Politecnico di Torino

Master of Science in Computer Engineering

Master Degree Thesis

LLMs in the SIEM Loop: A Contract-Based Framework for Threat Detection with an Evaluation on Windows Telemetry and MITRE ATT&CK Mapping

Supervisors
prof. Andrea Atzeni

Candidate
Gabriele ESPOSITO

October 2025

Summary

Security Information and Event Management (SIEM) platforms centralize and correlate heterogeneous telemetry to surface suspicious behavior. Yet a stubborn gap remains between raw events and analyst-ready claims about what actually happened—claims that align with operational abstractions such as MITRE ATT&CK techniques. Large language models (LLMs) are a natural candidate for this semantic bridge: they read unstructured text well and can map descriptions to controlled vocabularies. However, the usual way LLMs are applied—open prompts, long contexts, free-form outputs—sits uneasily with security operations. Hallucinated details, brittle formatting, unclear provenance, and privacy constraints make naïve integration impractical. This thesis asks a practical question: how can a SIEM pipeline employ LLMs to transform telemetry into attack-informed, auditable artifacts under constraints of accuracy, privacy, and governance?

Rather than proposing a single "LLM for security," the thesis advances a modular architecture in which multiple LLM operators—potentially different models with different inductive biases—are composed under narrow contracts and surrounded by validation, retrieval, and feedback. Each operator performs one disciplined transformation of evidence (e.g., condense noisy events; map a behavior to ATT&CK; justify a claim), and each speaks a constrained interface, so that downstream components can enforce schema, check consistency, and keep provenance. The design goal is not maximum model power, but governability: the ability to reason about, audit, and evolve the system as models change.

To make this design concrete without overreaching, the thesis exercises one thin, end-to-end path through the architecture. A disciplined Windows lab (Hyper-V, clear snapshot per run) executes one Atomic Red Team technique per run; Sysmon and selected Windows channels are exported to XML to retain structured fields; a reporter operator compresses each run into one neutral sentence in technical English; a mapper operator outputs only a list of ATT&CK IDs as a plain string. The reporter is instantiated with GPT-4o (used via API for budget reasons in the lab); the mapper is instantiated as a compact, locally deployable Mistral-7B adapted with LoRA. GPT-4o is also used as a comparator mapper under the same "IDs-only" instruction. This path is intentionally austere—no retrieval, no validator, no feedback loop—so that behavior is attributable to the reporter-mapper pair rather than to orchestration.

Evaluation is separated by input regime. In-domain, immediately after fine-tuning, the adapted Mistral-7B is tested on the CTI dataset's held-out split—clean, human-written prose that matches the training style. Using the checkpoint at the minimum validation loss, Mistral-7B attains micro-F1 0.5748, macro-F1 0.2985, Exact Match 0.5808, and Hamming loss 0.0026. This establishes that a small, governable model does learn the narrative-to-ATT&CK mapping on its home turf. Cross-domain, the same mapping role is exercised on single-sentence summaries distilled from Windows logs—short, behavior-centric language that is terser than CTI prose and sensitive to omissions. On 70 Windows runs with single-label ground truth, GPT-4o (as a mapper) shows micro-F1 0.2806, Exact

Match 0.0286, Hamming 0.0408, while the fine-tuned Mistral-7B shows micro-F1 0.2707, Exact Match 0.2429, Hamming 0.0198. The contrast is operationally meaningful: GPT-4o is recall-heavy, often surfacing the correct ID among plausible neighbors but adding extras (hurting Exact Match and Hamming); Mistral-7B is precision-leaning, emitting tighter lists that improve Exact Match and reduce label-wise error at the cost of some misses. Because API budget precluded running GPT-4o across the entire CTI test split, a budget-limited subset of 70 CTI items (multi-label; not aligned to the previous runs) provides a prose-only head-to-head: GPT-4o achieves micro-F1 0.2215, Exact Match 0.0286, Hamming 0.0527; Mistral-7B achieves micro-F1 0.2535, Exact Match 0.1714, Hamming 0.0248. The same precision-versus-recall split persists on concise human prose.

These findings support three claims. First, summary fidelity sets the ceiling. When the sentence is explicit about actor, flags, object, and locus (e.g., regsvr32 with /i /s against a scriptlet; a Run-key write; a scheduled task), mapping is reliable; when the sentence is vague, the general model hedges and the adapted model declines to guess. This places a premium on reporter discipline: improving the prompt (e.g., obliging actor $\rightarrow$ flags $\rightarrow$ object $\rightarrow$ locus) is a high-leverage lever that requires no model changes. Second, model biases can be composed. A recall-heavy proposer and a precision-leaning confirmer become dependable once a Validation layer enforces schema, reconciles outputs with simple evidence cues, and gates promotions into a Detection Record. The goal is not a single perfect model but controlled complementarity. Third, small models under tight contracts are enough to shoulder SIEM-aligned roles with reproducible practice: reset-per-run lab hygiene, fixed prompts, short outputs, and archived per-run artifacts make behavior inspectable rather than anecdotal.

The work is intentionally scoped. It does not build correlation rules nor a full SIEM; it designs and evaluates LLM operators that fit inside a SIEM pipeline. Outputs are SIEM-ready: one sentence and a set of technique IDs are exactly the kind of governed artifact a validator can wrap into a Detection Record and forward to an alert store, where existing rules can corroborate, contextualize. The thesis also records real-world constraints: a small CTI training corpus (especially for rare techniques), a hosted reporter in the lab that a production SOC would replace with on-prem/VPC inference or sanitized inputs, and no retrieval/validation/feedback in the prototype path. These are not oversights; they are transparent boundaries that convert a sprawling problem into a tractable experiment.

The implications are pragmatic. In the near term, the path forward is to tighten the two contracts that define the prototype: strengthen the reporter's discipline and constrain the mapper's decoding against an explicit schema (valid ATT&CK IDs; optional short rationales kept separate from the ID list). Even without changing models, this reduces brittleness and gives a validator room to act. In parallel, modest data investments—a small paired set (logs, sentence, ATT&CK) and targeted augmentation with paraphrases and hard negatives—would improve cross-domain robustness without increasing model size. With those pieces in place, reintroducing validation, retrieval, and feedback turns the prototype into a governed product surface: validation to enforce format and consistency; retrieval to inject slow-moving organizational knowledge and reduce stale guesses; feedback to recycle confirmed hits, misses, and near-misses into periodic adapter refreshes. Finally, deployment posture matters: the lab used a hosted reporter for convenience, but a SOC needs local or VPC-isolated inference, prompt/model versioning, dataset lineage, and canary rollouts—engineering practice that makes LLM-augmented detection sustainable.

Overall, the thesis contributes: a survey that situates LLMs across reliability, DFIR/log analysis, and ATT&CK-aligned SIEM practice; a governed design framework with narrow

contracts and explicit guardrails; and a working instantiation with reproducible procedures and transparent limits.

Acknowledgements

Giunto al termine di questo percorso, sento il bisogno di esprimere la mia profonda gratitudine a tutte le persone che, in modi diversi, hanno contribuito alla realizzazione di questo lavoro e mi hanno sostenuto durante questi anni di studio.
Un ringraziamento speciale va al mio relatore, Prof. Andrea Atzeni, per la disponibilità, la guida preziosa e il suo approccio che mi ha ispirato a seguire questo tema di ricerca con passione e dedizione.

Contents

List of Figures 10

List of Tables 11

1 Introduction 13

1.1 Thesis organization 15

2 Background 16

2.1 Security Information and Event management (SIEM) 16

2.1.1 How SIEM works 16

2.1.2 Benefits of SIEM 17

2.1.3 Key to successful SIEM implementation 17

2.2 Artifical Intelligence 19

2.2.1 Natural Language Processing (NLP) 19

2.2.2 Generative AI 20

2.2.3 Large Language Models (LLMs) 22

2.2.4 Prompt Engineering 27

2.3 MITRE ATT&CK Framework 29

2.3.1 Why ATT&CK Framework is Important 29

2.3.2 The ATT&CK Matrix 30

2.3.3 Cyber Threat Intelligence 30

2.4 Virtualization 30

2.4.1 The main components of virtualization 32

3 State-of-the-Art 34

3.1 Hallucinations and Bias in LLMs 34

3.1.1 Prompt Engineering 35

3.1.2 Model Development 36

3.1.3 Other Techniques 37

3.2 Digital Forensics Reports written by LLMs 39

3.3 Log Analysis with LLMs . . . 40
3.3.1 SuperLog: Adapting LLMs for Log Analysis . . . 40
3.3.2 LogGPT . . . 43
3.3.3 Leveraging Language Models for Automated Shell Log Analysis: The LogPrécis Approach . . . 44
3.3.4 LoGBabylon: A Unified Framework For Cross-log File Integration And Analysis . . . 46
3.3.5 Overview of Log Analysis with LLMs . . . 48
3.4 MITRE ATT&CK: State of the Art and Way Forward . . . 48
3.5 SIEM solutions in modern cybersecurity . . . 50
3.6 SIEM rules with MITRE ATT&CK . . . 52
3.7 Summary-strengths and weaknesses . . . 54

4 Design 56
4.1 Introduction and Motivation . . . 56
4.1.1 Goals. . . . 56
4.1.2 Scope and non-goals. . . . 57
4.2 Architectural Overview . . . 58
4.3 Components and responsibilities . . . 61
4.4 Data models and Artifacts . . . 66
4.4.1 Design principles for artifacts . . . 66
4.5 Orchestration and operational considerations . . . 69
4.5.1 Event-driven orchestration in this context . . . 69
4.5.2 A concrete walk-through . . . 71
4.6 Positioning vs. the state of the art . . . 72
4.7 Limitations, risks, and mitigations . . . 73
4.7.1 Privacy, compliance, and data residency: local-only vs. API-based inference . . . 75
4.7.2 Limits that remain . . . 76

5 Implementation 77
5.1 Objectives and Scope . . . 77
5.2 Data and Models . . . 78
5.3 Simulation environment . . . 81
5.4 Testing methodology and results . . . 83
5.5 Discussion . . . 86

6 Conclusions 88

6.1 Challenges and Limitations 89

6.2 Future Improvements 90

A Mistral-7B fine-tuning 93

A.1 Overview and intent 93

A.2 Environment and dependencies 93

B Set-up lab environment 99

B.1 Sysmon and Log capture configuration set-up 99

B.2 Invoke-AtomicRedTeam set-up and execution 102

Bibliography 111

List of Figures

2.1 SIEM architecture [1] 17
2.2 Transformer-based models architecture 22
2.3 Mistral sliding window attention mechanism 26
2.4 Mistral 7B performance on various benchmarks 26
2.5 MITRE ATT&CK Enterprise Matrix (a subsection). 31

3.1 Taxonomy of hallucination mitigation strategies for LLMs 36
3.2 Architecture Diagram for Custom Hallucination Detection and Mitigation 37
3.3 Illustration on different temperature resulting in different possibility distributions 39
3.4 Illustration on the interpretable knowledge construction and continual pretraining of SuperLog. 42
3.5 LogBabylon's architecture 47
3.6 Behavioral analytic with the ATT&CK matrix: We identified three main relations with the ATT&CK matrix, i.e., Input, Support,and Comparison. 49
3.7 Workflow of the proposed AI agent: the first half 51
3.8 Workflow of Relationship Graph Construction 52
3.9 AI Agent-based RAM pipeline 54
3.10 Overview of prompt structure used in all steps of the pipeline. 55

4.1 Architecture of the proposed system. 59
4.2 Detailed architecture of the proposed framework, highlighting data flow, artifact placement, and validation guardrails. 65

5.1 Instantiation of the architecture from Chapter 4 in the lab, showing how data flows between components and where artifacts are placed in the process. 80
5.2 Comparison of GPT-4o and Mistral-7B performance on LLM-generated prompts (single TTP ground truth). 85
5.3 Comparison of GPT-4o and Mistral-7B performance on a CTI subset (multi-label, not aligned to Windows runs). 86

B.1 Listing available Atomic Red Team test for technique T1082 103
B.2 Checking prerequisites for Atomic Red Team test T1082 104

List of Tables

2.1 **Comparison of Mistral with LLaMA.** Metrics are split across two panels to keep body font size and respect margins. 26

3.1 Summary of serging findings for the forensic report analysis. 41

3.2 PLMs with context chunking and domain adaptation. CodeBERT with token classification task offers the best results (HaaS dataset). 46

5.1 Results using LLM-generated prompts (single TTP ground truth) 84

5.2 Results using human-written prompts (multi-TTP ground truth) 86

A.1 Hardware configuration and resource utilization 94

A.2 Training parameters 94

A.3 Training dynamics with minimum validation loss at step 2200 96

A.4 Test set evaluation metrics 98

B.2 Per-run predictions (plain strings returned by models). 107

B.1 Per-TTP detection under exact ATT&CK ID match. A checkmark indicates the model returned the ground-truth ID for that run. 110

Chapter 1

Introduction

In the age of digital transformation, cybersecurity has become a critical concern for organizations worldwide. With the increasing sophistication of cyber threats [2], it is essential for organizations to adopt comprehensive strategies to protect their assets, ensure data integrity, and maintain operational continuity.
The growing frequency and complexity of cyberattacks have led to the development of numerous frameworks and methodologies aimed at improving threat detection, response, and mitigation. To gain visibility into their digital environments and to effectively manage security incidents, organizations have increasingly adopted Security Information and Event Management (SIEM) solutions. SIEM platforms play a crucial role in aggregating and analyzing security data from diverse sources, enabling real-time detection, response, and mitigation of threats. Security operations have always been a race against asymmetry. Defenders must sift through high-volume, heterogeneous telemetry to spot a handful of meaningful behaviors, while adversaries need only one overlooked weakness to gain a foothold. Security Information and Event Management (SIEM) platforms were created to rebalance that race by centralizing logs, normalizing formats, and correlating events across hosts, networks, and applications. Yet modern SIEMs still face a familiar bottleneck: turning raw events into interpretable claims about what actually happened on a system, at a level of abstraction that aligns with an analyst's mental model.
Large language models (LLMs) appear, at first glance, to be ideal for this semantic bridge [3]. They are adept at reading unstructured text, composing explanations, and mapping descriptions to controlled vocabularies. But the way LLMs are typically deployed—general prompts over long contexts with open-ended outputs—does not transfer cleanly to regulated, high-stakes security workflows. The risks are well known: hallucinated details, fragile prompt surfaces, unpredictable formatting, unclear provenance, and non-trivial privacy implications if telemetry leaves the organizational perimeter. The challenge, therefore, is not "can an LLM read a log?" but rather how to integrate LLMs into a SIEM pipeline in a way that is effective, governable, and auditable.

This thesis is motivated by that practical question. Instead of proposing a single, monolithic "LLM for security," it argues for roles, interfaces, and contracts. LLMs should be treated as operators—potentially multiple, possibly different models—each responsible for a narrow transformation (condensing noisy events, extracting salient behavior, aligning outputs to ATT&CK Framework, or justifying a claim), each speaking a constrained interface that downstream components can validate. With that posture, we can reason about behavior, swap models, and scale capabilities without losing control of the surrounding system.
The central problem this dissertation addresses is: how can a SIEM pipeline employ

LLMs to transform raw telemetry into attack-informed, auditable artifacts, under constraints of accuracy, privacy, and operational discipline? Concretely, the work pursues three objectives. First, it seeks to design a conceptual framework for LLM-augmented detection that is modular and enforceable. The design must specify the roles played by different operators (not necessarily one model), the contracts at their boundaries (inputs and outputs), and the governance around them (validation, retrieval, feedback), so that the system remains explainable and maintainable as models evolve.
Second, it aims to instantiate one thin, end-to-end path through that framework to make behavior empirically inspectable. The path chosen in this thesis uses two LLM operators as a concrete example: a reporting operator that compresses Windows event logs into a single neutral sentence, and a mapping operator that translates that sentence into MITRE ATT&CK techniques. This is one instantiation among many the framework admits; it is chosen because it isolates the influence of summarization on detection and produces outputs that are easy to validate and score.
Third, it intends to evaluate this path under realistic constraints. The mapper is implemented as a compact, locally deployable model (Mistral-7B adapted with LoRA) to respect privacy and cost; GPT-4o is used where appropriate as a comparator and, for budget reasons, as the reporter in the lab via API calls. The evaluation is deliberately split across two input regimes: in-domain (CTI prose, immediately after fine-tuning) to establish that the mapping can be learned at all, and cross-domain (single-sentence summaries distilled from Windows logs) to probe how that mapping behaves when fed compressed, behavior-centric language from real telemetry. Where API budget precludes full-dataset comparisons, a budget-limited CTI subset is used and its limitations are made explicit.
By pursuing these objectives, the thesis does not claim to "solve" SIEM detection with LLMs. Instead, it proposes a governed way of using them—one that constrains inputs, expects narrow outputs, and relies on validation and feedback to keep errors contained. The results show that even a small slice of such a pipeline can be made legible and useful: summary fidelity proves to be the ceiling for downstream mapping; different models naturally separate along a recall-versus-precision axis; and once a validator is introduced, those complementary biases become an asset rather than a liability.

The approach taken in this work is intentionally incremental. Rather than treating the SIEM as a black box and bolting an LLM onto its side, the thesis decomposes the detection problem into operators connected by contracts. Each operator has a single, narrow responsibility and a constrained interface: what it may read, what it must produce, and which invariants its output must satisfy. This orientation allows the pipeline to remain intelligible even when individual models are replaced or chained.
Within this general architecture, the experimental program exercises one thin path end to end to make behavior observable. The path begins with telemetry capture in a disciplined Windows lab: A set of Atomic Red Team techniques is executed from a clean snapshot; a bounded time window of events is collected from Sysmon and selected Windows channels; and those events are exported as XML without premature flattening. This produces a run-scoped bundle that is easy to archive and inspect. The next operator is the reporter, which compresses the bundle into one neutral sentence in technical English. The sentence is not a narrative report and it does not name ATT&CK labels; it is a compact, behavior-centric description intended to surface the essentials—who acted, with which flags, on what object, and in which locus. The sentence becomes the input to the mapping operator, which returns only a list of ATT&CK technique IDs as a plain string. That austerity is deliberate: it separates learning from formatting, makes scoring unambiguous, and allows a future validator to enforce a schema with minimal friction.

1.1 Thesis organization

The document is structured to move from principles to practice without losing sight of operational realities. The Background chapter 2 gathers the technical context: how SIEMs ingest and normalize telemetry, how ATT&CK organizes adversary behavior, and how LLMs behave under different prompting and adaptation regimes, including the risks—hallucinations, prompt injection, brittle formatting—that matter in security settings. The State of the Art chapter 3 surveys emerging uses of LLMs in log analysis and threat intelligence and identifies the gaps this thesis chooses to address: the lack of governed integration patterns, the tendency to present single-model solutions, and the scarcity of reproducible end-to-end evaluations on real telemetry.
Building on that foundation, the Design chapter 4 presents the conceptual framework: a modular pipeline with several LLM operators, each defined by a role and a contract, surrounded by validation, retrieval, and feedback. It explains how evidence flows, how artifacts are shaped, and how the system remains auditable as components evolve. The Implementation chapter 5 then instantiates one thin path through that design—Windows lab, Atomic Red Team, Sysmon export, single-sentence reporting, IDs-only mapping—and evaluates it carefully in two regimes: CTI in-domain and Windows cross-domain, with a budget-limited CTI subset for head-to-head comparisons.
The Appendices A B provide the reproducibility backbone: fine-tuning details for Mistral-7B, environment setup and scripts for the Windows lab, and instructions to regenerate the per-run tables and figures.
This organization mirrors the argument of the thesis itself. Start with why and how; define who does what under which contract; run an instantiation, honest experiment; read the results through the lens of governance; and leave behind enough artifacts that another practitioner can pick up where the work stops.

Chapter 2

Background

2.1 Security Information and Event management (SIEM)

Security Information and Event Management (SIEM) these type of solutions collect, aggregate, and analyze large volumes of data from organizational systems and networks in real time, SIEM solutions provide a comprehensive view of an organization's security posture, empowering security operation centers (SOC) to detect, investigate, and respond to security incidents swiftly and effectively [4].
Key components of SIEM include:

- **Log Management:** SIEM solutions collect logs and events from various sources, including servers, network devices, applications, and security appliances. The goal of this data collection is to uncover anomalies that indicate a potential threat. Many SIEM solutions also ingest threat intelligence feeds, which allow security teams to identify and block emerging threats.
- **Event Correlation:** SIEM systems analyze the collected data to identify patterns and correlations that may indicate security incidents. This involves applying rules and algorithms to detect suspicious activities, such as unauthorized access attempts, malware infections, or data exfiltration. Event correlation helps detect complex attacks that may not be evident from individual events.
- **Incident response and monitoring:** To detect and respond to security incidents, SIEM solutions provide real-time monitoring and alerting capabilities. When a potential threat is identified, the SIEM system generates alerts that notify security analysts for further investigation, this enables rapid response to security incidents, minimizing potential damage.

2.1.1 How SIEM works

SIEM system continuously gather data from various sources, including servers, network devices, applications, and security appliances. This data is then normalized and stored in a centralized repository for analysis. Using algorithms and correlation rules, the system analyzes the data to identify patterns and anomalies in the normalized data, which may indicate security incidents. When a potential threat is detected, the SIEM system generates alerts that notify security analysts, these alerts are centralized in a dashboard, allowing security teams to monitor the security posture of the organization in real time.

Security analysts can then investigate the alerts, determine the severity of the incident, and take appropriate actions to mitigate the threat. This may involve blocking malicious IP addresses, isolating compromised systems, or initiating incident response procedures.

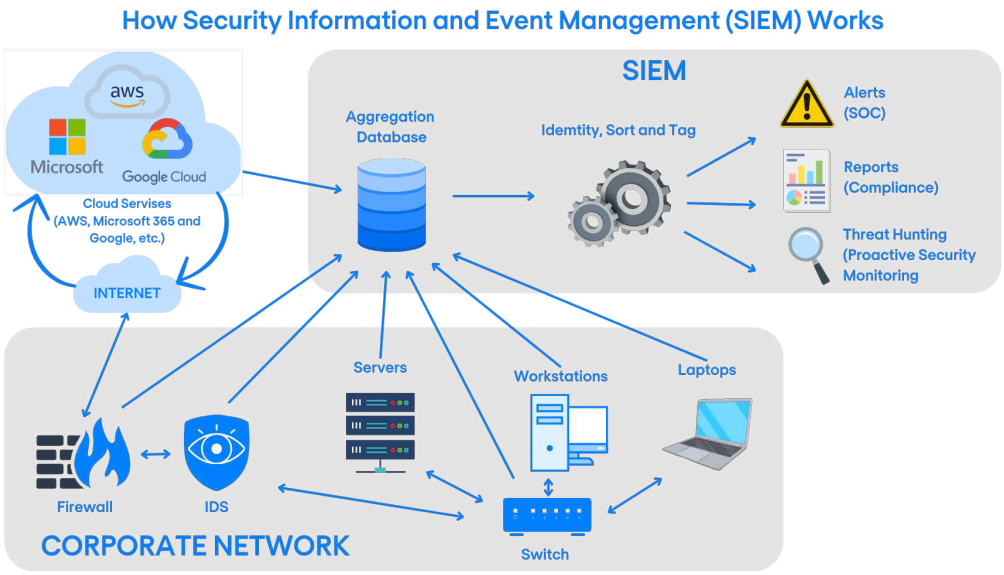

Figure 2.1: SIEM architecture [1]

2.1.2 Benefits of SIEM

SIEM solutions offer several benefits to organizations, including:

- **Enhanced Threat Detection:** By aggregating and analyzing data from multiple sources, SIEM solutions can detect complex threats that may go unnoticed by traditional security measures.
- **Improved Incident Response:** SIEM systems provide real-time alerts and centralized dashboards, enabling security teams to respond quickly to potential threats.
- **Compliance and Reporting:** Many organizations are required to comply with industry regulations and standards. SIEM solutions help automate compliance reporting by providing detailed logs and audit trails.

2.1.3 Key to successful SIEM implementation

Achieving effective SIEM implementation can be challenging, it demands carefull planning, ongoing maintenance, and alignment with organizational objectives. Key factors susch as clear goal-setting, appropriate tool selection, data management, tuning, automation, and skilled analysts all contribute to a successful SIEM deployment. Below are some key factors to consider for successful SIEM implementation [5]:

1. **Define Clear Security Objectives:** Establish clear security objectives and requirements that align with the organization's risk tolerance and compliance needs.

Objectives should include reducing indicent response times, achievieng specific compliance, mandates, or enhanciing detection of advanced threats. Clear objectives help the selection of use cases, data sources, and correlation rules, ensuring the SIEM's outputs align with business needs ans risk appetite.

2. **Select the Right SIEM Solution:** Theare are various SIEM solutions available, organization should evaluate their needs before selecting a solution. A good system must provide real-time monitoring, advanced analytics, integration with other security tools, autometed alert notifications, and flexible growth bespoke to the organization's needs. When selectint a SIEM solution, consider factors that integrate with existing security tools, such as firewalls, intrusion detection systems, and endpoint protection solutions. The more various data sources the SIEM can integrate with, the more comprehensive the security monitoring will be.

3. **Continuous Monitoring And Tuning for Optimal Performance:** SIEM systems require continuous monitoring and tuning to ensure optimal performance. Regulary review and update are essential to adapt to evolving threats and changing business requirements. It also means that tuning the system to reduce false positive alerts and improve the accuracy of threat detection is an ongoing process. Even the Endpoint Detection and Response (EDR) solutions require the enhancement of their detection capabilities through continuous learning and adaptation.

4. **Integrating Threat Intelligence:** Integrating threat intelligence feeds into the SIEM system can enhance its ability of detection and response to emerging threats. Threat intelligence provides valuable context about known threats, vulnerabilities, and attack patterns, enabling the SIEM to correlate events with external threat data. Having threat intelligence integrated into the SIEM system can help security teams to understand external threats and respond more effectively. This can identify more advanced threats (such as Advanced Persistent Threats (APTs)), where traditional security measures may not be sufficient.

5. **Automating Incident Response:** Automating incident response processes can significantly improve the efficiency and effectiveness of security operations. By integrating automation into the SIEM system, organizations can streamline their response to security incidents, reduce response times, and minimize the impact of attacks. Automation can also help to ensure consistent and repeatable incident response procedures, reducing the risk of human error.

6. **Establishing Comprehensive Reporting**: Consistent reporting ensures that security incidents are documented, analyzed, and communicated effectively. Reports may also include metrics that quantify the effectiveness of the SIEM system, such as the number of incidents detected, response times, and the impact of incidents on the organization. The reports should be easy to understand and provide actionable insights into security posture. In compliance they can also help to demonstrate adherence to regulatory requirements and industry standards.

7. **Fostering A Culture of Security Awareness:** Security awareness is crucial for the success of SIEM implementation. Professionals require training and education in utilizing the system effectively, understanding its capabilities, and recognizing potential threats. This is achieved by providing regular training sessions, workshops, and resources to keep security teams updated on the latest threats and best practices.

2.2 Artifical Intelligence

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and learn like humans, AI encompasses a wide range of technologies and techniques, including machine learning, natural language processing, deep learning, computer vision, and robotics. The goal of AI is to create systems that can perform tasks that typically require human intelligence, such as understanding natural language, recognizing patterns, making decisions, and solving complex problems. Each of these techniques has its own strengths and applications, and they can be combined to create more powerful AI systems. For example, natural language processing (NLP) is often used in conjunction with machine learning to enable machines to understand and generate human language, while computer vision is used to enable machines to interpret and analyze visual data.

- **Machine Learning (ML):** A subset of AI that focuses on the development of algorithms that allow computers to learn from and make predictions or decisions based on data. ML algorithms can be supervised, unsupervised, or semi-supervised, depending on the nature of the training data.
- **Natural Language Processing (NLP):** A field of AI that enables machines to understand, interpret, and generate human language, NLP techniques are used in applications such as chatbots, language translation, sentiment analysis, and text summarization.
- **Deep Learning:** A subset of machine learning that uses neural networks with multiple layers to model complex patterns in large datasets. Deep learning has been particularly successful in tasks such as image and speech recognition, natural language processing, and game playing.

2.2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on the interaction between computers and humans through natural language. The goal of NLP is to enable machines to understand, interpret, and generate human language in a way that is both meaningful and useful. NLP combines computational linguistics, machine learning, and deep learning techniques to process and analyze large amounts of natural language data. NLP encompasses a wide range of tasks, the most common of which include:

- **Text Analysis:** Analyzing and extracting information from text data, such as sentiment analysis, topic modeling, and named entity recognition.
- **Language Generation:** Creating human-like text based on input data, such as chatbots, automated content generation, and summarization.
- **Speech Recognition:** Converting spoken language into written text, enabling voice-activated systems and virtual assistants.

How NLP works

NLP works by combining linguistic rules with statistical and machine learning techniques to process and analyze natural language data. Here is an overview of a typical NLP pipeline and its steps [6]:

1. **Text Preprocessing**
   This step involves cleaning and preparing the text data for analysis. Common preprocessing tasks include tokenization (breaking text into words or phrases), after which the text is often converted to lowercase to standardize it to ensuring that words are treated uniformly. Stemming or lemmatization may also be applied to reduce words to theis base or root form (e.g., “running" to “run"), removing stop words (common words that do not carry significant meaning, such as “the," “is," “and”), and removing punctuation and special characters. After preprocessing, the text is clean, standardized and ready for machine learning models to interpret effectively.

2. **Feature Extraction**
   In this step, relevant features are extracted from the preprocessed text data, this may involve converting text into numerical representations, such as Bag of Words which quantify the presence and importance of words in a document. Other techniques include word embeddings (e.g., Word2Vec, GloVe) that capture semantic relationships between words by representing them as dense vectors in a continuous space. These representations allow models to understand the context and meaning of words based on their usage in sentences.

3. **Text analysis**
   The extracted features are then analyzed using various computational techniques. This process includes tasks sush as part-of-speech tagging (identifying the grammatical role of each word), named entity recognition (identifying and classifying entities such as names, dates, and locations), and sentiment analysis (determining the emotional tone of the text, assessing whether it is positive, negative or neutral). These analyses help in understanding the structure and meaning of the text.

4. **Model Training**
   Processed data is used to train machine learning models for specific NLP tasks. During this phase it adjusts its parameters to minimize errors ans improve its performance. Once trained, the model can be used to make predictions or generate outputs on new, unseen data. The effectiveness of NLP models is continuallt refined through iterative training and evaluation, ensuring that they can generalize well to different contexts and datasets.

2.2.2 Generative AI

Generative AI refers to a subset of artificial intelligence that focuses on creating new content, such as text, images, music, or other forms of media, using algorithms and models. Unlike traditional AI systems that primarily analyze and classify existing data, generative AI models are designed to generate new data that resembles the training data they were exposed to. This is achieved through techniques such as deep learning, neural networks, and probabilistic modeling. It has gained significant attention in recent years due to its ability to create realistic and high-quality content, leading to applications in various fields, including art, entertainment, marketing, and even scientific research.

How Generative AI works

Generative AI works by training models on large datasets to learn patterns and structrures within existing data to generate new and original new content. One of the breakthroughs

with generative AI models is the ability to leverge different learing approaches, such as unsupervised, semi-supervised learning for training. This has given the ability to more easily create models that can generate new content without requiring extensive labeled data to create foundation models. This foundational models can then be fine-tuned for specific tasks or applications, such as text generation, image synthesis, or music composition. An example of foundation models is the Generative Pre-trained Transformer (GPT) series, which are large language models that can generate coherent and contextually relevant text based on a given prompt. These models are trained on vast amounts of text data and can be fine-tuned for specific applications, such as chatbots, content creation, or language translation [7].

Types of Generative AI models

Generative AI models can be categorized into several types based on their architecture and the type of data they generate. Some common types include:

- **Diffusion Models:** These models are generative models that determine vectors in a latent space through a two-step process during training. The first step is called the forward diffusion where slowly adds noise to training data, while the second one is called the reverse diffusion where the model learns to denoise the data to reconstruct the data samples. Diffusion models have shown impressive results in generating high-quality images and other types of data. However they can be computationally intensive and require significant resources for training and inference.

- **Variatonal Autoencoders (VAEs):** VAEs consist of two main components, an encoder and a decoder. The encoder converts input data into a smaller latent space representaion. This compressed representation preserves the essential features of the data while reducing its dimensionality. The decoder then reconstructs the original data from this latent representation. This two components work together to learn a probabilistic model of the data, allowing the VAE to generate new samples by sampling from the learned latent space. VAEs are particularly useful for tasks such as image generation, anomaly detection, and data compression.

- **Generative Adversarial Networks (GANs):** Discovered by Ian Goodfellow in 2014, GANs consist of two neural networks, a generator and a discriminator, that work against each other in a game-theoretic framework. The generator creates synthetic data samples, while the discriminator evaluates whether the samples are real (from the domain) or fake (generated). The generator aims to produce data that is indistinguishable from real data, while the discriminator tries to correctly identify real and fake samples. Through this adversarial process, both networks improve over time, leading to the generation of high-quality synthetic data. GANs have been widely used for image synthesis, video generation, and other creative applications. While GANs can produce impressive results, they can be challenging to train and may suffer from issues such as mode collapse, where the generator produces limited diversity in its outputs, thus making GANs better suited for domain-specific data generation.

