex multi-modal planning and autonomous system orchestration. Frameworks like *AutoGen* and *CrewAI* allow to deploy such systems in practice.

2.4 Frameworks for agents development

There are different technologies available for developing AI agents that leverage Large Language Models (LLMs) to perform complex tasks. The selection of a specific framework influences the level of flexibility, scalability and the ease of integration with external tools or services. In the context of this project, LangGraph, a framework built on top of LangChain, was adopted to design the AI agent. LangGraph introduces a graph-based abstraction, where each step in the agent's reasoning process is represented as a node within a directed graph (DG). This architecture supports features such as conditional routing, which allows to manage even complex workflows. Nevertheless, there are several other alternatives, each offering different capabilities and abstraction levels. The following is a more in-depth analysis of the most prominent existing technologies.

- **LlamaIndex**
  LlamaIndex stands out as one of the most adopted frameworks for agents development. It is specifically designed for context-augmented LLM applications. It enables agents to access external data beyond their initial training sets, facilitating more informed and accurate responses. The most popular example of

context-augmentation is Retrieval-Augmented Generation or RAG, which combines context with LLMs at inference time. The framework is available in both Python and TypeScript, offering user-friendly tools for beginners as well as advanced customization capabilities for more experienced developers. LlamaIndex is an open-source project, fostering a collaborative environment for developers worldwide. To support the transition from development to production, LlamaIndex provides LlamaCloud, an end-to-end managed service available both as a hosted solution on their infrastructure and as a self-hosted deployment. Among its components, LlamaParse serves as the document parsing engine, enabling the extraction and structuring of data from various document formats. Users can sign up and parse up to 1,000 pages per day for free, with the option to scale usage through a pay-as-you-go model by adding a credit card. It is ideal for situations where an AI agent needs to understand natural language requests and be able to draw information from a library or index, like a knowledge base, such as chatbots, knowledge assistants, or question-answering systems. However, the setup can be slow, with developers needing to undertake data organization tasks, such as adding filters and analyzing any logs to check agent behavior before the system is truly optimized.

- **AutoGen**
  AutoGen is an open source framework by Microsoft for developing AI agents and facilitating cooperation among multiple agents to solve tasks. It offers extensive customization capabilities, along with AutoGen Studio, a no-code, drag-and-drop interface designed to support users who may not be proficient in programming. Extensive documentation is available online, including hands-on tutorials, examples and a rich library of pre-built, open source agents that can be adapted and customized to many different use cases. The documentation available online is extensive and provides many tutorials and open source pre-built agents that can be easily customized. The framework is currently compatible with both Python and .NET, enabling cross-language agent collaboration and broader integration within enterprise environments. AutoGen's architecture is based on an event-driven messaging infrastructure, supporting asynchronous communication and multiple message types (e.g., prompts, results, code execution, etc.), which simplifies the orchestration of sophisticated multi-agent workflows. From a deployment perspective, AutoGen supports both local hosting and enterprise-scale cloud deployment, offering integration with Microsoft Azure services for secure, production-ready workflows. Microsoft also provides additional tools to enhance AutoGen-based applications:

  – **Azure AI Studio** to deploy LLM-powered applications
  – **Azure OpenAI** to integrate GPT models in a secure environment
  – **OpenTelemetry** for observability and debugging of multi-agents applications

  AutoGen is ideal for tasks requiring multiple AI agents, such as customer support, data analysis or IT support. While optimized for Microsoft tools, it's open source.

The simple UI and ability to use the 'low-code' studioGen appeals to developers looking for a standardized, modular framework for creating intelligent agents. The designed architecture is extensible, allowing users to customize systems with pluggable components. However, it offers less flexibility in designing custom logic, agent autonomy and intricate workflow than other frameworks.

- **LangChain - LangGraph**
  LangChain is the most widely adopted open-source frameworks for developing applications powered by Large Language Models (LLMs). It is designed to facilitate the creation of context-aware and tool-augmented agents. LangChain is particularly well-suited for building agents that require step-by-step reasoning, integration with external APIs or access to long-term memory. The framework supports both Python and JavaScript/TypeScript, offering an high flexibility level. LangChain also includes native support for key LLM capabilities such as tool usage, retrieval-augmented generation (RAG), function calling and agent execution. LangChain is often used in combination with LangGraph, a complementary library that introduces a graph-based architecture for managing complex workflows through directed graphs (DGs). This integration allows developers to define more complex multi-agents architecture or conditional routing to support also parallel execution. For deployment, LangChain applications can be hosted locally or in the cloud. Additionally, LangChain offers LangSmith, a dedicated observability and debugging platform that enables developers to trace, evaluate and fine-tune their LLM workflows. Also in this case, LangChain's documentation is available online with many hands-on tutorials and pre-built agents that can be customized and connected to generate complex behaviors. The project is supported by a highly active and growing community of developers and contributors, which fosters rapid innovation and support. While LangChain excels in automating complex processes and aids users with prompt engineering and modular development, it's often noted as complex for beginners, requiring a significant time investment to learn the nuances.

- **Botpress**
  Botpress is an all-in-one platform for building AI agents powered by LLMs, specifically introduced to facilitate the development and deployment of chatbots within self-hosted websites. Given the specificity of the task Botpress is designed for, the documentation is mainly related to examples and tutorials for realizing chatbots and customizing their interface to ensure visual consistency with the host application. Botpress includes features like intent recognition, slot filling, multi-turn conversations, and memory management, making it suitable for a wide range of use cases from customer support to lead generation. It also provides a graphical interface that allows developers and non-technical users to visually design conversational flows, making it highly accessible for rapid prototyping and production use. Botpress has recently evolved to support LLM-based agents, offering a hybrid approach that combines rule-based logic with generative AI capabilities. Developers can define workflows and conditions for generation, retrieval or decision-making.

It also supports integrations with third-party tools and APIs. The platform is open-source and self-hostable, but also offers a cloud version with advanced features like analytics, monitoring, and versioning that is called Botpress Studio. The community is active, although its primary focus remains in chatbot applications rather than general-purpose agent architectures. Botpress relies on rigid, predefined paths, which can be effective for simple queries or highly structured workflows. This is especially true for enterprises looking to get started with AI-assisted customer support, such as automating customer service, answering FAQs, and offering live chat. While effective for simple queries or highly structured workflows, unanticipated questions or scenarios can be a challenge. This framework may appeal to enterprises keen to start offering customer support, but scalability may be an issue.

- **CrewAI**
  CrewAI is a multi-agent platform built for enterprises, which allows to orchestrate powerful AI agents with ease and scale. Custom agents can be defined for data research, automation or generation, assigning, depending on the context, different tasks to be accomplished. The platform is specifically developed to build AI agents that work together to tackle complex tasks. It is completely developed in Python and it is independent of LancgChain or other agent frameworks. CrewAI provides two distinct paradigms for building agents: Crews and Flows. The first one allows for an high-level abstraction, where each agent is assigned a role, a goal and, optionally, a tool, the Crew class handles orchestration. The latter allows for a more fine-grained control at a lower level, where Flows gives full control over the execution path, task sequencing, and condition-based branching. It is possible to define a flow of tasks, each executed by a specific agent or LLM call, allowing for fine-tuned logic and inter-agent communication. Additionally, CrewAI allows to easily integrate tools created with LangChain or LlamaIndex, or even to develop custom tools. Among available frameworks for agent development, CrewAI stands out for its rich and comprehensive documentation, offering tutorials and examples for virtually every possible use case. Although it lacks a native observability layer, it can be integrated with external tools for logging and monitoring (such as LangSmith if using LangChain tools). Being fully open-source, CrewAI is an ideal choice for developers and researchers looking to build customizable and autonomous AI teams without vendor lock-in For enterprises, CrewAI provides an Enterprise Edition that includes advanced features such as orchestration dashboards, integrations with over 700 applications, enhanced observability, and deployment tools. Recently, CrewAI Studio has been introduced, a no-code interface designed to simplify the development and orchestration of AI agents and multi-agent workflows. While CrewAI Studio enhances accessibility, it is important to note that it requires an enterprise subscription. Despite being relatively new, CrewAI has quickly fostered an active and supportive community, with regular updates and open-source contributions. For defined goals and tools and teams familiar with task orchestration and AI agent workflows, it is a great framework

that can integrate with a variety of external tools to gather data, analyze trends and generate reports or be used to manage projects. It can be used for tasks like content creation, with research and writing articles done by specialized agents, or project management, breaking down complex projects into smaller tasks, assigning tasks to team members, and tracking progress. CrewAI excels in structured workflows but lacks adaptability for tasks requiring autonomous decision-making or real-time role reassignment.

Framework	Year	Language	Key Features	Weaknesses	Stars
LLamaIndex	2022	Python, TypeScript	Advanced document indexing; integrations with external data; document parser (LlamaParse)	Struggles with real-time updates, frequent index refreshes	40.9K
AutoGen	2023	Python, .NET	Multi-agent orchestration; human-in-the-loop; event-driven execution	Less flexible for enterprise-scale tasks requiring reasoning	43.2K
LangChain	2022	Python, JS/TS	Modular; third-party tool integration; full observability	Limited debugging; complex for beginners	106K
Botpress	2017	JavaScript	Hybrid logic engine (LLM + rules); NLU support	Rigid responses; poor contextual understanding	13.5K
CrewAI	2024	Python	Agent grouping (roles/goals/tools); optimized for collaborative agents	Rigid task assignment; less adaptable	30.1K
LangGraph	2023	Python, JS/TS	Recursion control; supports long-running agents with RAG	Too complex for simple agents; hard for beginners	11.5K

2.5 Development framework

Among the many frameworks analyzed, LangGraph was selected as the core technology for this project due to several compelling reasons.

First and foremost, LangGraph builds upon the robust and widely adopted LangChain ecosystem, allowing seamless integration with pre-existing tools for retrieval, tool use, memory, and function calling. Its graph-based architecture introduces a highly modular and flexible way of designing agents, where each node represents a step in the agent's reasoning process. This structure makes it easier to define conditional logic, recursion, and parallel workflows, which are essential for complex agent behavior.

Another major factor in the decision was the availability of a large number of pre-built components. These significantly reduce development time and allow developers to focus on the logic and reasoning capabilities of the agent rather than on infrastructure or boilerplate code.

LangGraph also benefits from being part of an active and fast-growing community, which ensures continuous improvements, regular updates, and quick support through forums, GitHub discussions, and community-driven resources. The comprehensive documentation and numerous open-source examples made onboarding and experimentation significantly more efficient.

Finally, LangGraph offers a high degree of customization. Developers can fine-tune agent workflows at every level—from memory handling and tool integration to the definition of execution paths and failure handling strategies—making it a highly versatile choice for both research and production use cases.

In summary, LangGraph was chosen not only for its advanced technical capabilities, but also for the strong ecosystem and support it provides, making it a future-proof and scalable foundation for the development of intelligent, multi-step agents.

2.6 Agent evaluation

Evaluating the performance of an agent is a key point for improvement and it's also critical to understand its behavior in a real-world scenario. There are different possible evaluation strategies than can be used, most of them rely on the creation of ad-hoc benchmarks.

These benchmarks differ depending on the specific capabilities being assessed in AI agents. For example, some researchers focus on evaluating Self-Reflection, Memory or Planning capabilities (sometimes in combination) [54].

In the context of AI agents for Cybersecurity forensics, the objective was to evaluate the agent's effectiveness in facing a near-real-world scenario.

To this end, we use CFA-bench [7], a benchmark specifically designed to test AI agents in cybersecurity forensic tasks, which serves as the foundation of our evaluation framework.

Moreover, the agent realized for the aforementioned paper is the baseline of this work. In that case, they evaluated different LLM-powered agent architectures:

- **ReACT Reasoning** that requires to the LLM, for each step, to provide both a thought and an action for the next step. The environment then responds with an observation that is appended to the Scratchpad
- **ReACT + Summary** is built upon the same concepts of the Reasoning agent, but the LLM is also required to report a summary of the process till the ongoing step. This summary becomes the new input, so that the context window is not overcame by too many tokens
- **Decoupled** where the idea is to enhance reasoning capabilities by decoupling thought and action at each step

The agents, in their different configuration, were required to provide a final report highlighting 4 key elements:

1. Affected service
2. Service vulnerable (True or False)
3. Attack successful (True or False)

4. Identified CVE

Thus, the agent is first required to find the name of the service under evaluation and the type of attack being attempted by the adversary. Then, it must assess whether the service is effectively vulnerable to that type of attack and whether the attack was successful.