- **Transformer-based Models:** These models, such as the GPT series, use self-attention mechanisms to process and generate sequential data, particularly text. They are capable of capturing long-range dependencies and context in the data,

making them effective for tasks like language generation, translation, and summarization. Transformer-based models have revolutionized natural language processing and are widely used in various applications. Two mechanisms that are commonly used in transofermer-based models are the self-attention mechanism and the positional encodings. The self-attention mechanism allows the model to weigh the importance of different words in a sentence when generating or understanding text, enabling it to capture context and relationships between words effectively. Positional encodings are used to provide information about the position of words in a sequence, allowing the model to understand the order of words and their relationships within a sentence.

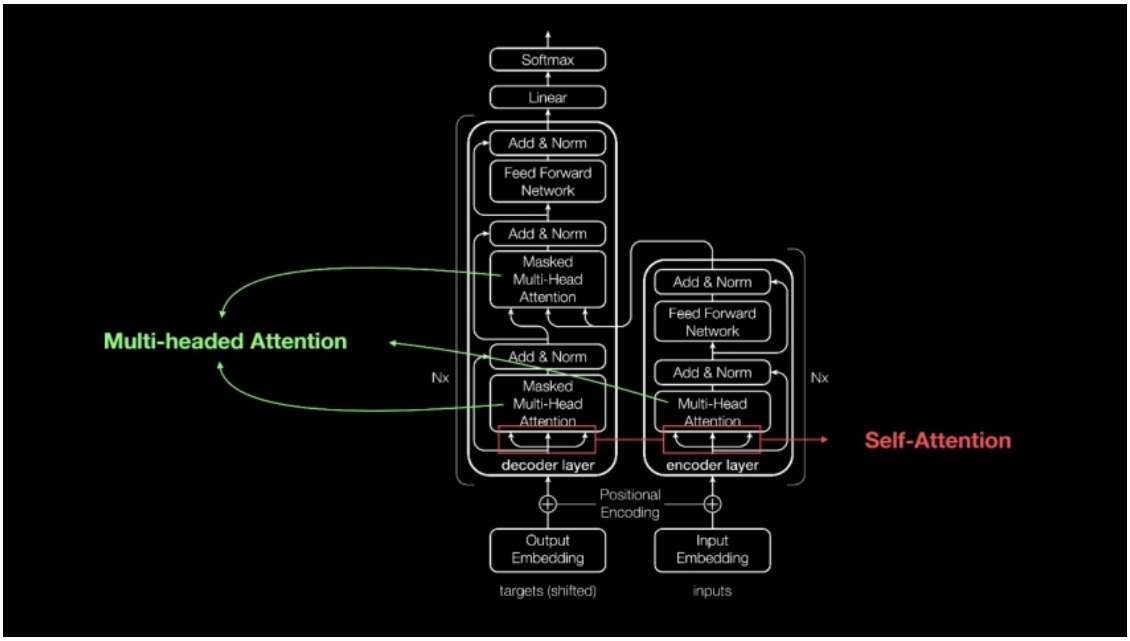

Figure 2.2: Transformer-based models architecture

2.2.3 Large Language Models (LLMs)

Large language models (LLMs) are a category of foundation models trained on immense amounts of data making them capable of understanding and generating natural language and other types of content to perform a wide range of tasks [8].
LLMs are a class of foundation models which are trained on vast amounts of data to understand and generate natural language and other types of content, in addition to other types of content based on the training data. They have the ability to infer from context, generate coherent text and contestually relevant responses, translate languages, summarize text, and even perform tasks like question answering and text completion. They are able to do this thanks to bilion of parameters that are used to capture the nuances of language and context, allowing them to generate human-like text and understand complex queries.

How Large Language Models (LLMs) work

LLMs operate by leveraging deep learning techniques and vast amounts of training data. These models are typically based on transformer architectures, like the generative pre-trained transformer (e.g. GPT-3, GPT-4, LLama, etc.), which excels at processing sequential data, such as text.
They consist of multiple layers of neural networks, each with parameters that can be fine-tuned during training which are enhanced by the self-attention mechanism, during the training phase, LLMs learn to predict the next word in a sentence based on the context provided by the preceding words. The model assigned probability score to the recurrence of words that have been tokekinezed (broken down into smaller units, such as words or subwords), and these tokens are then transformed into numerical representations (embeddings) that capture their semantic meaning.
The training process involves exposing the model to vast amounts of text data, allowing it to learn patterns, grammar, and context. Once trained, LLMs can generate text by sampling from the learned probability distribution, selecting the most likely next word based on the context provided by the input, the result is coherent and contextually relevant text that resembles human language. The model can also be fine-tuned, prompt-tuning (like reinforcement learning from human feedback (RLHF)) to remove biases, hallucinations, improve performance on specific tasks and not less important for do not expose organizations to unwanted liability, or cause damage to the organization or its customers.

Applications of Large Language Models (LLMs)

LLMs have a wide range of applications across various domains, including:

- **Text Generation:** LLMs can generate coherent and contextually relevant text, making them useful for content creation, storytelling, and creative writing.
- **Content Summarization:** LLMs can summarize long documents or articles, extracting key information and presenting it in a concise format.
- **Language Translation:** LLMs can translate text from one language to another, enabling cross-lingual communication and understanding.
- **AI assistants:** LLMs can power chatbots and virtual assistants, providing users with information, answering questions, and assisting with various tasks.
- **Sentiment Analysis:** LLMs can analyze text to determine the sentiment or emotional tone, which is useful for understanding customer feedback, social media sentiment, and market trends.

GPT-3 and GPT-3.5 Models

Gpt-3, introduced in 2020 by OpenAI, is a large language model based on the Transformer architecture, designed to capture long-term dependencies in text data. GPT-3 is generative pre-trained transformer model that has been trained on a large and diverse dataset to achieve a high level of natural language understanding and generation. It has 175 billion parameters, making it one of the largest language models. This has allowed GPT-3 to improve its performance on NLP tasks. A key feature of GPT-3 is the introduction of in-context learning, a method that allows the model to adapt to new tasks,

this do not require fine-tuning or additional training, but rather relies on prompting the model to manipulate it and generate contextually relevant responses. The GPT-3 architecture uses a unidirectional transformer decoder, which processes text in auto-regressive manner, meaning that it generates text one token at a time based on the previous tokens in the sequence. This happens because the attention matrix prevents that model from attending to future tokens during training, allowing it to generate text in a left-to-right manner. The large size of the network, with its 175 billion parameters, enables a strong generalization capability, allowing it to perform well on a wide range of tasks without the need for task-specific training.
GPT-3 has some limitations, such as the inability to handle long-term dependencies effectively, complex prompts, and the potential for generating biased or harmful content.

On the other hand, GPT-3.5 is an improved version of GPT-3, addressing some of the limitations of GPT-3. It employs a subfield of AI known as Reinforcement Learning from Human Preferences (RLHF), which means that human feedback is used to improve machine-learning algorithms [9]. Like GPT-3, GPT-3.5 is a Transformer-based autoregressive language model, it uses a similar multi-layer decoder stack, attention mechanism, and positional encodings. It is also optimised for coding tasks, but also to increase the ability to understand the prompt, in addition to natural language modelling and generation.

GPT-4 Model

GPT-4 is a large-scale, multimodal language model developed by OpenAI, which can accept both text and image inputs, generating text outputs. GPT-4 exhibits human-level performance on various professional and academic benchmarks. GPT-4 is a Transformer-based model pre-trained to predict the next token in a document. The post-traineing alignment process results in improved performance in measures of factuality and adhrence to desire behavior [10]. In addition the dense focus is enriched with advanced mechanism for processing long context, allowing it to handle documents with thousands of words, exapanding the number of tokens that can be processed simultaneously. The multimodal archietecture introduces an integrated data fusion pipeline, enabling the model to process and understand images and text together, enhancing its ability to generate contextually relevant responses based on both types of input. The distinction between GPT-3.5 and GPT-4 comes out when the complexity of the task increases, as GPT-4 demonstrates superior performance in understanding and generating complex text.
Despite its capabilities, GPT-4 is not without limitations. It can still produce incorrect or nonsensical answers, and it may be sensitive to input phrasing, meaning that slight changes in the input can lead to different outputs. Additionally, while GPT-4 has been trained on a diverse range of data, it may still exhibit biases present in the training data, which can affect its responses. OpenAI continues to work on improving the model's performance and addressing these limitations through ongoing research and development. Greate care should be taken when using GPT-4 in sensitive applications and contexts (such as security, healthcare, and legal domains), where the accuracy and reliability of the generated content are critical.
The training of GPT-4, like its predecessors, was conducted to predict the next word in a document, using a large corpus of publicly available data (such as internet data). The data is a web-scale dataset including correct and incorrect solution to mathematical problems, self-contradictory statements, and representing a great variety of domains and topics. So when prompt with a question, GPT-4 can generate a variety of ways that might be far from the user's intent, to align its responses with user intent, the model has

been fine-tuned using reinforcement learning from human feedback (RLHF).

Mistral7B model

Mistral 7B is an open-source large language model released by Mistral AI in September 2023. Despite having only 7.3 billion parameters, it consistently outperforms larger models—such as Llama2 13B—on a wide range of benchmarks, and even rivals Llama1 34B in many reasoning and code-generation tasks [11]. It shipped in both a “base" and an “instruction-tuned" version, the latter being fine-tuned to follow user prompts in a chat setting. of the most discussed language models in the AI community because of its performance with just 7B parameters [12]. Mistral has been engineered for superior performance and efficiency, outperforming the best open 13B models across all evaluated benchmarks. It is a decoder-only transformer model, which means that it resembles the decoder part of the transformer architecture. A lot of language models are decoder only since they are designed to generate text, which does not require bidirectional processing. Each token it is represented by a vector of 4096 dimensions, the attention block are 32 different heads, whereas the transformer had 8 heads but Mistral uses a different attention mechanism Knows as Grouped Query Attention (GQA) with sliding windows. Context length is 8192 tokens, which is the maximum number of tokens that can be processed simultaneously.
In the normal self attention mechanism, each token attends to every other token in the sequence that comes before it, which can be computationally expensive for long sequences. Mistral applies a sliding window approach of some size w that does not let a token attend to all the previous tokens. As it shows in Figure 2.3, a windows size of 3 has been maintained along with the casual mask end this significantly reduces the computational cost for faster inference and training since it has to compute less dot products. This might lead to a loss of information since it is not able to capture long-term dependencies between tokens, but it is a trade-off that Mistral has made to achieve better performance and efficiency. But even though the sliding windows attention mechanism restricts direct attention to a local context, the multiple layers of the transformer blocks enable information to propagate through the sequence, allowing the model indirectly to capture long-term dependencies.
Another important feature of Mistral is KV cache with rolling buffer, the inference process performed is that it starts with a special token after which it generates the first word based on the start token, then from the previous tokens it generates the next word, and so on until another special token is reached. This process encounters a lot of redunant computations so MIstral uses a KV cache to optimize it only when the key and value vectors are cached while the query vector is recomputed for each step. BUt since it has a sliding window attention mechanism, it is not needed to recompute tokens that do not fall under the windows size, so it uses rolling buffer that limits the size of the cache to the size of sliding window.

Mistral 7B was compared to other models such as Llama models on various benchmarks with their own evaluation pipeline as it is shown in Figure 2.4 [12].

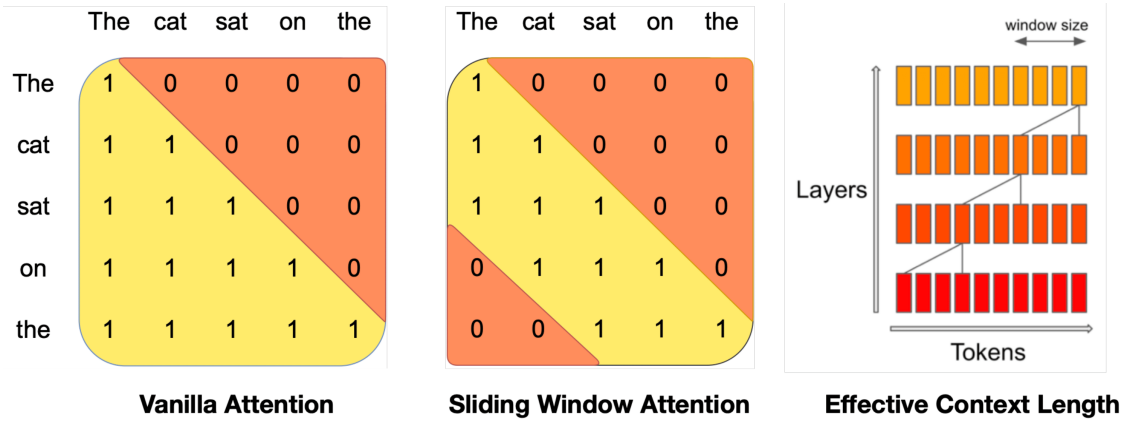

Figure 2.3: Mistral sliding window attention mechanism

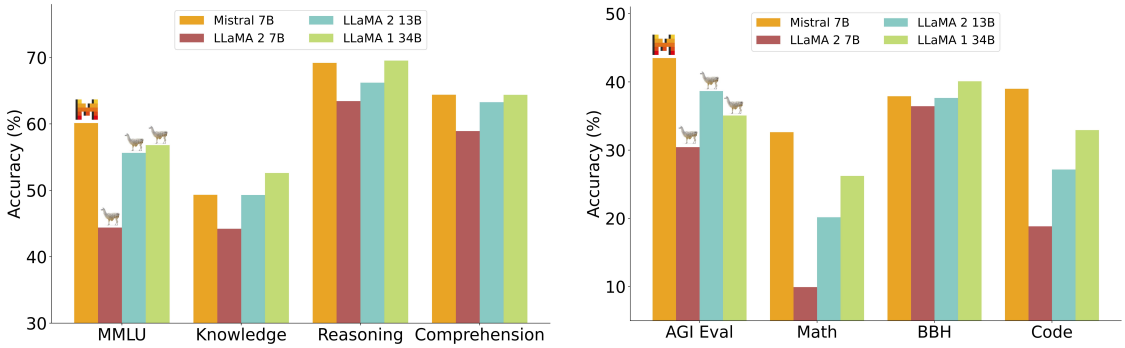

Figure 2.4: Mistral 7B performance on various benchmarks

Model	Modality	MMLU	HellaSwag	WinoG	PIQA	ARC Easy	ARC Chall.
LLaMA 2 7B	Pretrained	44.4%	77.1%	69.5%	77.9%	68.7%	43.2%
LLaMA 2 13B	Pretrained	55.6%	**80.7%**	72.9%	80.8%	75.2%	48.8%
Code-llama 7B	Finetuned	36.9%	62.9%	62.3%	72.8%	59.4%	34.5%
Mistral	Pretrained	**60.1%**	**81.3%**	**75.3%**	**83.0%**	**80.0%**	**55.5%**

Model	Modality	NQ	TriviaQA	HumanEval	MBPP	MATH	GSM8K
LLaMA 2 7B	Pretrained	24.7%	63.8%	11.6%	26.1%	3.9%	16.0%
LLaMA 2 13B	Pretrained	**29.0%**	**69.6%**	18.9%	35.4%	6.0%	34.3%
Code-llama 7B	Finetuned	11.0%	34.9%	**31.1%**	**52.5%**	5.2%	20.8%
Mistral	Pretrained	**28.8%**	**69.9%**	30.5%	47.5%	**13.1%**	**52.2%**

Table 2.1: **Comparison of Mistral with LLaMA.** Metrics are split across two panels to keep body font size and respect margins.

Mistral 7B has demonstrated that smaller model can achieve competitive performance by leveraging advanced techniques. Despite having only 7 billion parameters, it has shown that it can outperform many larger models across a wide range of benchmarks, including reasoning, code generation, and natural language understanding tasks. These strengths make Mistral 7B a compelling choice for researchers and developers looking for a balance between performance and computational efficiency, particularly suited in settings with

limited resources.

2.2.4 Prompt Engineering

Prompt engineering is the process of designing and optimizing input prompts to guide large language models (LLMs) in generating desired outputs. It involves crafting specific instructions, questions, or examples that effectively communicate the task or information needed from the model. The goal of prompt engineering is to elicit accurate, relevant, and contextually appropriate responses from LLMs, enhancing their performance on various tasks such as text generation, question answering, and language translation.
Prompt engineering represent an important aspect of working with LLMs, beacuse Large models are trained without any explicit knowledge of user intent, a good prmpt engineering will prove effective in guiding the model to produce their desired outputs. Infact much of the quality of the generated content depends on the the effective prompt.

How Prompt Engineering works

Prompt engineering works by carefully crafting input prompts that provide clear instructions and context to the LLM. There are different techniques and strategies that can be used to improve the effectiveness of prompts. Below are some common techniques used in prompt engineering [13]:

Zero-shot prompting

Zero-shot prompting is a technique where the model is asked to perform a task without any prior examples or training specific to that task. The model relies on its pre-existing knowledge and understanding of language to generate a response. This approach is useful when there are no labeled examples available for the task at hand. It has some limitations, most of the time, the output may not be as accurate or relevant as desired, where the answer could be vague or not aligned with the user's intent. This could be frustrating for users who expect more precise or contextually relevant responses and can lead to prompt multiple times to get the desired output.

Few-Shot Prompting

Few-shot prompting is a technique where the model is provided with a few examples of the desired output format or task before generating a response. It can be used as a technique to enalbe in-context learning where the user provides demonstrations of the desired output format or task within the prompt. These demonstrations help as a guide for the model to understand the expected output and generate responses that are more aligned with the user's intent. Below is an example of few-shot prompting [14]:

Prompt: A “whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is: We were traveling in Africa and we saw these very cute whatpus.
To do a “farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:
Output: When we won the game, we all started to farduddle in celebration.

It is important to note that the model has somehow learned to perform the task by providing just one example(i.e. 1-shot). Of course it is still not a perfect solution, especially when the task is complex or requires a deep understanding of the context, but it can significantly improve the quality of the generated content.

Chain-of-Thought Prompting

Chain-of-thought prompting is a technique that encourages the model to generate intermediate reasoning steps before arriving at a final answer. The prompt enables comples reasoning capabilities through the use of intermediate steps, it can be combined with few-shot prompting to provide examples of the reasoning process. Example:

Prompt: "The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.

The odd numbers in this group add up to an even number: 17, 10, 19, 4, 8, 12, 24.
A: Adding all the odd numbers (17, 19) gives 36. The answer is True.

The odd numbers in this group add up to an even number: 16, 11, 14, 4, 8, 13, 24.
A: Adding all the odd numbers (11, 13) gives 24. The answer is True.

The odd numbers in this group add up to an even number: 17, 9, 10, 12, 13, 4, 2.
A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is False.

The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:
Output: Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer is False.

However, chain-of-thought prompting can be computationally expensive and may not always lead to the desired output, its ability is still dependent on its training data and the complexity of the task.

Role-Based Prompting

Role-based prompting is a technique that involves assigning specific roles or personas to the model to guide its responses. Instead of simply asking a question, the user can frame the prompt in a way that encourages the model to respond as if it were a specific character or expert. This helps shape the style and tone of the response, making it more relevant to the user's intent. Example of role-based prompting:

Prompt: Act as a senior software engineer interviewing me for a backend role. Ask me two coding questions, then evaluate my answers and provide feedback.

2.3 MITRE ATT&CK Framework

The MITRE ATT&CK framework is a comprehensive knowledge base of adversary tactics, techniques, and procedures (TTPs) based on real-world observations of cyber attacks. It provides a structured approach to understanding and analyzing the behavior of threat actors, enabling organizations to improve their cybersecurity posture by identifying potential vulnerabilities and implementing effective defenses. The framework is organized into matrices that categorize adversary behaviors across different platforms, such as enterprise, mobile, and cloud environments. Each matrix consists of tactics, which represent the high-level goals of an adversary, and techniques, which describe the specific methods used to achieve those goals. The framework also includes sub-techniques that provide more granular details about how a technique can be executed. In detail, the framework is divided into several components: **tactics**, **techniques**, and **procedures**.
Tactics represent the high-level objectives that adversaries aim to achieve during an attack, these objectives outline the “why" behind an adversary’s actions and are categorized into stages of an attack lifecycle, such as Initial Access, Execution, Persistence, Privilege Escalation, Exfiltration, etc. Techniques define the specific methods or approaches that adversaries use to achieve their tactics, these techniques provide the “how" of an attack and include various methods such as phishing (to gain initial access), command and scripting interpreter usage (for executing commands on a compromised system), credential dumping (to obtain sensitive credentials), etc. Procedures are the specific implementations of techniques that adversaries use in real-world attacks, these procedures provide detailed information on how a technique is executed in practice, including tools, scripts, and workflows used by adversaries.

2.3.1 Why ATT&CK Framework is Important

ATT&CK framework was created to provide a common language and understanding of adversary behaviors, enabling organizations to share threat intelligence and collaborate on defense strategies. MITRE decided, based on their research and analysis of real-world cyber attacks, to create a framework that would help to address four main issues [15]:

1. Adversary behavior is complex and constantly evolving, making it difficult for organizations to keep up with the latest threats. Typical indicators such as domains, IP addresses, file hashes, registry keys, etc. were easily changed by adversary to evade detection, making it difficult to identify and respond to attacks.

2. The existing lifecycle models were too high-level and did not provide enough detail about adversary behaviors. The existing models were not comprehensive enough to cover the wide range of tactics and techniques used by adversaries.

3. Applicability to real enviroments. TTPs need to be based on real-world observations of adversary behavior, rather than theoretical models or assumptions. This ensures that the framework is relevant and applicable to real-world scenarios.

4. Common taxonomy. TTPs need to be organized in a way that allows for easy understanding and communication among security professionals.

An organization’s ability to detect and defend against cyber threats is greatly enhanced by strong offense and defense teams that work together to understand and mitigate risks, so the ATT&CK became the go-to tool for adversary emulation team to plan

events and for the detection team to verify the effectiveness. With this useful process the MITRE's reasearch program realeased the ATT&CK framework to the public in 2015, and it has since become a widely adopted standard in the cybersecurity community.

2.3.2 The ATT&CK Matrix

The ATT&CK matrix is a visual representation of the tactics and techniques used by adversaries during cyber attacks. For instance, as shown in Figure 2.5, the matrix is organized into columns representing tactics and rows representing techniques, the tactic Persistence (the adversary's goal of maintaining access to a compromised system) is represented by the first column, while the techniques used to achieve this goal are listed in the corresponding rows, where each cell in the matrix represents a specific technique (this technique may have sub-techniques as well), such as "Account Manipulation" or "Boot or Logon Autostart Execution", which are used by adversaries to maintain persistence on a compromised system. Of course each technique is bound to one or more tactics, like the "Account Manipulation" technique is bound to the Persistence and Privilege Escalation tactics.

This matrix is the most recognizable part of the ATT&CK framework because it is commonly used to show things like detection capabilities in security products, defensive coverage on an environment, and results of an incidents or red team engagements [15].

2.3.3 Cyber Threat Intelligence

Another important aspect of the ATT&CK framework is its focus on cyber threat intelligence (CTI). ATT&CK documents adversary groups—such as APT28, APT29, and FIN7—based on real-world observations of their tactics, techniques, and procedures (TTPs). By mapping these groups and their TTPs to the framework, organizations can gain a deeper understanding of threat actor behavior and prioritize defenses against the most commonly used techniques. Examples of how specific adversary employ techniques are documented in the framework page, which outlines each group's procedure for using the technique. A procedure represents a concrete instance of use and can be invaluable for understanding exactly how a technique is used in practice, for replicating an incident through adversary emulation and for pinpointing the specific indicators needed to detect that instance in action.

2.4 Virtualization

Virtualization is a process that allows multiple virtual instances of a computer system to run on a single physical machine. It involves creating virtual machines (VMs) that can run their own operating systems and applications, while sharing the underlying hardware resources of the host machine. Virtualization is widely used in cloud computing, data centers, and software development environments to improve resource utilization, flexibility, and scalability. It allows cloud service providers (CSPs) such as Amazon Web Services (AWS), IBM Cloud,Microsoft Azure, and Google Cloud Platform (GCP) to optimally utilize their IT infrastructure to deliver scalable resources. Virtualization offers numerous benefits to both on-premises and cloud environments, including [16]:

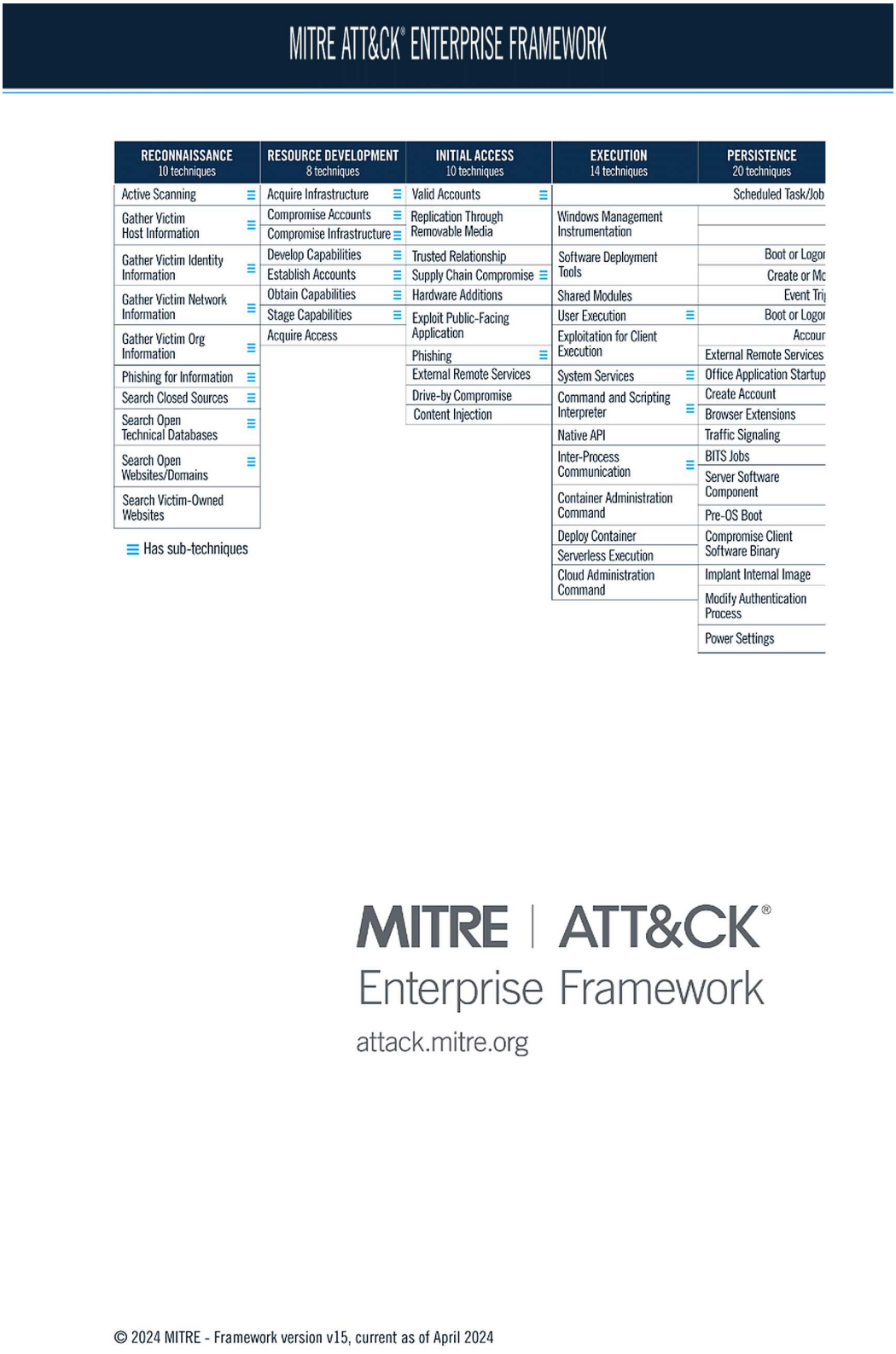

Figure 2.5: MITRE ATT&CK Enterprise Matrix (a subsection).

- **Resource Optimization:** In the past organizations used to allocate a dedicated physical CPU for each application, which resulted in underutilization of resources. Virtualization allows multiple virtual machines to run on a single physical server, optimizing resource utilization and reducing hardware costs.
- **Easier Management:** Replacing physical servers with virtual machines simplifies management tasks such as provisioning, scaling, and monitoring. Virtual machines can be easily created, modified, and deleted. For example, automated deployment and configuration tools allow administrators to quickly set up new virtual machines and Applications as services in software templates, reducing the time and effort required for manual setup. Additionally security policies can leverage security configurations based on the virtual machine's role.
- **Minimal downtime:** It allows administrators to run multiple redundant virtual machines on the same physical server to failover between them in case of a problem occurs.
- **Faster provisioning:** Provisioning new virtual machines is faster than setting up physical servers, as it involves copying existing virtual machine images or templates rather than installing an operating system and applications from scratch. VM management tools can automate the process of creating and configuring new virtual machines, reducing the time required to deploy new applications or services.
- **Disaster Recovery:** Virtual machines can be easily backed up and restored, facilitating disaster recovery and business continuity planning.
- **Cost-effectiveness:** Virtualization reduces hardware acquisition costs, maintenance and energy consumption, as fewer physical servers are needed to run multiple virtual machines.

2.4.1 The main components of virtualization

VMs are usually referred to as guests, with one or more guest machines running on a single host machine. In practice virtual machines consist in several files (configuration files, virtual hard drives, etc.) that are stored on the host machine. These files are used by the hypervisor to create and manage the virtual machines. A hypervisor is a software layer that coordinates the VMs, it serves as an interface between the VM and the host machine's hardware resources, ensuring that each of them has access to the necessary resources such as CPU, memory, storage, and network. It also manages the allocation of resources to each VM, ensuring that they do not interfere with each other.
There are two main types of hypervisors [16]:

- **Type 1 Hypervisor (Bare-metal):** This type of hypervisor runs directly on the host machine's hardware, without an underlying operating system. It provides better performance and resource utilization, as it has direct access to the hardware resources. Examples of Type 1 hypervisors include VMware ESXi, Microsoft Hyper-V, and Xen.
- **Type 2 Hypervisor (Hosted):** This type of hypervisor runs on top of a host operating system, relying on the host OS for resource management. It is generally easier to set up and use, but may have lower performance compared to Type 1 hypervisors. Examples of Type 2 hypervisors include VMware Workstation, Oracle VirtualBox, and Parallels Desktop.

Beyond server virtualization, there are other types of virtualization that can be used to optimize IT infrastructure management and resource utilization, these type of virtualization include Desktop virtualization, application virtualization, storage virtualization, network virtualization, and data virtualization.

Virtualization versus Containerization

Virtualization and containerization are two distinct technologies used to optimize resource utilization and improve application deployment. While both approaches aim to create isolated environments for running applications, they differ in their architecture and implementation.

Containerization involves packaging applications and their dependencies into lightweight containers that share the host operating system's kernel. Containers are more efficient in terms of resource utilization, as they do not require a full OS instance for each application. This allows for faster startup times and better scalability. However, containers provide a lower level of isolation compared to VMs, as they share the same kernel and may be more susceptible to security vulnerabilities.

Virtualization and Security

Virtualization provides numerous security benefits by isolating applications and services within virtual machines (VMs). Compared to containers and traditional deployment, VMs provide a higher level of isolation, as each VM runs with strong hardware isolation and has its own operating system instance, so one VM can be compromised without affecting others. This makes VMs a better choice for running untrusted applications or services, as the impact of a security breach can be contained within the compromised VM. If a VM is compromised or infected with malware, the impact is contained within that VM and can be rolled back to a previous state (snapshot) when the VM was stable, they can be easily restored to a known good state, thus minimizing the risk of data loss or corruption. Of course, this is not a silver bullet, as the hypervisor itself can be vulnerable to attacks, and if the hypervisor is compromised, it can lead to a security breach across all VMs, potentially allowing an attacker to access sensitive data or disrupt services. However, virtualization still provides a significant security advantage over traditional deployment methods, as it enables better isolation and containment of security incidents.

Security features for protecting VMs and the underlying physical hardware include access control, encryption, network segmentation, monitoring, logging, patch management, and regular security assessments. Software-based security solutions provide virtual monitoring tools that can address compliance, real-time threat detection, and incident response.

Chapter 3

State-of-the-Art

In this chapter, we examine the current state of the art in Large Language Models (LLMs) and their applications, with a particular focus on SIEM (Security Information and Event Management) systems and the integration of LLMs within them. We review various integration strategies proposed in recent years, alongside a survey of Log Analysis techniques and how LLMs can enhance their performance.

The chapter also discusses the main challenges and limitations associated with current approaches. Building on this foundation, in the next chapter, the work proposes a novel method that leverages the capabilities of LLMs to improve the efficiency and effectiveness of SIEM systems. Specifically, it focuses on post-attack log analysis and the automatic generation of a summary report detailing the nature and impact of the attack. This report is then mapped to the MITRE ATT&CK framework.

3.1 Hallucinations and Bias in LLMs

Nowadays, Large Language Models (LLMs) are widely used in various applications such as chatbots, content generation, and question-answering systems. However, these models often struggle with factual accuracy and consistency (bias and hallucination problem) in their responses. To address these challenges, several techniques have been developed to enhance the reliability of LLM outputs.
LLMs continue to evolve in their ability to generate human-like text, a key challentge remains the issue of hallucinations, where models produce confident but incorrect or nonsensical information. This issue is the biggest barrier to safetly deploying LLMs into real-world applications. Unlike traditional AI systems (focused on specific tasks), LLMs are trained on vast amounts of data, making them more prone to generating false information. This allows them to generate text that is coherent and contextually relevant, but it also means they can produce outputs that are factually incorrect or misleading. This is alarming when we rely on NLP capabilities for critical tasks such as medical diagnosis, legal advice, financial support, and cybersecurity. Small errors in these domains can lead to significant consequences, including financial loss, legal issues, and even threats to human life [17]. To mitigate these risks, researchers first introduce a structured classification of more than 32 mitigation strategies, grouped into three primary categories: Prompt Engineering, Model Development, and Supervised Fine-Tuning 3.1.

3.1.1 Prompt Engineering

Cybersecurity experts have long know that it takes a combination of human behavior and technology to battle phishing and ransomware attacks. The same is true for LLMs, where prompt engineering plays a crucial role for battling dangerous hallucinations responses from aberrant generative algorithms and the human prompts that trigger them.
Chain-of-feedback (CoF) prompting is a method where users iteratively guide AI by providing specific constructive feedback on its outputs, similar to mentoring a child. Instead of vague demands like “Try again" effectively prompts detail what the AI did well and what’s missing, steering it towards more accurate and relevant results. The tips for an optimal CoF prompting strategy include [18]:

- **Balance Specificity**: Avoid overly detailed prompts that stifle creativity but clarify desired tone, format, or style.
- **Focus on “Do" over “Don’t”**: AI often misinterprets negative instructions, so frame prompts positively.
- **Use Examples**: Showcasing desired outputs from other platform helps refine responses.
- **Collaborative Refinement**: Treat AI as a partner, combining human intuition with algorithmic precision for higher-quality results.

CoF and mindful prompt engineering bridge the gap between raw AI capability and context-aware, reliable outputs, emphasizing that human interaction is essential to harness AI’s potential responsibly.

The are several other prompt engineering techniques that can be used to mitigate hallucinations in LLMs, we list some of the most notable ones below [13]:

- **Chain-of-Verification (CoVe)**: Uses iterative verification steps to cross-check outputs for consistency and factual correcteness.
- **Step-Back Prompting**: encourages deeper reasoning processes within Large Language Models (LLMs) before arriving at final answers. By using prompts like “think through this task step-by-step,” you guide the model to engage in higher-level reasoning (for example analyze system logs step by step) rather than jumping to conclusions.
- **Feedback Loop Integration for Continuous Improvement in AI Systems**: Leverages user feedback to iteratively refine model performance and address errors.
- **Fact-Checking with external sources**: Cross-references outputs against trusted databases to validate accuracy (can be used to validate common attack patterns in cybersecurity).

Implementing these techniques may reduce hallucinations, enhance the reliability of AI-generated content. Of course the importance is to combine these techniques (if it is possible) with ongoing refinement to ensure safe and effective AI deployment.

Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a technique that incorporates external knowledge sources to ground model outputs in verifiable information. This approach address the key challenge of accuracy and currency in LLMS ouputs [19]. RAG effectively mitigates hallucinations by providing responses that are not only coherent but also factually accurate and verifiable.
Other notable methods include LLM-Augmenter, which iteratively refines prompts with feedback from external sources. Given a user query the framework first retrieves evidence from extarnal knowledge and performe reasoning to form evedince chain, then LLM-Augmenter queries an LLM using a prompt that contains the evidence for the LLM to generate a response based on the retrieved information. LLM-Augmenter validates the response (e.g. by checking if the response is consistent with the evidence) and generates a feedback message to refine the prompt. This process is repeated until the response meets the desired quality standards.
Other approaches include the use of FreshPrompt which generates up-to-date web search queries results into model responses. Real time strategies like EVER validate and correct hallucinations during the generation phase, while post-generations methods like RARR retrofit responses with research-backed.

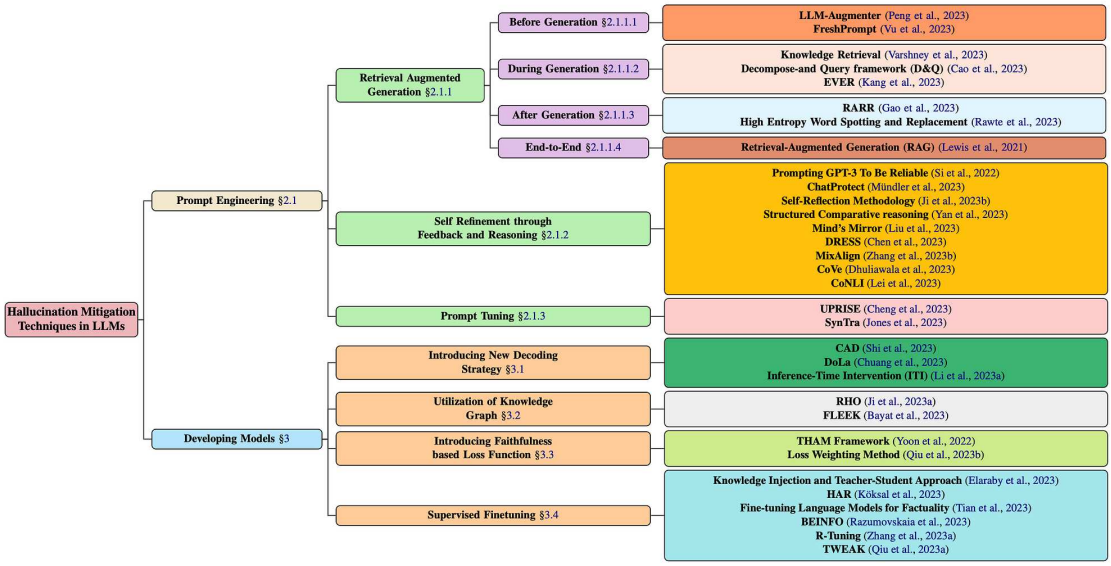

Figure 3.1: Taxonomy of hallucination mitigation strategies for LLMs

3.1.2 Model Development

Here, the focus is on architectural and algorithmic innovations. New Decoding strategies such as Context-Aware Deciding (CAD) and DoLa contrast models' outputs from various internal layers to enhance factual consistency. Knowledge graph-based methos like RHO and FLEEK integrate structured factual database into the generation process. Some models use faithfulness-based loss functions, including THAM and Loss Weighting, to penalize hallucinations during training.

Supervised Fine-Tuning

Supervised fine-tuning involves training LLMs on curated datasets that emphasize factual accuracy. SFT serves as a vital phase in aligning LLMs for specific tasks using labeled data.
It helps to follow human commands more accurately for specific tasks and eventually increases the faithfulness of the model's responses. Of course, the quality of the fine-tuning data is crucial, as it directly impacts the model's ability to generalize and produce accurate outputs. During this phase, the LLM's weights are adjusted based on the gradients from a task-specific loss function, which measures the difference between the model's predictions and the ground truth labels. This technique has proven particularly effective in improving the adaptability of LLMs, enabling them to perform at previously unseen tasks with high accuracy [19].

3.1.3 Other Techniques

As we said, in domains where incorrect information can have serious consequences, such as healthcare, finance, legal applications, and cybersecurity. Unchecked LLM outputs can undermine the reliability and trustworthiness of the system. We introcuded RAG, an approach that aims to reduce hallucinations in language models by incorporating the capability to retrieve external knowledge and making it part of the prompt that's used to generate the response, where the model uses the retrieved information in conjunction with prompts to generate the final output.
Amazon Web Services (AWS) has introduced a new feature called Amazon Bedrock Guardrails [20], which is designed to offer hallucination detection with contextual grounding checks, seamlessly applied using the APIs or embedded in applications workflow. After the model generates a response, Guardrails checks the output to see if hallucinations occurred. Amazon Bedrock Guardrails helps accelerate generative AI application development by orchestrating multiple tasks, it uses the reason capability of LLMs to break down user-requested tasks into smaller steps. The following illustrates the solution architecture:

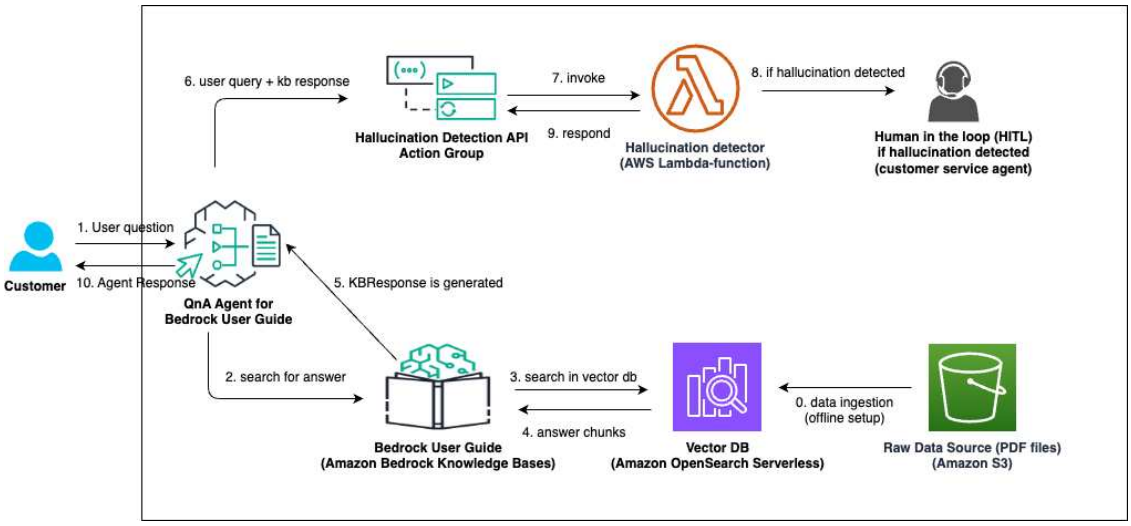

Figure 3.2: Architecture Diagram for Custom Hallucination Detection and Mitigation

The architecture consists of the following steps:

1. Data ingestion which involves raw data stored in Amazon S3 bucket.
2. The user provides relevant questions to the Amazon Bedrock User Guide, which are handled by the Amazon Bedrock Agent (set up to handle user queries).
3. The agent creates a plan and indentifies the need to use a knowledge base. The request is sent to the knowledge base, which retrieves relevant information from the underlying vector database. The agent retrieves and answer through RAG technique, in detail:
   - The query is directed to the vector DB.
   - Relevant answers chuncks are retrieved.
   - The knowledge base is generated from the retrieved chunks back to the agent.
4. The user query in conjuction with the knowledge base response are sent to invoke the correct action group.
5. This tuple is then passed to a Lambda function that calculate a hallucination score.
6. A notification (Amazon uses SNS service) is triggered to alert if the score is lower than a certain threshold.
7. Instead if the score is above the threshold, the hallucination detector set up in Lambda responds with a final knowledge base response. Otherwise, it generates a default response to the user asking to wait until a custom service agent joins the conversation.
8. The final agent response is sent back to the user in the chatbot UI.

Other techniques that are foundations of reliable AI interactions especially in an automed system are:

- **Input Validation**: Decrease the risk of malicious prompt (injection attacks).
- **Context Window Management**: Smart token distribution between user inputs and system prompts leads to better context management. Models can access external information through retrieval augmented generation (RAG) without overloading the context window.
- **Temperature and Top-p Sampling Control**: Language models use temperature settings to balance creativity and predictability. Tasks that need accuracy work best with lower temperature (close to 0) because they produce more focused and deterministic outputs. Higher temperatures let the model explore different possibilities, but can lead to less reliable results. The Figure 3.3 illustrates the impact of temperature on LLM outputs [21].
- **Implementing Guardrails in Prompts**: safeguard against AI Hallucinations used for proper validation mechanisms. This reduces the risk of generating harmful or misleading content. Some examples of basic guardrails include:
  - **Data Validation Checks**: Provides detailed validation through multiple layers of protection. Undesirable content is filtered, and any personal data is redacted by the system. Then checks verify information using mathematical processes to match outputs with expected results.

  - **Output Format Enforcement**: Structured outputs help prevent hallucinations by ensuring that the generated content adheres to a specific format, making it easier to validate and verify. (e.g. JSON schema implementation lets developers define expected structures for API responses).

- **Testing and Monitoring**: Regularly test and monitor LLM outputs to identify and address potential issues help prevent AI hallucinations in production environments. Team can maintain high-quality outputs and minimize risks with proper evaluation. Piepelines with continuous integration and continuous deployment (DevOps) help teams check AI responses automatically.

- **Fine-Tuning**: Adjusting model parameters or retraining models with additional data that specifically targets identified weaknesses or biases can help improve the model's performance and accuracy to reduce the possibility of hallucinations. This helps align the model outputs with the desired behavior, addressing any gap that initially led to hallucinations.

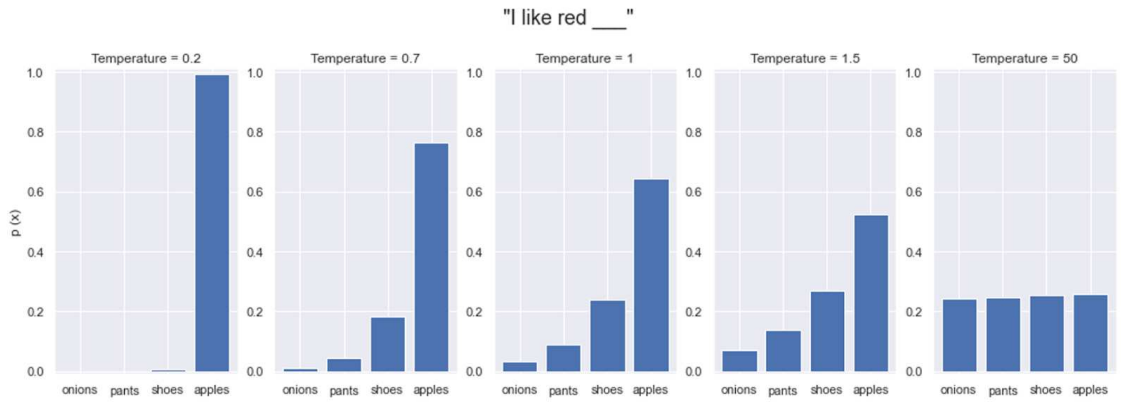

Figure 3.3: Illustration on different temperature resulting in different possibility distributions

In conclusion, preventing AI hallucinations just needs a complete approach that combines prompt engineering, resilient guardrails, and continuous monitoring. Researchers shows that good prompt engineering can significantly boost response accuracy by 30% [22]. Building reliable AI applications that users trust needs constant watchfulness, commitment and adaptation to new challenges. Rigorous testing and evaluation before LLM deployment, establishing clear operational boundaries and incorporating user feedback are essential to mitigate the risk of LLM hallucinations. Furthermore, continuous monitoring and improvement of LLMs (strategic fine-tuning) are vital to ensure that LLM applications not only meet but exceed standards of reliability and safety [23]. The importance of these measures is not only for technical reasons, but protecting information's integrity and keeping users safe from harmful misinformation. Put these techniques into practice today to build a safer and more reliable AI future.

3.2 Digital Forensics Reports written by LLMs

The advent of LLMs has has introduced new possibilites for automating complex language-based tasks across a variety of domains. In digital Forensics, where the articulation of technical findings into clear, coherent, and legally sound reports is critical and time-consuming task, LLMs are beginning to demonstrate potential as supportive tools. It is

crucial to enphasize that LLMs are not intended to replace human experts, but rather to assist them in generating reports that are more efficient and effective. Concerns around factual reliability, data confidentialitiy, and legal admissibility persist, especially given the high-stakes nature of digital evidence in juridical context.

While automation has been successfully integrated into many early phases of digital forensics investigations (e.g. data acquisition), report generation remains one of the least supported yet most critical tasks. This stage demands clarity, precision, and legal defensibility, all of which are tradittionally grounded in expert human judgment and domain knowledge. However, the potential of LLMs to assist examiners in drafting reports more efficiently, particularly in high-volume environments where backlogs are common.
Recent studies [24] have shown empirically that LLMs can assist in writing digital forensics reports. Their findings reveal that certain sections, such as the Introduction, Items Received, and Methodology, are well-suited for LLM assistance due to their strucured input data and relatively low variability. Conversely, sections requiring evaluative judgment or interpretative reasoning, like the Discussion and Conclusion, pose greater challenges for automated text generation due to the reliance on investigator experience, context, and nuanced interpretation 3.1.
The study also highlights that LLMs should be regarded not as autonomous report writers but as assistive tools. Generated texts often require exstensive proofreading, factual verification, and stylistic adjustment before they can be included in a final report. In some cases, as we said in the previous section 3.1, LLMs introduce hallucinated content or make plausible yet inaccurate assumptions, which could compromise the reliability of evidence presentation if unchecked. Nonetheless, when appropriately integrated, LLMs can significantly reduce the time spent on report drafting, allowing forensic experts to focus on higher-level analysis and decision-making. Their role fits within a broader trend of leveraging AI for augmentation rather than replacement in critical decision-making domains.

3.3 Log Analysis with LLMs

Log analysis is a critical component of cybersecurity, enabling organizations to monitor, detect, and respond to security incidents. LLMs can enhance log analysis by automating the extraction of relevant information from vast amounts of log data, identifying patterns, and generating insights. By leveraging LLMs, security teams can improve their incident response times and reduce the cognitive load associated with manual log analysis.

3.3.1 SuperLog: Adapting LLMs for Log Analysis

As modern IT infrastructures scale in complexity and heterogeneity, the volume, variety, and velocity of log data generated by systems pose significant challenges to traditional log analysis approaches. Conventionally, engineers have relied on manual inspection or rule-based systems to extract insights from system logs, which is not only time-consuming but often fails to scale or generalize across environments. The advent of large language models (LLMs) has sparked new interest in automating log-related tasks such as parsing, anomaly detection, root cause analysis, and interpretation. However, the application of general-purpose LLMs (such as GPT-3.5, GPT-4) to log analysis reveals a major shortcoming: a fundamental mismatch between the structure of system logs and the natural language these models are trained on. Logs are typically terse, highly domain-specific, syntactically

Section	Purpose and Content	Structure and Elements	Input data source	LLM-potential
Introduction	Provides a summary of the mandate and the investigation context, includes crime description, suspect(s), investigator(s), transmitted items, and the prosecutor questions	Text that usually follows the mandate structure	Mandate and Lab Log	High
Received Items	Description of seized items including characteristics: size, hash (if forensic image), or physical condition (if device)	Combination of text, tables, and lists describing each item	Mandate, Lab Log and Tool report (note that this last source only presents partial data)	High
Methodology	Details analysis procedure, i.e., steps followed (incl. justification) and tools used (incl. versions)	Text or list of taken steps/applied tools in chronological order	Literature and Lab Log	High
Results	Provides an overview of the results, e.g., a list of every artifact of interest identified, and their characteristics	Combination of texts, tables, and lists with varying structure	Lab Log and Tool report	Medium*
Discussion	Discusses the meaning of the results in the context of the investigation, and the limits of the undertaken analysis†	Text with varying structures for each mandate's question (the evaluation usually comes at the end)	Investigator knowledge, experience, and opinion (parts of the data may be in the lab log)	Low
Conclusion	Summarizes the important elements of each prior section	Text following the overall structure of the report	Prior elements	Medium–Low

* Low for the whole section but high for various components within the section.
† An evaluation of the results under the light of each hypothesis is also present if the report is evaluative.

Table 3.1: Summary of serging findings for the forensic report analysis.

irregular, and lack the narrative context that LLMs excel at processing. To address this gap, the paper [25] introduces SuperLog, a novel adaptation of open-source LLMs for the domain of log analysis that successfully bridges the divide between structured operational logs and the natural language interface of LLMs.

The core innovation lies in a newly proposed Continual Pre-Training (CPT) paradigm that injects interpretable domain knowledge into a general-purpose LLM, enabling it to

understand, process, and reason over log data while maintaining its fluency in natural language. Instead of relying solely on raw log corpora, which offer limited semantic clarity, the authors constructed NLPLog, a large-scale dataset containing over 250,000 question-answer pairs derived from real-world logs across 14 diverse domains (including Windows, Linux, MacOS, Hadoop, Spark, and more). Each Q&A pair is designed to provide rich interpretive context across five dimensions of operational understanding: Grok pattern parsing, event insight generation, root cause analysis, component correlation, and failure forecasting. Importantly, these pairs are expressed in natural language, ensuring that the model internalizes structured knowledge in a format that aligns with its pretraining corpus, thereby avoiding the interpretability degradation often seen in LLMs retrained on raw logs.

To prepare the data, the authors developed a multi-stage (Figure 3.4) pipeline involving deduplication, log event reconstruction, and interpretable knowledge generation. They utilized existing tools such as LogPPT to extract log templates from massive volumes of semi-structured logs, then populated these templates with representative variables to simulate realistic events. GPT-4 was employed via carefully designed prompts to generate meaningful natural language explanations, questions, and answers around each log instance. In contrast to prior work—such as BigLog or LogRobust, that employed domain-adapted pretraining with raw logs or structured anomaly labels, this method enriches the model's domain understanding with human-like interpretability [25].

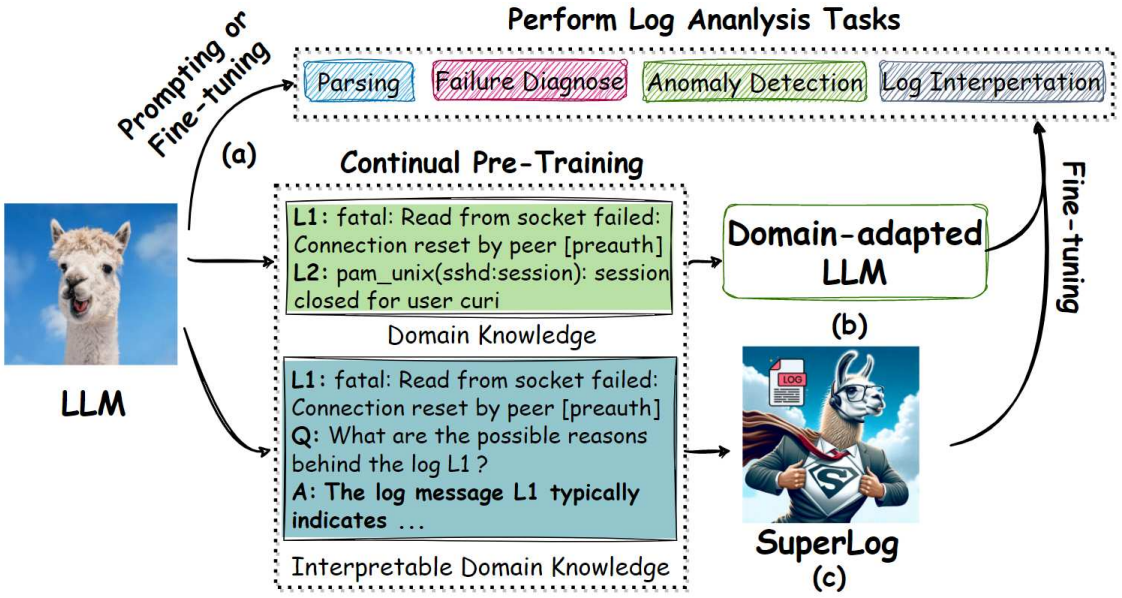

Figure 3.4: Illustration on the interpretable knowledge construction and continual pre-training of SuperLog.

During the continual pre-training phase, the authors trained SuperLog using the 7-billion-parameter LLaMA2 model, incorporating NLPLog to shift the model's attention toward domain-specific log features while preserving its original language generation capabilities. Fine-tuning was conducted in two distinct modes: task-based fine-tuning, where the model is trained on specific datasets for log parsing and anomaly detection (e.g., LogHub, BGL, Spirit); and instruction-following fine-tuning, where the model learns to generalize across log tasks by following user prompts in natural language (using instruction datasets like Alpaca enriched with clustering and ranking). This dual fine-tuning strategy makes SuperLog both domain-specialized and interactive, capable of following diverse instructions without task-specific retraining.

The model's performance was evaluated on four core tasks [25]: log parsing, anomaly detection, log-based failure diagnosis, and log interpretation. In all cases, SuperLog outperformed baseline methods, including state-of-the-art LLMs such as GPT-4 and Claude-3, as well as log-specific models like OWL-7B and DevOps-14B. For log parsing, it achieved near-perfect RandIndex and F1-scores across five datasets. For anomaly detection, it showed significant gains in both session- and template-level evaluations, surpassing prior works like LogBERT and LogAnomaly. Notably, in log interpretation tasks—measured via GPT-4-based grading of readability and usefulness—SuperLog demonstrated the highest average scores, confirming that the integration of interpretable domain knowledge enhances not only task performance but also the clarity and human-alignment of model outputs. Additional ablation studies confirmed that removing the CPT phase or replacing interpretable data with raw logs led to notable performance degradation, particularly in interpretive tasks, highlighting the importance of natural-language supervision.
An especially compelling aspect of SuperLog's capabilities lies in its generalization to unseen domains. The model was evaluated on Apache and OpenStack logs—datasets excluded from the CPT and fine-tuning processes. Compared against high-performing baselines, SuperLog achieved significantly higher ROUGE scores, demonstrating its capacity to transfer learned reasoning to novel environments. These results indicate that the model's internalization of log reasoning patterns is not domain-locked, but rather general and reusable—an essential property for practical deployment in dynamic IT environments. Moreover, the model's architecture supports offline deployment, unlike proprietary LLMs that require API access and present latency, cost, and security concerns.

3.3.2 LogGPT

Unlike existing approaches that rely on individual models tailored for specific tasks such as parsing, anomaly detection, or root cause analysis, LogGPT aims to provide a general-purpose, instruction-following log analysis framework based on the powerful capabilities of LLMs. To overcome the semantic gap between structured log data and natural language, and to ensure the model can understand, reason, and generate human-interpretable explanations, the authors [26] develop a comprehensive training methodology that includes multi-stage instruction tuning and domain adaptation via synthetic and real-world logs. At its core, LogGPT is trained to accept task-specific prompts that include both a high-level instruction (e.g., "parse this log") and input log content, and to produce structured or textual outputs in response. This prompt-based interaction model enables it to flexibly support a wide range of tasks including log parsing, anomaly detection, log summarization, fault localization, root cause analysis, and more—offering a unified solution rather than a fragmented toolset.

A critical contribution of the paper is the design of a multi-stage fine-tuning strategy that guides the LLM to generalize across diverse log analysis tasks while maintaining high interpretability and reliability. The authors construct a large-scale log-centric instruction dataset that includes synthetic logs with known structures and patterns, as well as real-world logs obtained from public datasets such as BGL, HDFS, and Thunderbird. This dataset includes thousands of task-formulated prompt-output pairs spanning multiple log formats, failure scenarios, and operational contexts. The training process involves several stages: starting with a base LLM (LLaMA2), the model is adapted first through domain exposure using synthetic logs, and then through task-specific instruction tuning with human-generated or simulated outputs. Importantly, the authors use chain-of-thought prompting to encourage the model to output interpretable reasoning traces—particularly

valuable for tasks like root cause analysis or anomaly justification. To prevent overfitting and ensure generalization, they also implement input diversity strategies and task randomization across training batches.

LogGPT's architecture and training methodology enable it to tackle both structured prediction and generative reasoning tasks, something that traditional log analysis models cannot do [26]. For instance, in log parsing, the model is capable of identifying log templates and variable fields directly from free-text logs; in anomaly detection, it can flag unusual log sequences based on learned patterns; and in summarization, it can generate natural language explanations of log segments or sequences. Additionally, the model supports log-based question answering, enabling interactive diagnostic scenarios where an engineer can query a system state in plain English and receive structured or textual insights. The versatility of LogGPT is validated through comprehensive evaluations on multiple benchmarks. In log parsing, it outperforms strong baselines such as Drain and LogPai. In anomaly detection, it shows superior performance to both rule-based and neural methods (e.g., LogAnomaly, DeepLog), particularly in scenarios involving unseen failures or noisy data. For root cause analysis and fault localization, LogGPT not only achieves high accuracy but also provides traceable, explainable reasoning steps, which is a significant improvement over black-box models.
It demonstrates strong generalization to unseen log formats, with few-shot prompting or zero-shot capabilities that allow it to handle novel system logs with minimal tuning. The authors also show that the model's instruction-following nature makes it adaptable to operator feedback or domain-specific constraints, making it suitable for integration into interactive debugging tools or DevOps platforms. A notable advantage of LogGPT over proprietary APIs (e.g., GPT-4) is its offline deployability and transparency—an important consideration for enterprise environments where data privacy, latency, and customization are critical. By releasing a publicly available version and accompanying training data, the authors contribute a significant foundation for further research in log-intelligent systems.

3.3.3 Leveraging Language Models for Automated Shell Log Analysis: The LogPrécis Approach

Here the researchers [27] propose LogPrécis, an innovative framework that utilizes pre-trained language models (PLMs) to automate the analysis of malicious Unix shell logs, offering a novel methodology for extracting high-level semantic information from raw logs in the form of attack fingerprints.
LogPrécis is designed to parse raw shell sessions—typically collected through honeypots—and assign MITRE ATT&CK tactics to each command or token in the session. These tactics act as interpretable, high-level labels representing the attacker's goals (e.g., Execution, Persistence, Discovery, Defense Evasion). This process results in a tactic sequence, or fingerprint, that characterizes each attack session. Through a combination of domain adaptation, few-shot fine-tuning, and a careful comparison of language model architectures, LogPrécis demonstrates that it is possible to reduce over 400,000 raw attack logs to approximately 3,000 unique fingerprints, thereby enabling efficient summarization, pattern detection, and forensic analysis.
The study rigorously evaluates various pre-trained models—such as BERT, CodeBERT, CodeBERTa, and GPT-3—and experiments with multiple tokenization strategies (tokens, words, statements), chunking methods, and training regimes (from-scratch, domain-adapted, fine-tuned). Among the models tested, CodeBERT with token-level classification, context-aware chunking, and domain adaptation performed best, achieving over 85% ROUGE-1 score and 66.9% fidelity on the classification task 3.2. The results show

that PLMs pre-trained on code significantly outperform generic NLP models, and that adapting them to the specific shell log domain substantially improves performance.

The process starts with domain adaptation, where the PLMs are further trained on Unix shell logs in an unsupervised fashion to better understand the peculiarities of shell syntax, attacker tools, and command chaining. The fine-tuning step then uses a small but curated set of 360 labeled attack sessions to train a classifier to assign MITRE tactics to log tokens. Despite the limited labeled data, the model shows strong generalization thanks to few-shot learning, and continues to improve as more samples are added. An important insight from the study is that contextual chunking, where each segment of a shell session is fed to the model along with a few preceding and following commands, greatly enhances the model's ability to correctly classify the intent behind each command—showing that context is crucial in understanding the semantics of attacker behavior.
The paper also addresses the challenges of ambiguous commands. For example, a command like rm (remove) can be part of Persistence, Impact, or Defense Evasion, depending on context. LogPrécis can disambiguate such cases by analyzing surrounding commands and system state. It is demonstrated that while classic approaches like static rule matching or Word2Vec-based embeddings fail in these scenarios, PLMs with attention mechanisms excel by capturing semantic relationships and positional dependencies. In addition to the per-command classification, LogPrécis builds higher-level representations by grouping commands with the same tactic and generating session-level tactic fingerprints. These fingerprints facilitate several downstream applications:

- Clustering and deduplication of similar attack sessions.
- Tracking the evolution of malware families (e.g., variants of the DOTA cryptominer).
- Anomaly and novelty detection, by flagging the emergence of new, unseen tactic sequences.
- Real-time alerting in SOC pipelines with reduced false positives and better prioritization.

An illustrative use case involved the emergence of a new fingerprint involving the **lockr** command, which LogPrécis flagged in December 2022. This command was later confirmed by external reports to be part of a novel SSH-based persistence attack targeting WordPress and Drupal environments. The fact that LogPrécis detected this shift months in advance highlights its potential as a proactive tool for cyber threat intelligence. From a performance standpoint, the study shows that LogPrécis offers the best trade-off between effectiveness, efficiency, and cost. It runs faster and more accurately than GPT-3, with inference costs near zero since it uses open-source models. In contrast, GPT-3, while competitive in terms of ROUGE-1, incurs substantial computational and monetary costs, making it less suitable for real-time or large-scale deployment. The authors also built a practical implementation of LogPrécis using Python, Elasticsearch, and Kibana, creating an interactive dashboard where analysts can explore log sessions by tactic sequences, time trends, or individual commands. This integration shows that LogPrécis is not just a proof-of-concept but a deployable system with tangible benefits for operational security.