This benchmark simulates a near-real-world scenario, as the PCAP files contain network traffic already filtered to include only the service of interest, complemented by Docker log files, i.e., the application logs produced within the corresponding Docker containers. Therefore, this setup can be interpreted as a post-identification stage, where an issue has already been localized to a specific service, and the agent is expected to conduct further investigation in a fully autonomous manner. However, several cases in the benchmark would present a significant challenge even for a human expert encountering them for the first time. Indeed, understanding some of them would require access to the system to perform some type of conclusion, and current agents are not meant to do so. Moreover, determining that a service is not vulnerable to a specific CVE based solely on an observed attack attempt is not always straightforward. This often requires precise knowledge of the deployed software version and a deep understanding of the nuances between different vulnerabilities affecting the same service.
The list of all the evaluated incidents is reported in Table 2.1.

Table 2.1: List of collected incidents for benchmark.

Service	**CVE**	**Vulnerable**	**Attack success**
Couchdb	CVE-2022-24706	True	True
Grafana	CVE-2021-43798	True	True
Apache HTTP Server	CVE-2021-41773	True	True
Apache HTTP Server	CVE-2021-42103	True	False
Jenkins	CVE-2024-23897	True	True
Joomla	CVE-2023-23752	True	True
Apache Solr	CVE-2021-44228	True	True
phpMyAdmin	CVE-2016-5734	True	True
phpMyAdmin	CVE-2018-12613	True	True
Cacti	CVE-2022-46169	True	True
Apache Airflow	CVE-2020-11981	True	False
Apache Airflow	CVE-2020-11981	True	True
SaltStack	CVE-2020-11651	True	True
SaltStack	CVE-2020-11651	False	False
Apache APISIX	CVE-2021-45232	True	True
Apache APISIX	CVE-2021-45232	False	False
Apache ActiveMQ	CVE-2017-15709	False	False
Apache ActiveMQ	CVE-2017-15709	True	True
GitLab	CVE-2021-22205	False	False
GitLab	CVE-2021-22205	True	True

My approach was to iteratively improve the agent guided by the observations made on the tests performed on the aforementioned benchmark. Thus, in order to understand whether the architecture was effective also in other scenarios we collected a set of new events. We decided to limit the dimension of our new independent evaluation set to 10, including only vulnerabilities discovered in 2025, so that the chance of being in the agent's knowledge was lower. This allowed to evaluate the actual effectiveness of our findings while also ensuring the generality of the final proposed solution. The objective was to demonstrate that, by iteratively adapting our agent to better face the benchmark, we did not overfit on its characteristics. The set of new events is reported in Table 2.2.

Table 2.2: List of newly collected incidents (2025).

Service	**CVE**	**Vulnerable**	**Attack success**
Erlang/OTP	CVE-2025-32433	True	True
Erlang/OTP	CVE-2025-32433	False	False
Vite	CVE-2025-30208	True	True
Vite	CVE-2025-30208	False	False
Next.js	CVE-2025-29927	True	True
Next.js	CVE-2025-29927	False	False
Langflow	CVE-2025-3248	True	True
Langflow	CVE-2025-3248	True	True
Langflow	CVE-2025-3248	False	False
Tomcat	CVE-2025-24813	True	False

Chapter 3

Related works

In recent years, the growing complexity and scale of cyber threats have exposed the limitations of traditional approaches to digital forensics and incident response. As these challenges evolve, Artificial Intelligence has started to play a key role in supporting and enhancing forensic investigations. Among the most promising developments are AI agents, autonomous systems capable of analyzing evidence, reconstructing timelines and generating detailed forensic reports. These tools are designed to boost analyst productivity, speed up response times and bring greater consistency to investigations.

This section explores current work at the intersection of AI, cybersecurity and digital forensics, with a particular focus on autonomous agents. It looks at early applications of LLMs in forensic analysis, recent efforts to combine large language models with rule-based systems and the growing interest in agent-based architectures for automating security tasks. Moreover, it touches on some of the key challenges ahead, including concerns around safety, transparency and governance when deploying these systems in sensitive or high-stakes environments. Finally, it presents a brief overview about evaluation benchmark for autonomous agents in this context.

3.1 Digital forensics and LLMs

In the world of cybersecurity, Large Language Models (LLMs) are positioned as tools to assist rather that replace human investigators. In particular, in the context of forensic analysis, it is often difficult for experts to explore in detail the huge amount of traffic they are exposed to. Thus, it is fundamental to have access to an instrument that can facilitate this task.

This research field is relatively recent in time. One of the earliest attempt was made evaluating directly ChatGPT across a range of tasks, including interpreting digital artifacts [37]. In this study by Scanlong and Breitinger (2023), their conclusions highlight both the potential and limitations of LLMs in this domain. On the positive side, ChatGPT can assist experienced analysts by accelerating routine tasks and offering educational support. However, they also emphasizes significant risks: hallucinations, limited reasoning capabilities and lacks of data privacy. The authors conclude that, while LLMs

hold promise as forensic assistants, they require careful supervision and are not yet reliable for unsupervised investigations in critical environments. Obviously, this was a first trial to open a new research direction, since prompting directly ChatGPT to evaluate its performance makes it difficult to replicate results or make conclusions fully reliable. However, it was fundamental to demonstrate the first versions of generalist huge models have knowledge in the context of cybersecurity and can actually assist forensic experts.

3.2 Specialized models

Following the direction of the previous work, where the capabilities of an LLM are directly evaluated when collaborating with a human, Sharma et al. proposed a more specialized model for digital forensics: ForensicLLM [55]. Specifically, to address the accuracy and privacy issues of general LLMs,they deployed a custom 8-billion-parameter model fine-tuned for digital forensics. ForensicLLM is built on Meta's LLaMA and trained on an extensive corpus of more than 1,000 forensic research papers and artifact metadata using a retrieval-augmented fine-tuning (RAFT) pipeline. It is also provided with a vector database as a tool to fetch relevant domain knowledge. Also in this case, the model acts as a Q&A assistant for a forensic analyst that is performing an in-depth analysis in network traffic. Despite not being specifically tailored for reducing the overhead required for the analysis of a huge amounts of data, it is still a solution improving the quality of the results. According to author's evaluation, their strategy significantly reduced hallucinations and factual errors compared to a general LLM. However, the model is not fully autonomous, and an 8B-parameter local model may not match the raw performance of larger proprietary LLMs. Its contribution lies in demonstrating a practical, privacy-preserving assistant tuned for forensic analysts, with retrieval techniques to ground its responses in verified forensic knowledge.

3.3 Autonomous agents

GenDFIR [56] is a framework that integrates a rule-based expert system with an LLM-based agent to automate cyber incident timeline forensics. In this work, data (logs, file timestamps, etc.) go through an expert system, where if-then-else conditions have been defined by rule engineers. This reduces the data volume and focuses the analysis mainly on suspicious artifacts. Finally, an LLM with Retrieval-Augmented Generation (RAG) processes those filtered data, treating them as a knowledge base for the LLM. The LLM agent is prompted to act as an autonomous digital forensic analyst, tasked with correlating events and even suggesting likely incident causes or outcomes. This work introduces an important concern related to the amount of data that are required to be analyzed, which is complex not only for humans, but also for LLMs when considering costs and focus. Indeed, API calls are expensive when the number of input tokens grow too much, while hosting a local model can alleviate this problem, it is still convenient to reduce input tokens to keep inference times constrained. Moreover, also the ability to fully grasp important details is strictly related to the amount of data to be analyzed.

These considerations highlight the importance of designing agents that are not only accurate, but also resource-efficient, capable of prioritizing relevant information without being overwhelmed by the scale of forensic data.

The main limitation of this approach is the same that distinguishes rule-based systems: they rely on predefined rules. In the context of forensic analysis, it is important to keep updated with a fast-evolving landscape: novel attack patterns such as zero-day exploits or obfuscated techniques emerge often. In this case, a rule-based system may fail to detect them unless new rules are added. Updating these rules is not only time-consuming, but also error-prone, as overlapping or conflicting conditions may reduce system performance. Moreover, such programs often struggle to generalize across unseen scenarios, limiting their adaptability and generality in a real-world use-case.

Despite progress towards autonomy in many recent works, most of the practical implementations used in real use-cases, still rely on human-in-the-loop designs. This is the case of the agent developed by Microsoft Research in 2023 [17], which is intended to support cloud operations engineers during incidents. In this case, rather than acting autonomously, the system is able to reason and propose hypotheses, suggesting also possible next steps that have to be authored by a human.

In particular, they designed a modular agent with three main LLM-driven components: a Hypothesis Generator (given evidences, proposes possible causes), a Hypothesis Tester (suggests commands to be run in order to verify the previous hypothesis) and a Mitigation Planner (recommends fix actions once the cause is confirmed). The system is designed in such a way that, in order to run the mitigation step, the supervisor engineer has to check and confirm it.

The difference in this case is that decisions and actions are all left to human operators, ensuring accountability in this context. This is fundamental when working with mission critical systems, where even minor errors can have significant consequences. This is the direction of most of the works in the context of AI agents applied to real use-case scenarios: enhance performance while still prioritizing safety and control. As such, hybrid approaches that combine automated reasoning with human validation are increasingly seen as the most viable path toward integrating LLM-based agents in real-world operations.

3.4 Multi-agent collaboration for incident response

In the context of fully autonomous agents, Zefang Liu developed a multi-agent system to handle incident response. In his work [24], multiple agents collaborate like a cyber defense team to face a threat. The study experimented with different team structures: centralized teams (an Incident Captain agent directs others), decentralized teams (agents work more independently) and hybrid setups, to see which coordination style yields the best outcome in the simulated incidents. The results showed that, when using a multi-agent paradigm, decision-making improves with effective communication. In particular, since in that case the plan is provided by one single agent, the centralized approach showed more coherent actions. In the other hand, decentralized agents showed more

diverse exploration of hypotheses. However, the author also observed challenges: the agents sometimes miscommunicated or got stuck without human guidance. Indeed, it is complex to make an agent fully adhere to its role and task, without losing focus and overlapping with the others. The results of this research suggest that multi-agent LLMs hold significant promise when a collaborative workflow is needed, as it happens for incident response. However, their effectiveness is still limited if the role definition, as well as the communication method, is not robust enough. To unlock full potentials of specialized agents, the future direction may be to fully constraint their objective to a role and make communication possible only in predefined steps by defining a pipeline of agents.

3.5 Positioning of this Work

In light of this evolving landscape, the contribution presented in this thesis aims to advance and understand how to integrate LLM-based agents into practical forensic workflows. Previous works have designed partially autonomous systems, which required humans to correct and guide the agent or programs working in isolated tasks such as anomaly detection.

In this work the agent operates over structured forensic data while incorporating both retrieval and tool-use mechanisms. Thus, the system is fully autonomous and able to perform a complete reasoning leading to a compact report that can be useful as first guidance for a forensic expert. Moreover, the multi-agent architectures developed are intended to face the limitations highlighted in section 3.4, avoiding communication in each round and bounding the scope of each agent.

The study shows promises and drawbacks of many different approaches, highlighting their key differences in terms of structure and performance. This is done through a clear path of enhancements in terms of architectures and provided tool in order to allow the agent to face the tasks to be performed.

Unlike existing models that rely purely on open-ended prompts or static context, the agent developed here leverages token-aware memory management and a FIFO queue to maintain conversational context within operational bounds. It is further enhanced by semantic retrieval components that allow the agent to recall and reason over past evidence, even when omitted from the active context window. Moreover, the human enters the loop only at the end of the execution, with the possibility to analyze reports and logs, fully understanding each reasoning step performed by the agent. This allows for more flexibility both in terms of usability and human feedback, also going in the same direction as LLM-based agents in real-world operations.

Overall, this work contributes to bridge the gap between experimental agentic architectures and deployable forensic tools, offering a modular and extensible framework for integrating LLM agents in real-world cybersecurity investigations.

3.6 Benchmark for evaluation

There are several benchmarks assessing the capabilities of LLMs in static security-related tasks, such as vulnerability detection and debugging. In my work, I used CFA-bench [7] to evaluate the performance of the agent on 20 different tasks. For each proposed architecture, the agent was run three times to reduce the impact of randomness in executions and an average of the evaluation metrics is computed.

However, there are several benchmarks that have been proposed over years to evaluate agents in specific security tasks. The approaches are different depending on the type of ability that is assessed. In Table 3.1, there is a summary of the main contributions in this areas, highlighting their focus and methodologies.

Table 3.1: Cybersecurity Benchmark Datasets for LLM Evaluation

Benchmark Dataset	**Date**	**Task**	**How they use LLM**	**What they measure**
Securityeval [39]	2022-11-09	Evaluate ML-based code generation on vulnerable examples	Prompt to generate code snippets	Vulnerability detection and patching
LLMSecEval [46]	2023-03-16	Natural language security evaluations	Prompt to generate code snippets	Accuracy on security-related prompts
OWL-Bench [16]	2023-09-17	Operational cybersecurity tasks	Multiple-choice, Q&A	Task-solving ability
NetEval [25]	2023-09-19	Network operations tasks	Multiple-choice, Q&A	Network operations skills
SecEval [20]	2023-12-20	Cybersecurity knowledge assessment	Multiple-choice, Q&A	General cybersecurity knowledge
SecQA [23]	2023-12-26	Computer security QA	Multiple-choice, Q&A	Problem-solving and security knowledge
DebugBench [44]	2024-01-11	Debugging capability assessment	Code snippets pass rate	Bug detection and fix accuracy
CyberMetric [45]	2024-02-12	Cybersecurity QA	Multiple-choice, Q&A	Domain understanding
OpsEval [22]	2024-02-16	IT operations benchmark	Multiple-choice, Q&A	Operations performance
PythonSecurityEval [2]	2024-02-19	Patch security issues in Python code	Prompt to generate code snippets	Patch correctness and impact
Can LLMs Understand Computer Networks? [9]	2024-04-22	Virtual sysadmin tasks on networking	Q&A	Network reasoning and troubleshooting
SECURE [4]	2024-05-30	Cybersecurity advisory generation	Multiple-choice, Q&A	Advisory quality and relevance

Chapter 4

Method

The first version of the agent developed to tackle cybersecurity forensic tasks is inspired by the ReACT Agent proposed with the CFA-bench [7]. In their work, explained more in detail in subsection 2.6, one of the main challenges was managing the Scratchpad characterizing the short-term memory: continuously appending messages from the conversation could lead to a premature abort of the execution. Indeed, they tried to solve this problem by introducing a second version called Summary + ReACT, where the prompt at each step is a summary of the current and all the previous steps in terms of actions and corresponding observations. The problem with this alternative approach is that it makes the agent lose the focus and performance tends to decrease. Among the three versions they implemented, the most promising one was the simple ReACT Agent, which I also adopted as the baseline with some improvements in order to reduce the aforementioned limitation.

4.1 ReACT Agent

As already explained in subsection 2.2.3, the ReACT agent is tasked with providing both a thought and an action as a response to the previous observation. The environment, once the action is executed, responds with an observation which is then appended to the scratchpad. The agent proceeds step by step till it is confident enough to provide a final report or until the maximum number of iterations is reached. In the latter case, the agent is prompted to provide a final answer based on the information gathered so far, even if not complete.

The system is able to interface with data provided and external sources through tools. In this first version, these are the tools available:

- **PCAP Reader**: this tool allows the agent to interact with the network traffic associated with each event stored in a PCAP file by executing `tshark` commands from the command line.

  Specifically, at the beginning of the reasoning process, the agent is provided with

a summary list of the packets extracted from the PCAP using the following command:

```
tshark -r {pcap_file} -T fields -e frame.number -e frame.time -e
frame.protocols -e _ws.col.Info
```

This output includes, for each packet:

- the frame number (used as identifier),
- the timestamp of capture,
- the protocol stack (e.g., eth:ip:tcp:http),
- and the summary line as shown in Wireshark's "Info" column.

These lines are presented to the agent directly in the prompt to provide a compact overview of the entire capture file.

The agent can then call the PCAP Reader tool to access the full content of a specific packet by specifying the corresponding frame number. Internally, the following command is executed:

```
tshark -r {pcap_file} -Y "frame.number=={frame_number}" -T
fields -e data
```

This allows the agent to inspect the payload of any specific frame for deeper forensic analysis relevant to exploit detection.

- **Information Retriever**: a web search tool that allows to complement the internal knowledge of the LLM with external, up-to-date information. To optimize the process, a Retrieval-Augmented Generation (RAG) is used. The agent is encouraged to search for relevant information on the web providing a query (e.g., 'Common Vulnerabilities and Exposures (CVEs) potentially associated with a specific service or a malicious attack'). The RAG leverages a smaller language model to enrich the query using a *query transformation* approach [13]. The resulting expanded query is then used on DuckDuckGO, a search engine to extract content from the top 10 web pages. Each webpage is then segmented into smaller chunks using a sliding window approach, sampling a 512-token segment every 256 tokens. These chunks are embedded into vector representations using the OpenAI text-embedding-3-small model [28]. To select the most relevant content, a semantic ranking is performed by computing the cosine similarity between the query embedding and the embeddings of all chunks. The top-5 ranked chunks are then selected and incorporated into the prompt provided to the main LLM for final reasoning.

The agent is provided with the aforementioned tools and the following prompt to execute the task:

> **Role:** You are a specialized network forensics analyst.
> You are working towards the final task in a step-by-step manner.
> **Instruction:** I will give you the task context, the previous steps, and the last thought, action, and the corresponding observation. By thinking in a step-by-step manner, provide only one single reasoning step in response to the last observation and the action for the next step.
> **Context:** {context}
> **LastStep:** {last_step}

ReACT-style prompt, taken from [7].

4.1.1 Limitations of the ReACT Agent

This version, despite showing promises, is affected by several key limitations, as explained in the reference work [7].
Indeed, the management of the prompt through a Scratchpad causes the abortion of many events in the benchmark because the context window is overcame. Additionally, providing the agent with the full list of packet has two main drawbacks:

- The analysis doesn't even start if the list of packets is extended. This typically happens in realistic use cases, where the length of the list of packets exchanged is greater than the average context window size allowed by the transformer architecture

- The system has to operate blindly opening packets with a few information about them, without even knowing IP addresses involved in the conversation. This often results in calling the `PCAP Reader` tool with random numbers, without a clear execution strategy

Moreover, the retrieval strategy adopted by the RAG system inherently limits the amount of information it can extract from an external web page. In many cases, relevant content may appear both at the beginning and at the end of an article, positions that are distant in terms of chunking. As a result, important details that would be valuable for the agent may not be included in the final prompt. Since chunk selection is based on semantic similarity to the query, semantically distant but complementary chunks are less likely to be retrieved together, even though their combination would significantly enrich the contextual information provided to the agent.

4.2 Baseline, the Single Agent (SA)

The first version of the program has been developed to reproduce the ReACT agent described in Section 4.1 with two main improvements. As anticipated in Section 2.4, the agent has been implemented using LangGraph, which allows to represent it through a graph of nodes and edges. Each node performs a step, stores the result in the State and passes it to the next node depending on the routing defined. Here is the structure of SA:

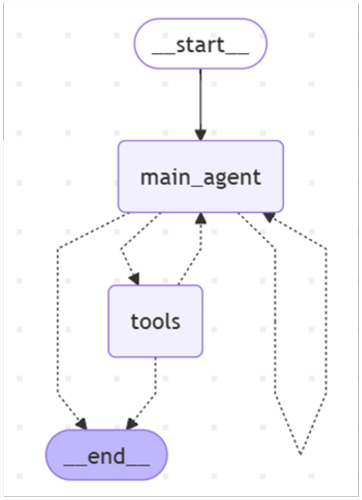

Figure 4.1: SA graph

There are two main nodes in the graph:

1. **main_agent**: this node calls the LLM, it's the brain of the agent. With three optional edges, it may decided to:
   - Call a tool
   - Iterate on itself continuing the reasoning process
   - Stop the execution by reaching the **end** node and returning a result
2. **tools**: this abstract box includes all the tools that the LLM can decide to call based on the task to be performed. In this version, there are 3 tools available:
   - Web search tool, explained in subsection 4.2.2
   - Store memory, deepened in subsection 4.2.1
   - Frame opener, the same provided to the ReACT agent 4.1

Since there is only one intelligent node responsible for planning and taking actions, we call this first version *Single Agent (SA)*.

To overcome the limitations of the first ReACT system, a MemGPT [35] like prompt has been adopted. The objective was to allow the agent to perform complex and lengthy reasoning despite the context window size.

4.2.1 Mem-GPT Prompt

The structure of the prompt has been changed in order to adapt to iterative reasoning processes. In particular, it is realized by taking inspiration from the work done for Mem-GPT [35]. Their OS-inspired design of the memory allows to manage many rounds of conversation or, as in this case, many subsequent steps without nor aborting the execution, neither losing relevant early details in the process.

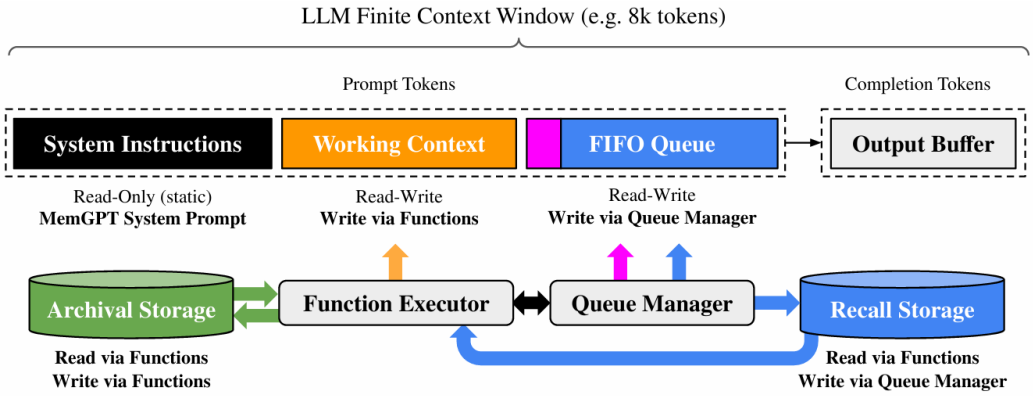

Figure 4.2: High-level architecture of MemGPT, illustrating how memory management is handled through controller logic and memory modules. Figure adapted from [34].

The prompt is divided into three main sections:

- **System instructions**: read-only and fixed instructions provided to the LLM to make it understand the task to be executed, the role and available tools.
- **Working Context**: section containing relevant memories of the conversation. It is updated at each step based on the context and stored messages by the LLM.
- **FIFO Queue**: queue that reports the most recent steps in the reasoning process and flushes away older messages when there's no available space anymore

The agent is provided with the following **System Instructions** section of the prompt:

> **Role:** You are a specialized network forensics analyst.
> You are working towards the final task on a step by step manner.
>
> **Instruction:**
> You are provided with a queue of the most recent steps in the reasoning process.
> Store the most relevant information as soon as you get them, because the queue has a limited size and older messages will be removed.
> By thinking in a step by step manner, provide only one single reasoning step in

response to the last observation and the action for the next step.
You have tools available to search online for CVEs and extract frames from the pcap file.
When you are ready to provide the final answer, stop the reasoning and format the result.

Context: Analyze the provided PCAP (Packet Capture) file to accomplish the following tasks:

1. Identify the name of the service or application involved.
2. Determine the relevant CVE (Common Vulnerabilities and Exposures) based on the captured data.
3. Gather evidence of malicious activities associated with the identified CVE.
4. Assess whether the service or application is vulnerable to the identified attack.
5. Evaluate whether the attack was successful.

Guidelines:

- Begin by thoroughly analyzing the PCAP file to extract relevant information.
- After the exploratory analysis, ensure that the CVE identification is accurate by cross-referencing details from external sources with the evidence found in the PCAP files.
- Use the online search tool only after the exploratory analysis has been completed to verify the findings and gather additional information.

System instruction prompt

The agen is provided with an additional tool that allows it to store memories in a vector database. At each step, the memories are retrieved from the vector database based on the semantic similarity to the last 3 messages in the conversation. To compute embeddings, the OpenAI text-embedding-3-small model is used [28]. The 10 most relevant memories retrieved from the database are then placed in the **Working context** section of the prompt.

The **FIFO Queue** is managed by monitoring the token count of each message. When the accumulated messages exceed the model's context window, the oldest ones are removed sequentially until sufficient space is available.

4.2.2 Improved Web Search Tool

When testing more in depth the web search tool introduced in Section 4.1, I observed that query expansion was not effective in this specific context. In fact, asking an LLM to expand a query without access to any background knowledge often results in a loss of

meaning or the introduction of irrelevant terms. Since the query expander operates in isolation, without access to PCAP data or any additional context, it lacks the necessary understanding of the forensic scenario. As a result, the expanded queries may diverge from the analyst's intent, ultimately degrading the quality of the retrieved web content.

Thus, the refined version of the web search tool has been inspired by an official Open AI guide [30] and using Google's custom search APIs [15] to increase the quality of the results. The Web Search Tool acts as a subagent explicitly looking for CVEs. In particular, each web page retrieved by the search engine is passed to an LLM to be summarized. The Web Search Agent receives the following prompt:

> You are an AI assistant tasked with summarizing content relevant to '{query}' for a forensic analyst
> that is trying to identify the CVE related to a specific service/application under analysis.
>
> Please provide a concise summary in {character_limit} characters or less where you highlight your findings for each CVE detected in the web page.
>
> The summary should be in the following form for each CVE identified:
> `'CVE-XXXX-YYYY: Description of the CVE and its relevance to the service/application under analysis.'`

Ultimately, the LLM is enriched with a final aggregated summary that synthesizes the content of the ten individual summaries, each derived from a distinct web page retrieved during the search. This aggregated summary follows the same structured format defined in the prompt used for summarizing each single page, i.e., a list of identified CVEs along with their corresponding descriptions and relevance to the investigated service or application. This approach enables the agent to reason over a compact and coherent list of vulnerabilities, without relying on raw chunks of text. Compared to traditional chunking-based retrieval methods, where long passages are segmented and may lose semantic coherence, this strategy leverages the language model's summarization capability to preserve critical information and context. As a result, the final summary tends to retain more relevant details, improving both precision and usability in the reasoning process.