Model	Entity	Accuracy	ROUGE-1	Fidelity
CodeBERT	token	**0.912**	**0.853**	**0.669**
CodeBERT	word	0.896	0.823	0.594
CodeBERTa	token	0.889	0.817	0.506
BERT	token	0.902	0.811	0.556
BERT	statement	0.909	0.807	0.614
BERT	word	0.885	0.791	0.486
CodeBERTa	statement	0.885	0.788	0.553
CodeBERTa	word	0.863	0.781	0.406
CodeBERT	statement	0.877	0.739	0.522

Table 3.2: PLMs with context chunking and domain adaptation. CodeBERT with token classification task offers the best results (HaaS dataset).

3.3.4 LoGBabylon: A Unified Framework For Cross-log File Integration And Analysis

LogBabylon introduces a unified framework for log file integration and analysis that leverages advances in Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG). The framework addresses the challenges of heterogeneous log formats, enabling seamless integration and analysis across diverse log sources. At its core, LogBabylon is designed to automate the entire pipeline of log analysis (from classification to interpretation, Figure 3.5) across heterogeneous and unstructured log sources. Its architecture is composed of three main stages [28]:

1. **Classification and Template Extraction**: LogBabylon classifies incoming logs using a prefix parse tree structure and a lightweight preprocessing method. Instead of relying on hand-crafted rules or regular expressions, the system employs LLMs to extract log templates dynamically. This strategy enables high adaptability to new log types and reduces the need for domain-specific expertise. A novel matching algorithm—combining strict, loose, and fallback LLM-based matching—ensures that both known and unseen log patterns can be parsed accurately. The system organizes logs into clusters based on semantic similarity and maintains a template pool that evolves over time.

2. **Consolidation via Retrieval-Augmented Generation (RAG)**: To overcome the limitations of static language model knowledge, LogBabylon integrates RAG, allowing it to retrieve relevant past logs from a vector database. Incoming logs are embedded into vector representations and compared against stored log entries to find similar examples. These examples are used as **context** for the LLM to produce context-aware and semantically accurate interpretations. This RAG-enhanced approach improves anomaly detection by allowing the system to detect even subtle deviations from normal behavior and facilitates a richer understanding of system state.

3. **Interpretation and Human-Readable Insights**: The final stage focuses on making the output of log analysis accessible and actionable. LogBabylon applies variable-aware prompting and in-context learning (ICL) to fine-tune LLM responses. By identifying and classifying variables (e.g., timestamps, IPs, codes) and using carefully selected k-shot examples, the system improves template generalization and reduces hallucinations. The output includes natural-language summaries, anomaly classifications, and root-cause analyses, all designed to aid system operators in decision-making.

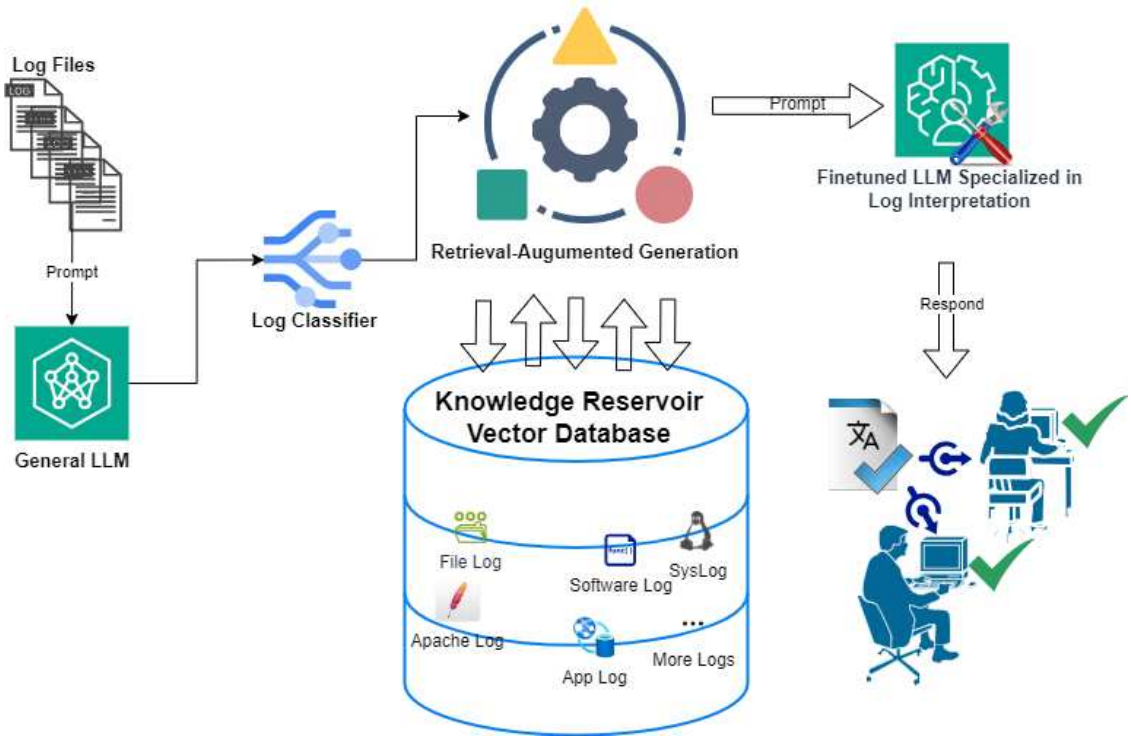

Figure 3.5: LogBabylon's architecture

LogBabylon's performance was evaluated on two benchmark datasets: loghub-2k, which includes structured logs from 16 systems, and logPub, a more diverse and large-scale collection. Compared to both rule-based tools (e.g., Drain, Uniparser) and LLM-based baselines (e.g., ChatGPT and DivLog), LogBabylon showed superior results across multiple metrics, including Grouping Accuracy (GA), Parsing Accuracy (PA), F1 score on Template Accuracy (FTA), and Granularity Distance (GD). Notably, even when operating with zero or very few labeled examples, LogBabylon outperformed other systems in robustness and adaptability [28].

From a usability perspective, LogBabylon requires minimal configuration and no extensive manual tuning, making it highly suitable for real-world deployment. The system supports a wide range of log types—including application logs, system logs, and network logs—and can scale to large volumes with high accuracy. Its design also anticipates future extensions, such as integration with confidential computing environments and interactive user feedback loops for template refinement.

3.3.5 Overview of Log Analysis with LLMs

The surveyed works demonstate that LLM-based log analysis is taking shape along three reliable lines and a few still-emerging ones. First, domain adaptation with log-centric continual pre-training plus instruction tuning now works consistently: it lets a model move across environments while keeping explanations that an analyst can read and trust, as exemplified by SuperLog. Second, when the data are shell sessions, models pre-trained on code can compress activity into compact, MITRE ATT&CK-aligned "fingerprints" that help clustering and triage; this is the core idea behind LogPrécis. Third, task-conditioned "generalist" models can cover parsing, anomaly detection, summarization, and fault localization through a single interface with competitive results on public datasets such as BGL, HDFS, and Thunderbird, as shown by LogGPT. By contrast, unified pipelines that classify, template, retrieve context, and interpret heterogeneous logs—like LoGBabylon—are promising but not yet fully validated at SOC scale, and they still need stronger safeguards against hallucinations and model drift. Engineering constraints are also underexplored: we lack systematic evidence on latency, throughput, and cost in always-on SIEM settings, and the community does not yet share robust benchmarks for log understanding, multi-step reasoning, and ATT&CK mapping. These gaps suggest practical next steps: couple LLMs with retrieval-grounded checks and simple rules to bound errors in CTI/IOC extraction; run active-learning loops so analyst feedback steadily improves prompts, templates, and instructions; and prefer privacy-preserving, on-prem deployments via distillation and quantization when data cannot leave the SOC. In practice, use SuperLog-style adaptation for cross-domain reasoning with readable rationales; prefer LogPrécis for shell telemetry when ATT&CK fingerprints and clustering matter; choose LogGPT when one model must span multiple log-analytics tasks; and adopt LoGBabylon-like pipelines when you need to stitch diverse sources together with evidence-grounded analysis.

3.4 MITRE ATT&CK: State of the Art and Way Forward

The framework continues to expand and now supports multiple domains, including Enterprise, Mobile, and Industrial Control Systems (ICS). Each domain has unique tactics and techniques, reflecting the differing threat landscapes. For example, the Enterprise matrix includes over 190 techniques and 385 sub-techniques, covering diverse platforms like Windows, macOS, Linux, and cloud environments. The ICS matrix reflects the convergence of operational and information technologies, highlighting the growing vulnerabilities in cyber-physical systems.
Notable developments in the framework include the addition of PRE-ATT&CK tactics (later absorbed into the main matrix) to capture adversary behavior before initial access, and the expansion into domains like mobile platforms (2017) and cloud services (2019).

To contextualize ATT&CK's utility, the authors [29] compare it with other prominent cybersecurity frameworks: the Lockheed Martin Cyber Kill Chain (CKC), Microsoft STRIDE, the Unified Kill Chain (UKC), and the Diamond Model (DM). They emphasize that while ATT&CK serves as a detailed behavioral database, CKC is a linear attack model emphasizing temporal sequence, and STRIDE is a threat identification methodology grounded in six threat categories (Spoofing, Tampering, etc.). ATT&CK is uniquely suited for detailed modeling of Tactics, Techniques, and Procedures (TTPs), especially when used in combination with other models for risk assessment and response planning.

The surveyed works are organized into four major use-case categories:

1. **Behavioral Analytics:** The works analyze attacker behavior by mapping observed data to ATT&CK techniques (Figure 3.6). For example, clustering algorithms have been used to detect relationships among techniques, while machine learning and AI-driven systems such as HOLMES or MAMBA correlate log data with ATT&CK entries to identify and respond to APTs in real-time. Other studies propose frameworks for ICS threat hunting, mobile APT detection, and explainable AI models that help visualize and interpret adversarial behavior.

2. **Red Teaming and Adversary Emulation:** These studies use the ATT&CK framework to simulate attacks and evaluate security systems. Several domain-specific languages (DSLs) such as enterpriseLang and powerLang have been proposed to model attack scenarios. Automated tools like CALDERA and **Atomic Red Team** help emulate real-world attack sequences for training and testing. Studies also extend ATT&CK with new techniques relevant to emerging domains like 5G networks.

3. **Defensive Gap Analysis:** Here, ATT&CK is used to identify blind spots in existing defenses. Studies map vulnerabilities, assess cyber hygiene, and analyze the efficacy of existing countermeasures by cross-referencing defense capabilities with ATT&CK's documented techniques. For example, some works link ATT&CK to CVEs to estimate risk profiles across systems.

4. **CTI Enrichment:** The framework is often integrated into CTI pipelines to improve sharing and analysis of threat intelligence. Tools have been developed to automatically extract ATT&CK-relevant indicators from threat reports using NLP and knowledge graphs. Furthermore, some works 3.5 align ATT&CK with frameworks like NIST to improve security posture assessments.

The ATT&CK framework has proven to be a flexible and powerful resource across various cybersecurity domains. It supports standardized terminology, enriches threat modeling, and improves communication between stakeholders.

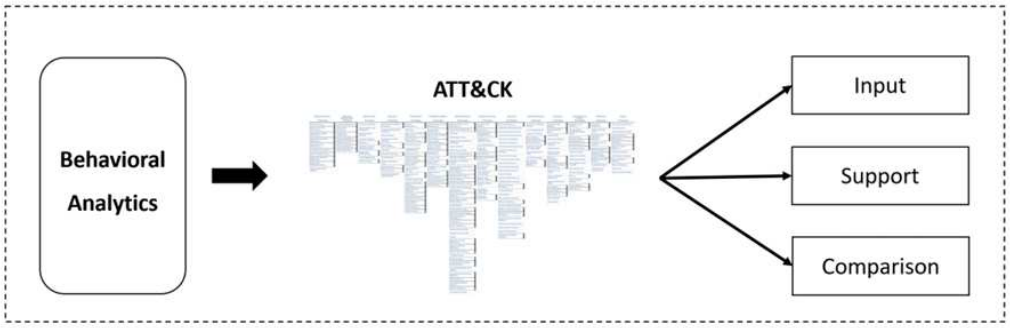

Figure 3.6: Behavioral analytic with the ATT&CK matrix: We identified three main relations with the ATT&CK matrix, i.e., Input, Support,and Comparison.

However, the survey also identifies several open issues:

- **Scalability and Complexity:** As the framework expands, maintaining its usability and relevance becomes challenging. The sheer number of techniques can overwhelm analysts, especially in large organizations with complex environments.

- **Dynamic Threat Landscape:** New attack vectors and techniques emerge rapidly, necessitating continuous updates to the framework. Ensuring timely incorporation of new threats while maintaining backward compatibility is a key challenge.
- **Integration with Other Frameworks:** While ATT&CK is widely adopted, integrating it with other models (e.g., MITRE D3FEND) for comprehensive risk assessment remains an open area of research.
- **Automated Mapping and Analysis:** Automating the mapping of observed behaviors to ATT&CK techniques is still an active area of research, particularly in heterogeneous environments where logs may not be standardized.
- **Benchmarking and Evaluation:** There's a need for standardized benchmarks to evaluate the effectiveness of ATT&CK-based systems.
- **Emerging Domains:** There's limited ATT&CK coverage for domains like IoT, automotive, and edge computing, which are increasingly vulnerable.

3.5 SIEM solutions in modern cybersecurity

In the modern cybersecurity landscape, Security Operation Centers (SOCs) serve as the central nevous system for organizational defense, continuously monitoring systems, analyzing events, and responding to potential threats. A core component of their operational workflow involves the use of SIEM systems, which aggregate and analyze security data to identify suspicious activities and potential threats (chapter 3.5). SIEM systems have become increasingly advanced, incorporating real-time correlation engines and custom rule systems, but they still face significant limitations, especially when dealing with unstructured textual data such Cyber Threat Intelligence (CTI) reports, incident logs, and alerts. CTI documents, often published in prose by security firms or disseminated across analyst forums, contain critical indicators of compromise (IOCs) sush as file paths, process names, registry keys, and command line instructions. However, because this information is embedded in natural language, its extraction and operationalization (such as converting it into SIEM correlation rules) remains a highly manual and repetitive task for analysts. Recognizing the inefficiencies of this manual process, this work proposes a fully automated, AI-powered solution that harnesses the capabilities of LLMs to bridge the gap between unstructured CTI reports and structured, actionable threat detection rules. Unlike prior domain-specific approaches, which often rely on traditional NLP techniques or narrowly trained Named Entity Recognition (NER) models, the proposed agent by the researchers leverages the generalization power of frontier models like GPT-4. It is designed to operate without human oversight, significantly reducing the cognitive and time burdens placed on security personnel and allowing them to focus on higher-level analytical and strategic tasks [3].

The system architecture is based on a multi-stage pipeline that addresses several core technical challenges. First, the CTI report is divided into smaller, manageable segments, typically paragraphs, and each is processed independently to extract potential IOCs. Given the risk of factual inaccuracy in LLM outputs, the system implements a purification mechanism that combines majority voting across multiple LLM invocations with retrieval-augmented filtering (RAG). This filtering stage cross-references the LLM-generated outputs against a curated vector database built from Windows and Linux documentation to confirm the validity of file paths, command formats, registry structures, and system binaries.

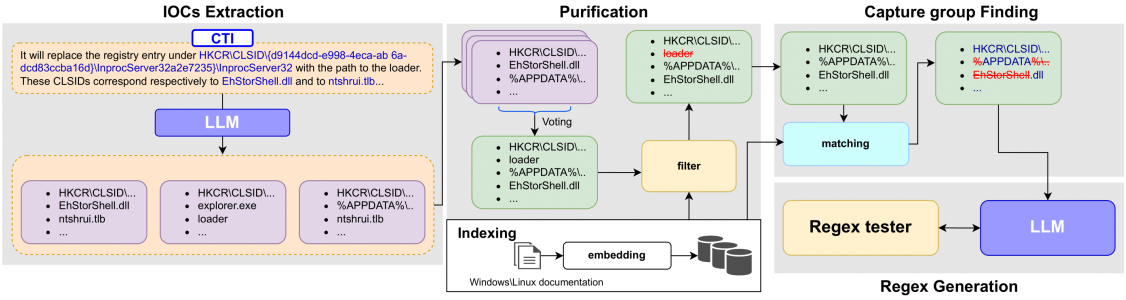

Figure 3.7: Workflow of the proposed AI agent: the first half

Once purified, the agent processes the IOCs further by distinguishing between "capture" and "non-capture" groups within each string-an essential step for building flexible and robust regular expressions (RegEx). Capture groups include predictable or invariant components like system paths or command verbs, while non-capture groups encompass attacker-defined elements such as filenames or GUIDs that may change across incidents. This distinction enables the generation of generalizable RegEx patterns that can match variations of malicious activity. A built-in RegEx tester then validates each generated pattern to ensure syntactic correctness and semantic relevance.
Beyond individual indicators, a key innovation of this project is the automated extraction of relationships between IOCs. Using both linguistic parsing and semantic role labeling, the agent identifies subject-object-verb dependencies to uncover causal or sequential links—e.g., whether a given executable "drops" a second payload, or if a script "modifies" a registry key. These relationships are then normalized through a verb-mapping schema that categorizes similar actions (e.g., "create", "drop", "establish") under common behavior labels to facilitate consistent analysis. A final verification step ensures that relationships make logical sense within cybersecurity contexts (e.g., a registry key should not "create" an executable), and invalid links are either corrected or discarded.

The culmination of this process is the generation of a Relationship Graph 3.8: a directed, structured representation of how various threat components (expressed as RegEx-formatted IOCs) interact within the reported incident. These graphs provide SOC analysts with an intuitive, at-a-glance understanding of attack chains, malware behavior, and propagation logic, which are invaluable for threat hunting and proactive defense.
To validate the effectiveness of the agent, the system was tested on more than 50 real-world CTI reports collected from diverse sources. The agent successfully extracted over 2,900 candidate IOCs, of which approximately 2,300 were validated as correct through purification. It produced around 2,200 RegEx patterns, a slightly lower number due to shared structures across different reports (e.g., the recurring use of the "AppData" directory or registry keys such as HKLM\Software\Microsoft\Windows\CurrentVersion\Run). Notably, the system achieved a near-complete recall rate, failing to extract only about 3% of the ground-truth IOCs, and many of those omissions were due to edge cases or ambiguous phrasing in the source text [3].

This research represents one of the first comprehensive attempts to fully automate CTI analysis using LLMs, from raw text ingestion to the creation of actionable intelligence in the form of correlation rules and threat graphs. Unlike earlier efforts that required continuous human oversight or were limited to specific IOC types, this AI agent demonstrates broad coverage, domain adaptability, and operational utility. By integrating it into SOC workflows, organizations can dramatically accelerate the speed of response, reduce analyst fatigue, and improve their readiness against evolving cyber threats. Moreover,

the generalizability of the architecture suggests that similar agents could be trained or fine-tuned for other threat intelligence domains, including fraud detection, vulnerability reporting, and software supply chain analysis.

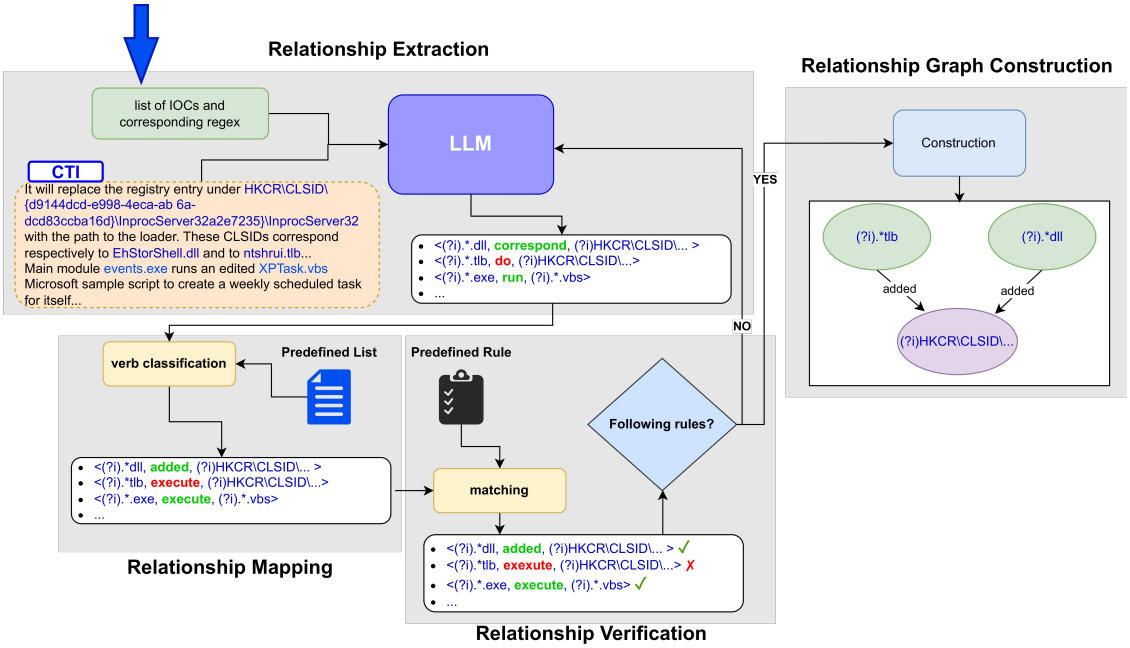

Figure 3.8: Workflow of Relationship Graph Construction

In summary, the work illustrates the transformative potential of LLMs in cybersecurity operations, not merely as assistive tools but as autonomous agents capable of executing complex analytical tasks with precision and consistency. It opens the door to a new class of intelligent, self-improving security infrastructure that scales with both the volume and complexity of modern threat landscapes.

3.6 SIEM rules with MITRE ATT&CK

In the face of escalating cyber threats, security operations increasingly rely on SIEM (Security Information and Event Management) systems to monitor and analyze log data via structured rules. These SIEM rules are critical in identifying malicious activity (chapter 2.1). However, to be truly effective, these rules must be accurately mapped to the tactics, techniques, and procedures (TTPs) defined in the MITRE ATT&CK framework-a globally recognized taxonomy of adversarial behavior. Unfortunately, this mapping process is often manual, error-prone, and inefficient (chapter 2.3).

To address these limitations, the researchers [30] introduce Rule-ATT&CK Mapper (RAM), a novel, automated framework that leverages large language models (LLMs) to map SIEM rules to their corresponding MITRE ATT&CK techniques. Unlike existing solutions that primarily focus on unstructured data such as cyber threat intelligence (CTI) reports, RAM implements a multi-stage, prompt-chained pipeline that transforms structured SIEM rules into unstructured natural language descriptions, enhances them with contextual information, and then uses LLMs to recommend corresponding MITRE ATT&CK techniques and sub-techniques. This approach is particularly significant because prior methods—mostly rule-based systems or supervised learning models—have either required constant retraining or have been limited to unstructured data (e.g., threat

intelligence reports), making them unsuitable for the structured nature of SIEM rules.

The RAM pipeline consists of six key steps (Figure 3.9):

1. **Indicator of Compromise (IoC) Extraction:** Automatically identifies entities such as IP addresses, process names, registry paths, etc., in the SIEM rule.
2. **Contextual Information Retrieval:** Uses LLM agents (e.g., REACT (REasoning & ACTing) framework [31]) to enrich IoCs with additional data through web searches or external knowledge sources.
3. **Natural Language Translation:** Converts the SIEM rule and its contextual information into a readable, unstructured description, making it more interpretable and compatible with the LLMs.
4. **Technique Recommendation:** The LLM proposes probable MITRE ATT&CK techniques based on the natural language representation of the rule.
5. **Technique Refinement and Explanation:** Filters irrelevant techniques and generates chain-of-thought rationales, offering explainability and justifications for the mappings.

The framework was tested using the Splunk Security Content dataset, comprising SIEM rules written in SPL (Search Processing Language), with careful attention paid to avoid data leakage from LLM knowledge cutoffs. Several LLMs were evaluated within RAM, including GPT-4-Turbo, GPT-4o, Qwen, IBM Granite, and Mistral, demonstrating that GPT-4-Turbo achieved the highest accuracy and recall in mapping SIEM rules to MITRE techniques.
An ablation study highlighted that enriching the rules with contextual information substantially improved mapping performance. Specifically, translating rules into plain English with context boosted the average recall from 0.46 to 0.75, and average precision from 0.39 to 0.52. This underscores the importance of combining both implicit LLM knowledge and explicit external information for domain-specific tasks.
The study also compared RAM with several baselines, including zero-shot GPT-4 prompting, BERT and CodeBERT classifiers, and TTPxHunter (a SecureBERT-based system). RAM outperformed all baselines in both recall and precision, demonstrating its effectiveness and adaptability.
Importantly, RAM offers clear advantages:

- **It eliminates the need for labeled training datasets**, thus overcoming a major limitation in cybersecurity applications where data is sensitive and scarce.
- **It produces interpretable outputs**, by providing reasoning and explanations for each mapping decision, which is crucial for analysts to understand the rationale behind the mappings (Figure 3.10).
- **It is format-agnostic** supporting SIEM rules across various platforms and rule definition languages (e.g., SPL, Lucene, KQL).

There are some limitations to consider. For instance, SIEM rules often lack sufficient embedded context, which can hinder even LLM-based systems. Additionally, issues such as ambiguous technique descriptions, mislabeled ground truth datasets, and challenges in differentiating techniques from sub-techniques can impact accuracy. Overall, RAM

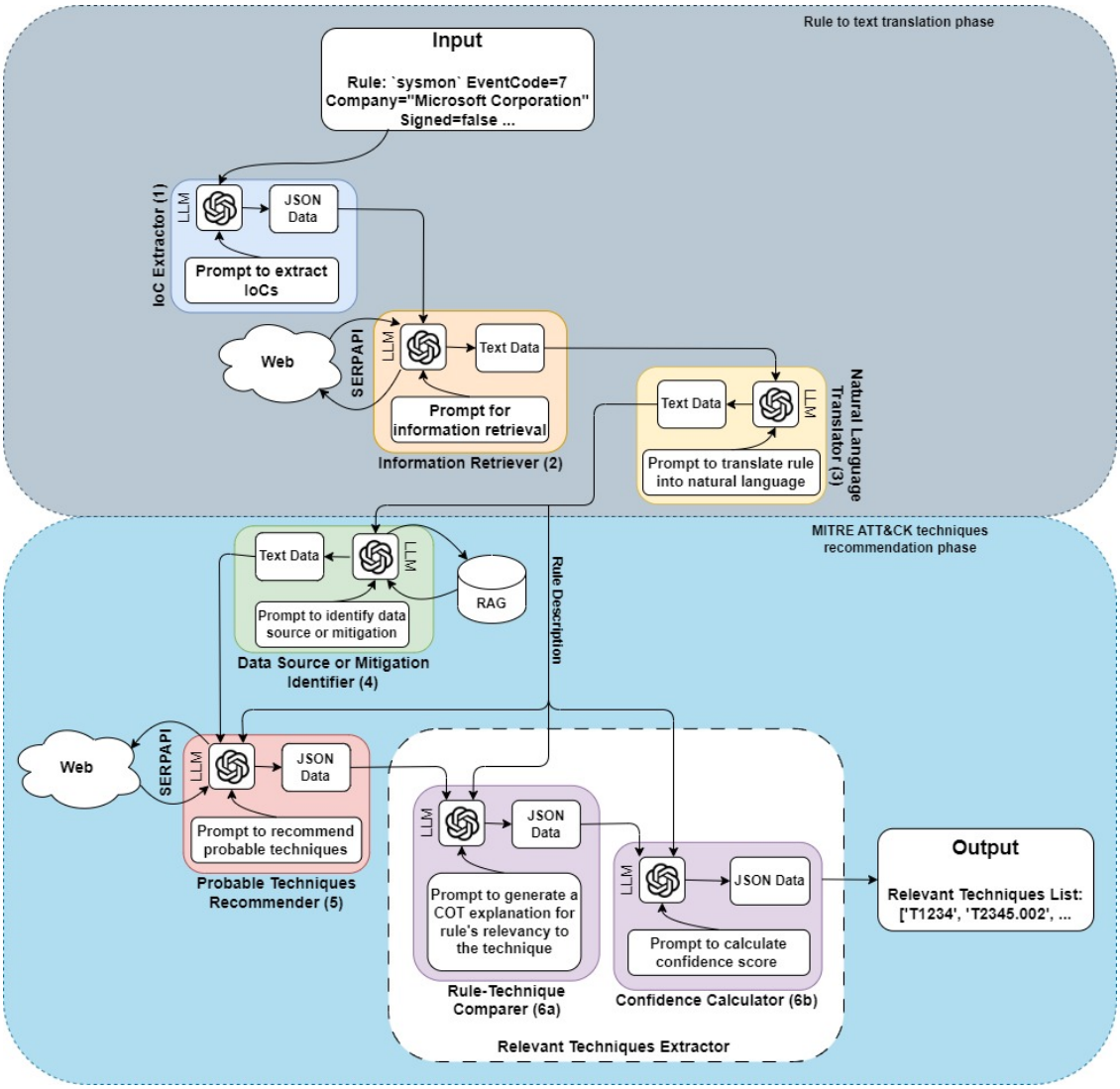

Figure 3.9: AI Agent-based RAM pipeline

represents a significant advancement in automated cyber threat detection. By bridging the gap between structured SIEM data and the semantically rich MITRE ATT&CK framework through LLMs, RAM enhances both the efficiency and accuracy of cyber defense strategies.

3.7 Summary-strengths and weaknesses

These articles show that applying large language models (LLMs) to cybersecurity and digital forensics can accelerate critical workflows—including extraction of indicators of compromise (IoCs), normalization and interpretation of heterogeneous logs, mapping to MITRE ATT&CK, and drafting structured sections of incident reports—thanks to natural-language interfaces and domain-aware pipelines (e.g., model adapters for logs, retrieval-augmented generation, and agentic orchestration with guardrails). Prompt-engineering patterns, RAG, and supervised/instruction fine-tuning, together with operational controls (input validation, context management, temperature/top-p settings, and output schemas), improve coverage, coherence, and traceability, reducing analyst

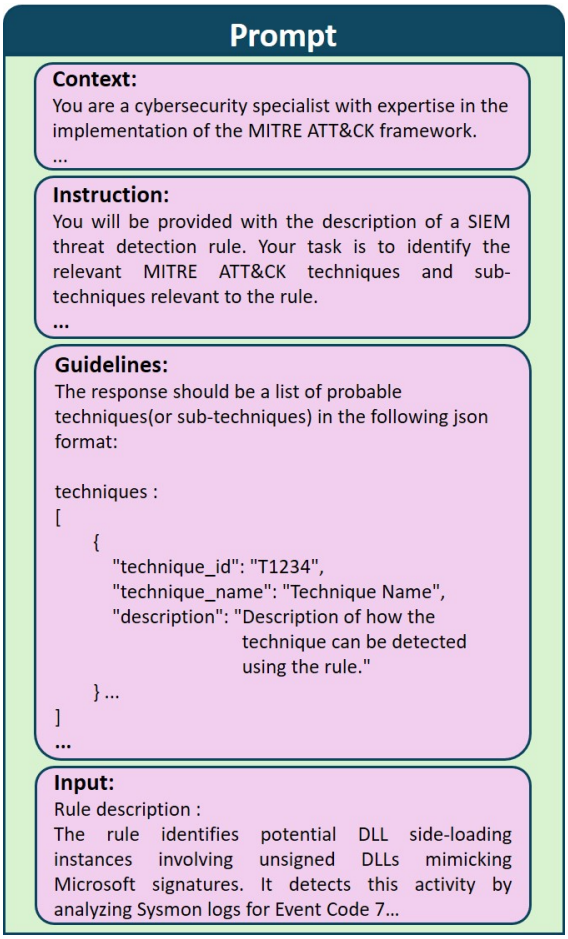

Figure 3.10: Overview of prompt structure used in all steps of the pipeline.

workload and easing integration with SIEM and threat-intel sources. However, material weaknesses remain: hallucinations and bias are mitigated but not eliminated and depend on the quality and freshness of curated knowledge; nondeterminism and data opacity hinder reproducibility and legal auditability; security risks (prompt injection, data leakage) and privacy/compliance constraints persist; costs and latency can be significant at scale; robust domain adaptation still requires task-specific data and evaluation; and both ATT&CK mapping and multi-agent orchestration can introduce error propagation, drift, and ongoing maintenance. In short, LLMs provide tangible gains in efficiency and coverage, but they should be embedded in grounded, well-governed workflows with *human-in-the-loop*, strong monitoring, and verifiable outputs.

Chapter 4

Design

4.1 Introduction and Motivation

This chapter presents a conceptual framework for integrating Large Language Models (LLMs) into a Security Information and Event Management (SIEM) pipeline. The framework is deliberately modular and extensible, so that individual capabilities (ingestion, normalization, retrieval, reasoning, validation, and reporting) can evolve over time without forcing a redesign of the whole system. The primary objective is to support the analysis of heterogeneous operating-system and application logs and broader security telemetry, while producing explainable and actionable results that fit the operational workflows of a Security Operations Center (SOC).