4.2.3 Limitations of the Single Agent

This first version of the program was developed to reproduce the results reported in the reference work [7], trying also to address their main limitation: aborted cases caused by the restricted context window.
Although the adoption of a MemGPT-inspired prompt in the Single Agent partially mitigated this issue, the solution proved insufficient in some cases. Specifically, 2 out of 20 events in our reference benchmark failed to start the execution phase entirely, as the number of packets exceeded the available context window when using GPT-4o model [29].

Additionally, one of the main limitation described in the previous section is still present in this agent: it blindly opens packets without being provided with more metadata to perform an informed decision.

This drawback could not be effectively overcame by changing the architecture: the structure of the data and the tools provided had to be adapted to a more realistic use-case scenario.

4.3 Multi-agent architecture, TShark Expert Agent (TEA)

Given the limitations highlighted in the previous section, the system has been enhanced to avoid appending the full list of packets to the prompt.

In particular, the architecture changed towards a multi-agent structure. The rationale was to manage more complex workflows needed to allow the agent executing arbitrary TShark commands through a more powerful system.

I observed that, when providing the system with a tool to analyze PCAP files through the command line, many errors raised because of missing arguments and wrong syntax.

The background knowledge of the LLM used (GPT-4o, provided by OpenAI [29]) was not enough to face this complex task. To address these limitations, a dedicated subagent was introduced to specialize in interpreting and executing tshark commands. This subagent acts as an intermediary, translating high-level instructions into syntactically correct command-line operations. By isolating this responsibility, the overall robustness and reliability of the system significantly improved, especially when dealing with non-trivial use cases.

Overall, the structure of the agent didn't change; in this case, there is just one additional tool: the Tshark expert, acting as a subagent. For this reason, I call it the *TShark Expert Agent (TEA)*.

4.3.1 TShark expert

Also in this case, the subagent has been implemented using LangGraph. Here is the structure of the agent:

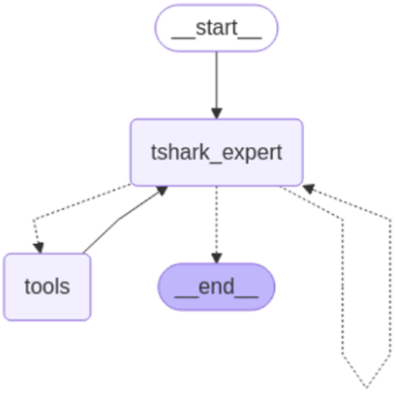

Figure 4.3: TShark Expert subagent

The capabilities of this system are provided as a tool to the main_agent, which can call it to perform an analysis. In particular, the expert requires as input the description of an high level inspection to be performed on the PCAP file, and then it returns the output of the executed command and a summary of the operations performed to get to that.

As showed in the structure of the subagent, it is allowed to iterate and call tools to improve its analysis and execute refined commands. The subagent is provided with 2 tools:

- **TShark manual**: a semantic search tool built from the official `tshark` manual pages [42] and the Wireshark display filter guide [43]. The content of both documents was parsed and chunked using a sliding window strategy, where each chunk includes 1000 characters with an overlap of 200 characters to preserve context across adjacent chunks. Each chunk was embedded using a transformer-based sentence embedding model [28], and the resulting vectors were stored in a FAISS (Facebook AI Similarity Search) index for efficient approximate nearest-neighbor search.

  When the subagent requires additional knowledge (e.g., about syntax or parameters of a `tshark` command or about display filters), it formulates a natural language query. This query is embedded using the same embedding model and used to perform a semantic search over the FAISS index. The top-3 relevant chunks are then retrieved and appended to the subagent's context before the next reasoning step.

  This mechanism enables the agent to retrieve precise technical documentation without needing to include the full manuals in the prompt. It effectively augments the agent's capabilities with an up-to-date and scoped external knowledge base.

- **TShark command executor**: a wrapper that allows executing `tshark` commands. The subagent is expected to generate the command string in the correct syntax, which is then passed to the executor. When an error occurs, the system extracts additional information regarding missing parameters or syntax issues by

processing the content of the standard error stream (stderr). Given the size of PCAP files of some of the events in the benchmark and in realistic use-cases, the length of the output of the command is checked before returning it to the agent. It is allowed to append the output to the prompt as long as it does not exceed the context window, which depends on the specific LLM used for the forensic analysis. If the output is too large to fit within the context limit, the subagent is prompted to refine its commands. This may be done, for example, by applying more restrictive filters or adopting an alternative strategy to reduce the amount of extracted data.

4.3.2 Interaction Flow between Agents

The main agent is initially provided with a high-level overview of the network traffic contained in the PCAP file. This summary is obtained by executing a preliminary `tshark` command that extracts the list of TCP conversations, offering a compact representation of communication flows that is appended to the prompt.
The command used to generate this summary is shown below:

```
tshark -r <pcap_file> -q -z conv,tcp
```

This output provides the agent with essential information such as IP addresses, ports and the amount of data exchanged in each connection.
When the main agent needs to perform a more detailed analysis of the PCAP file, it delegates the task to the subagent. This happens by passing a textual description of the high-level analysis goal expressed in natural language. For instance, the main agent might submit the following instruction:

```
Follow the second TCP stream in the PCAP file and report relevant
details.
```

Upon receiving such a request, the subagent initiates an iterative reasoning process in which it plans a sequence of actions. It may consult the `Tshark Manual` tool to resolve any ambiguities in the command syntax or display filter expressions, and it uses the `Tshark Command Executor` to run the actual commands.

The result of each command is parsed and interpreted by the subagent to refine its next steps. For example, in response to the instruction above, the subagent might:

- Parse the high-level instruction (e.g., "second TCP stream") to identify the corresponding stream index.
- Construct an appropriate display filter (e.g., `tcp.stream == 1`) to isolate the desired stream.
- Execute follow-up commands to extract relevant information such as payload content, packet timestamps and TCP flags.

When the subagent is satisfied by the results obtained in relation to the high level goal required, it returns the `tshark` command executed and the corresponding output. Obviously, also in this case, it is allowed for a maximum number of iterations before being prompted to return the result of its analysis even if not complete.

In order not to make execution times explode because of the two running agents, the number of iterations for the Tshark expert are limited to 15.

4.3.3 Limitations of the Tshark Expert Agent

This version, despite allowing us to face all the events in the benchmark without saturating the context window with too many packets, presents several limitations. The principal issue lies in the main_agent's inability to explore large volumes of data without losing focus. During testing and log analysis, it became evident that the agent often fixates on irrelevant details, failing to maintain a coherent investigation strategy. As a result, it struggles to abstract from the noise and may be unable to complete the analysis or to inspect the entire network traffic in a meaningful and structured way.

This behavior highlights the need for a more guided and goal-oriented reasoning process, as well as for improved mechanisms to identify, prioritize and focus on relevant information. Such improvements are essential to enable the agent to carry out complex, multi-step forensic analyses over large network traces in a structured and effective manner.

4.4 Multi-agent architecture, Flow Reporter Analyst (FRA)

Despite the improvements introduced through the TShark_expert, the main limitation related to the inability to autonomously explore a huge amount of data without losing the focus is still present. To address this, I revised the architecture to allow the agent just to draw conclusions from data, avoiding the step of devising techniques to explore the network traffic effectively. Indeed, from the analysis of the log files obtained through the tests of the previous versions, it is clear that the agent lacks causality and ability to plan when there are too many possibilities and there's not a standard strategy coming from the literature.

Thus, the new approach is inspired by one of the possible workflows of human forensic analysts. In practice, cybersecurity experts open a PCAP file with Wireshark and examine the entire network trace, looking for unusual requests or abnormal responses. Their method is often straightforward yet effective: they analyze TCP flows to identify suspicious patterns or anomalies within the communication, observing them one at a time if there are no additional indicators that suggest where to start from.

The problem is that, when leaving complete freedom to the agent, it typically focuses on some irrelevant detail found in early iterations instead of analyzing the entire traffic. In this case, it may be lucky and find the real evidence of exploitation, but it could also lose time on noisy data (e.g., misconfiguration alerts in the traffic not related to the attempted attack).

In order to solve the aforementioned problem, I introduced a guided search to ensure that the traffic is completely inspected and findings are correlated. This is a common pattern in realizing agents: drawing the path and leaving freedom to make conclusions and/or hypothesis.

Starting from the network traffic summary obtained with the command:

```
tshark -r <pcap_file> -q -z conv,tcp
```

the output is first examined to ensure that TLS traffic is excluded. Encrypted traffic is removed because it does not contribute useful information to the analysis while increasing complexity and execution time. After this filtering step, a dedicated subagent (Section 4.4.1) analyzes each remaining TCP flow to identify relevant indicators or potential anomalies.

4.4.1 PCAP flows reporter

The PCAP flows reporter inspects each flow of TCP traffic and generates a structured forensic report that highlights the following key elements:

- **Service**: the service involved (or a list if more than one is visible from the traffic). If possible, the agent has to highlight the version of the service, which may be particularly useful to call the web search with more accurate queries
- **Relevant events**: point out actions performed by the different IP addresses involved in the conversation. If needed, direct quotations can be added in order to give a more in-depth overview to the `main_agent`
- **Malicious activities**: report malicious or suspicious activities found in the flow and the service they involve (among those present, if more than one)
- **Attack success**: state whether there has been an attack in the current flow and if it was successful.

The subagent analyzes each TCP flow by following the corresponding stream and extracting the relevant information previously defined. The traffic is extracted from each stream using the command:

```
tshark -r <pcap_file> -q -z follow,tcp,ascii,{stream_number}
```

This command reconstructs the full bidirectional TCP conversation associated with the given stream number, displaying the exchanged payloads in ASCII format. It separates the content sent by the two endpoints, making it easier to analyze application-layer protocols (e.g., HTTP, FTP) or detect suspicious sequences of commands and responses. The output is particularly useful for inspecting raw communication without the need to open Wireshark's graphical interface, but giving exactly the same information to the agent that works through analyzing natural language data.

For each flow, the subagent generates a structured report that highlights the key details outlined in the predefined list. Each field in the report is explicitly filled in

by the LLM, ensuring that all relevant information is consistently captured. Defining a clear and uniform report structure is crucial to enable reliable conclusions across different flows.

Additionally, it receives the report of the immediately preceding flows in order to correlate findings when necessary. Indeed, some attacks are exploited through a request and a response that comes later in another flow, which would be impossible to be detected without having a complete overview.

Since TCP traffic may be particularly long, a chunking technique has been introduced: when the number of tokens is too high, the agent iterates receiving one chunk at a time and refining the report at each iteration. This way, I ensured that the context window is not overcame, while the traffic is fully inspected. This structured and iterative analysis enables the system to handle complex forensic tasks with greater precision. The PCAP flows reporter plays a crucial role in the architecture, as the quality and completeness of its reports directly influence the reasoning and final assessments performed by the main_agent. By ensuring consistency and depth of inspection, this subagent significantly enhances the reliability of the overall investigation workflow. Finally, the report is passed to the main_agent as first message in order to start the reasoning process. Since the architecture builds upon the previous version, the tools available to the main agent remain the same as those used in the multi-agent system described in Section 4.3, without the possibility of executing arbitrary commands:

1. Web search tool
2. Store memory

The entire flow of execution of the agent is reported in Figure 4.4.

4.4.2 Limitations of the Flow Reporter Analyst

As it has been previously mentioned, the entire context of the attack is injected into the first agent in the pipeline, which is obliged to analyze the totality of the captured traffic to produce reports. The first problem associated with this strategy is the amount of tokens that it requires to input within the LLM. Indeed, in real use-cases, captured traffic can become huge in terms of exchanged data, mainly because of HTML pages exchanged or long communication protocols. The amount of tokens, which directly translates into costs, is not the only limitation: the chunking strategy to analyze huge amount of data make the agent lose the focus on irrelevant details. With the strategy explained in the previous section, when the TCP flow is too long, it is divided into partially overlapping chunks. During the first iteration the *PCAP flows reporter* generates a partial version of the report, then it iterates again receiving the previous report and the new chunk to refine previous conclusions and so on till the TCP flow has been completely analyzed. With such an approach, the risk of losing relevant details because they were not properly included in the previous version of the report is high. When observing the traffic, I noticed that very long flows were full of noisy data, such as HTML pages that are useless for the analysis.

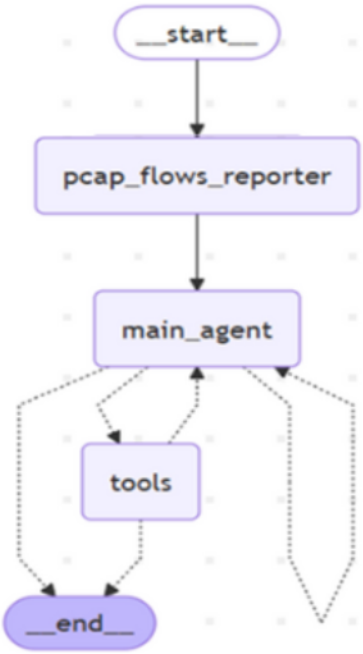

Figure 4.4: TEA

4.5 Multi-agent architecture, Log Reporter (LR) and Flow Reporter Analyst (FRA)

Up to the previous version of the agent, we only considered PCAP files as input data. However, as explained in Section 2.6, the benchmark provides not only network traffic captures but also Docker log files, enabling additional insights for each event. The possibility to provide them to the agent was left as a suggestion and future work by the reference paper [7].

The main challenge in this context was to devise an efficient strategy to make the agent access logs. Indeed, some events are characterized by many files, which can also be lengthy. The idea of providing the agent with a tool to open and append them to the prompt step by step would have not been effective. Indeed, it would have wasted many iterations, with the risk of saturating more and more the context window. Thus, also in this case, the idea is to face a such complex workflow with a multi-agent architecture in LangGraph.

Starting from the previous version of multi-agent system, there is the need of an additional subagent focusing on analyzing log files and reporting relevant details. Here is the structure of the complete agent:

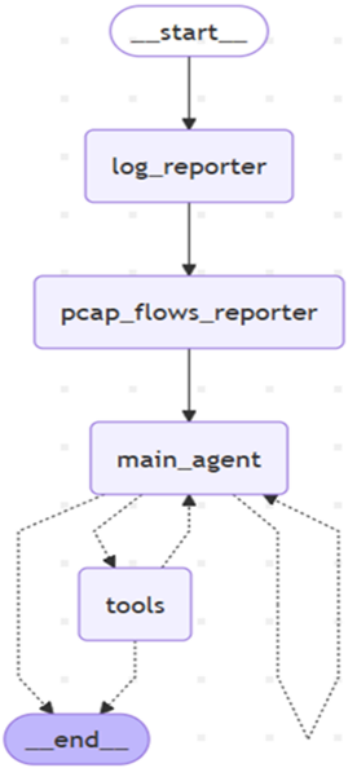

Figure 4.5: Multi-agent system, FRA + LR

As showed in the graph 4.5, before getting to the PCAP flows reporter, the flow passes through the log_reporter.

This subagent receives all log files directly appended to the prompt and generates a concise forensic report. It is explicitly instructed to focus on the service mentioned in the logs, identifying it by name and, when possible, also reporting the version. The analysis is tailored for a forensic analyst and aims to highlight suspicious or anomalous events, referencing specific log entries to support its conclusions. If no relevant findings are detected, the subagent is required to state this explicitly, avoiding any speculative assumptions, particularly regarding the success of an attack, which typically cannot be reliably inferred from log data alone.

The resulting report is passed to the main_agent and appended to its prompt, representing the foundation on which the system bases the following steps in the analysis.

To improve performance, the complete version of the agent receives two different reports:

1. **Logs report**: summary of relevant events and services from log files produced by the `Log_reporter` 4.5
2. **PCAP flows report**: message containing a list of reports (one for each TCP flow in the traffic) produced by the `PCAP_flows_reported`

The main_agent in figure 4.5 is provided with the same tools that characterized the orchestrator of the previous version, with the most important one being the web search.

4.6 Constrained Flow Reporter Analyst

To avoid burning an amount of tokens that is not sustainable for this analysis and reduce the noise, I introduced a new version of agent provided with a *token budget* for the data that can be analyzed in the TCP flows.

Given the budget, I first count the totality of tokens within TCP flows. In case it is under the budget, then the traffic is fully examined flow after flow by the *PCAP flows reporter*. Conversely, if it overcomes the budget, it is necessary to allocate, for each flow, an amount of tokens that is proportional to its initial dimension.

To compute the amount of tokens for each flow we use:

$$\text{allocation}_i = \text{budget} \times \frac{\sqrt{\text{tokens}_i}}{\sum_j \sqrt{\text{tokens}_j}}$$

This square root normalization strategy has been introduced to reduce the impact of huge flows. Indeed, in case one stream of traffic is much larger than the others, if we use proportional allocation, then we are focusing only on data of the biggest flow, ending up ignoring the others. Since mathematically it may happen, with the previous formula, that the allocation is greater than the actual dimension of a small flow, we select the minimum to make things correct:

$$\text{final\_allocation}_i = \min(\text{tokens}_i, \text{allocation}_i)$$

Any remaining budget is then iteratively redistributed among the flows that have not yet reached their maximum allocation, balancing the distribution further while preserving the global token limit.

This strategy ensures a more balanced and fair analysis across all TCP flows, preventing large streams from monopolizing the token budget and allowing smaller, potentially relevant flows to be included in the inspection.

Finally, given the allocation budget, there's the need of a strategy to select actual rows of text from the traffic, moving from tokens to words. The approach used in this case is to divide by two the final allocation budget for the i-th flow. Half of the total is used to select words from the beginning of the traffic, while the other half is used for the end of the flow, increasing the probability of selecting relevant details within data. Indeed, the ratio is that, in the middle, there could be noise related to specific exchanged data, but not details to detect the type of attack or its success.

The purpose of the agent is not to fully analyze what is exchanged by the service and the attacker, but to detect anomalies and report them to a cybersecurity expert that can go deeper in the analysis, if needed.

4.6.1 Limitations of the Constrained Flow Reporter Analyst

This version of the agent has been introduced to reduce the costs of execution, as an effective system is not only evaluated by the result returned, but also based on many other metrics. Indeed, it is important to account also for input/output tokens, execution times, security etc.

However, the square root allocation strategy has a main drawback: it is not informed, it makes a decision that does not depend on the specific flow, but it is fixed. Thus, it may happen, even when considering events outside our reference benchmark, that the evidence of the success of an attack is in the middle of a flow that is not seen by the agent. To mitigate this limitation, future versions of the agent could incorporate lightweight heuristics or attention mechanisms to guide allocation based on flow semantics or preliminary inspection. Nevertheless, this constrained approach remains a valuable tool for scalable and cost-aware pre-filtering, enabling analysts to focus their efforts where they matter most.

4.7 Architectures overview

To conclude the methodology, a complete overview of the four agent architectures is provided in Fig. 4.6. I study four different architectures that consistently improve results over evaluation metrics. The solutions proposed varies both in terms of structure, operational flow and type of input.

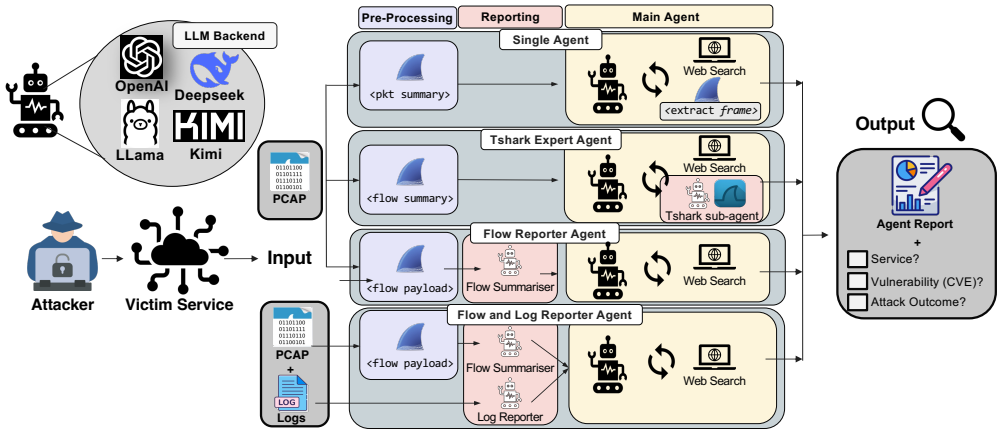

Figure 4.6: Overview of agent architectures. Each architecture receives traffic data (and logs) and processes it through a pipeline of pre-processing, reporting, and an LLM reasoning agent. The agents connect to the desired LLM backend via API, interact with tools and sub-agents (`tshark`, Flow Reporter, Log Reporter and Web Search) to extract evidence and produce the forensic report.

Chapter 5

Experimental evaluation

Given the architectures introduced in the previous chapter, I evaluate all of them on CFA-Bench [7] using GPT-4o [29] as reference model. I then further test the best-performing architecture on a held-out dataset of unseen events and across several additional LLMs. During development, benchmark feedback informed several design choices, that's why I validate the findings on unseen events to assess robustness to forensic variability.

For the experiments, I rely on well-known LLMs available through public APIs.[1] I evaluate four architectures, two input modalities, and six LLMs. Unless otherwise specified, I use each model's default temperature,[2] and repeat each incident three times to account for stochasticity. The time to complete an incident analysis ranges from a few minutes to 15–20 minutes.

5.1 Results over architectures, GPT-4o

In this first section, I introduce the results obtained with the different architectures. For each architecture, I report several metrics:

- **Affected service**: percentage of events for which the correct service have been identified
- **Identified CVE**: percentage of correct CVE identification
- **Attack successful**: correct identification of the success of the attack, binary output given by the agent (True or False)
- **Aborted cases**: number of events that fails to run in the benchmark because they overcome the context window

[1] https://platform.openai.com; https://www.together.ai

[2] The temperature controls generation randomness and ranges from 0 to 1; 0 yields deterministic output. See, e.g., [19, 47].

- **MCC attack success and $\mathbf{F1_{macro}}$ attack success**: Matthews Correlation Coefficient and $F1_{macro}$ score to better evaluate the binary label previously introduced
- **Steps**: number of steps performed by the agent
- **Input and Output tokens per run**: number of tokens given as input/output over one run of the benchmark
- **Cost per run**: upper bound of the cost for each run

Note. Throughout this section, a "run" denotes a single execution of the benchmark. All the reported metrics in the following are the mean over three independent runs to reduce stochastic variability.

I evaluate the performance of the three agent architectures explained in 4, excluding the one equipped with the *Log reporter* that will be better analyzed in the following 5.1.4. I keep the back-end LLM fixed to OpenAI GPT-4o and set the maximum number of steps to 25. I summarise results in Tab. 5.1.

Best Architecture: Start with the Single Agent (SA) and compare its performance against the CFA-bench baseline [7]. Both approaches achieve similar accuracy in identifying the service under attack. However, thanks to the improved Web Search tool and the MemGPT-style memory management, SA achieves substantially better results in other dimensions: CVE identification improves by +14%, and the attack success evaluation by +34%. These gains are attributed to SA's ability to:

1. retrieve better targeted information about candidate CVEs from the web
2. accumulate and reuse contextual information over multiple reasoning steps to refine its conclusions

Despite these improvements, the overall SA performance remains unsatisfactory. Notably, the high number of steps reflects the SA difficulties in using *tshark* efficiently: it sequentially inspects packets until it finds some evidence, causing a waste of input tokens, and finally loosing focus. Additionally, the MCC is negative, indicating that its predictions are biased in the wrong direction: worse than random guessing. This effect stems from SA's tendency to return "attack unsuccessful" whenever it fails to gather sufficient evidence. Because the benchmark is unbalanced (14 successes versus 6 failures), this systematically biases SA's predictions toward the wrong class, explaining its poor MCC despite reasonable accuracy and F1 scores (recall – a random guess would have 50% accuracy).

Next, I analyse the performance of the Tshark Expert Analyst (TEA) and Flow Reporter Analyst (FRA) architectures. Both outperform SA in identifying the correct service. However, only FRA achieves a consistent improvement in CVE detection and attack success evaluation. The limited TEA gains can be traced back to a coordination gap between the main agent and the *Tshark sub-agent*. Indeed, the main agent often issues overly broad requests (e.g., "explore the HTTP traffic"), which triggers the *Tshark*

Table 5.1: Comparison of different agent architectures. best in **bold**; second-best underlined.

	Metric	CFA-bench [7]	SA	TEA	FRA
	Service (↑)	0.42	0.45	0.58	**0.67**
	CVE (↑)	0.14	0.28	0.35	**0.45**
Success	**Acc (↑)**	0.13	0.47	0.48	**0.62**
Success	**F1 (↑)**	—	0.49	0.46	**0.62**
Success	**MCC (↑)**	—	-0.11	0.00	**0.45**
	Steps (↓)	—	18.11	11.32	**5.48**
	In. Tokens (↓)	—	5.48M	4.37M	**3.86M**
	Out. Tokens (↓)	—	**78k**	123k	112k
	Cost [$] (↓)	—	14.48	12.15	**10.78**

sub-agent into performing inconclusive analyses, eventually losing direction. As a result, many runs fail to converge to useful evidence. Nonetheless, when *Tshark sub-agent* succeeds in extracting a meaningful clue, TEA is able to resolve the task efficiently. For instance, TEA correctly identifies the service in 35 runs out of 60, and in 21 of these it also pinpoints the correct CVE. Once the service is known, the CVE is typically identified within just three reasoning steps – in a (query → reasoning → second query) pattern. This shows that, although the TEA design provides fine-grained packet analysis capabilities, its practical effectiveness is hampered by coordination issues between its two agent layers.