Modern SIEM deployments face three simultaneous pressures. First, the volume and heterogeneity of telemetry continue to grow, stressing rule-based correlation and manual triage. Second, attacker techniques change quickly, and knowledge encoded in static rules and dashboards becomes stale. Third, operational environments (especially in regulated sectors) require traceability: every detection and every automated action must be justified, reproducible, and auditable.
LLMs offer new capabilities for summarization, pattern abstraction, and semantic mapping (e.g., linking traces of behavior to MITRE ATT&CK techniques). However, naïvely inserting an LLM into a SIEM can create risks: hallucinations, inconsistent output formats, leakage of sensitive information, and difficulty validating or reproducing results. The design in this chapter addresses those risks by structuring how LLMs interact with data and by surrounding them with guardrails.

4.1.1 Goals.

Modular, event-oriented processing The framework organizes the SIEM pipeline into discrete processing stages (ingestion, pre-processing, prompt orchestration, LLM analysis, validation, and outputs). Each stage consumes well-defined inputs and produces well-defined outputs. This organization allows deployments to be event-driven (i.e., each new item of telemetry triggers a cascade of processing steps) without hard-wiring the implementation to a particular technology stack. The consequence is lower latency for detections and a clearer separation of responsibilities between stages.

Reduced hallucinations and stale knowledge. The framework reduces generation errors and outdated reasoning through three complementary techniques:

- **Prompt engineering and context management.** The framework uses structured prompts, tasks are defined by reusable instruction templates with explicit output schemas (e.g., JSON fields for findings, candidate TTPs, confidence, and evidence references). Where appropriate, few-shot exemplars and controlled decoding (low temperature) guide the model toward stable behavior.
- **Fine-tuning and model selection.** models can be adapted to the security domain using curated data (e.g., forensic traces with known TTP labels), improving alignment with SOC tasks.
- **Retrieval-augmented generation (RAG).** optional retrieval of recent and authoritative context (ATT&CK pages, CTI reports, internal policies) grounds the model's answers in verifiable sources. Retrieval is governed by allow-listed repositories and freshness checks to limit drift.

Be auditable and safe by design. The system ensures every result is traceable to its inputs, including the exact prompt, model version, and parameters used. Auditability must meet these properties: (i) reproducibility, given the same inputs and configuration, the system produces the same output; (ii) provenance, all artifacts carry tamper-evident metadata about their origin, transformations, and integrity; (iii) explainability, outputs are structured (not free text only) and cite the evidence supporting each claim; and (iv) governance, a validation layer enforces schema adherence, consistency constraints, and policy gates before any automated action can proceed.

Support multiple LLM roles. The design anticipates scenarios where more than one LLM-based role is beneficial, for example, one role to summarize raw logs into a concise analyst report, and another to map behaviors to MITRE ATT&CK techniques. The framework permits these roles to be composed sequentially (prompt chaining) or run in parallel, while exposing a single, consistent interface to downstream components.

Scalability and replaceability. Each stage communicates through explicit contracts (structured artifacts), enabling independent scaling and replacement. For instance, a preprocessing component optimized for Windows Event Logs can be substituted later with a broader normalizer without changing the design of the LLM stages or the validation logic.

Privacy-aware operation. Because security telemetry may include sensitive or forensic data, the framework is compatible with on-premise or private-cloud inference and with strict data-minimization practices. Optional retrieval is governed by source allow-lists, and prompts are constructed from sanitized, normalized representations of events rather than raw logs.

4.1.2 Scope and non-goals.

The present chapter focuses on conceptual design: roles, interfaces, and the behavior of the pipeline as a whole. It does not prescribe a specific vendor technology, queueing system, or model; those are implementation choices that can vary across deployments. Likewise, this chapter does not describe datasets, training recipes, or evaluation metrics, those belong to the subsequent Implementation chapter where only where only a small proof-of-concept instance of the framework is realized.

4.2 Architectural Overview

This section describes the end-to-end behavior of the framework, from the moment security telemetry is produced by an information system to the moment a detection record is made available to analysts and response tooling. The description is technology-agnostic: it defines roles, data exchanged between roles, and the sequence of transformations, not the specific software products or deployment patterns. Figure 4.1 (Framework architecture) provides a visual summary that we will reference while introducing the six phases.

From telemetry to normalized evidence. The boundary of the system starts where security-relevant data is generated: operating systems, applications, identity providers, network devices, middleware, and security tools such as EDRs. The framework assumes that these sources can be accessed through standard collection mechanisms (agents, connectors, or export interfaces). The Input Retriever is the logical role responsible for bringing raw artifacts into the pipeline and attaching basic provenance: where the artifact came from, how it was obtained, when it was observed, and—when possible—an integrity hash to make later tampering detectable.
Raw artifacts are seldom uniform. A single host may emit Windows Event XML, PowerShell transcripts, and application logs with free-form lines; a network sensor may provide NetFlow, PCAP slices, and IDS alerts. The Pre-process role brings structure to this heterogeneity. First, it parses and normalizes events to a canonical representation with stable field names (timestamp, host identity, user identity, action, resource, network endpoints, process metadata, etc.). Second, it applies sanitization steps so that the normalized representation can be safely embedded in prompts without allowing the content of logs to steer the model (e.g., escaping model-like tokens and quoting untrusted strings). Third, it may derive simple features that are useful later (for example, a normalized command line, a process tree summary, or a rarity score for an event type). The output of this phase is a structured artifact we denote [Y] Normalized Event. [Y] is intentionally compact and deterministic: two identical raw events should lead to the same [Y], which helps with reproducibility and caching.

Prompt orchestration The framework separates what we ask the model to do from what it must reason over and from which external knowledge is allowed to guide that reasoning. This separation appears in three artifacts:

- **[X] Pre-set Instruction** encodes the task. It can be as simple as "summarize the security-relevant behavior in [Y] and output valid JSON with fields A, B, C," or as precise as "given [Y], return candidate MITRE ATT&CK techniques with confidence scores and evidence references." The key point is that [X] is reusable across many events.

- **[Y] Normalized Event** is the evidence produced earlier in 4.2.

- **[Z] Context (Optional)** is external knowledge fetched on demand when the task benefits from grounding—pages from ATT&CK, snippets from an internal hardening policy, a short CTI extract about an IOC that appears in [Y]. Retrieval is governed by source allow-lists and freshness rules, and it is bounded in size so that the model remains focused on the event at hand. A small context cache can avoid repetitive retrieval when many events require the same background.

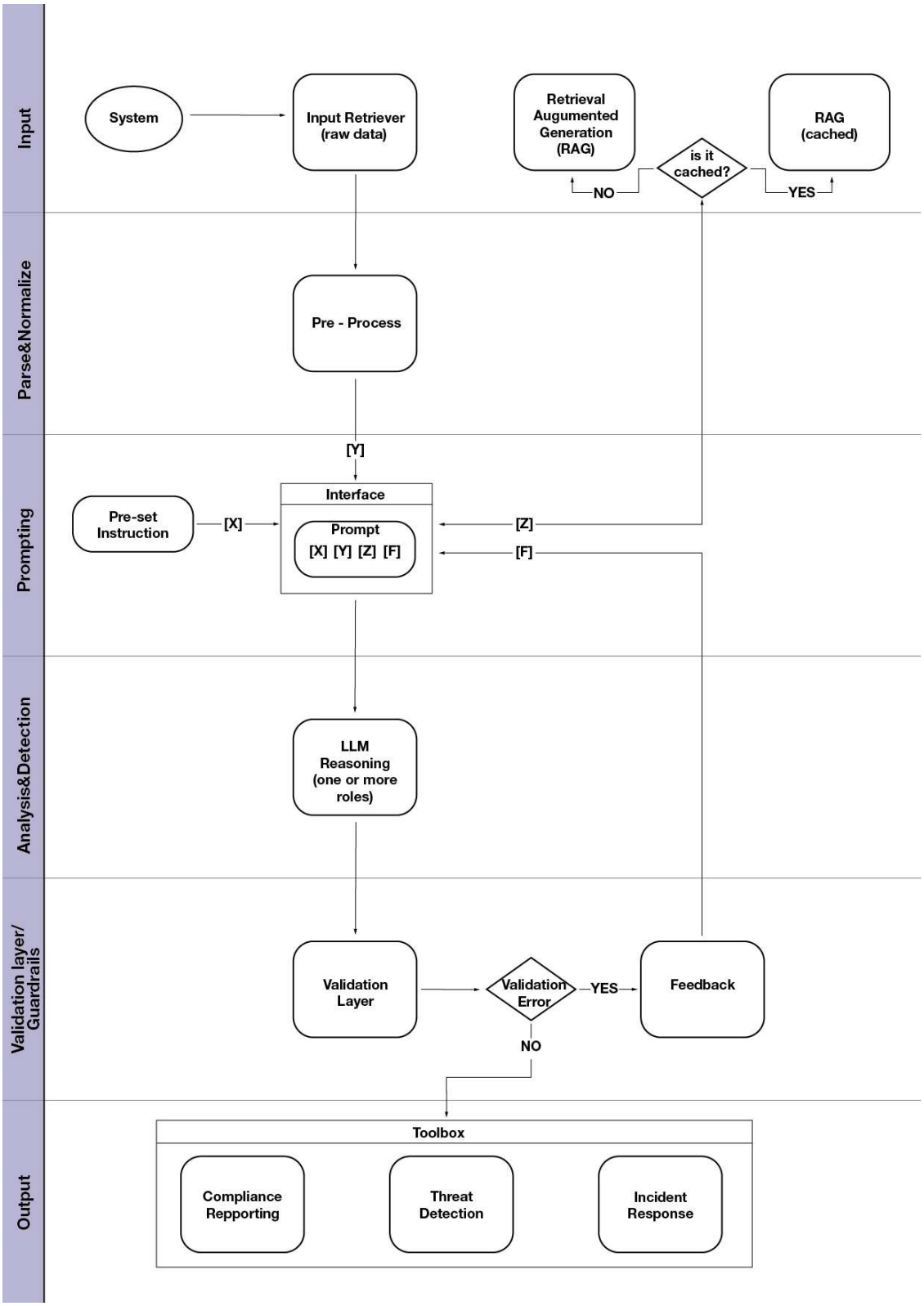

Figure 4.1: Architecture of the proposed system.

An Interface composes [X], [Y], and (when present) [Z] into a concrete prompt according to ordering and size rules. The Interface also communicates to the model an output contract (for instance, a JSON schema), so that downstream components can rely on a stable structure rather than free text. The interface can also apply [F] Feedback

Directives, produced later by analysts or the validator, that patch instructions (tighten schemas, add exemplars) or append short vetted context. Evidence [Y]remains immutable and is never edited by feedback.

Reasoning over behavior The LLM Layer is the reasoning core. It is presented here in general terms, independent of a specific model or decoding strategy. The framework allows one or more roles to be configured, depending on the analytic question an organization wants to answer.

At its simplest, a single role consumes [X] and [Y] (and optionally [Z]) and returns a structured answer with findings, evidence references, and—where applicable—candidate mappings to a knowledge structure such as MITRE ATT&CK. In more elaborate setups, a first role may transform [Y] into a concise analyst-oriented report (reducing noise and highlighting anomalies), while a second role consumes that report to perform classification (e.g., selecting likely techniques and explaining the rationale). The roles can be chained without changing the rest of the pipeline because they all speak the same contract: inputs are explicit artifacts and outputs must respect the same schema family.

The framework does not require a specific prompting style. Zero-shot, few-shot, or prompt chaining are implementation choices. What the design does require is that the chosen style be captured in [X], that the model operate on sanitized [Y], and that any additional knowledge come through governed [Z], so that the reasoning remains auditable.

Guardrails and acceptance LLM outputs are not delivered directly to detection dashboards or response playbooks. Instead, they pass through a Validation Layer whose purpose is twofold: to protect the rest of the system from malformed or unsafe outputs, and to convert model suggestions into reliable, auditable detection records.

Validation proceeds in stages. First, the output is checked against the promised format: the JSON must parse; mandatory fields must be present; identifiers must meet expected patterns; numerical ranges (such as confidence values) must make sense. These checks are quick and objective, either the contract is respected or it is not.

Second, the content is examined for internal consistency. Timestamps referenced in the explanation must fall within the time window of [Y]; hostnames and user identities must match what appears in the evidence; ATT&CK technique identifiers must be valid entries; if the model claims that a process contacted a given IP, that IP must be visible in the network fields of [Y] or in the retrieved context [Z] with an explicit citation. This stage does not re-do the model's reasoning; it verifies that the reasoning is grounded in observable facts.

Third, policy and risk controls are applied. Organizations can set thresholds (for example, "do not auto-escalate to response unless the confidence is above 0.9 or unless two independent roles agree"). They can forbid certain automatic actions for high-risk assets, or require human sign-off for specific families of techniques. The Validation Layer is where such rules live, ensuring that machine-generated suggestions cannot bypass governance.

When a result passes validation, it is transformed into a canonical Detection Record and handed to the output services. When a result fails, the validator records a Validation Issue that explains why and routes the case to a Feedback space where analysts can review it, correct labels if needed, and provide examples that may later be used to refine prompts, update retrieval indices, or fine-tune models. The feedback path closes the loop between model behavior and operational reality without forcing immediate retraining.

From detection to action The accepted Detection Record becomes the single source of truth consumed by three families of services. **Compliance reporting** turns it into a human-readable narrative that cites evidence and aligns with regulatory expectations for audit trails. **Threat-detection** views aggregate and correlate records across hosts and time windows, enabling analysts to see campaigns rather than isolated events. **Incident-response** hooks expose the record to playbooks that can open tickets, notify responders, or, under the constraints enforced by validation, request containment actions from integrated tools. Because all three services consume the same canonical artifact, they remain consistent and reproducible, and investigators can always trace a dashboard widget or a response action back to the exact evidence, prompt, and model output that motivated it.

4.3 Components and responsibilities

This section characterizes each component of the framework in terms of its role, the artifacts it consumes and produces, and the quality properties it is responsible for. The components are presented as logical roles rather than fixed products or services. An implementation may map several roles into a single process or service, or split them further, but the interfaces described here should remain stable across such choices. The figure 4.2 shows the detailed architecture of the proposed system, illustrating how data flows between components and where artifacts are placed in the process.

Input Retriever The Input Retriever is the controlled entry point of the system. Its responsibility is to acquire security-relevant artifacts from heterogeneous sources and attach minimal provenance so that later stages can reason about trust and timing. Typical sources include host operating systems, applications and middleware, identity providers, network sensors, and security tools such as EDR/IDS. Acquisition may occur through agents, built-in export interfaces, or connectors already present in an enterprise SIEM.

Two properties matter here. First, provenance: every raw artifact is tagged with "who/where/when/how", such as the source system, the collector identity, the observation time, and (when feasible) a cryptographic hash of the exact bytes received. Second, non-interference: collection should be read-only or otherwise forensically sound, to avoid modifying the very systems one aims to observe. The retriever does not attempt to interpret the content beyond basic integrity and format checks. Its output is simply a RawEvent with attached provenance.

Pre-process Raw artifacts vary widely in syntax and semantics. The Pre-process component transforms them into a stable, compact representation suitable for downstream reasoning and safe to embed in prompts. This transformation has three layers.

Parsing to a canonical shape. Events are projected onto a common set of fields: timestamps, host and user identities, process metadata, file or network descriptors, and a normalized notion of "action". Standards such as ECS or OCSF can inspire the field naming and typing, but the canonical shape must also accommodate source-specific structures. In Windows environments, for example, Event Viewer exposes events as XML with rich, event-ID specific payloads (e.g., process creation 4688, logon 4624). The parser should preserve those details—like NewProcessName, ParentProcessName, LogonType, SIDs, and event IDs—while also mapping them to portable fields (e.g., process.executable, process.parent.executable, user.id, event.code). This dual view lets the pipeline benefit from Windows-specific semantics without binding the rest of the design to Windows only.

Normalization and enrichment. After parsing, obvious ambiguities are resolved (time zone normalization, hostname canonicalization), and lightweight enrichments are added where inexpensive and stable (for instance, deriving a process tree snippet from parent/child relations, or marking whether an IP is private vs. public). These enrichments are simple and deterministic; heavy analytics are intentionally deferred to later stages.

Prompt-safety and sanitization. Pre-process treats input as untrusted text. Any string that will later appear inside an LLM prompt is escaped or quoted to prevent the content itself from acting like instructions (a risk when logs contain fragments that resemble prompts). The result of Pre-process is the [Y] Normalized Event, an immutable, schema-validated record that captures essential facts and is safe to reference inside prompts.

Retrieval and Context Cache Not every analytic task requires external knowledge, but when grounding improves reliability the framework can attach a small amount of curated context. The Retrieval component receives a focused query derived from the task and the event (for example, a technique keyword present in [Y], or an indicator observed in the event) and searches allow-listed repositories: MITRE ATT&CK technique pages, corporate security policies, curated CTI notes, vulnerability descriptions, and similar sources. A Context Cache stores frequently used chunks under a short time-to-live so that repeated lookups for the same technique or policy do not add latency. Two constraints keep Retrieval predictable. First, governance: only approved sources are searched, with recency checks to avoid anchoring on outdated material. Second, budgeting: the amount of context is bounded so that the model remains focused on the event at hand. The output is [Z] Context, a compact set of cited snippets with metadata (source, date, identifiers) that can be safely injected into prompts.

Prompt Interface The Prompt Interface is the point where the framework turns artifacts into a concrete query for the model, while also announcing an output contract that downstream components can verify. It takes three inputs and one optional adjunct: [X], [Y], [Z], and [F]. The Interface enforces ordering (instructions before evidence, evidence before context), applies size limits, and ensures that what the model returns must be valid according to a declared schema family (for instance, an LLMInference v1 object). It does not decide what the model should infer; it decides how the question is asked and how the answer must be shaped so that the pipeline remains auditable.

LLM layer The LLM Layer performs the actual reasoning under the constraints set by the Interface. Conceptually, it exposes one or more roles that can be composed (for example):

- **A reporter** role that distills [Y] into a concise, structured mini-report for analysts, emphasizing anomalies, unusual sequences, and the minimal evidence needed to understand them.
- **Classifier** role that reads [Y] (or the reporter's output) and proposes candidate mappings to a knowledge structure such as MITRE ATT&CK, with confidence and explicit references to the evidence that motivated each candidate.

These roles are illustrative rather than prescriptive, the framework equally accommodates alternative roles, such as a triage summarizer or a policy-conformance checker, provided that the roles accept the same artifact family as input and produce outputs that satisfy the same schema contract.

Three design choices make this layer dependable:

Determinism where it matters. For steps that feed automation or validation, decoding parameters favor stability (e.g., low temperature) and outputs are restricted to structured function calls or JSON. Where ambiguity is expected—say, exploring alternative explanations—n-best generation is permitted but must still meet the schema contract.

Grounded justifications. Every claim in the output (a finding, a technique candidate) carries references to fields in [Y] or to snippets in [Z]. This is what allows the Validation component to verify that the reasoning is anchored in observable facts rather than free association.

Composability without entanglement. Because roles consume and emit standardized artifacts, they can be chained without hidden couplings. For example, a reporter can be swapped for a different summarization strategy without changing the classifier or the validator.

Validation Layer The Validation component is the system's gatekeeper. It receives the model's output and decides whether the result is acceptable as a detection artifact. It does so by applying a sequence of checks that move from formal to semantic to policy:

- **Contract conformance.** The output must parse and satisfy the declared schema: required fields present, correct types, and sensible ranges.
- **Consistency with evidence and context.** References in the justification must point to facts present in [Y] or to cited snippets in [Z]; timestamps must align with the event's time window; identifiers such as host, user, and technique codes must be valid and coherent.
- **Risk and policy controls.** Organizational rules are enforced here: minimum confidence thresholds for automated escalations, asset-sensitivity constraints, and constraints on which response actions are even eligible for automation.

If a result passes these checks, it is transformed into a canonical, tamper-evident Detection Record. If it fails, the validator records a Validation Issue with explicit reasons and routes the case to Feedback. The validator never silently fixes semantic errors; it either accepts with full traceability or rejects with a reason, keeping the pipeline honest about what the model did or did not establish.

Feedback and Continuous Improvement Feedback is the controlled place where human understanding corrects or augments machine suggestions, and where those corrections are turned into durable improvements without rewriting history. It serves three functions.

Case review and labeling. Analysts review flagged cases, annotate errors ("invalid JSON," "incorrect technique," "unsupported claim"), and, when appropriate, attach corrected labels or minimal teaching examples. This is not free-form editing of outputs; it is structured feedback captured under a small ontology of error types and label schemas. Crucially, the original [Y] remains immutable.

Prompt patches and hints ([F]). Some errors are best addressed by adjusting how questions are asked rather than by retraining a model. Feedback can therefore emit [F] Feedback Directives that instruct the Prompt Interface to modify [X] (for example, enforce stricter JSON formatting, add a domain-specific exemplar, or focus the task on a particular facet of [Y]) or to add a small piece of vetted context to [Z]. Because [F] targets the instruction and context, not the evidence, it improves the next run without altering the record of what happened.

Curating improvements for later. Over time, labeled cases and analyst-approved snippets form a high-quality dataset for fine-tuning and for refreshing retrieval indices. This path is explicitly offline: changes are staged, reviewed, and applied under version control, so that the behavior of the online system changes only through deliberate updates.

Controls in Feedback. To remain trustworthy, Feedback itself is governed. Labels carry the identity of the annotator and timestamps; sampling strategies avoid overrepresenting easy cases; proposed prompt patches can be canaried on a subset of traffic before becoming default; and every change to validation thresholds or retrieval allow-lists is tracked so that analysts can correlate system behavior with governance decisions.

Output Services. Once accepted, the Detection Record becomes the single source of truth for downstream consumers.

Compliance reporting. transforms it into a narrative that cites evidence and justifications in a human-readable layout while preserving references to the underlying artifacts. Because reports are derived from the same canonical object that drives detection, they are reproducible and auditable.

Threat-detection views and correlation. assemble records across hosts and time windows to reveal campaigns and lateral movement rather than isolated events. Since every record carries stable correlation keys (host, user, process lineage), such aggregation does not require re-parsing the original logs.

Incident-response hooks. expose records to playbooks that can open tickets, notify responders, or request containment actions from integrated tools. The hooks honor the decisions already made by Validation (e.g., actions allowed only above a threshold, or only after human approval), so automation remains inside the boundaries set by policy.

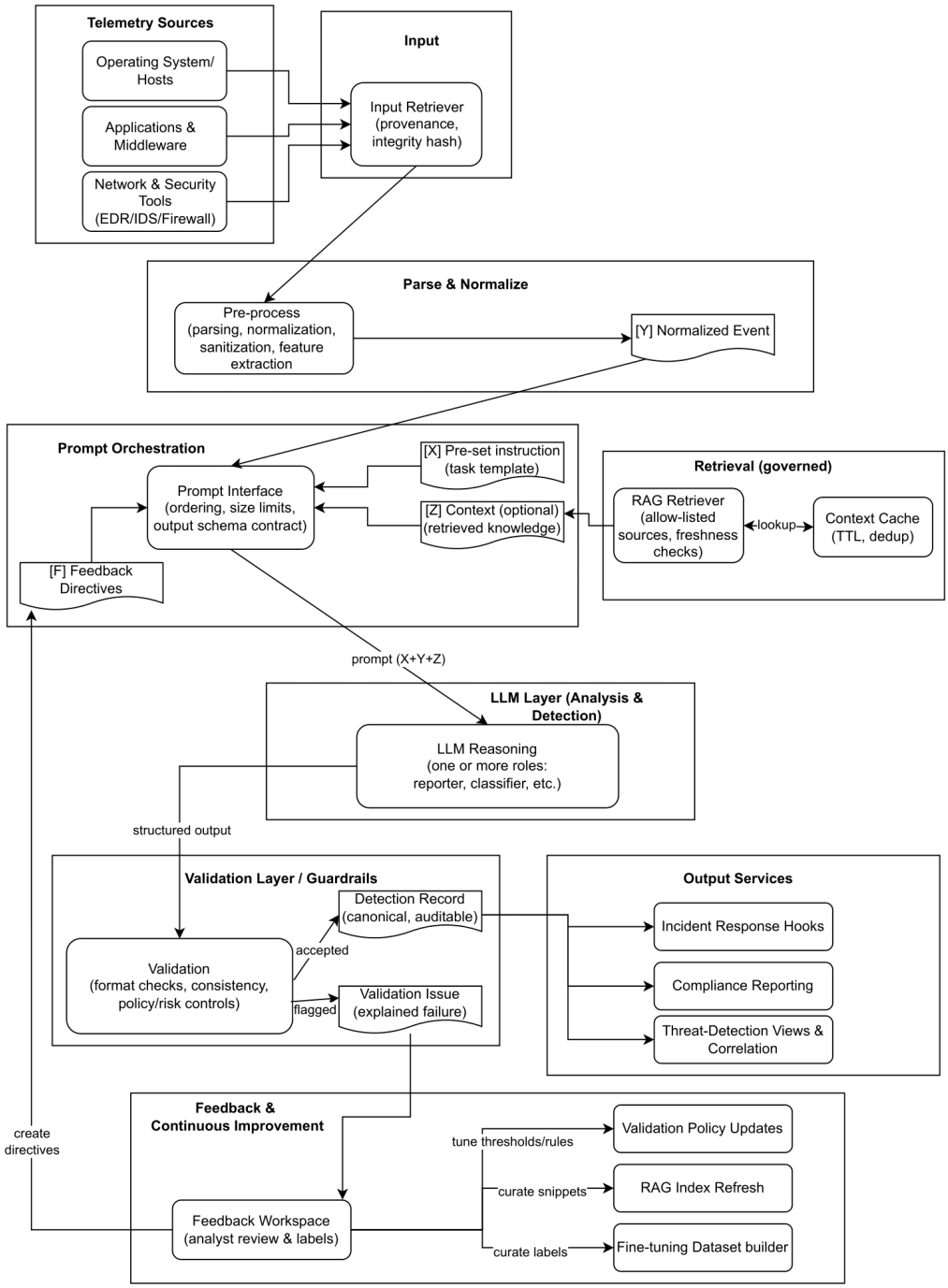

Figure 4.2: Detailed architecture of the proposed framework, highlighting data flow, artifact placement, and validation guardrails.

4.4 Data models and Artifacts

The framework relies on a small set of versioned artifacts exchanged between components. Keeping these artifacts explicit has two goals. First, it makes the pipeline auditable and reproducible: given a particular event and configuration, the same output can be re-created later. Second, it isolates concerns: instructions can evolve without changing evidence, and context can be refreshed without rewriting prompts. This section defines the artifacts and explains the Detection Record, which is the canonical output consumed by reporting, correlation, and response tooling.

4.4.1 Design principles for artifacts

Artifacts follow four principles. They are immutable once created, so downstream reasoning always refers to a stable object. They are schema-validated and versioned, so the system can evolve without ambiguity. They carry provenance and privacy annotations, so investigators know where information came from and what handling rules apply. And they include references rather than copies whenever possible, so that larger payloads (e.g., raw logs, long context documents) remain in dedicated storage while the artifacts keep the flow lightweight.

[Y] Normalized Event. [Y] is the compact, deterministic representation of a raw item of telemetry. It is produced by the pre-processing stage and designed to be safe for inclusion in prompts. The record captures timing, source host and user, action semantics, and the minimal fields required to reconstruct the evidence in analyst tools. In Windows environments, this structure preserves native Event Viewer XML semantics—such as event IDs (e.g., 1234 for process creation, 2345 for logon), SIDs, NewProcessName, ParentProcessName, and logon types—while mapping them to portable fields like event.code, user.id, process.executable, and process.parent.executable. By keeping both the Windows-specific and the normalized view, [Y] remains faithful to the source and remains cross-platform.
Two properties of [Y] are important for later stages. It is sanitized so that strings which will enter prompts cannot act as instructions (for example, tokens resembling model directives are quoted or escaped). And it is hash-addressed: the artifact ID encodes a content digest and provenance (collector, source, capture time), which supports forensic integrity checks.

[X] Pre-set Instruction. [X] specifies the task to be performed on [Y] and how the result must be shaped. It is a reusable template ("map behaviors in [Y] to candidate ATT&CK techniques and output valid JSON with fields ...") and can include few-shot exemplars when stability is more important than openness. The template also records decoding policy (for instance, low temperature and max-tokens ceilings) and an output contract by reference to a schema, so that downstream validation can check conformance mechanically. Because [X] is data, not code, changing a prompt or adding a new exemplar is a governance decision that leaves the rest of the system untouched.

[Z] Context (optional, governed). [Z] is a bounded collection of snippets that grounds the model's reasoning in verifiable sources. Typical entries include ATT&CK technique descriptions, short extracts from vulnerability advisories, or paragraphs from

internal security policies. Each snippet carries a citation (source, identifier, URL or document handle, publication date) and freshness metadata, and the artifact as a whole records the retrieval policy used (allowed sources, TTL, deduplication parameters). The pipeline favors small, topical context over large document dumps: the goal is to help the model disambiguate specific behaviors observed in [Y], not to expand the search space.

[F] Feedback Directives. [F] is the artifact that carries structured guidance from analysts and from the validation layer back to the Prompt Interface. It does not modify [Y]; instead, it patches [X] (for example, enforcing a stricter JSON schema or adding a domain-specific exemplar) and may append a short vetted snippet to [Z] when that snippet represents institutional knowledge worth injecting consistently. Feedback directives are small and composable: each directive states a reason (e.g., "format non-compliant," "wrong technique family," "insufficient evidence linkage") and a minimal fix (e.g., "require function-call output," "prioritize process lineage evidence," "insert ATT&CK T1105 exemplar"). Because [F] targets instruction and context rather than evidence, it improves future runs without rewriting history.

LLMInference. The LLMInference object is the immediate result of model reasoning. It is deliberately machine-first: a JSON structure with fields for findings, evidence references, candidate mappings (e.g., ATT&CK techniques), confidence, and a compact rationale. Each claim links back to specific fields in [Y] or to citations in [Z], which is what enables downstream validation to check that explanations are grounded.
An inference object also records the model lineage used to produce it: model family and version, decoding parameters, and the exact [X]/[Z] digests. These details make reruns comparable and support A/B evaluations of alternative prompts or models without ambiguity.

Detection Record. The Detection Record is the canonical artifact produced when an inference passes validation. It is the single source of truth for reporting, correlation, and response, and it is designed to be tamper-evident, traceable, and operationally useful. At a high level, a Detection Record binds together five strands:

1. **What happened.** A reference to the [Y] Normalized Event (or to a small set of [Y] records if the detection covers a short sequence), with enough context to re-open the raw evidence in analyst tools. The record keeps [Y] immutable and addressable by hash so investigators can verify integrity later.

2. **What the model concluded and why.** The accepted LLMInference payload, including findings, candidate techniques, confidence, and the evidence references and citations that support each claim. The record stores the exact [X] template digest and [Z] context digest that framed the question, and it stores the model lineage (model ID, version, decoding parameters) so that the same reasoning can be reproduced.

3. **Why this was accepted.** The Validation section captures the checks performed (contract conformance, consistency with [Y]/[Z], policy and risk controls) and their outcomes. Rather than a simple "pass/fail", the record includes a short narrative of the validator's decision and the thresholds or rules that were in force (e.g. accepted as medium-risk finding because confidence 0.88 >= 0.80 threshold; auto-response not authorized because asset sensitivity = high).

4. **How to correlate it.** A set of correlation keys (host identity, user identity, process lineage fingerprints, time window identifiers) computed deterministically from [Y]. These keys allow dashboards and hunt queries to aggregate Detection Records into campaigns and to follow lateral movement, without reparsing raw logs or re-running models.

5. **What can act on it.** The Response policy section, which states what actions, if any, are permitted automatically, which require human approval, and which are explicitly barred for this record. These decisions are inherited from organization policy and from the validator's risk assessment; the record stores them so that response tooling does not need to re-implement validation logic.

Two additional facets make the Detection Record fit for regulated environments. First, it carries privacy and retention annotations (for example, whether personal data is present, and the latest date on which redaction or deletion may be required). Second, it is append-only: follow-up information (such as an operator's comment or the ticket ID of an incident) is added as linked entries with their own provenance rather than by editing the original content. This approach preserves a clear audit trail.

A concise example, stripped to essentials for readability:

```
{
  "record_id": "det-41af...",
  "created_at": "2025-03-17T12:04:10Z",
  "status": "accepted",
  "evidence": {
    "y_refs": ["y:host-42:evt-7a12..."],
    "window": {"start": "2025-03-17T12:02:00Z", "end": "2025-03-17T12:05:00Z"}
  },
  "inference": {
    "id": "inf-902b...",
    "model": {"name": "ModelX", "version": "v1.3", "temperature": 0.1},
    "x_digest": "sha256:...",
    "z_digest": "sha256:...",
    "findings": [{"id": "f1", "summary": "Download-and-execute via bash",
        "evidence_refs": ["Y.process.command_line", "Y.network.dst_ip"]}],
    "attck": [{"technique": "T1059.004", "confidence": 0.86, "justification":
        "Command interpreter used with remote script"}],
    "confidence": 0.84
  },
  "validation": {
    "contract": "pass",
    "consistency": ["timestamps_aligned", "host_consistent"],
    "policy": {"risk": "medium", "auto_response": false, "reason":
        "high-sensitivity asset"},
    "validator_version": "v2.1"
  },
  "correlation": {"host_id": "host-42", "user_id": "u-1001",
      "proc_fingerprint": "p:4312@host-42"},
  "privacy": {"contains_pii": false, "retention_class": "security-incident"},
  "audit": {"by": "validation-service", "hash": "sha256:..."}
}
```

In practice, a production record will also include a links section (to tickets, dashboards, or playbook runs) and may reference a bundle of [Y] artifacts when the detection emerges from a short sequence rather than a single log line.