The FRA architecture resolves the coordination gap by decoupling the *PCAP flows reporter* from the main agent. The sub-agent independently conducts a systematic analysis of the PCAP, producing a high-quality report of suspicious activities without being influenced by intermediate reasoning steps. The main agent then refines these findings through targeted reasoning and web searches. Although this limits the agent's action space since it cannot query back the flows reporter, FRA achieves the best overall performance at the lowest cost. On average, it converges in just ∼5 steps, significantly reducing input token usage. Notice the positive MCC testifying that it takes well-grounded decisions based on the evidence collected by the *PCAP flows reporter* sub-agent.

Result Breakdown Fig. 5.1 reports, for each incident in the dataset, how many times (out of three runs) the agents correctly identify the service (top) and the CVE (bottom). Incidents are ordered by difficulty. SA can correctly identify the service of the first 8 incidents only. In 5 of them, the agent immediately guesses the exact service at the first reasoning step. In fact, for those incidents, the answer is self-contained in the `packet list summary`. As previously stated, once the service is found, the agent can get the correct CVE with one or two web searches. In contrast, when finding the service

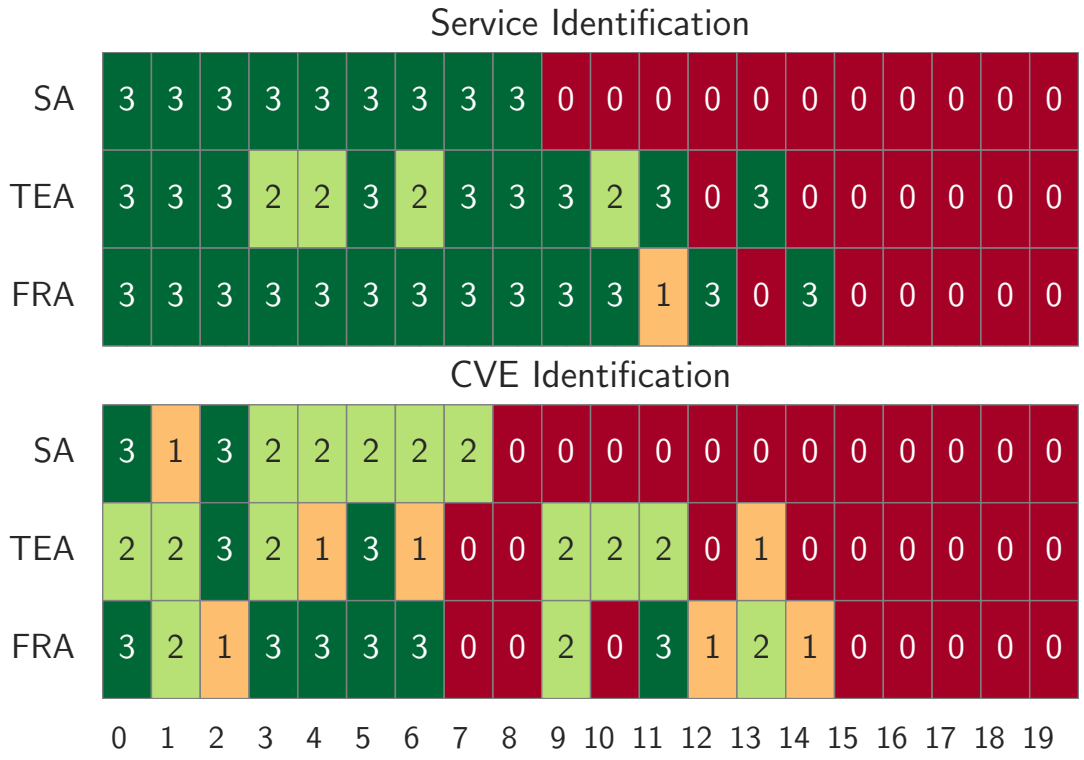

Figure 5.1: Breakdown of performance in Service (top) and CVE (bottom) identification over the 20 incidents.

requires using the *PCAP reader tool*, the reasoning trace becomes much longer (18 steps on average, Table 5.1), a clear sign that the agent starts losing focus and finally fails in both tasks. TEA shows improved service identification thanks to the *TShark sub-agent*, but the coordination issues discussed earlier often prevent it from translating partial findings into successful CVE detection. By contrast, FRA's decoupling strategy ensures more consistent progress: the Flow Reporter provides focused evidence, enabling the main agent to refine its reasoning and achieve higher accuracy in both service and CVE identification.

Looking at the hardest incidents (15-19), these involve multi-container systems where the attack generates a large volume of traffic (Tab. 2.1), causing the agents to lose focus. Incident 17-Cacti is a notable exception: here, the difficulty stems not from traffic overload but from the scarcity of online information. In this scenario, the Web Search tool provides little support, and agents ultimately fail. Conversely, TEA and FRA fail to identify the CVE in incidents 7 and 8. These incidents involve Apache ActiveMQ exposing OS and kernel versions in plain text. The attacker simply uses a telnet connection to issue a standard HTTP GET request, leaving almost no forensic evidence of malicious activity. This lack of distinctive signals hampers the coordination

between the *Tshark sub-agent* (in TEA) or the *PCAP flows reporter* (in FRA) and the main agent.
As we can observe, results keep increasing when moving towards a more complex architecture in the table. When focusing on `SA`, we notice that results are close to be a random guess for what concerns the identification of the success of the attack, while the agent is able to detect the correct service or the CVE less than half of the times.

When introducing the `TEA` there are many improvements, in particular because it allows to face all the events, but also because the agent is able to extract more information from PCAP files. This improves even more when considering the version that also includes logs as source of information. Indeed, log files are particularly useful for service identification, improving not only that metric, but also CVE detection, as it is a direct consequence.

5.1.1 Web Search Tool usage

I further analyze how agents leverage the Web Search tool by examining a subset of increasingly complex incidents (Fig. 5.2). Each row reports the last query issued in one run, with cell colours reflecting whether the correct CVE was identified (green) or not (red).

I identify three typical failures: (A) query asking for the wrong service, (B) query broadly looking for attacks, and (C) query asking for the wrong attack. At large, a "correct" query does not guarantee success, and a wrong query implies failure. For instance, service misidentification (A) makes the whole analysis fail (see SA looking for `JetDirect` and `etcd` in the *Couch DB* and *APISIX* incidents). In the Gitlab incidents, SA gets lost in investigating the long packet list, reaching the maximum step number. TEA performs slightly better, e.g., correctly looking for `CouchDB`. Yet, its queries are too broad to succeed in complex scenarios. Only FRA manages to recover CVEs for some challenging incidents. Curiously, in the `12-APISIX` incident, queries are correct, and the true CVE is present in the web search results. Yet the agent picks the right CVE in only one run, suggesting that GPT-4o struggles at interpreting web search results.

Observing how the queries performed by the agent evolve over time it is easier to measure the actual understanding. Indeed, the Single Agent performs queries that are always different among executions and that are too general, resulting in researches that don't even include the type of attack attempted, as it was not able to detect it. On the contrary, the degree of detail increases towards the Flow Reporter Analyst, which includes the type of attack (RCE, path traversal, etc.) most of the times and also demonstrates an higher stability over execution for what concerns evidence found.

As the query becomes more specific, I observe that the number of results returned by the *Web Search Tool* reduces. The overall result is:

- Fewer steps reduce execution time and cost.
- A more focused, context-aware analysis increases the likelihood of retrieving the correct CVE during online search.

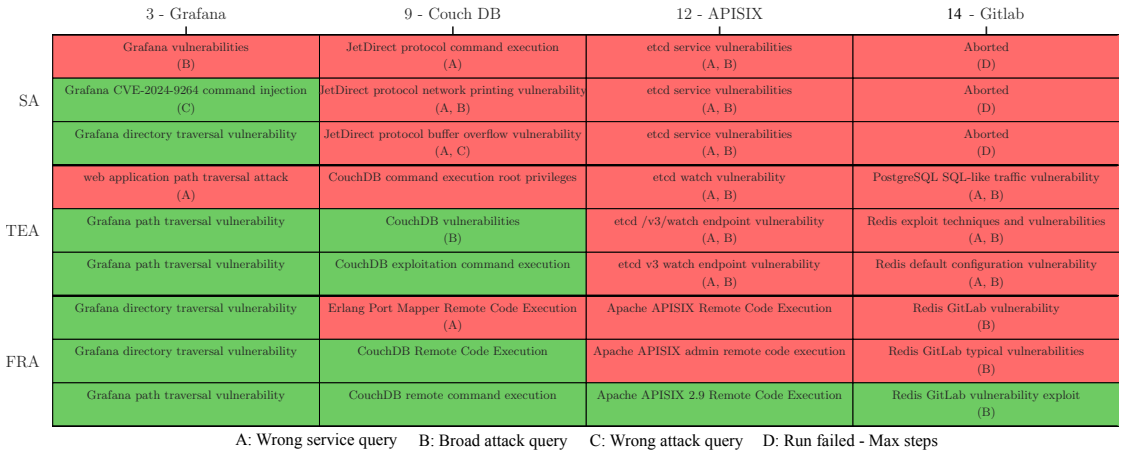

Figure 5.2: Agent's web searches in increasingly complex incidents. Red cells indicate that the agent failed in CVE identification.

- With fewer plausible candidates, the agent can more reliably select the correct one.
- Accuracy improves not only because there are fewer CVEs to consider, but also because the agent develops a deeper understanding of the overall incident.

5.1.2 Model ablation over FRA architecture

I study the effects of different back-end LLMs on the performance while providing some quantitative interpretation of their reasoning. I test only the FRA architecture and compare six LLMs.

Table 5.2: Study over different LLMs.

Metric		OpenAI GPT-4o [33]	OpenAI o3 [32]	OpenAI GPT-5 [31]	Deepseek R1 [8]	Kimi K2 [41]	LLama-4 Maverik [1]
Service (↑)		0.67	0.75	0.80	**0.85**	0.82	0.75
CVE (↑)		0.45	0.48	**0.68**	0.67	0.63	0.53
Success	**Acc (↑)**	0.62	**0.80**	0.78	0.78	0.78	0.60
	F1(↑)	0.62	**0.79**	0.77	0.76	0.76	0.63
	MCC (↑)	0.45	**0.65**	0.63	0.55	0.55	0.35
Steps (↓)		5.48	**3.55**	3.93	5.42	6.70	17.12
In. Tokens (↓)		3.86M	**3.67M**	3.92M	4.66M	4.35M	8.69M
Out. Tokens (↓)		112k	245k	373k	555k	**94k**	215k
Cost [$] (↓)		8.60	9.30	8.63	3.78	4.63	**2.52**

In Tab. 5.2 I detail the results. Focusing first on the closed-source OpenAI models, both `o3` and `GPT-5` outperform the baseline `GPT-4o` in service identification (+8% and +13%, respectively). This aligns with expectations: `o3` is a dedicated *reasoning model*, designed to generate an internal reasoning trace before producing the final answer, at

the cost of higher latency. This makes it well-suited for detailed forensic investigations. `GPT-5`, in contrast, is a *hybrid model* that can dynamically invoke reasoning when needed and benefits from a more recent training process. For CVE detection, however, `o3` performs only on par with `GPT-4o`. As I will show in Fig. 5.3, this is due to `o3` often being overconfident in its own knowledge and relying less on the Web Search tool. Still, thanks to its reasoning capabilities, `o3` achieves the highest performance in attack success evaluation (0.64 MCC), even surpassing the newer `GPT-5`. Both models also require fewer reasoning steps on average, but their verbosity makes the overall cost comparable to `GPT-4o`.

Turning to open-source models, both `DeepSeek R1` and `Kimi-K2`[3] deliver impressive results, matching or surpassing OpenAI's best models across most metrics. This is particularly encouraging, as they can be deployed on-premise at inference time, avoiding the need to outsource sensitive data. Even when accessed via the cloud (as in our experiments), these models remain significantly cheaper than their closed-source counterparts—roughly half the cost. The open source `Llama-4 Maverick` improves over `GPT-4o`, but ranks only second to last overall.

In short, open-source LLMs rival or even exceed the performance of the best proprietary models, while offering greater flexibility and substantially lower cost.