Lifecycle and mutability. Only three things can happen to a Detection Record after creation. It can be linked to other records (for correlation), it can accrue annotations (for example, incident IDs or human comments), and it can be referenced by reports and response artifacts. None of these operations changes the original body of the record. If a later re-analysis yields a different conclusion, the system creates a superseding record that points back to the earlier one and explains the change (for instance, "technique updated from T1059.004 to T1105 based on analyst confirmation"), keeping history intact.

Storage, indexing, and queries. Artifacts live in tamper-evident storage. For efficiency the system maintains two layers of indexing. A document index supports retrieval by identifiers and time windows, and a graph or columnar index supports correlations over the keys stored in records (hosts, users, process fingerprints). Analysts search and pivot in the record space, not in raw logs; when they need full fidelity, the record provides the pointers and hashes necessary to retrieve the original evidence.

Why this separation matters. The separation of [X], [Y], [Z], and [F] is not stylistic, it is what minimizes accidental leakage and maximizes reproducibility. Evidence remains evidence and is never rewritten by feedback. Instructions can be tightened or enriched without touching the event. Context enters through an audited gate and can be refreshed on its own schedule. And feedback travels as directives rather than edits, so the pipeline improves while the historical record remains clean.

4.5 Orchestration and operational considerations

This section explains how the pieces of the framework "move" together in practice. The emphasis is not on a particular product or middleware, but on the operational behavior of the pipeline: how events trigger work, how stages hand off artifacts, how limits are enforced, and how the system remains observable, safe, and evolvable.

4.5.1 Event-driven orchestration in this context

By event-driven we mean that new telemetry causes the next step to run, rather than the system waiting for a periodic batch. When a log record (or a short burst of related records) arrives, the pipeline reacts: it is parsed and normalized into [Y], a prompt is assembled from [X] + [Y] + (optional) [Z], the model reasons, the validator judges, and, if all goes well, a Detection Record is issued. Nothing in this description requires a specific transport; what matters is causality (each artifact triggers the next transformation) and decoupling (each stage depends only on the artifact contract, not on who produced it). Why this matters for a SIEM scenario:

- **Latency.** It keeps latency low for detections that depend on fresh context (e.g., a suspicious process tree that should be flagged before lateral movement succeeds).
- **Backpressure.** It enables backpressure: if a downstream stage slows down (for example, a burst of inference work), upstream stages can continue to normalize and queue artifacts without dropping data or blocking the whole system.
- **Explainability.** It improves explainability: every output corresponds to a small, traceable chain of artifacts rather than to a large, opaque batch job.

In an implementation, the hand-off between stages can be synchronous or asynchronous. The design remains the same either way because the "contract" is the artifact itself.

To avoid overloading terms like "service" or "microservice," the chapter uses **operator** to denote a logical processing role: a piece of the pipeline that accepts one artifact type and emits another, while satisfying a small set of invariants. Pre-process is an operator that turns raw logs into [Y]; the Prompt Interface is an operator that combines [X], [Y], and [Z] into a model call; the LLM layer is an operator that turns that call into an LLMInference; the Validator is an operator that turns an LLMInference into either a Detection Record or a Validation Issue; and so on.
Thinking in operators has two advantages. First, it clarifies responsibility boundaries: if the output is malformed, we know exactly which operator violated its contract. Second, it enables idempotence and replay: re-running an operator on the same inputs should yield the same output, which is crucial for forensics and audits. Whether a given deployment maps each operator to a separate process, or co-locates several in a single runtime, is an implementation choice; the operator abstraction stays the same.

Every real system runs under constraints. We use the word **budget** to make those constraints explicit, so that the pipeline can make predictable trade-offs when load spikes or when an event is unusually complex.
We say **Latency Budget** the maximum time we are willing to spend from ingestion to an accepted Detection Record under normal conditions. It is split across stages (normalization, optional retrieval, inference, validation). If retrieval is slow or the model is busy, the system can choose to degrade gracefully, for example, run without [Z] for low-risk assets, or route the event to a smaller model, so that the total time stays inside the budget. Time budgets differ by use case: real-time containment needs faster paths than nightly compliance reporting.
Token/size budget refers that Prompts cannot grow without bound. The Interface enforces a size budget for [Y] (e.g., the salient parts of a Windows 1234/4321 pair rather than entire XMLs) and a context budget for [Z] (a handful of cited paragraphs, not a full ATT&CK site dump). When inputs exceed budget, the Interface applies deterministic compression (summaries that preserve key fields, process-tree skeletons, or top-k salient lines) and records in the output that truncation occurred. This keeps model behavior stable and makes validation tractable.
Cost budget refers to inference and retrieval consume compute (and, in some deployments, money). The orchestrator can apply tiering: first a compact "triage" role on all events; escalate only the ambiguous or high-risk ones to a larger role or to retrieval. Caches (for [Z] context and for repeated prompt patterns) further reduce cost without changing semantics.
Finally **Risk budget**, every organization chooses how much automation risk it will accept. The validator encodes this as thresholds and gates: "no automatic response unless confidence $\geq \theta$ and the asset is not marked high-sensitivity"; "two independent roles must agree for containment." These are not mere settings; they are part of the governance story and are written into the Detection Record so that actions remain auditable.

Many environments benefit from micro-batching: grouping a few related events that occur within a short window and treating them as a unit of reasoning. For example, a process creation (4600) followed by a network egress and a new scheduled task is more informative together than in isolation. Micro-batching still respects event-driven causality (the arrival of the first event starts the timer), but it allows the Interface to construct a single [Y] bundle with richer structure and better signal-to-noise.

Security telemetry is messy: duplicates appear, clocks skew, and delivery may be at-least-once. Operators therefore behave idempotently: if the same RawEvent is seen twice,

its [Y] identifier (derived from content hash + provenance) is the same, so downstream correlation does not double-count it. Where ordering matters (e.g., reconstructing a process tree), the Pre-process operator uses timestamps and parent/child identifiers to rebuild causal order rather than trusting arrival order. The Detection Record itself is append-only and hash-addressed, which makes "exactly-once" less fragile: duplicates may be ingested, but they collapse onto the same identifiers.

Because the pipeline is event-driven, observability is natural: every operator emits metrics (throughput, error rates, queue depth), logs (artifact IDs, schema-violation reasons), and traces (a correlation ID that follows an event through the operators). Two patterns are especially useful:

- **Guardrail metrics:** fraction of inferences rejected by contract checks vs. by policy checks; average confidence over time; drift in the distribution of mapped techniques.
- **Governance timelines:** when [X] changed, when validation thresholds changed, when a new model version was adopted. These markers explain sudden shifts in acceptance rates or response behavior.

Failures are inevitable; what matters is that they be contained. If retrieval fails, the Interface proceeds without [Z] and sets a "context-degraded" flag; the validator may then lower the action privileges of the resulting record. If the model fails (timeout, invalid JSON), the Interface can re-prompt once with stricter settings; if it fails again, the case is routed to Feedback and, optionally, to a rule-based fallback. Importantly, failures do not mutate [Y]; they produce separate artifacts (ValidationIssue, Feedback) with clear provenance.

An LLM-enhanced SIEM evolves: prompts are refined, validation thresholds adjusted, retrieval sources updated, and model versions rotated. The orchestration treats these changes as versioned configuration, not as silent tweaks. New versions are introduced alongside old ones (canarying), with a small fraction of traffic routed to them. Detection Records capture which versions were in force, making before/after comparisons easy. When a change backfires, rollback is mechanical: switch the routing, not the code.

Operationally, privacy is enforced at three crossings. First, Pre-process performs data minimization and optional PII (Personally Identifiable Information) redaction before anything touches a prompt. Second, Retrieval uses an allow-list and logs the exact citations injected into [Z]. Third, the LLM Layer runs under a placement policy: on-prem or VPC inference for sensitive tenants; no prompts leave the organization's boundary unless policy explicitly permits it. These decisions are recorded in the Detection Record's privacy annotations so audits can verify compliance.

4.5.2 A concrete walk-through

To make the orchestration less abstract, consider a Windows host that emits a 1234 process creation for **powershell.exe** followed by an outbound connection. The Input Retriever attaches provenance and passes the raw XML to Pre-process, which extracts the process tree, normalizes user and host identifiers, and produces [Y]. The Interface composes [X] ("identify suspicious execution and map to candidate ATT&CK techniques; output JSON with findings, evidence_refs, candidates, confidence, rationale") with [Y]; it decides to query [Z] for the "Command and Scripting Interpreter" page because of the presence of PowerShell, but it keeps only a short excerpt. The model returns an inference

with a candidate technique and cites the command line and the network destination from [Y]. Validation parses the JSON, checks that the cited fields exist and the timestamps align, enforces the policy that no containment can be suggested on high-sensitivity hosts, and accepts the result. A Detection Record is issued, visible immediately to analyst views; a cache entry is created for the ATT&CK page in case similar events arrive. If later a burst of similar events appears, the Interface may switch to a micro-batch strategy (grouping the next few events by process lineage) to stay within time and cost budgets without losing context.

4.6 Positioning vs. the state of the art

This section positions the proposed framework with respect to the research surveyed in Chapter 3. In brief, the works reviewed there advance modeling and analytics for logs (summarization, prompt strategies, domain adaptation, RAG over heterogeneous sources, ATT&CK mapping). The contribution of this chapter is architectural and operational: it specifies how such analytics are safely integrated into a SIEM by means of explicit artifact contracts ([X]/[Y]/[Z]/[F]), a Validation Layer with policy/risk controls, and a canonical Detection Record that makes outputs reproducible, auditable, and actionable.

LogPrécis focuses on malicious Unix shell sessions and uses language models to segment sessions into tactic-level behaviors, yielding compact "fingerprints" that help analysts understand families of attacks [27]. It demonstrates that LLMs can abstract noisy command sequences into meaningful behaviors at scale.
In our framework, it aligns naturally with a reporter role in the LLM layer: it can distill [Y] (e.g., command transcripts) into analyst-friendly summaries or tactic sequences. The novelty of our proposal is what surrounds that role: governed [Z] context (allow-listed, time-bounded), schema-first outputs (an LLMInference contract), Validation (format → consistency → policy gates), and a Detection Record that downstream tools and playbooks can trust. In other words, LogPrécis provides a powerful analytic; the framework provides the safe plumbing that carries it into a SIEM.

Works on prompt strategies for interpretable, online log analysis (e.g., [26]) and agentic patterns like ReAct for retrieval-and-reasoning [31] show that careful prompt design and tool use improve both accuracy and interpretability without heavy task-specific training.
These ideas live in our [X] instruction library (task templates, few-shot exemplars, decoding policies) and in [F] feedback directives, which let analysts tighten or amend instructions after validation (e.g., "require function-call JSON," "prioritize process-lineage evidence"). The key step forward here is discipline around prompting: [X] and [F] are versioned, auditable artifacts; [Y] (evidence) is sanitized and immutable; [Z] (context) is governed for source and size. That separation reduces accidental leakage and makes prompt evolution traceable—properties typically outside the scope of prompt-strategy papers but essential in SOC operations.

Research on adapting LLMs to logs (e.g., instruction-tuned or continually pre-trained models for log QA and anomaly tasks [25]) and on LLM-assisted forensic report drafting [24] demonstrates that domain-specialized models can reason more reliably over telemetry and produce analyst-useful prose.
As we said earlier, the framework is model-agnostic and role-based: a domain-adapted model can simply serve as the reporter or classifier role. What the framework adds is the governance boundary around the model: governed [Z] retrieval, schema-checked outputs,

Validation (including confidence thresholds and asset-sensitivity rules), and a Detection Record with correlation keys and privacy/retention annotations.

LogBabylon argues for cross-log integration and uses RAG to help models handle heterogeneous formats and improve analyst workflows [28]. It highlights the practical value of consolidating sources and using retrieval to inform interpretation.
We agree with the integration + RAG premise, but we govern retrieval explicitly: [Z] is built only from allow-listed sources, is bounded in size, carries citations and freshness metadata, and is cached with short TTLs to avoid cross-tenant leakage. Crucially, retrieval and caching decisions are recorded in the Detection Record, so investigations can replay the exact context that influenced a decision. This turns “RAG helps" into “RAG helps within auditable boundaries."

RAM shows that multi-stage prompting can map structured SIEM rules to MITRE ATT&CK techniques across multiple models without fine-tuning, and that curated external context boosts performance [30]. It is a concrete instance of classifier-style reasoning over structured inputs.
This maps one-to-one onto a classifier role in the LLM layer. The framework then contributes the post-classification safety path: schema-checked JSON, Validation for consistency with [Y]/[Z] and for risk/policy gates, and the emission of a Detection Record suitable for correlation and governed response. RAM demonstrates that ATT&CK mapping is feasible; the framework ensures such mappings become operationally trustworthy.

Chapter 3 also surveys hallucination mitigation (temperature control, verification chains, detection/grounding checks, and guardrail services) [17, 19, 23, 20, 22, 13, 21]. Our design operationalizes those ideas at the system level: low-temperature, function-call/JSON-only outputs for safety-critical steps; governed [Z] retrieval; explicit Validation (format → consistency → policy); and feedback loops ([F] directives for prompt tightening now; curated labels for fine-tuning later). Rather than a grab-bag of techniques, the framework makes them part of the contract.

4.7 Limitations, risks, and mitigations

No design that incorporates generative models into security operations is free of trade-offs. Here we discuss the main limitations and risks of the proposed framework and outline mitigations that reduce (but do not eliminate) their impact. Where appropriate, the text connects the risk to specific elements of the framework (e.g., [X]/[Y]/[Z]/[F], Validation, the Detection Record) so that mitigations are concrete rather than merely aspirational.

LLMs are pattern recognizers trained to produce plausible continuations, not formal verifiers. Even with domain adaptation, careful prompting, and governed context, they may still hallucinate a fact, over-generalize from weak evidence, or assign confidence poorly. In our pipeline this risk is bounded by design: outputs must be schema-conformant and every claim must cite where in [Y] or [Z] it comes from. That does not guarantee truth, but it ensures that an unverifiable claim cannot pass Validation undetected.
Some mitigations could be to constrain outputs to JSON/function calls; keep decoding deterministic for safety-critical steps; require evidence references for every finding; gate high-impact actions on consensus (e.g., two roles agreeing) or on human approval; maintain offline evaluation suites with seeded ground truth and periodically re-test models and prompts.

Telemetry is untrusted. Attackers can deliberately craft log lines or command arguments that look like instructions to a model. If raw text is concatenated with instructions, the model can be steered. The framework assumes this threat and responds in two places: Pre-process performs prompt-safety sanitization (quoting and escaping untrusted strings in [Y]); and Prompt Interface keeps a hard boundary between [X] (instructions) and [Y] (evidence). Because [Y] is immutable and sanitized, Feedback cannot retroactively alter evidence; it can only patch [X] or contribute vetted snippets to [Z].
Mitigations: Treat all inputs as adversarial; sanitize before prompting; keep allow-lists for function calls/tools; block patterns known to cause model jailbreaks; include “deny directives" in [X] that instruct the model to ignore any embedded instructions found in [Y].

Retrieval helps, but it is not a silver bullet. Knowledge bases can be stale, biased, or simply off-topic for a specific event. Over-long context can distract the model; underscoped context can leave ambiguity unresolved. The framework therefore keeps [Z] small, cited, and freshness-checked, with a short-TTL cache. Detection Records note which context was used so that investigators can replay or contest its relevance.
Mitigations: Enforce allow-lists and TTLs; prefer short, technique-specific snippets over large dumps; monitor cache hit/miss and stale-context rates; regularly curate sources; fall back to no-[Z] operation with a “context-degraded" flag when retrieval is unavailable.

High-volume SIEM environments can produce more candidate events than a large model can analyze within tight SLAs (Service Level Agreements). Retrieval adds variable latency; deterministic decoding reduces variability but may still be slow. The orchestration addresses this with budgets (time, size/tokens, cost) and tiering (small “triage" roles on all traffic; escalate only ambiguous or high-risk items).
Mitigations: Micro-batch related events; summarize before prompting; cap [Z] size; use model cascades; cache common contexts; route non-urgent tasks to batch windows; monitor end-to-end latency and degrade gracefully under load (e.g., skip [Z] for low-risk assets).

Not every behavior maps cleanly to ATT&CK techniques; sometimes the correct answer is “unknown" or “novel." Over-confident mappings create a false sense of precision. The framework separates mapping confidence from detection confidence and allows the model to return “unknown/novel," provided that evidence is still summarized and linked.
Mitigations: Require explicit confidence intervals; allow multiple candidate techniques with rationales; prefer consensus for automated actions; maintain an internal extension taxonomy for behaviors not covered by ATT&CK and document mappings over time.

Ground truth for real attacks is scarce and costly to label. Synthetic datasets help but may not capture operational messiness. The risk is deploying a pipeline that performs well in curated tests and underperforms in live conditions.
Mitigations: Combine red-team replays, curated CTI-derived cases, and live-traffic canaries; evaluate at multiple levels (JSON validity, grounding correctness, TTP precision/recall, time-to-signal); store model lineage and prompt versions in Detection Records so that regressions can be traced to specific changes in [X], [Z], or model versions.

Automated response can reduce dwell time, but it introduces a blast-radius risk when a detection is wrong. The framework’s Validation component is designed as a governance gate: it decides what actions, if any, are eligible for automation given confidence, asset sensitivity, and policy.
Mitigations: Default to notify or isolate-with-revert rather than destructive actions; insist on human approval for high-impact changes; simulate playbooks in a “dry-run" mode before enabling automation; record every action decision inside the Detection Record.

Caches improve latency and cost, but they can create cross-tenant or cross-environment leakage if not scoped. Retrieval indices can also inadvertently mix data if built from multi-tenant sources.
Mitigations: Scope caches and indices by tenant/environment; include [X]/[Y] digests in cache keys; apply short TTLs; audit cache usage in Detection Records; separate fine-tuning datasets per tenant unless an explicit data-sharing agreement exists.

Prompt templates, validation thresholds, retrieval allow-lists, and model versions evolve. Untracked changes create silent drift: acceptance rates shift and nobody can explain why.
Mitigations: Treat [X], [Z] policies, validator rules, and model versions as versioned configuration; roll out changes via canaries; capture versions and digests in every Detection Record; review guardrail metrics (schema-violation rates, grounding errors, confidence distributions) as part of routine operations.

4.7.1 Privacy, compliance, and data residency: local-only vs. API-based inference

Security telemetry and forensic artifacts often contain personally identifiable information, trade secrets, or confidential incident details. In many organizations, and in some jurisdictions, sending such material to an external API is either prohibited or impractical to govern. For this reason, the framework is intentionally compatible with local-only (on-prem or private-cloud/VPC) inference. This subsection clarifies the trade-offs.

Local-only inference. Benefits:

- **Data control and residency.** Prompts ([X]+[Y]+[Z]) and outputs never leave the organization's boundary; storage and retention are under the same governance as the rest of the SIEM.
- **Chain of custody.** Forensic integrity is easier to argue when evidence never traverses a third-party boundary.
- **Policy alignment.** It is straightforward to enforce redaction, minimization, and per-tenant isolation. Easier to audit and certify for regulated environments.

Trade-offs:

- **Hardware and scalability.** Running capable models requires GPUs/accelerators, careful capacity planning, and ongoing maintenance (drivers, runtimes).
- **Model freshness.** Upgrading to new model families or safety features is slower; you own the lifecycle (quantization, fine-tuning, evals).
- **Cost profile.** Up-front capital and ongoing ops may exceed API usage for small volumes; for large, steady workloads on-prem can be cheaper, but only if utilization stays high.

Operational guidance:

- Start with a tiered cascade: small local model for triage + larger local model for escalations; keep a clear path to swap models as hardware evolves.

- Use quantized or distilled variants for first-pass roles; reserve heavier models for classifier roles where accuracy materially changes decisions.
- Keep artifact contracts stable so that changing the model does not ripple through the pipeline.

When an external API is considered Some organizations will still evaluate hosted APIs (e.g., for specialized models). If so, the bar for use must be unusually high. Minimum controls:

- No-retention commitments and region pinning contractually enforced, prompts and outputs must not be used for provider training.
- Transport encryption with mutual authentication; tenant-scoped keys.
- Strict minimization: never send raw logs; send only sanitized [Y] extracts and only when policy allows; consider two-stage abstraction (local reporter role produces a heavily redacted mini-report; only that report is sent for classification).
- Independent logging: keep a local ledger of every prompt digest sent and every response received to preserve auditability.

Residual risk. Even with controls, metadata and timing can be sensitive; legal/regulatory regimes can change; and incident confidentiality can be compromised by subpoenas outside the organization's control. For forensic contexts, the safest default remains local-only inference.

4.7.2 Limits that remain

Even with the mitigations above, residual limitations persist. The framework cannot guarantee truth—only that claims are traceable and governed. It cannot remove all latency variance, nor can it make ATT&CK mapping complete or unambiguous. And while local-only inference addresses major privacy concerns, it imposes costs and capacity limits that some organizations may find prohibitive. The purpose of this chapter is not to claim perfection but to make the trade-offs explicit and to encode safety in the architecture: immutable evidence ([Y]); governed context ([Z]); disciplined instructions and feedback ([X]/[F]); a Validation gate; and a Detection Record that binds decisions to evidence, configuration, and policy.

Chapter 5

Implementation

5.1 Objectives and Scope

This chapter implements a minimal, working slice of the architecture introduced in Chapter 4, it exercises one thin, concrete path through those generic roles to see how the idea behaves on real telemetry. The core hypothesis is simple: if we can compress the noise of Windows event logs into one short, technical sentence that describes what actually happened, then a small model should be able to map that sentence to one or more MITRE ATT&CK techniques. The experimental loop therefore has three moving parts: execute a single, well-scoped Atomic Red Team technique; capture a bounded slice of telemetry; and run a summarize → map pipeline that consumes the exact same artifacts on every run. Retrieval, validation, and feedback, the components (that would govern a production system) remain outside this chapter on purpose.

The lab setting is intentionally modest. Experiments run on a Windows 10 Pro virtual machine under Hyper-V, always reverted to the same snapshot before each test so that observed effects can be attributed to the current technique rather than to residue from earlier runs. Telemetry comes from Sysmon plus selected Windows channels and is exported per channel to XML in a folder named after the technique (for example, T1218.010/...). This keeps evidence easy to inspect later and avoids introducing a heavy normalization stage that the chapter does not aim to build. Operational details—Hyper-V configuration, VM sizing, the export script (to retrieve telemetry)—are documented in the appendix A and are recalled here only when they affect methodology.
Within this loop we instantiate two of the abstract roles from Chapter 4. First, a summarization role turns a per-run XML bundle into one neutral, formal sentence that surfaces the behavior without naming ATT&CK techniques or dumping raw events. For budget and time reasons only, this role is implemented in the lab with GPT-4o via API; the lab did not have the resources to host a large model locally. That is a pragmatic choice, not an architectural requirement: in a production SOC the same role should run on-prem/VPC or be fed sanitized inputs.
Second, a mapping role reads the sentence and returns only the list of ATT&CK IDs (e.g., T1059.001, T1112). We evaluate two inductive biases on the same sentence and under the same instruction ("output only IDs"): a fine-tuned Mistral-7B (LoRA, 4-bit) adapted on CTI-style summaries mapped to ATT&CK, and GPT-4o used directly as a classifier. The first choice reflects a governance constraint: a compact mapper is realistic to host locally and keeps cost and privacy compatible with forensic workloads. The second gives a strong general-purpose comparator under identical prompting, which helps explain where a general model tends to favor recall and where an adapted small model tends to favor

precision. The quantitative results—both on the CTI test split of the fine-tuning run and on transfer to Windows summaries—will be presented later; for now it suffices to note that the adapted classifier does learn the mapping on CTI text and that, on Windows runs, the two models behave complementarily along the precision/recall trade-off.
To keep comparisons fair while acknowledging practical constraints, the evaluation proceeds along three channels. First, an in-domain baseline on the CTI dataset is taken immediately after fine-tuning and is computed only for the fine-tuned Mistral-7B (clean human prose). Second, a Windows pipeline evaluates cross-domain transfer on one-sentence summaries from logs (runs), where both Mistral-7B and GPT-4o are tested under the same "IDs-only" instruction. Third, to obtain a head-to-head comparison on concise human prose with multiple labels without incurring prohibitive API costs, both classifiers are evaluated on a budget-limited CTI subset (the first 70 items). This subset is not aligned one-to-one with the Windows runs.

The scope of this chapter is deliberately narrow. There is no governed retrieval ([Z]), no Validation Layer that checks format, consistency, or policy before promoting a result to a Detection Record, no feedback loop, and no structured output beyond a plain string of IDs. Those omissions are intentional and make the experiment legible: we can observe how summary quality drives mapping quality without interference from other stages. "How-to" material—installation steps, scripts, training and inference settings—lives in the Appendix A, so that the main text can focus on motivations and implications.

5.2 Data and Models

The prototype stands on two kinds of text. On one side there are short CTI-style narratives written by humans, each paired with one or more MITRE ATT&CK techniques. On the other side there are single-sentence summaries distilled from Windows event logs by the reporter. The classifier is trained on the former and tested on both. This asymmetry is intentional: it lets us separate what the model learns on clean prose from how well it transfers to compressed, behavior-centric sentences produced from raw telemetry.

The classifier is Mistral-7B adapted with LoRA techniques. Base weights remain frozen and only small low-rank adapters are learned, which keeps memory and compute requirements within reach of a lab (with modest performance) while preserving most of the capacity of the base model. Despite its relatively compact size, Mistral 7B surpasses larger models—outperforming LLaMA 2 (13B) across all major benchmarks and even exceeding LLaMA 1 (34B) in tasks involving reasoning, mathematics, and code generation [32] [33]. The task is framed as multi-label mapping from a short description to a set of ATT&CK technique IDs. To avoid pushing schema decisions into the learning problem, the output contract is deliberately austere: the model is asked to return only a plain string listing technique IDs, separated by commas or newlines (for example, T1059.001, T1112). There is no JSON envelope, no confidence score, and no rationale. That constraint mirrors the runtime interface used later in the Windows loop and keeps the discussion focused on behavior rather than formatting.

Training follows the now standard "instruction-following" recipe: each example is rendered as a brief instruction to "map the following activity to MITRE ATT&CK techniques", followed by a compact narrative, with the target rendered as the plain list of IDs. The checkpoint used in the rest of this chapter is the one taken at the minimum validation loss A.3; decoding on the CTI test split is configured to be deterministic (a small beam rather than fully greedy) because exact-match improves without changing the output format. For the Windows experiments we keep variance low for the same reason:

the outputs are short strings and reproducibility is a practical concern, even if this chapter does not claim formal stability across re-runs. All low-level settings (optimizer and schedule, precision/quantization, batch/accumulation, hardware envelope) are moved to the Appendix A; what matters here is that the adaptation is lightweight and the model is small enough to be hosted on-prem in a future SOC setting.
The Figure 5.1 shows the instantiated architecture in the lab, illustrating the components that are active in this chapter, how they interact.

Two prompts define the runtime behavior. The reporter prompt takes a bundle of Windows Event XML and asks for one formal, label-free sentence that filters routine noise and surfaces the salient behavior: which process acted, with which flags, against which object, and in which locus (for example, a Registry hive or a Task Scheduler entry). This role is implemented with GPT-4o via API purely for budget/time reasons—a pragmatic lab choice, not a production requirement. In a mature deployment, the same role should run on-prem/VPC or be fed sanitized inputs.
The prompt pre-build I provided([X1] object text in the figure 5.1) is the following:

> **Role:** You are a cybersecurity assistant integrated within a SIEM platform. **Context:** You are analyzing a batch of Windows Event XML logs. Filter out routine system noise. Focus exclusively on behaviors indicating initial compromise or post-exploitation activity (e.g., privilege escalation, unusual persistence mechanisms, suspicious remote access). **Task:** Generate a single, highly concise summary (1-2 sentences, strictly under 100 words) describing the core malicious activity on the system. Use formal, technical English. **Constraints:** - Focus solely on the high-level malicious behavior. - Do not reference MITRE ATT&CK techniques, frameworks, or specific TTPs. - Do not list, cite, or enumerate individual log events or raw data. - Avoid specu lation; state observable behaviors indicative of compromise. Examples:- "A Tiny Web Shell was deployed post-exploitation of CVE 2019-19781, with payloads dropped to specific Citrix directories."- "Fake TLStraffic carrying RC4-encrypted system beacons indicates a likely backdoor communicating with a remote C2."

This prompt shows a possible application of prompt engineering techniques and in this case it fits best into the following categories:

- **Few-shot prompting:** because it provides explicit examples to guide the model's output.
- **Instructional prompting:** the task is clearly defined with specific constraints and expectations.
- **Role-based prompting:** a specific role is assigned to the model ("cybersecurity assistant within a SIEM platform").
- **Contextual prompting:** a well-defined operational context is provided (analyzing Windows Event XML logs for malicious activity).

The [X1] object is attached to the [Y1] object, which refers to the TTP's logs extracted from the previous phase, and the final prompt is sent to the GPT-4o model for analysis and report generation part. Finally it produces a concise behavioral summary describing the malicious activity related to the TTPs executed(in this experiment case only one

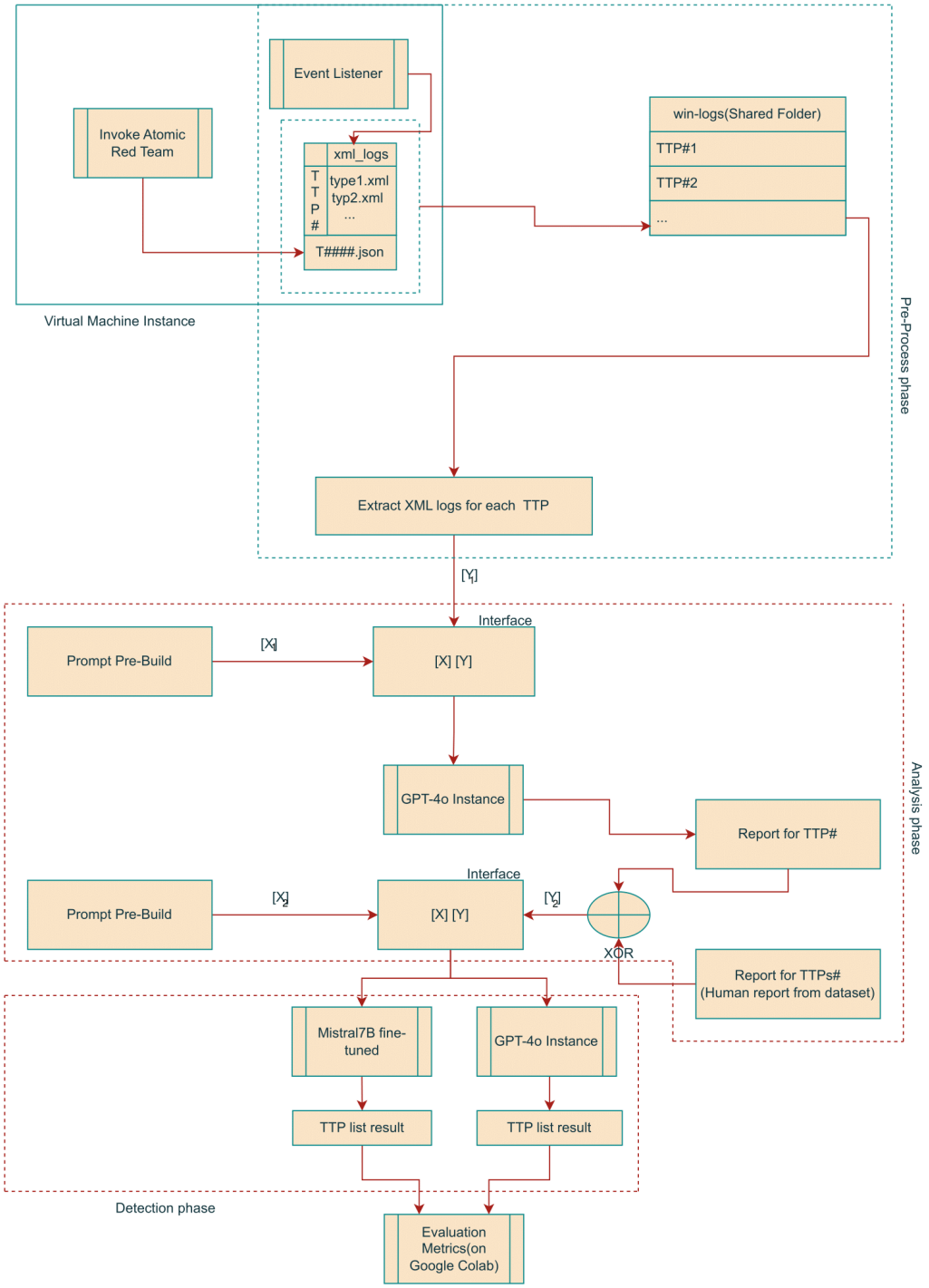

Figure 5.1: Instantiation of the architecture from Chapter 4 in the lab, showing how data flows between components and where artifacts are placed in the process.

TTP is executed at a time) without mentioning the specific TTPs or MITRE ATT&CK framework. Here is an example of a summary generated by the GPT-4o model for the TTP T1003.001(OS Credential Dumping: LSASS Memory):

> "An account with SYSTEM privileges reconfigured Image File Execution Options for lsass.exe and launched multiple debugger binaries (nanodump.x64.exe, rdrleakdiag.exe via WerFault) to generate full memory dumps of the LSASS process into user-temp folders, evidencing automated credential dumping post-compromise."

Of course this summary generation is prone to errors due to the nature of the LLMs, which may not always produce accurate or relevant summaries.
The detector prompt given to the mapping role is intentionally spartan: analyze the sentence and output only the applicable MITRE ATT&CK technique ID(s) as a plain string. Keeping the instruction fixed lets us attribute differences to the models rather than to prompt variation. The prompt interface is composed of two text objects, [X2] and [Y2], where [Y2] represents either the prompt generated by the LLM in the previous phase or the human-written report from the HuggingFace dataset, and [X2] is a predefined prompt that provides the model with the necessary context to perform the detection task. The full prompt is the following:

> **Instruction:** You are a cybersecurity assistant. Analyze the system log below and assign all applicable MITRE ATT&CK technique ID(s), separated by commas. You must output only the list of elements—nothing else.
> **Log:** [Y2]
> **Response:**

Note that the instruction prompt is designed to be simple and straightforward, with a clearly defined task and a strict response format. [Y2] serves as an input attachment to [X2], providing the log or report that is to be analyzed.

The mapping role is evaluated in two families and one budget-driven subset. In-domain, the model is tested on the dataset's held-out CTI split to establish what has been learned; these full-dataset metrics are reported for Mistral-7B only. Cross-domain, the mapping role is exercised on one-sentence summaries from Windows runs (one executed TTP per run), where both Mistral-7B and GPT-4o are compared under the same instruction. Finally, because running GPT-4o across the entire CTI test split was not economically feasible, a head-to-head comparison on clean human prose is performed on a budget-limited subset: the first 70 CTI items. This subset is multi-label and not aligned to Windows runs.
One caveat runs through the whole section. The sentence produced by the reporter is the bottleneck. If it fails to mention the actor or the crucial flags, or if it omits the locus of the action, the mapper's ceiling is immediately capped—no amount of fine-tuning will recover information that is simply not present. For that reason, the effectiveness of the reporter prompt is flagged as a high-leverage axis for future work: contrasting the current instruction with a generic baseline and with a lightly structured variant (for instance, requiring process → object → locus) would quantify how much of the observed variance is due to prompt discipline alone, without changing any model code.

5.3 Simulation environment

The experiments run on a deliberately simple but disciplined Windows lab. Each trial starts from the same Windows 10 Pro virtual machine under Hyper-V, reverted to a known snapshot before execution so that side effects from previous runs do not leak into

the current one. This "reset-per-test" routine is more than a convenience: it makes the telemetry attributable. When a technique fires, the subsequent logs can be read as evidence for that action rather than background noise accumulated over a long-lived host. The VM is sized like a workstation, connected to a default virtual switch, and kept isolated from production networks to keep the risk surface small.
Adversarial activity is produced with the **Invoke Atomic Red Team:** a PowerShell-based framework that provides a collection of atomic tests to simulate adversary techniques from the MITRE ATT&CK framework. It allows to execute individual TTPs in a controlled manner while capturing relevant system logs and events. We chose this framework because it is widely used in the cybersecurity community to test and validate detection capabilities against known techniques, and it offers a robust testing experience—in other words, it simply imports the Invoke-AtomicTest module and downloads the tests you want to run, without writing custom scripts for each TTP. Before a technique is executed, prerequisites are checked and, when available, auto-installed. Techniques are run locally, one at a time, from the clean snapshot. In practice, the PowerShell session that drives the runs sometimes needs to load modules or execute scripts, to unblock that in the lab, the shell is launched with -ExecutionPolicy Bypass at process scope. That setting disables an important guardrail for that session only where in an isolated VM the trade-off is acceptable because the goal is to collect data, not to model a hardened estate, but the work records the implication openly because, in an enterprise setting, it would strongly prefer signed scripts (AllSigned) or Constrained Language Mode rather than bypassing policy.

Telemetry collection favors fidelity over cleverness. The machine runs **Sysmon** [34] applying the default merged profile provided by the project and relies on several Windows event channels (Security, System, Application, Windows PowerShell, WMI-Activity/Operational, Task Scheduler/Operational, and Sysmon/Operational) meanwhile a small listener brackets each run in time, then exports events per channel to XML into a folder named after the technique under test.
The repository organizes rules modularly by Sysmon Event ID (e.g., process creation, image load, registry activity, DNS, file operations) and supplies automation to merge selected modules into a single sysmonconfig.xml. This design offers two advantages relevant to the present study: (i) it yields broad coverage with sensible filters out of the box, reducing time-to-signal in a lab; and (ii) it remains auditable and maintainable, since modules can be added or excluded in a controlled manner as needs evolve.
From a semantic standpoint, the choice aligns with the kinds of evidence this chapter relies on. Sysmon Event IDs expose precisely the actors and artifacts that drive behavior-centric summaries and ATT&CK mapping: Event ID 1 (process creation, including command line), ID 7 (image load), IDs 12-14 (registry key/value create, modify, delete), ID 22 (DNS query), and newer additions such as ID 27-29 for executable file blocking/detection. These fields make it possible to state, in compact form, which binary acted, with which flags, against which object, and in which locus—information that the summarization role condenses into a single sentence, and that the mapping role subsequently interprets. Using an XML-first export preserves the nested attributes (e.g., CommandLine, Image, TargetObject) that would be flattened or lost in simplified formats [35]. There are, however, explicit trade-offs in adopting the default profile without estate-specific tuning. The default configuration is intentionally general and will capture benign high-volume patterns alongside the signals of interest. In a production environment this would warrant a second phase where frequent, demonstrably benign sources are excluded and high-value modules are refined to the organization's software baseline; in a lab setting with one Atomic technique per clean snapshot, the additional background

is manageable and, in fact, beneficial for reproducibility, since other readers can apply the same public profile and obtain comparable bundles. The modular structure is designed precisely to facilitate this evolution from "broad and sane defaults" to "curated and estate-aware" without rewriting the configuration from scratch. It is important to delineate what Sysmon does and does not claim. Sysmon provides observational evidence on the endpoint; it does not by itself ascribe ATT&CK techniques. The mapping reported in this chapter is performed by the pipeline's modeling components using the summarized evidence. In other words, Sysmon is the sensor, not the labeler—a distinction that mirrors the governance model introduced in Chapter 4, where validation and policy determine whether an observation is promoted to a detection artifact.

From this bundle, the summarization role, implemented with GPT-4o via API purely for budget and time reasons, produces one sentence in formal, technical English. The prompt 5.2 prepared sets the tone and the discipline: analyze Windows Event XML, filter out routine noise, and deliver a single neutral description of what happened; do not name ATT&CK techniques; do not dump raw events. The instruction includes a couple of very short exemplars to anchor style and granularity. The goal is not to produce a narrative report but to surface the minimal behavioral cues a mapper needs: which process executed, with which flags, on what object, and in which locus (for example, a Run key under the Registry or a scheduled task entry).

The mapping role consumes exactly that sentence and nothing else. For comparability, both variants—the fine-tuned Mistral-7B and GPT-4o used directly as a classifier—receive the same detection instruction 5.2: assign the applicable MITRE ATT&CK technique IDs and output only the list (a plain string, comma or newline separated). No schema is enforced and no post-hoc "repair" is applied; the outputs are stored as returned along with run metadata so that downstream scoring can be explicit about what was and wasn't produced.

Two concrete techniques make the data path tangible. When executing T1082 (System Information Discovery), logs show short-lived cmd.exe or PowerShell processes issuing benign-looking queries. The summarizer tends to produce a crisp line—essentially "a command-line session queried host identification and system details"—and, because the behavior is specific, the mapper can often hit the exact technique. By contrast, T1218.010 (Signed Binary Proxy Execution: regsvr32) is fertile ground for near-neighbor confusion: the evidence centers on regsvr32.exe with /i and /s; a general model may enumerate that family and add broader execution labels, whereas a small adapted model may return a single, conservative guess. These two ends of the spectrum—deterministic discovery vs. signed-binary proxy execution—reappear in the results as different precision/recall behaviors.
A total of 70 MITRE ATT&CK TTPs were tested. The orchestration for every run is identical: restore the snapshot, execute one Atomic technique, export a time-boxed set of XML files per channel, ask the summarizer for one sentence, pass that sentence to both mappers under the same instruction, and persist the plain-text lists the models return. That sameness is a strength: it removes "plumbing variance" from the picture so that differences in detection can be discussed in terms of what was said in the sentence and what the models made of it.

5.4 Testing methodology and results

Results are reported in two phases that mirror the life of the system: first, an in-domain baseline taken immediately after fine-tuning on the CTI dataset; second, the pipeline

evaluations on Windows, where inputs are compressed, behavior-centric sentences produced from logs (plus a complementary test on short human-written summaries).
All evaluations are framed as multi-label classification. We restate the metrics in the vocabulary of this chapter so that their operational meaning is clear.

- **Precision (micro, macro)**: Measures the proportion of true positive predictions among all positive predictions made by the model. *Micro precision* aggregates the contributions of all classes by counting the total true positives and false positives across the dataset. *Macro precision* computes precision independently for each class and then takes the unweighted average.
- **Recall (micro, macro)**: Measures the proportion of true positive predictions among all actual positive instances in the dataset. *Micro recall* aggregates across all classes, while *macro recall* calculates recall per class and then averages the results, giving equal weight to each class.
- **F1-score (micro, macro)**: The harmonic mean of precision and recall. *Micro F1-score* considers global true positives, false negatives, and false positives, while *macro F1-score* computes the F1-score per class and averages them equally, regardless of class frequency.
- **Exact Match**: Measures the proportion of instances where the predicted set of TTPs exactly matches the ground truth set. This metric is particularly relevant in multi-label classification settings, as it penalizes any partial mismatch between predicted and true labels.
- **Hamming Loss**: Measures the fraction of incorrectly predicted labels (both false positives and false negatives). It is computed as the average number of incorrect labels per instance, normalized over the total number of labels, and provides an indication of label-wise prediction error.

Before entering the Windows environment loop, the fine-tuned classifier was evaluated on the held-out CTI test split from the dataset used for training. Using the checkpoint at the minimum validation loss, Mistral-7B (LoRA, 4-bit) achieved micro-F1 = 0.5748 (micro-precision 0.6259, micro-recall 0.5314), macro-F1 = 0.2985, Exact Match = 0.5808, and Hamming loss = 0.0026 A.4. These numbers characterize what the adapted small model learns on clean, human CTI prose; they serve as a baseline only. The micro-macro gap reflects label imbalance. Decoding used low-variance settings (a small beam), which improved Exact Match over greedy while preserving the plain-string output contract.

Evaluation on LLM-generated summaries from Windows logs single-label TTPs. Each pipeline run executes one TTP from a clean snapshot, exports a time-boxed set of events to XML, and produces one formal sentence with the reporter. That same sentence is then given—unchanged and under the same instruction ("IDs only")—to two classifiers: the fine-tuned Mistral-7B and GPT-4o used directly as a classifier. Across 70 runs with single-label ground truth, the results are:

Model	$\mathbf{P}_\mu$	$\mathbf{R}_\mu$	$\mathbf{F1}_\mu$	$\mathbf{P}_M$	$\mathbf{R}_M$	$\mathbf{F1}_M$
GPT-4o	0.1875	0.5571	0.2806	0.2863	0.5571	0.3435
Mistral 7B	0.2857	0.2571	0.2707	0.1961	0.2571	0.2071

Table 5.1: Results using LLM-generated prompts (single TTP ground truth)

Hamming Loss: GPT-4o = 0.0408, Mistral7B = 0.0198
Exact Match: GPT-4o = 0.0286, Mistral7B = 0.2429

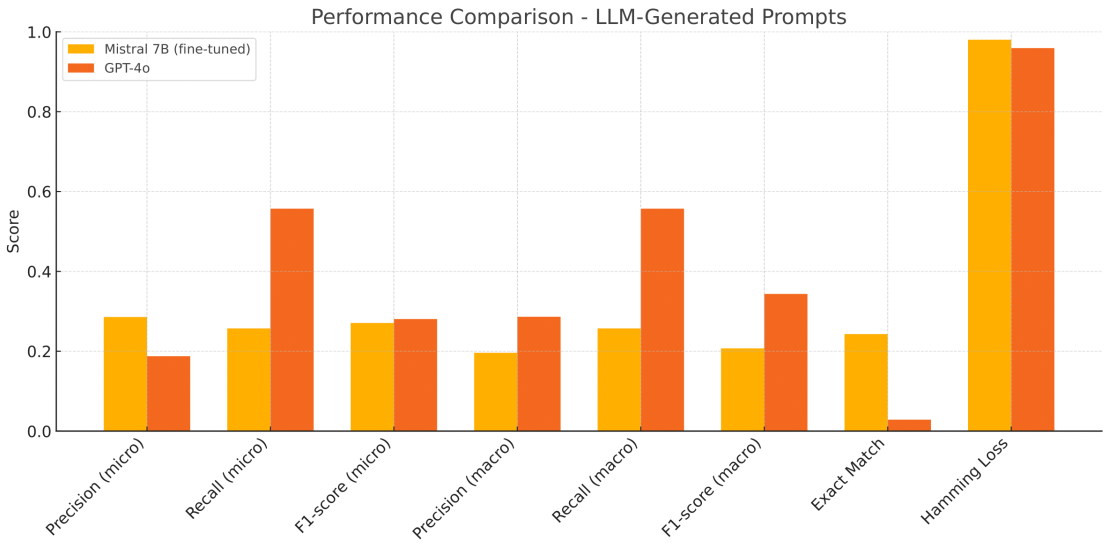

Figure 5.2: Comparison of GPT-4o and Mistral-7B performance on LLM-generated prompts (single TTP ground truth).

Two complementary behaviors emerge. GPT-4o favors recall: it often contains the correct ID somewhere in its output but tends to add plausible extra IDs, which depresses Exact Match and raises Hamming Loss when only one label is expected. Mistral-7B favors precision: it emits fewer IDs, which yields a much higher Exact Match and lower Hamming Loss, at the cost of missed positives (lower recall). Put simply: the general model casts a wider net; the adapted small model is choosier.
A per-run containment view makes this concrete. Over the 70 aligned runs, the breakdown is: both models surfaced the correct ID in 16 runs; Mistral-only in 2; GPT-4o-only in 23; neither in 29. Read alongside the strict metrics, this says that GPT-4o is more likely to surface the right candidate somewhere in a longer list (39/70 containments), while Mistral is more likely to match exactly when a single label is expected (Exact Match 0.2429 vs. 0.0286). Operationally, containment informs recall at triage time; Exact Match and Hamming Loss inform noise and analyst burden.
In the tables B.1 and B.2, the per-run detections and predictions are reported in detail. A qualitative glance at specific techniques is consistent with this split. Deterministic discovery such as T1082 (System Information Discovery) is mapped cleanly when the sentence states the actor and intent; by contrast, signed-binary proxy execution like T1218.010 (regsvr32) invites near-neighbor additions unless the sentence names the binary and salient flags (e.g., /i, /s). In the raw strings for those runs, GPT-4o tends to enumerate families; Mistral often picks a single candidate and stops.

Evaluation on a CTI subset (70 items, multi-label; not aligned to Windows runs) To compare the two classifiers on concise human prose without logging artefacts and within an API budget, I evaluated a subset of the CTI dataset: the first 70 items in dataset order. Each item is a compact analyst-style description with multiple ground-truth techniques (multi-label). This subset is not aligned 1:1 with the 70 Windows runs; it exists solely to provide a prose-only, multi-label head-to-head comparison under the same "IDs-only" instruction. Because it is the first 70 items rather than a stratified sample,

the figure 5.3 and table 5.2 below should be read as indicative, not as a statistically representative estimate for the entire dataset.

Model	$\mathbf{P}_\mu$	$\mathbf{R}_\mu$	$\mathbf{F1}_\mu$	$\mathbf{P}_M$	$\mathbf{R}_M$	$\mathbf{F1}_M$
GPT-4o	0.1624	0.3478	0.2215	0.1815	0.2880	0.2049
Mistral 7B	0.3600	0.1957	0.2535	0.1461	0.1694	0.1444

Table 5.2: Results using human-written prompts (multi-TTP ground truth)

Hamming Loss: GPT-4o = 0.0527, Mistral7B = 0.0248
Exact Match: GPT-4o = 0.0286, Mistral7B = 0.1714

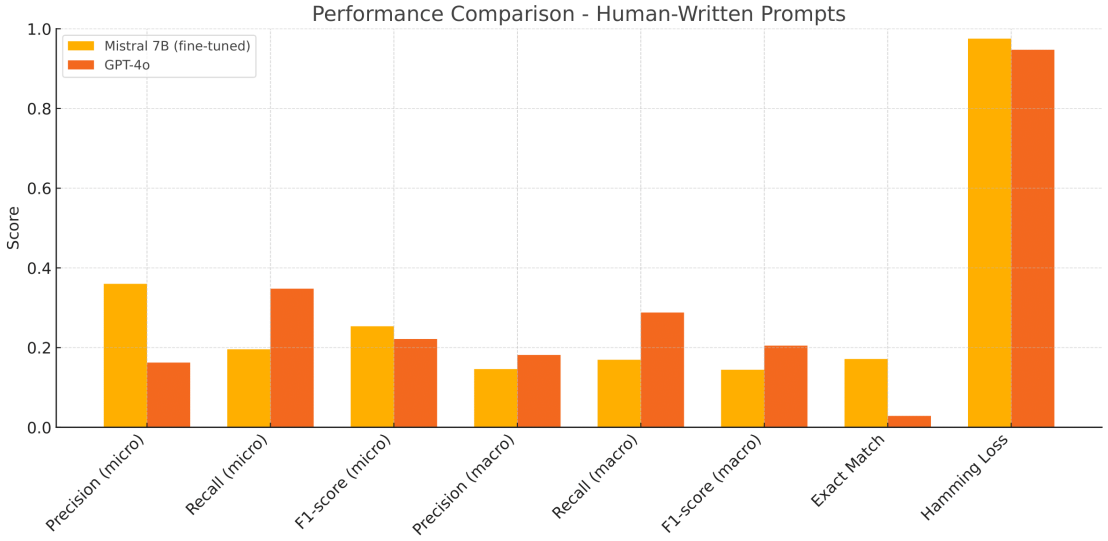

Figure 5.3: Comparison of GPT-4o and Mistral-7B performance on a CTI subset (multi-label, not aligned to Windows runs).

GPT-4o exhibits higher recall in both micro (0.3478) and macro (0.2880) settings, indi cating that it identifies a larger number of relevant TTPs per example. However, its lower precision leads to modest F1-scores—micro F1 at 0.2215 and macro F1 at 0.2049—sug gesting it tends to over-predict, introducing more false positives. In contrast, Mistral 7B achieves a much higher micro precision (0.3600 vs. 0.1624 for GPT-4o), but its lower recall (0.1957) results in a slightly better micro F1-score (0.2535), but worse macro F1 (0.1444). This indicates again that Mistral is more conservative and favors precision over coverage, particularly at the class level. The task is harder for both models under multi-label evaluation (as expected), and absolute values decrease. The relative behaviors nevertheless persist: GPT-4o keeps a recall-heavy profile; the adapted Mistral maintains tighter outputs (higher micro precision, lower Hamming Loss, higher Exact Match).

5.5 Discussion

The fine-tuned classifier used in the pipeline was adapted on a small CTI corpus. This constraint is worth stating explicitly because it interacts with every observation in 5.4. A limited training set narrows the distribution of phrasings, artifacts, and technique co-occurrences the model can see during adaptation; as a result, the in-domain baseline on

the CTI test split reflects competence on clean, familiar prose, whereas the Windows loop demands cross-domain transfer to one-sentence summaries distilled from telemetry. The gap reported in 5.1 is therefore not surprising. With fewer examples per label, rare techniques remain under-represented, which depresses macro-averaged metrics and pushes the model toward conservative outputs at inference time. This conservative bias shows up in the Windows results as higher Exact Match and lower Hamming Loss—the classifier predicts fewer IDs and avoids speculative additions—but also as lower recall when the one-line summary is ambiguous.

A small data regime also sharpens the role of prompting and decoding. Because the output contract is a short, rigid string, decoding choices that favor determinism (such as a small beam) help the model settle on the most plausible single list rather than exploring diverse alternatives; this improves strict metrics but does not manufacture information that the summary does not contain. Likewise, the reporter prompt becomes a principal lever: when training data are scarce, the sentence fed to the classifier must surface the decisive cues—actor, flags, and locus—so that the mapper can align the summary with the patterns it actually learned. In other words, the ceiling in 5.4 is jointly set by data volume during adaptation and by the information content of the sentence at runtime.

These constraints do not invalidate the pipeline; they delimit its claims and suggest concrete next steps. Modest data augmentation on the CTI side (paraphrases that preserve technique semantics; curated hard-negatives to separate near-neighbors) would broaden the model's exposure without changing its size. A small paired Windows set—log → one-sentence summary → ATT&CK—would reduce domain shift directly and allow the adapter to learn from the same style of input it will face in production. Even without collecting new labels, tightening the reporter contract (requiring the sentence to include, at minimum, process → object → locus) would raise the informative content of the input the classifier sees and therefore its attainable recall. Read in this light, the precision-heavy behavior observed in 5.4 is not a flaw of the architecture; it is a stable consequence of a small, careful adaptation coupled with a terse input. The framework anticipated exactly this division of labor: a recall-oriented proposer followed by a precision-oriented confirmer, with a Validation step to arbitrate promotions.

Chapter 6

Conclusions

This thesis set out to answer a simple, practical question: can a SIEM pipeline use large language models to turn raw security telemetry into attack-informed claims that an analyst can trust? Rather than proposing yet another monolithic "LLM for security," the work focused on roles, interfaces, and contracts. Chapter 4 formalized a conceptual architecture for a modular, governed pipeline: evidence is ingested and normalized; a reporter condenses noisy events into a short, neutral sentence; a mapper translates that sentence into MITRE ATT&CK techniques; a validator and policy layer decide what gets promoted to a Detection Record; and feedback and retrieval are designed to keep the system current without entangling runtime logic with training. The emphasis throughout was on clear boundaries—what each component is allowed to consume and produce—so that behavior can be reasoned about and audited.

Chapter 5 then exercised one thin, end-to-end path through that design. The path was deliberately frugal—no RAG, no validator, no feedback loop—so that the behavior of the reporter-mapper pair could be observed without confounders. A compact model, Mistral-7B adapted with LoRA, served as a locally deployable mapper; GPT-4o served both as a reporter (to produce the one-sentence summaries from Windows logs) and as a second mapper for head-to-head comparison under the same "IDs-only" instruction. The evaluation was staged on two distinct input regimes. In-domain, immediately after fine-tuning, Mistral was tested on the CTI dataset's held-out split, establishing that the small model does learn the narrative-to-ATT&CK mapping on clean human prose. Cross-domain, the same mapping role was exercised on single-sentence summaries distilled from Windows events collected in a disciplined lab loop (one executed Atomic technique per run, export to XML, summarize once, map once). Because API cost precluded running GPT-4o on the entire dataset, a budget-limited subset of 70 CTI items provided from the dataset itself, multi-label head-to-head comparison that is explicitly not aligned with the Windows runs.

Across these settings, the results are coherent. On CTI text, the adapted Mistral reaches a credible baseline and shows that a small, governable model can shoulder the mapper role. In the Windows loop, the two classifiers separate along a line that is operationally meaningful: GPT-4o favors recall, often surfacing the right technique among several plausible candidates, while Mistral favors precision, emitting tighter lists that improve exact matches and reduce label-wise error at the cost of some misses. On the 70-item CTI subset this contrast persists, now on concise human prose rather than sentences distilled from logs. Read through the lens of the Chapter-4 design, this is not a quirk but a useful division of labor: a recall-heavy proposer paired with a precision-leaning confirmer becomes reliable once a validator is in place to check format, reconcile obvious inconsistencies against simple evidence cues, and enforce policy before promotion to a Detection

Record.
Two broader contributions emerge from this work. First, it clarifies the contracts that make LLMs safer to use in a SIEM context: narrow inputs and outputs, deterministic decoding for short targets, and the deliberate separation of reporting, mapping, and validation. Second, it grounds the discussion in reproducible practice: a clean, reset-per-run Windows lab; a well-known Sysmon configuration; fixed prompts; plain-string outputs; and published per-run tables that make successes and failures inspectable. The thesis does not claim completeness—several pieces of the architecture (retrieval, validation, feedback) remain future work by design—but it shows that even a small slice, if properly governed, already exposes the central trade-offs and provides a path for a SOC to modernize detection without surrendering control.

6.1 Challenges and Limitations

The most immediate limitation of this work is data regime. The mapper was adapted on a small CTI corpus, and the full-dataset evaluation after fine-tuning was conducted only for the adapted model. That choice made the study feasible in a student lab, but it constrains what the model could reasonably learn—especially for rare techniques—and explains part of the gap between the neat in-domain baseline and the Windows pipeline. When the summaries become terse, behavior-centric sentences distilled from logs, the classifier's conservative bias surfaces: it prefers to say less rather than guess, which improves exact matches but suppresses recall. This is not a defect of the approach so much as a consequence of pairing limited supervision with compressed inputs.
A second constraint is economic and operational. GPT-4o was used via a hosted API for the reporter role and, where budget permitted, as a comparator mapper. That delivered strong baselines quickly, but it is not the right posture for a SOC handling sensitive or regulated data: forensic evidence is not meant to leave the perimeter. The thesis therefore argues for an on-prem or VPC-isolated deployment in production, but it did not implement it. The same budget constraint explains why the head-to-head on CTI prose covers only the first 70 items of the dataset rather than the full split and why that subset is not stratified. These choices are recorded explicitly; they make the comparison informative rather than definitive.
The governance surface is intentionally incomplete. The pipeline exercised here omits retrieval, validation, and feedback—precisely the components that, in Chapter 4, were meant to keep the system current and safe. Without retrieval there is no principled way to inject fresh organizational knowledge or CTI context; without validation there is no policy gate to catch inconsistent or ill-formed outputs; without feedback there is no structured path to correct systematic errors or to retrain adapters when the estate drifts. The prototype compensates by narrowing the contracts (IDs-only outputs, deterministic decoding), but that narrowness is itself a limitation: analysts receive no confidence or rationale, which would help justify promotions to a Detection Record and accelerate triage.
There are also measurement and methodology limits worth stating plainly. The Windows loop favored single-label ground truth (one technique per run) to keep attribution clean; real incidents can, of course, involve multiple techniques in quick succession. The study therefore adds a multi-label view on concise human prose, but those items are dataset entries, not aligned to Windows runs, and the two views are compared only at the level of aggregate metrics. Exact-match, a deliberately strict score, punishes any extra label; Hamming loss, a label-wise error rate, does not capture semantic "distance" between near-neighbors (e.g., proxy execution variants). Both metrics are useful, but neither says

much about how helpful a sentence was for an analyst or how easily a validator would reconcile a plausible near-miss against simple evidence cues.
On the instrumentation side, the lab relied on the default sysmon-modular profile to shorten time-to-signal and maximize reproducibility. That choice yields broad coverage quickly but is not estate-tuned; in production one would exclude known benign chatter and promote modules aligned with the organization's baseline to keep noise—and SIEM cost—in check. The lab sometimes launched PowerShell with -ExecutionPolicy Bypass at process scope to unblock module loading, a decision acceptable in an isolated VM whose goal was to capture evidence, but not representative of hardened enterprise posture. These trade-offs are recorded transparently, and they do not affect the logic of the results, yet they limit the external validity of the measurements.
Finally, the study does not attempt to stress the system adversarially. Prompt-injection hardening, schema-enforced outputs, and function-calling guardrails were discussed in the design but not implemented. The mapper's interface was kept intentionally austere to isolate behavior, but that also means brittleness: a stray token or formatting quirk can turn a good forecast into a scoring error. Likewise, the study does not quantify stability across repeated runs (variance under identical inputs) beyond adopting deterministic decoding where possible. None of these choices undermines the central finding—that the reporter-mapper pair can be made legible and useful under tight contracts—but they mark where engineering effort must flow before a SOC can rely on such a pipeline as part of its primary detection surface.

6.2 Future Improvements

The most direct path forward is to tighten the two contracts that define the prototype path—the sentence that enters the mapper and the artifact that leaves it—and then to reintroduce, in a disciplined way, the governance pieces that Chapter 4 anticipated but the experiments deliberately omitted. On the input side, the reporter must evolve from a well-phrased instruction to a measurable contract. The single sentence worked because it forced the model to foreground essentials, but it also hid variation the mapper could not overcome. A structured probe makes this explicit: hold the mapper fixed and vary the reporter along a narrow ladder, from a lightly templated sentence that obliges at least actor → flags → object → locus to a more structured template that also enforces tense and subject-verb-object order. For each rung, measure not only downstream detection but the sentence fidelity itself—did those fields actually appear? The lighter variant is easy to adopt and preserves stylistic flexibility, yet leaves more room for omission; the structured variant is clearer for the mapper and tends to lift recall on rare patterns, at the cost of some brittleness when upstream evidence is thin. Starting light and promoting fields to "required" only where they demonstrably improve recall keeps discipline without overfitting the prose.
A complementary improvement is to constrain decoding against the ontology and to return structured artifacts. The mapper currently emits a plain string; that made scoring easy, but it also made the system brittle. In a production pipeline the output should be a small, schema-conformant record—valid ATT&CK IDs enforced at generation time, a machine-checkable shape, and an optional human-readable rationale kept separate from the ID list. The rationale need not be a chain-of-thought; a short, anchored explanation ("regsvr32 executed with /i /s against a remote scriptlet; signed-binary proxy execution is likely") is enough to support validation and analyst trust. Constrained decoding or grammar-guided generation would all but eliminate formatting errors and open the door to policy checks in the validator: reconcile IDs against the sentence, reject contradictions,

and apply minimal heuristics (for instance, forbid multi-family mixes that are implausible given the observed actor and locus).

Given schema-constrained outputs, promotion should be gated by a validator that is opinionated but light. A rule-first validator (schema checks plus a few sentence-ID consistency rules) is transparent, deterministic, and cheap, yet may miss subtle semantic slips; adding a small discriminative checker over (sentence, IDs) catches near-neighbor confusions, though it requires labels and care to avoid opacity. A third option is a proposer/confirm-er pattern: a recall-leaning proposer nominates candidates and a precision-leaning confirmer prunes them under calibrated thresholds; this balances errors and routes disagreements to the validator or a human, at the price of extra latency and orchestration. For an incremental path, adopt rule-first now, instrument reasons for rejection, and introduce the confirmer once thresholds are stable.

The next class of improvements concerns data. Training on human CTI prose was sufficient to show feasibility, but cross-domain performance is capped by how much the mapper has seen of the sentence style it will face at runtime. Two data investments stand out. First, assemble a paired set of the form (logs $\rightarrow$ one-sentence summary $\rightarrow$ ATT&CK), even if small, so that adaptation can directly internalize the reporter's style and omissions. Second, repair the imbalance that depresses macro performance by targeted augmentation: paraphrases that preserve technique semantics, and hard negatives that force the model to separate near neighbors. Neither requires an exotic training regime; both make the small adapter more robust without changing its footprint. In parallel, repeat the GPT-4o head-to-head on a stratified CTI sample rather than the first N items to reduce sampling bias and to sharpen confidence in the observed recall-versus-precision trade-off.

As the footprint widens beyond Windows logs, downstream components benefit from a small, explicit canonical schema (stable IDs and timestamps, subject-verb-object fields, host/user/process keys, plus a concise "fact" string) with per-source adapters that only fill what they can prove. A community schema buys interop quickly but invites field-guessing and silent mapping errors; a narrow local schema tightens contracts and audits but needs more glue when new sources arrive; an hybrid approach keeps the core strict while projecting community-compatible views for tools. Similarly, context should prefer a slow-moving, versioned cache (baselines, allow-lists, policy snippets, recent ATT&CK deltas) over open-ended retrieval: the cache is predictable and low-drift but less helpful on long-tail behaviors; full semantic retrieval is richer but increases operational burden and the risk of spurious context. A mixed posture—cache by default, retrieval only behind clear guardrails—keeps blast radius small while covering rare cases.