5.1.3 Result Breakdown

To provide a more interpretable view of model performance, Fig. 5.3 analyses CVE identification outcomes across all 60 runs (20 incidents, each repeated 3 times). For each model, I report the fraction of runs:

1. where the model issues a web query (green bar);
2. where the last web response contains the correct CVE identifier (blue bar);
3. the final CVE successfully detected, distinguishing correct answers supported by web evidence (cream bars) from those obtained through prior knowledge (red bar).

All models use the Web Search tool in over 80% of cases, except `o3`, which does so in only 50% of runs. This reflects its distinctive behaviour: despite the prompt explicitly encouraging web searches when uncertain, `o3` tends to rely heavily on its internal knowledge, often displaying overconfidence. The quality of the queries is captured by the blue bar. Here, `GPT-4o` performs poorly: it searches in 95% of runs, yet only 54% of responses contain the correct CVE. By contrast, `Llama-4 Maverick` emerges as the most effective searcher: it always queries the web, and 80% of its results include the correct CVE. The crucial step is then selecting the right CVE among candidates. `GPT-5`, `DeepSeek R1`, and `Kimi K2` excel here: whenever the correct CVE is present, they pick it 90-95% of the time (blue bar $\approx$ cream bar). `Llama-4 Maverick`, however, struggles to

[3]Open-source models make their code, architecture, and trained weights freely available for use, modification, and redistribution.

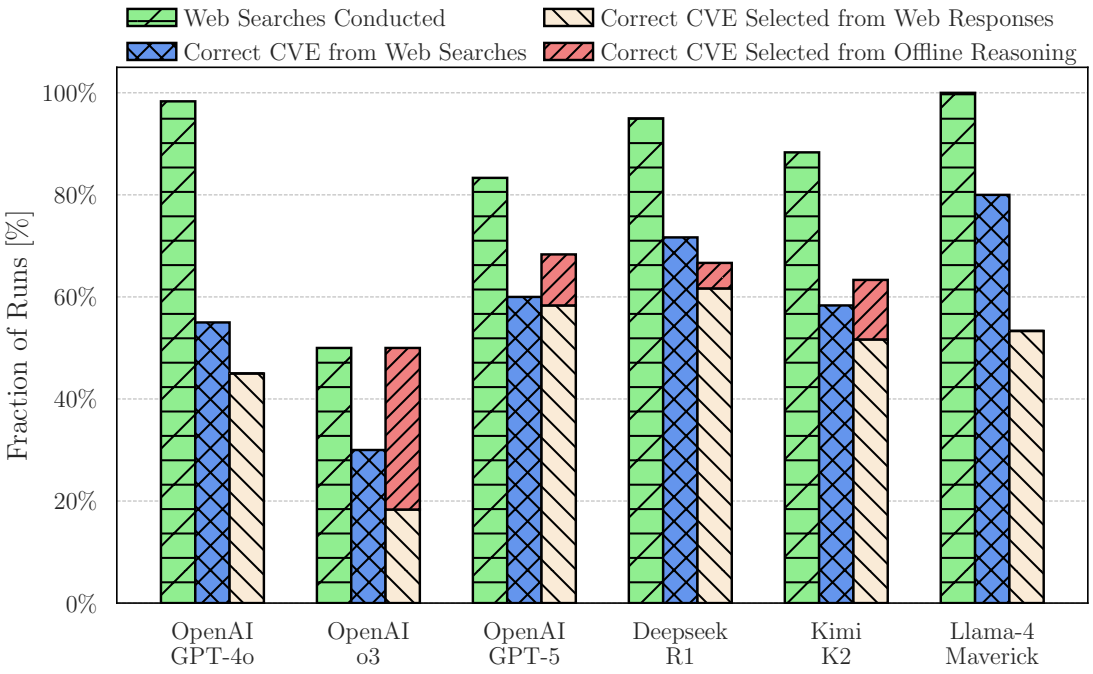

Figure 5.3: Detailed outcomes of web searches and CVE identification across 60 runs.The sum of CVE detection success rates corresponds to the accuracy values of Table 5.2.

distinguish between near-identical CVEs, choosing the wrong one in 30% of such cases showing a clear reasoning limitation. Finally, the red bar highlights cases where the correct CVE is predicted without it appearing in web search results, i.e., by relying on LLM's prior knowledge. As said, `o3` frequently succeeds this way. Interestingly, `GPT-5` shows a more systematic use of external evidence, only occasionally complementing it with prior knowledge.

Overall, the figure reveals three distinct behaviours: `o3` relies primarily on offline priors; `GPT-5` and `DeepSeek R1` combine web evidence with strong reasoning in a balanced manner; `Llama-4 Maverick` and `GPT-4o` suffer from mismatches between retrieval and reasoning.

5.1.4 Including the Log Reporter

Finally, I study the effects of offering the application logs to the agent as in the Flow and Log Reporter Agent (FRA + LR) in Fig. 4.6. I only consider the best and worst performing LLM backend (GPT-5 and GPT-4o) in Tab. 5.3.

Consider GPT-4o. Application logs improve the service identification of the services (+18%), as the agent can better resolve ambiguous cases with the service information frequently available in logs. This partially improves the CVE detection and evaluation

Table 5.3: Effects of including the application logs as input for the worst and best model in the previous study. Results in **bold** are the best for that metric; results underlined are the second best.

		OpenAI GPT-4o		OpenAI GPT-5	
	Metric	**FRA**	**FRA + LR**	**FRA**	**FRA + LR**
Success	**Service (↑)**	0.67	0.85	0.80	**0.87**
	CVE (↑)	0.45	0.50	**0.68**	0.65
	Acc (↑)	0.62	0.70	**0.78**	0.68
	F1 (↑)	0.62	0.70	**0.77**	0.69
	MCC (↑)	0.45	0.54	**0.63**	0.51
	Steps (↓)	5.48	7.10	3.93	3.77
	In. Token (↓)	**3.86M**	4.34M	3.92M	3.87M
	Out. Token (↓)	**112k**	132k	373k	398k
	Cost [$] (↓)	**8.60**	12.18	8.63	8.80

of attack success (+5% and +8%, respectively). This is due to GPT-4o's reasoning limitations that still apply: while the additional information is highly beneficial for simple retrieval tasks (e.g., finding a service name directly from the input logs), it contributes little when deeper reasoning is required. The need to analyse more data increases the number of steps, tokens and finally cost.

GPT-5 benefits from logs in the service identification. Conversely, the additional information provided by the logs confuses the agent both in the CVE detection and evaluation of successful attacks (−3% and −10%). Recall that the logs offer evidence of attacks in only 7 incidents. This lack of evidence confuses the agent who neglects the evidence presentd by the *PCAP flows reporter*. This is reflected in the unexpectedly reduced number of steps, tokens and similar costs.

In a nutshell, logs are hardly beneficial, make the model more self-confident, and create the risk of making more errors. Future efforts shall focus on finding better prompts/strategies to actually account for the additional information provided by the log.

I finally evaluate the agent on 10 recent CVEs and benign traffic 2.2, using DeepSeek R1, OpenAI o3, and GPT-5 as back-end LLMs and FRA as architecture. As shown in Tab. 5.4, the agents achieve 90% service identification and 80% CVE detection accuracy – an impressive result considering these incidents involve both vulnerable and patched systems. GPT-5 and o3 deliver the strongest performance, whereas DeepSeek R1 struggles with attack success classification (3 failures out of 10), yielding an MCC of only 0.41, suggesting a more random guess approach. These results support the choice of GTP-5 as the final backend LLM for the agent.

Challenging the agent with 10 benign traces collected during simple browsing sessions, it correctly reports no evidence of malicious activity. In two cases, it flags potential brute-force attempts, supported by repeated and rapid failed login events, which, in fact,

Table 5.4: Results on 2025 CVEs (testset).

	Metric	Deepseek R1	OpenAI o3	OpenAI GPT-5
	Service (↑)	**0.90**	0.80	**0.90**
	CVE (↑)	**0.80**	0.70	**0.80**
Success	**Acc (↑)**	0.70	**0.90**	**0.90**
Success	**F1(↑)**	0.73	**0.90**	**0.90**
Success	**MCC (↑)**	0.41	**0.82**	**0.82**
	Steps (↓)	5.5	**4.3**	4.4
	In. Token (↓)	492k	**298k**	502k
	Out. Token (↓)	207k	**108k**	185k
	Cost [$] (↓)	**0.72**	1.46	2.47

is an accurate observation.

5.2 Human Report Assessment

I pooled human experts to evaluate the quality of the final report produced by OpenAI's o3 and DeepSeek R1 models and provide a qualitative assessment of agent performance. I ask experts to evaluate three aspects on a scale from 0 to 5 (0 - Totally Unsatisfied, 5 - Totally Satisfied):

- **Completeness**: Does the report include relevant and accurate information?
- **Usefulness**: Is the report helpful to a human analyst?
- **Logical Coherence**: Is the agent's reasoning clear and logically consistent?

I involve 22 volunteers. I select four incidents from the 2025 testset in which the agent was successful and show the agent's reasoning steps, the final report and the PCAP file. The volunteers self-declared their security forensic expertise and proceeded to compile an online questionnaire.The questionnaire is available at https://forms.gle/QsaJ1dVUSK7suu3K6.

At the time of running the survey, GPT-5 was not yet released. 22 students, experienced researchers, and professors responded. 9 self-declared low (rating 1-2), 8 medium (rating 3), and 5 high expertise (rating 4-5).

All participants find the produced reports to be complete, useful and logically coherent (average score of 4.33, 4.23, 4.31, respectively). When asked which report they prefer, Fig. 5.4 shows the breakdown of preference for Deepseek R1 (blue) or OpenAI o3 (orange) final report, broken down by level of expertise (the darker, the higher). Overall, participants show a slight preference for Deepseek R1, particularly among medium- and high-expertise participants. This is reflected by the all-metric average grade for Deepseek R1 of 4.39 versus a 4.20 for o3.

OpenAI o3 is preferred only on the first incident (20 - Vite). Some participants reported the Deepseek R1 comment to be verbose in this case. In the other cases, they

find OpenAI o3 to be overconfident (as in fact it is).

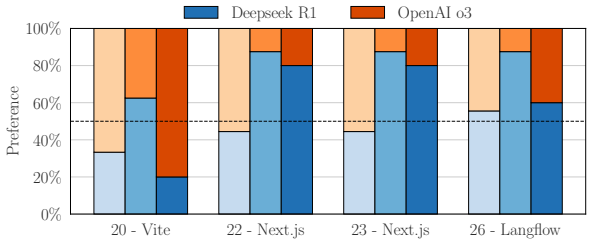

Figure 5.4: Human evaluation preferences per event, grouped by expertise level (low, medium, high).

5.2.1 Human evaluation details

On Fig. 5.5, I provide the in-depth results of my survey on the *Completeness*, *Usefulness* and *Logical coherence* of the agent in a subset of testing incidents. All criteria consistently receive average scores > 4, indicating overall appreciation. When asked to provide optional feedback, participants generally underlined that the responses of the o3-based agent were less verbose and therefore more functional (e.g., in 20 - Vite).

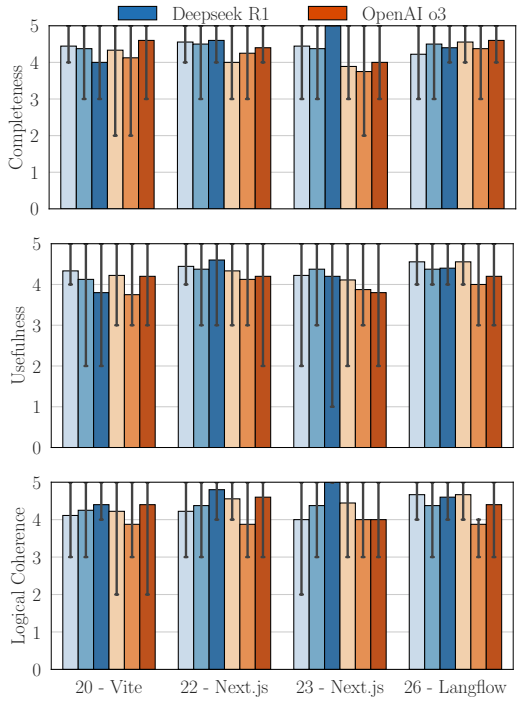

Figure 5.5: Average score for each criterion, grouped by incident and expertise level (low, medium, high). Whiskers indicate min and max scores.

Chapter 6

Conclusion and Limitations

I discuss the current limitations of the best architecture and provide some grounding for future exploration of this topic.

6.1 Limitations

6.1.1 Balancing Flexibility and Reliability

Network incidents require forensic agents to flexibly seek heterogeneous evidence. To approximate human practice, I designed the Tshark Expert Agent (TEA) architecture. However, in practice, granting TEA broad freedom led the agent to (i) lose direction, (ii) misuse `tshark` commands, and (iii) produce unfocused analyses. To mitigate this, I introduced Flow Reporter Agent (FRA), which restricts actions to inspecting connection payloads via a flows-summariser sub-agent. This improves reliability, but at a cost: some outcomes are encoded in transport-layer dynamics rather than payload content. For instance, in *Erlang/OTP* (incidents 8–9 in Tab. 2.2), determining the result of an SSH-based attack depends on whether the server terminates the connection with a TCP RST or continues. This evidence is not visible in payloads alone, and, therefore, FRA cannot resolve such cases. Future work should retain FRA's efficiency while recovering TEA's flexibility, e.g., by training a dedicated `tshark`-expert sub-agent that can execute and validate commands safely.

6.1.2 No Feedback Loop for Self-Correction

In my best-performing architecture (FRA), agents investigate incidents sequentially. The main agent receives the pre-processed output from the PCAP flows reporter and cannot invoke it again for targeted follow-ups. This contrasts with real-world practice, where teams of experts (i) jointly analyse and brainstorm on the same case and (ii) share access to the raw evidence. Consistent with prior observations [18], I found that tightly orchestrating multiple agents – especially over long interactions with evolving hypotheses – is challenging, reducing opportunities for self-correction. Future work should explore orchestration strategies that enable reflective feedback loops (e.g., cross-agent critique,

mediated tool calls) without sacrificing reliability, whether through more advanced architectures or improved prompting. In addition, iterative analysis challenges the LLM's natural ability to process a lot of data. Thus, splitting the data and analysis into chunks may impair the agent's ability to arrive at the correct conclusions [40], as noted already in [7].

6.1.3 Log Integration Pitfalls

Counter-intuitively, adding application logs to FRA architecture did not yield measurable gains. Logs often introduced noise, especially when they contained no incident-relevant evidence, causing the agent to over-attend to spurious details rather than decisive network signals. Future work should better harmonise the input information (e.g., salience-aware summarisation, confidence-weighted gating, or explicit "ignore-logs" decisions) to prevent bias from uninformative inputs.

6.2 Conclusion

In this thesis, I introduced an autonomous LLM-based agent for cyber forensics and incident response that supports Blue Team operations. By systematically analysing network traces, it identifies the targeted application, exploited vulnerability, and success of the intrusion, while generating structured forensic reports.

My results show that LLM-based agents can meaningfully support forensic investigations. I systematically compared different architectures: from single-agent baselines to multi-agent pipelines with expert sub-agents. Results reveal three main lessons:

1. specialised sub-agents combined in a multi-agent architecture outperform a single all-purpose agent;

2. simple orchestration pipelines are simpler and better than deeply inter-communicating agent designs, which often suffer from coordination failures;

3. more evidence is not always better, e.g., integrating system logs can distract from critical traffic patterns and reduce agent performance.

I found that GPT-5 and Deepseek R1 performed the best, allowing the agent to correctly identify exploited CVEs in up to 80% of the 10 most recent 2025 incidents. Finally, 22 experts confirmed the value of the reports generated, which were rated accurate, complete, and operationally useful.

Bibliography

[1] Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, 2025.

[2] Kamel Alrashedy, Abdullah Aljasser, Pradyumna Tambwekar, and Matthew Gombolay. Can llms patch security issues?, 2024.

[3] Leonardo Berti, Flavio Giorgi, and Gjergji Kasneci. Emergent abilities in large language models: A survey. *arXiv preprint arXiv:2503.05788*, 2025.

[4] Dipkamal Bhusal, Md Tanvirul Alam, Le Nguyen, Ashim Mahara, Zachary Lightcap, Rodney Frazier, Romy Fieblinger, Grace Long Torales, Benjamin A. Blakely, and Nidhi Rastogi. Secure: Benchmarking large language models for cybersecurity, 2024.

[5] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[6] Shuaihang Chen, Yuanxing Liu, Wei Han, Weinan Zhang, and Ting Liu. A survey on llm-based multi-agent system: Recent advances and new frontiers in application. *arXiv preprint arXiv:2412.17481*, 2024.

[7] Francesco De Santis, Kai Huang, Rodolfo Valentim, Danilo Giordano, Marco Mellia, Zied Ben Houidi, and Dario Rossi. Cfa-bench: Cybersecurity forensic llm agent benchmark and testing. In *To appear in the Proceedings of 10th International Workshop on Traffic Measurements for Cybersecurity (WTMC 2025)*, 2025.

[8] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, and Junxiao Song et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[9] Denis Donadel, Francesco Marchiori, Luca Pajola, and Mauro Conti. Can llms understand computer networks? towards a virtual system administrator, 2024.

[10] Jin Du, Menglin Liu, Shijie Zhang, Zekun Xiao, Yiwei Wu, Liangming Pan, Yankai Zhang, Bin Bi, and Minlie Huang. Ice cream doesn't cause drowning: Benchmarking llms against statistical pitfalls in causal inference. *arXiv preprint arXiv:2505.13770*, 2025.

[11] Mohamed Elaraby and Diane Litman. Arc: Argument representation and coverage analysis for zero-shot long document summarization with instruction following llms. *arXiv preprint arXiv:2505.23654*, 2025.

[12] Tingchen Fu, Mrinank Sharma, Philip Torr, Shay B. Cohen, David Krueger, and

Fazl Barez. Poisonbench: Assessing large language model vulnerability to data poisoning. *arXiv preprint arXiv:2410.08811*, 2024.

[13] Yujia Gao, Yuwei Xiong, Xiaohua Gao, Kang Jia, Jian Pan, Yutong Bi, Yufei Dai, Jian Sun, and Hao Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[14] Jianing Geng, Biao Yi, Zekun Fei, Tongxi Wu, Lihai Nie, and Zheli Liu. When safety detectors aren't enough: A stealthy and effective jailbreak attack on llms via steganographic techniques. *arXiv preprint arXiv:2505.16765*, 2025.

[15] Google Developers. Programmable search engine api. https://developers.google.com/custom-search/v1/overview, 2024. Accessed: 2025-06-30.

[16] Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, xu Shi, Tieqiao Zheng, liangfan zheng, Bo Zhang, Ke Xu, and Zhoujun Li. OWL: A large language model for IT operations. In *The Twelfth International Conference on Learning Representations*, 2024.

[17] Pouya Hamadanian, Behnaz Arzani, Sadjad Fouladi, Siva Kesava Reddy Kakarla, Rodrigo Fonseca, Denizcan Billor, Ahmad Cheema, Edet Nkposong, and Ranveer Chandra. A holistic view of ai-driven network incident management. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets '23)*, pages 180–188, Cambridge, Massachusetts, USA, November 2023. ACM.

[18] Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*, 2025.

[19] Nayeon Lee, Wei Ping, Peng Xu, Mostofa Patwary, Pascale Fung, Mohammad Shoeybi, and Bryan Catanzaro. Factuality enhanced language models for open-ended text generation. In *Advances in Neural Information Processing Systems*, 2022.

[20] Guancheng Li, Yifeng Li, Wang Guannan, Haoyu Yang, and Yang Yu. Seceval: A comprehensive benchmark for evaluating cybersecurity knowledge of foundation models. https://github.com/XuanwuAI/SecEval, 2023.

[21] Yuxuan Li, Yuxian Zhang, Qingxiu Liu, Yitong Li, et al. Fundamental capabilities of large language models and their applications in domain scenarios: A survey. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

[22] Yuhe Liu, Changhua Pei, Longlong Xu, Bohan Chen, Mingze Sun, Zhirui Zhang, Yongqian Sun, Shenglin Zhang, Kun Wang, Haiming Zhang, Jianhui Li, Gaogang Xie, Xidao Wen, Xiaohui Nie, Minghua Ma, and Dan Pei. Opseval: A comprehensive it operations benchmark suite for large language models, 2024.

[23] Zefang Liu. Secqa: A concise question-answering dataset for evaluating large language models in computer security, 2023.

[24] Zefang Liu. Multi-agent collaboration in incident response with large language models. *arXiv preprint arXiv:2412.00652*, 2024. Version 2.

[25] Yukai Miao, Yu Bai, Li Chen, Dan Li, Haifeng Sun, Xizheng Wang, Ziqiu Luo, Yanyu Ren, Dapeng Sun, Xiuting Xu, Qi Zhang, Chao Xiang, and Xinchi Li. An

empirical study of netops capability of pre-trained large language models, 2023.

[26] W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences of the United States of America*, 116(44):22071–22080, 2019.

[27] OpenAI. Gpt-4 technical report. Technical report, OpenAI, 2023.

[28] OpenAI. Openai text embedding models. https://platform.openai.com/docs/guides/embeddings, 2023. Accessed: 2025-06-30.

[29] OpenAI. Gpt-4o technical report. https://openai.com/research/gpt-4o, 2024. Accessed: 2025-07-03.

[30] OpenAI. Web search with google api (bring your own browser tool). https://cookbook.openai.com/examples/third_party/web_search_with_google_api_bring_your_own_browser_tool, 2024. Accessed: 2025-06-30.

[31] OpenAI. Introducing gpt-5, 2025.

[32] OpenAI. Reasoning models, 2025.

[33] OpenAI, Josh Achiam, Steven Adler, and Sandhini Agarwal et al. Gpt-4 technical report, 2024.

[34] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. Version updated 12 February 2024; Accessed: 2025-06-27.

[35] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[36] Recorded Future. State of ai and automation in threat intelligence. https://pages.recordedfutureext.com/State-of-AI-2025.html, 2025. Accessed: 2025-10-07.

[37] Mark Scanlon, Frank Breitinger, Christopher Hargreaves, Jan-Niclas Hilgert, and John Sheppard. Chatgpt for digital forensic investigation: The good, the bad, and the unknown. *arXiv preprint arXiv:2307.10195*, 2023.

[38] SentinelOne. Cybersecurity statistics: The ultimate list for 2024, 2024. Accessed: 2025-06-27.

[39] Mohammed Latif Siddiq and Joanna C. S. Santos. Securityeval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques. In *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security*, MSR4P&S 2022, page 29–33, New York, NY, USA, 2022. Association for Computing Machinery.

[40] Guijin Son, SangWon Baek, Sangdae Nam, Ilgyun Jeong, and Seungone Kim. Multi-task inference: Can large language models follow multiple instructions at once? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5606–5627, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[41] Kimi Team, Yifan Bai, Yiping Bao, and Guanduo Chen et al. Kimi k2: Open agentic intelligence, 2025.

[42] The Wireshark Team. tshark(1) - wireshark manual pages. https://www.wireshark.org/docs/man-pages/tshark.html, 2024. Accessed: 2025-07-03.

[43] The Wireshark Team. wireshark-filter(4) - wireshark display filter syntax. https://www.wireshark.org/docs/man-pages/wireshark-filter.html, 2024. Accessed: 2025-07-03.

[44] Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui, Weichuan Liu, Zhiyuan Liu, and Maosong Sun. Debugbench: Evaluating debugging capability of large language models, 2024.

[45] Norbert Tihanyi, Mohamed Amine Ferrag, Ridhi Jain, Tamas Bisztray, and Merouane Debbah. Cybermetric: A benchmark dataset based on retrieval-augmented generation for evaluating llms in cybersecurity knowledge, 2024.

[46] Catherine Tony, Markus Mutas, Nicolás E. Díaz Ferreyra, and Riccardo Scandariato. Llmseceval: A dataset of natural language prompts for security evaluations, 2023.

[47] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[48] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint*, arXiv:2201.11903, Jan 2022. Revised Jan 2023 (v6).

[49] Xingyu Wu, Kui Yu, Jibin Wu, and Kay Chen Tan. Llm cannot discover causality, and should be restricted to non-decisional support in causal discovery. *arXiv preprint arXiv:2506.00844*, 2025.

[50] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint*, arXiv:2401.11817, Jan 2024. Version 2 updated 13 Feb 2025.

[51] Kaiwen Yan, Hongcheng Guo, Xuanqing Shi, Jingyi Xu, Gu Yaonan, and Li Zhoujun. Codeif: Benchmarking the instruction-following capabilities of large language models for code generation. *arXiv preprint arXiv:2502.19166*, 2025.

[52] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint*, arXiv:2305.10601, May 2023. Preprint.

[53] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv preprint arXiv:2210.03629.

[54] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of LLM-based agents. *arXiv preprint arXiv:2503.16416*, March 2025.

[55] Zhipeng Yin, Zichong Wang, Weifeng Xu, Jun Zhuang, Pallab Mozumder, Antoinette Smith, and Wenbin Zhang. Digital forensics in the age of large language models. *arXiv preprint arXiv:2504.02963*, 2024.

[56] Xinyang Zhang, Kyusik Lee, Hrushikesh Karambelkar, Christopher Kruegel, Giovanni Vigna, Vineeth Avasarala, Sri Harsha Vemuru, Liron Werner, Todd Millstein, and Giovanni Vigna. Advancing cyber incident timeline analysis through rule-based ai and large language models. *arXiv preprint arXiv:2409.02572*, 2024. Version 3.

[57] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.