With those contracts and data in better shape, the pipeline is ready to reintroduce governance. The first component to add back is a Validation layer that is opinionated but lightweight: enforce the schema, check ID-sentence consistency against a few evidence cues, gate promotions to a Detection Record, and log decisions. The second is governed context. The mapper should be able to consult a stable cache of organizational knowledge—golden process baselines, known-good admin scripts, current ATT&CK changes—so that it can discount expected behavior and avoid stale mappings. Retrieval here is not a full semantic memory; it is a narrow, slow-moving context that reduces unnecessary guesses and helps the validator explain promotions. The third is feedback. Whether from analyst adjudications or from simple counters in the validator, the system should return a trickle of curated examples to the adapter: confirmed hits, confirmed misses, and representative near-misses. A plain supervised refresh at regular intervals—no complex reinforcement—would be enough to keep the mapper aligned with the estate and to prevent drift.

The privacy and deployment posture deserves equal attention. The thesis relied on a hosted reporter for budget reasons; a SOC cannot. The natural trajectory is local or VPC-isolated inference for both roles, with a redaction front-end for any optional external services. This is not merely a compliance box to tick: keeping inference close to the logs collapses latency, enables deterministic builds of the models in use, and makes it tractable to run shadow modes (new prompts, new adapters) on mirrored traffic before promotion. Surrounding this with routine MLOps—model and prompt versioning, dataset lineage, canary deployments, and rollbacks—turns the prototype path into a manageable product surface rather than an experiment.

Finally, the pipeline should be scaled outward and deepened. The contracts remain the same, but the reporter must learn different dialects. Deepened means moving beyond the one-technique-per-run simplification: multi-stage runs with interleaved techniques, time-aware summarization that carries forward state across a short horizon, and correlation that recognizes when several weak, precise hints add up to one strong claim. None of this contradicts the central insight of the thesis. It extends it. By keeping roles narrow, outputs structured, and validation explicit, the same small building blocks—one sentence in, one governed record out—can scale to richer evidence without surrendering control.

Appendix A

Mistral-7B fine-tuning

A.1 Overview and intent

This appendix records the practical details of adapting Mistral-7B-Instruct with LoRA for the task of mapping short security narratives to MITRE ATT&CK techniques. The goal is reproducibility: a reader should be able to recreate the training environment, regenerate the checkpoint at the validation minimum, and reproduce the in-domain test metrics reported in Chapter 5.

A.2 Environment and dependencies

The experiment aims to develop the analysis and detection component that will be deployed in the pipeline. The component is an assistant capable of analyzing system logs and identifying relevant MITRE ATT&CK techniques. Key components included:

- **Base model:** Mistral-7B-Instruct, a 7-billion parameter language model optimized for instruction-following tasks.
- **Dataset:** Security-TTP-Mapping (tumeteor/Security-TTP-Mapping from Hugging Face) [36].
- **Training Approach:** Parameter-efficient fine-tuning with LoRA.
- **Quantization:** 4-bit NF4 with double quantization.
- **Hardware:** Google Colab with A100 GPU (40GB VRAM).
- **Software:** Python 3.10, PyTorch, Hugging Face Transformers, torch library.
- **Objective:** Multi-label classification of ATT&CK techniques.

Hardware and training configuration The training was performed on Google Colab with an NVIDIA A100-SXM4 GPU (40GB VRAM).

Component	Specification
Platform	Google Colab
GPU	NVIDIA A100-SXM4 (40GB VRAM)
VRAM Utilization	22–24GB during training
Batch Size Support	16 (with gradient accumulation)
Quantization	Enabled (4-bit NF4)
Training Duration	∼6 hours (2400 steps)

Table A.1: Hardware configuration and resource utilization

Parameter	Value
Max steps	2400
Batch size	16
Gradient accumulation	2
Learning rate	1×10^{-4}
Warmup steps	100
LR scheduler	Constant
Early stopping patience	2
GPU Memory	40GB (A100)
VRAM Usage	∼24GB

Table A.2: Training parameters

Methodology The training process code is taken and adapted from the Digital Ocean Guide [32]. The model is fine-tuned on the Security-TTP-Mapping dataset, which contains labeled examples of security-related texts mapped to ATT&CK techniques. The training uses LoRA to efficiently adapt the model with a reduced number of trainable parameters, making it feasible to fine-tune large models on limited hardware.
Below are the key steps and configurations used in the training process.

Listing A.1: Loading the dataset

```
from datasets import load_dataset
ds = load_dataset("tumeteor/Security-TTP-Mapping")
```

Listing A.2: Creating the prompt template for training

```
def create_prompt(example):
bos_token = "<s>"
eos_token = "</s>"

system_message = (
    "You␣are␣a␣cybersecurity␣assistant.␣"
    "Your␣task␣is␣to␣analyze␣a␣system␣log␣and␣assign␣the␣most␣
        appropriate␣MITRE␣ATT&CK␣technique(s)."
)

log_text = example["text1"]
mitre_labels = "".join(example["labels"]).strip()
```

```
full_prompt = ""
full_prompt += bos_token
full_prompt += "\n␣Instruction:\n"
full_prompt += system_message
full_prompt += "\n\n␣Log:\n"
full_prompt += log_text
full_prompt += "\n\n␣Response:\n"
full_prompt += mitre_labels
full_prompt += eos_token

return full_prompt
```

Note that the prompt template is designed to provide clear instructions to the model, along with the log text and the expected response format. An example of the prompt generated is:

Listing A.3: Example of the prompt generated

```
create_prompt(ds['train'][0])

OUTPUT: <s>
### Instruction:
You are a cybersecurity assistant. Your task is to analyze a
   system log and assign the most appropriate MITRE ATT&CK
   technique(s).

### Log:
The command processing function starts by substituting the main
   module name and path in the hosting process PEB, with the one
   of the default internet browser. The path of the main browser
   of the workstation is obtained by reading the registry value

### Response:
['T1057']
</s>
```

The following code snippet shows the training configuration (adapted from the guide) and execution (for the load and configuration of the LoRa parameters see the guide [32]):

Listing A.4: Training configuration and execution

```
args = TrainingArguments(
output_dir = "/content/drive/MyDrive/siem-finetuned",
#num_train_epochs=5,
max_steps = 2400,
per_device_train_batch_size = 16,
gradient_accumulation_steps=2,
warmup_steps = 100,
logging_steps=50,
do_eval=True,
#save_strategy="epoch",
#evaluation_strategy="epoch",
```

```
    eval_strategy="steps",
    eval_steps=200,
    save_strategy="steps",
    save_steps=200,
    save_total_limit=5,
    learning_rate=1e-4,
    bf16=True,
    lr_scheduler_type='constant',
    metric_for_best_model="eval_loss",
    report_to=[]
    )

    trainer = SFTTrainer(
    model=model,
    peft_config=peft_config,
    formatting_func=create_prompt,
    args=args,
    train_dataset=ds["train"],
    eval_dataset=ds["validation"],
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
    )
    trainer.train()
```

The table below shows the training dynamics, with the minimum validation loss at step 2200. The model checkpoint at this step is saved and used for evaluation on the test set.

Step	**Train Loss**	**Validation Loss**
200	1.0859	0.9734
400	1.0455	0.9241
600	0.9938	0.9012
800	0.9434	0.8877
1000	0.9023	0.8812
1200	0.9153	0.8722
1400	0.8928	0.8563
1600	0.8081	0.8548
1800	0.8104	0.8462
2000	0.7239	0.8582
2200	**0.7328**	**0.8435**
2400	0.6491	0.8758

Table A.3: Training dynamics with minimum validation loss at step 2200

After training, the model is saved and can be used for inference on new data. The evaluation was performed on the test set of the Security-TTP-Mapping dataset, yielding the metrics reported in Table A.4. Below are the code snippets for loading the trained model, performing inference, and calculating evaluation metrics.

Listing A.5: Loading the trained model and performing inference

```
model_path =
    "/content/drive/MyDrive/siem-finetuned/checkpoint-2200"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForCausalLM.from_pretrained(
    model_path,
    device_map='auto',
    quantization_config=nf4_config
)
model.eval()

def create_inference_prompt(log_text: str) -> str:
bos = "<s>"
instruction = (
    "You␣are␣a␣cybersecurity␣assistant.␣"
    "Analyze␣the␣system␣log␣below␣and␣assign␣all␣applicable␣MITRE␣
       ATT&CK␣technique␣ID(s),␣separated␣by␣commas."
)
return (
    f"{bos}\n"
    "###␣Instruction:\n"
    f"{instruction}\n\n"
    "###␣Log:\n"
    f"{log_text}\n\n"
    "###␣Response:\n"
)

predictions = []
for example in test_data:
    prompt = create_inference_prompt(example["text1"])
    inputs = tokenizer(prompt,
       return_tensors="pt").to(model.device)
    output = model.generate(
        **inputs,
        max_new_tokens=32,
        do_sample=True,
        num_beams=5,
        early_stopping=True,
    )
    decoded = tokenizer.decode(output[0], skip_special_tokens=True)
    predictions.append(decoded)
```

The predictions are then processed to extract the predicted ATT&CK IDs, which are compared against the ground truth labels to compute evaluation metrics such as precision, recall, F1-score, Hamming loss, and Exact Match.
The code below shows how to process and compute these metrics:

Listing A.6: Processing predictions and computing evaluation metrics

```
all_techniques = sorted({tech for labs in true_labels_list for
   tech in labs}) # Unique sorted techniques
tech2idx = {tech: idx for idx, tech in enumerate(all_techniques)}
   # Technique to index mapping
```

```
def to_binary_vector(label_list, mapping): # Convert list of
    labels to binary vector
    vec = [0] * len(mapping)
    for lbl in label_list:
        if lbl in mapping:
            vec[mapping[lbl]] = 1
    return vec

y_true = [to_binary_vector(labs, tech2idx) for labs in
    true_labels_list] # True labels
y_pred = [to_binary_vector(labs, tech2idx) for labs in responses]
    # Predicted labels

prec_micro = precision_score(y_true, y_pred, average='micro',
    zero_division=0) # Micro precision
rec_micro = recall_score(y_true, y_pred, average='micro',
    zero_division=0) # Micro recall
#ETC ...
```

The final evaluation metrics on the test set are summarized in Table A.4.

Table A.4: Test set evaluation metrics

Metric	**Micro**	**Macro**	**Global**
Precision	0.6259	0.3578	-
Recall	0.5314	0.2923	-
F1-score	0.5748	0.2985	-
Hamming loss	-	-	0.0026
Exact match	-	-	0.5808

The current results validate our approach and provide a functional baseline for pipeline integration. The 58% exact match rate demonstrates the model's capability to correctly identify all relevant techniques in a majority of cases, which is acceptable for initial system integration and testing.
The primary limitation remains the model's performance on rare techniques (macro-F1 0.30), which we attribute to the limited training data available for these specific techniques, class imbalance in the training data, and the inherent complexity of multi-label classification tasks.

Appendix B

Set-up lab environment

The experimental phase focuses on the evaluation of large language models (LLMs) for detecting MITRE ATT&CK techniques based on Windows system logs. The primary goal is to assess whether LLMs, given a concise behavioral summary, are capable of inferring the correct TTP(s) associated with a specific adversarial activity.

Experiments were performed on both fine-tuned and general-purpose models:

- **Mistral 7B (fine-tuned)** - A custom-trained model based on the Security-TTP-Mapping dataset provided by Tumeteor [37].
- **GPT-4o** - a general-purpose model accessed via OpenAI chatGPT [38].

Experiments were conducted in a controlled environment using a Windows 10 virtual machine under Hyper-V. I have choosen to use the Hyper-V platform because is a native solution for Windows system and ease of snapshot management, which allows for quick resets between tests. The virtual machine was configured with the following specifications:

- **OS**: Windows 10 Pro
- **CPU**: 2 virtual cores
- **RAM**: 6 GB
- **Disk Space**: 100 GB
- **Network**: Default virtual switch for internet access

Each test was executed from the same system snapshot to minimize noise and ensure isolated behavior per TTP test. Everytime a test was executed the VM was reset to a clean state, ensuring no residual effects from previous tests.

B.1 Sysmon and Log capture configuration set-up

The Sysmon service [35] was installed and configured to capture detailed system activity. The configuration default file provided by [34] was used as a baseline. This configuration captures a wide range of events, including process creation, network connections, file modifications, and registry changes. To install and configure Sysmon:

Listing B.1: Sysmon installation and configuration

```
Sysmon64.exe -accepteula -i sysmonconfig.xml
# To update configuration later
Sysmon64.exe -c sysmonconfig.xml
# To uninstall Sysmon
Sysmon64.exe -u
```

A Powershell script was developed to automate the extraction of relevant logs from the Windows Event Viewer. The script starts to capture logs before executing the adversarial technique and stops immediately after the test completes. Below is the script used for log capture:

Listing B.2: PowerShell script for log extraction

```
param(
        [string]$OutputDir = ".\captured_logs"
    )

    if (-not (Test-Path -Path $OutputDir)) {
        New-Item -ItemType Directory -Path $OutputDir -Force | Out-Null
        Write-Host "Created␣output␣folder:␣$OutputDir"
    }

    Write-Host "Preparing␣to␣capture␣events.␣Press␣ENTER␣to␣start␣
        recording..."
    Read-Host

    $startTime = Get-Date
    Write-Host "`n[+]␣Capture␣started:␣$startTime"
    Write-Host "Run␣your␣Atomic␣Red␣Team␣tests.␣Press␣ENTER␣when␣
        you're␣done..."
    Read-Host
    $endTime = (Get-Date).AddSeconds(5)
    Write-Host "[+]␣Capture␣ended␣(with␣buffer):␣$endTime`n"

    $logs = @(
        "Microsoft-Windows-Sysmon/Operational",
        "Security",
        "System",
        "Application",
        "Windows␣PowerShell",
        "Microsoft-Windows-WMI-Activity/Operational",
        "Microsoft-Windows-TaskScheduler/Operational"
    )

    foreach ($log in $logs) {
        Write-Host "Extracting␣events␣from:␣$log"

        if (-not (Get-WinEvent -ListLog $log -ErrorAction
            SilentlyContinue)) {
            Write-Warning "Log␣$log␣not␣found␣or␣disabled"
            continue
```

```
}

try {
    $events = Get-WinEvent -FilterHashtable @{
        LogName = $log
        StartTime = $startTime
        EndTime = $endTime
    } -ErrorAction Stop

    $eventCount = if ($events) { $events.Count } else { 0 }
    Write-Host "$eventCount␣events␣found"

    if ($eventCount -gt 0) {
        $safeLogName = $log -replace '[^a-zA-Z0-9\-]', '_'
        $outputFile = Join-Path -Path $OutputDir -ChildPath
            "${safeLogName}_events.xml"

        $xmlWriterSettings = [System.Xml.XmlWriterSettings]@{
            Indent = $true
            IndentChars = "␣␣"
            Encoding = [System.Text.Encoding]::UTF8
        }

        $xmlWriter =
            [System.Xml.XmlWriter]::Create($outputFile, $
            xmlWriterSettings)
        $xmlWriter.WriteStartDocument()
        $xmlWriter.WriteStartElement("Events")
        $xmlWriter.WriteAttributeString("LogName", $log)
        $xmlWriter.WriteAttributeString("StartTime", $
            startTime.ToString("o"))
        $xmlWriter.WriteAttributeString("EndTime", $
            endTime.ToString("o"))
        $xmlWriter.WriteAttributeString("Count", $eventCount)

        foreach ($event in $events) {
            $eventXml = $event.ToXml()
            $xmlFragment = [System.Xml.XmlDocument]::new()
            $xmlFragment.LoadXml($eventXml)
            $xmlFragment.DocumentElement.WriteTo($xmlWriter)
        }

        $xmlWriter.WriteEndElement()
        $xmlWriter.WriteEndDocument()
        $xmlWriter.Flush()
        $xmlWriter.Close()

        Write-Host "All␣events␣exported␣to:␣$outputFile"
    }
}
catch {
```

```
            if ($_.Exception -match "No␣events␣were␣found") {
                Write-Host "No␣events␣found␣in␣the␣specified␣time␣
                    period"
            } else {
                Write-Warning "Error␣during␣extraction:␣$
                    ($_.Exception.Message)"
            }
        }
    }

    Write-Host "All␣logs␣exported␣to:␣$OutputDir"
```

This script captures events from multiple Windows logs, including Sysmon, Security, System, Application, PowerShell, WMI Activity, and Task Scheduler. It allows the user to specify an output directory for the captured logs, defaults to a folder named `captured_logs` in the current directory. The script prompts the user to start and stop the log capture, ensuring that only relevant events are recorded. The capture is designed to run manually, allowing the user to execute the desired adversarial technique during the capture window. This is important to ensure that the logs contain the events generated by the specific technique being tested. The captured logs are saved in XML format, which is suitable for further processing and analysis.

After, the captured events are stored in XML files within the TTP specific folder. It has the following structure:

```
$HOME
|- win-logs
    |- T1082
        |- Microsoft-Windows-Sysmon_Operational_events.xml
        |- Microsoft-Windows-WMI-Activity_Operational_events.xml
        ...
    |- T1083
    ...
```

The win-logs folder contains subfolders for each ATT&CK technique (TTP) tested (e.g. T1082, T1083, etc...). This folder is then shared with the host machine so that the log processing and analysis can be performed outside the VM environment. The XML files contain detailed information about each event, including timestamps, event IDs, process names, command lines, and other relevant metadata, which are essential for the analyzing system behavior and identifying the techniques used.

B.2 Invoke-AtomicRedTeam set-up and execution

The Atomic Red Team framework [39] was used to execute the adversarial techniques. The installation was done via Powershell using the following commands:

Listing B.3: Invoke-AtomicRedTeam installation

```
Install-Module -Name invoke-atomicredteam,powershell-yaml -Scope
    CurrentUser
```

If you get an error about the repository being untrusted, you can run:

```
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process
```

This command temporarily sets the execution policy to bypass for the current PowerShell session, allowing you to install the module without changing the system-wide policy. Of course this operation brings security risks, so it should be done only in a controlled environment, it remands to the chapter 5 for more details about security considerations.

Once installed, it needs to download the Atomic Red Team repository (this is where the atomic tests are defined), the default location is `C:\AtomicRedTeam\Atomics`, but you can specify a different path.

Listing B.4: Downloading the Atomic Red Team repository

```
IEX (IWR 'https://raw.githubusercontent.com/redcanaryco/
    invoke-atomicredteam/master/install-atomicsfolder.ps1' -
    UseBasicParsing);
Install-AtomicsFolder
```

Now that the framework is installed, you can execute a specific atomic test using the `Invoke-AtomicTest` command. A good starting point is to list all available techniques IDs and test names available for execution. The -ShowDetailsBrief flag provides a concise overview of each test given its technique ID.

Listing B.5: Listing available Atomic Red Team techniques

```
Invoke-AtomicTest T1082 -ShowDetailsBrief
```

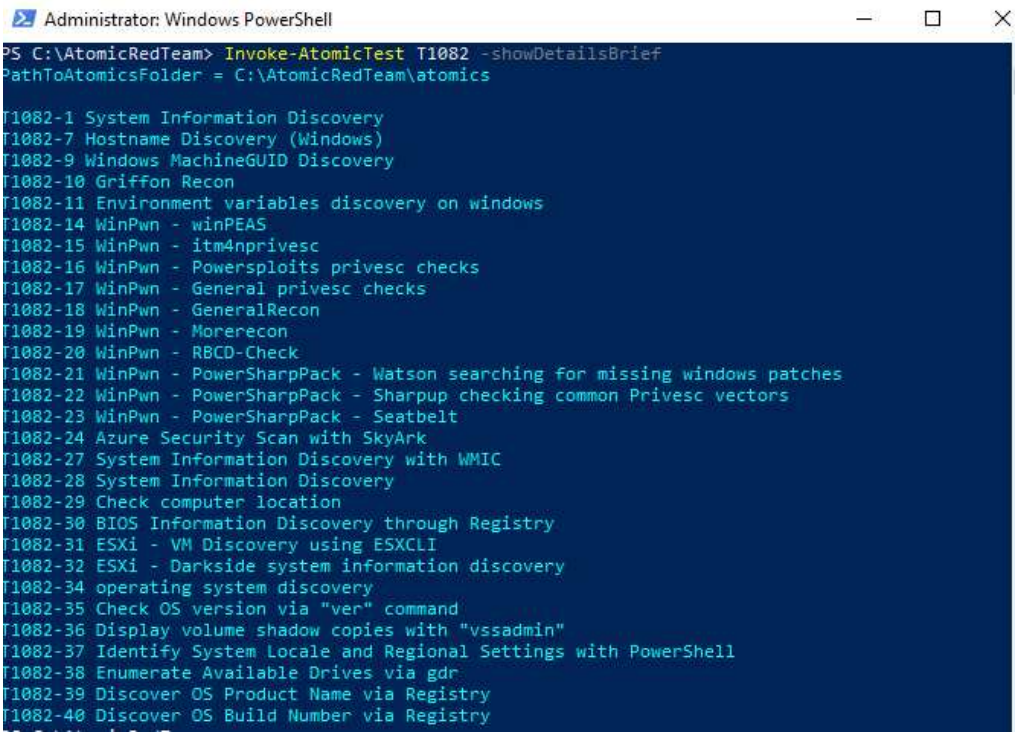

Figure B.1: Listing available Atomic Red Team test for technique T1082

Each atomic test definition specifies whether the test is intended to be run on Windows, Linux, or macOS. However, there may be finer-grained requirements for an atomic test.

For example, the test may be intended for execution on a Domain Controller or Server rather than on a Workstation. Other requirements (also called prerequisites) may include the existence of certain files or users, as well as the installation of specific tools. Note: Since a Domain Controller (e.g., Active Directory) was not set up in this environment, tests requiring such components were not executed.

To check if the actual system state meets the prerequisities for a specific atomic test, you can use the `-CheckPrereqs` flag. This is a useful step to ensure that the test can be executed successfully without errors due to unmet requirements.

Listing B.6: Checking prerequisites for an Atomic Red Team test

```
Invoke-AtomicTest T1082 -CheckPrereqs
```

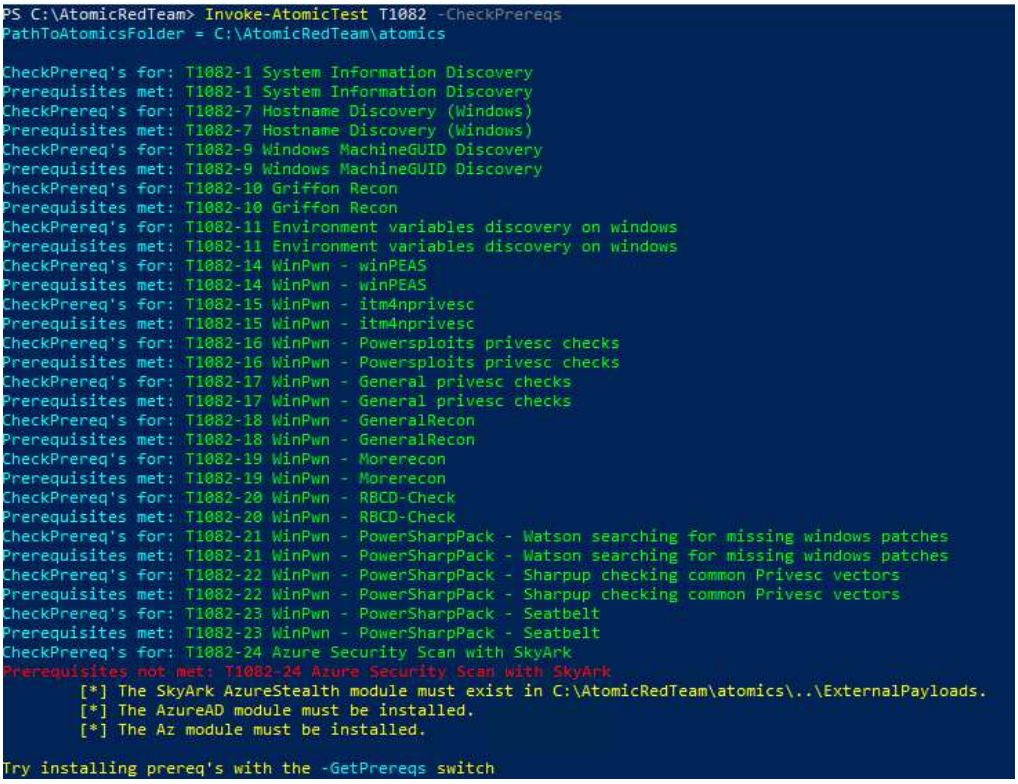

Figure B.2: Checking prerequisites for Atomic Red Team test T1082

If some prerequisites are not met, you can either manually adjust the system to satisfy them or insert the `-GetPrereqs` flag to have the framework attempt to automatically set up the necessary conditions. However, this automatic setup may not always be successful, especially if it requires complex configurations or installations. A full installation guide of the Invoke-AtomicRedTeam can be found at [39].

Executing Atomic Red Team tests Once the prerequisites are satisfied, you can execute the atomic test using the `Invoke-AtomicTest` command. The test can run on local or remote systems, but in this case, we are focusing on local execution.

Listing B.7: Executing an Atomic Red Team test

```
Invoke-AtomicTest T1082 -TestNumbers 1, 7, 9
```

This command executes the specified test numbers 1, 7 and 9 for technique T1082 (System information Discovery). The ouput shows the status of each test, indicating whether it was completed successfully or if there were any errors. For the purpose of this experiment, all tests were executed with the following commands:

Listing B.8: Executing all Atomic Red Team tests for a specific technique

```
PS C:\AtomicRedTeam\win-logs> powershell.exe -ExecutionPolicy
    Bypass -File ".\capture_events.ps1" -OutputDir ".\T1059.001"

PS C:\AtomicRedTeam> Invoke-AtomicTest T1059.001 -LoggingModule
    "Attire-ExecutionLogger" -ExecutionLogPath
    ".\win-logs\T1059.001\T1059.001.json"
```

The first command starts the log capture process generated by the system during the test execution and saves them in the specified output directory (e.g., `.\win-logs\T1059.001`). The second command executes the atomic test for technique T1059.001(Command and Scripting Interpreter: PowerShell) and logs the execution details in a JSON file in the specified path using the `Attire-ExecutionLogger` module. Note: For practical reasons, both commands must be executed in different PowerShell sessions.
The output from the framework provides a json file with the execution details, including the status of each test, any errors encountered, and the commands executed. This information is useful for verifying that the test ran as expected and for troubleshooting any issues that may arise during execution. Here is an example of the output from executing a test:

Listing B.9: Output from executing an Atomic Red Team test


```
{
    "mitre-technique-id": "T1082",
    "procedure-name": "Hostname Discovery (Windows)",
    "procedure-id": {
        "type": "guid",
        "id": "85cfbf23-4a1e-4342-8792-007e004b975f"
    },
    "procedure-description": "Identify system hostname for Windows. Upon
        execution, the hostname of the device will be displayed.\n",
    "order": 7,
    "steps": [
        {
        "order": 1,
        "time-start": "2025-06-17T09:38:21.000Z",
        "time-stop": "2025-06-17T09:38:21.000Z",
        "executor": "command_prompt",
        "command": "hostname\n",
        "process-id": 6480,
        "exit-code": 0,
        "is-timeout": false,
        "output": [
            {
            "content": "DESKTOP-T6SAS9U",
            "level": "STDOUT",
            "type": "console"
            }
        ]
        }
    ]
```

```
},
```


This JSON output provides a detailed record of test execution, including the technique ID, procedure name and description and the test steps with timestamps, commands executed, process IDs, exit codes, and output. In this case, the test successfully executed the `hostname` command, which retrieves the system's hostname as expected.

Another example of a TTP test was executed for T1218.010 (Signed Binary Proxy Execution: Regsvr32).

Listing B.10: Output from executing an Atomic Red Team test


```
{
    "mitre-technique-id": "T1218.010",
    "procedure-name": "Regsvr32 Silent DLL Install Call DllRegisterServer",
    "procedure-id": {
                        "type": "guid",
                        "id": "9d71c492-ea2e-4c08-af16-c6994cdf029f"
                    },
    "procedure-description": "Regsvr32.exe is a command-line program used to
        register and unregister OLE controls. Normally, an install is executed
         with /n to prevent calling DllRegisterServer.",
    "order": 5,
    "steps": [
        {
            "order": 1,
            "time-start": "2025-06-18T14:08:00.000Z",
            "time-stop": "2025-06-18T14:08:05.000Z",
            "executor": "command_prompt",
            "command": "C:\\Windows\\system32\\regsvr32.exe /s /i \"C:\\
                AtomicRedTeam\\atomics\\T1218.010\\bin\\AllTheThingsx86.dll\"\n
                ",
            "process-id": 2860,
            "exit-code": 0,
            "is-timeout": false,
            "output": [
                        {
                            "content": "",
                            "level": "STDOUT",
                            "type": "console"
                        }
            ]
        }
    ]
}
```


This technique is called **Signed Binary Proxy Execution: Regsvr32** and involves using a legitimate signed binary (in this case, `regsvr32.exe`) to execute a potentially malicious DLL. This technique is called **Signed Binary Proxy Execution: Regsvr32**, the regsvr32.exe is a command-line utility in Microsoft Windows for registering and unregistering DLLs and ActiveX controls in the operating system Registry. In this Test the regsvr32.exe [40] command is executed with the /s (silent) and /i (install) options to register a DLL file without displaying any user interface. Here an Attacker can abuse to run malicious DLLs without dropping EXEs (living off the land), Bypass application allowlists and some antivirus solutions, Avoid detection by using legitimate system utilities.

Test coverage A total of **70 MITRE ATT&CK TTPs** were tested. Each technique was executed individually with all the tests for which the requirements were satisfied, starting from a clean checkpoint. Events were collected via PowerShell scripts and stored in XML format.

The table B.1 shows the per-TTP detection results for both models under the exact ATT&CK ID match condition, and the table B.2 shows the predictions made by both models for each TTP tested. For the analysis and evaluation discussion, please refer to Chapter 5.

Table B.2: Per-run predictions (plain strings returned by models).

#	ATT&CK TTP	Mistral-7B FT (predicted IDs)	GPT-4o (predicted IDs)
1	T1003.001	T1056, T1056.001	T1003.001, T1112, T1055.012, T1547.009
2	T1005	T1074.001	T1005, T1074.001, T1059.001
3	T1007	T1059.003	T1059.001, T1087.002, T1057, T1033, T1112, T1547.001
4	T1010	T1112	T1059, T1204.002, T1547.001, T1112, T1055.001
5	T1012	T1059.001	T1086, T1012, T1112, T1546.003
6	T1016	T1046	T1086, T1046, T1083, T1057
7	T1021.001	T1562.004	T1112, T1562.004, T1059.001, T1021.001
8	T1021.006	T1059.001	T1059.001, T1021.006, T1087, T1078, T1550.002
9	T1033	T1033	T1059.001, T1033, T1087.001, T1087.002
10	T1036.003	T1036.005	T1036.003, T1059.001, T1204.002
11	T1036.004	T1546.013, T1059.003	T1050, T1059.003, T1547.010
12	T1036.005	T1059.001	T1059.001, T1027, T1055.001, T1547.001
13	T1047	T1055.004, T1055.003	T1033, T1047, T1059.001, T1055.001, T1078, T1547.001
14	T1049	T1059.001	T1086, T1059.001, T1087.002, T1018
15	T1053.005	T1047, T1069.001	T1059.001, T1059.005, T1087.002, T1220, T1047
16	T1055.001	T1059.001	T1059.001, T1105, T1055.001, T1055.002, T1543.003
17	T1055.002	T1055	T1059.001, T1055, T1055.001, T1055.012
18	T1055.003	T1059.001	T1059.001, T1055
19	T1055.012	T1055.012	T1055, T1055, T1036.004, T1204.002, T1548.002
20	T1056.001	T1059.001	T1056.001, T1059.001, T1548.001
21	T1057	T1059.001	T1059.001, T1068, T1112, T1547.006, T1543.003

#	ATT&CK TTP	Mistral-7B FT (predicted IDs)	GPT-4o (predicted IDs)
22	T1059.001	T1059.001	T1059.001, T1003.001, T1105, T1087.002
23	T1059.003	T1059.003	T1059.001, T1059.005, T1202, T1036, T1566.002
24	T1059.005	T1059.003	T1059.001, T1059.005, T1059.003, T1070.004
25	T1059.007	T1059.001	T1059.001, T1082, T1204.002
26	T1070.001	T1070.001	T1070.001, T1086
27	T1070.004	T1070.004	T1070.004, T1070.006, T1059.003, T1059.001
28	T1070.006	T1070.006	T1070.006, T1070.009, T1112
29	T1071.001	T1071.001	T1059.001, T1059.003, T1071.001, T1055.013, T1005
30	T1082	T1059.001	T1087.001, T1012, T1059.001
31	T1083	T1059.003	T1059.003, T1059.001, T1021.004, T1083, T1057, T1036.005
32	T1110.004	T1059.001	T1110.003, T1552.001, T1059.001
33	T1112	T1112	T1112, T1059.001, T1547.001, T1021.001
34	T1113	T1113	T1059.001, T1113, T1112, T1562.001
35	T1120	T1059.001	T1059.001, T1105, T1218.011, T1086
36	T1127.001	T1220	T1127.001, T1059.005, T1218.005
37	T1134.001	T1543.003	T1543.003, T1059, T1021.002, T1570
38	T1135	T1059.001	T1087.002, T1135, T1046, T1059.001
39	T1140	T1140	T1105, T1140, T1218.011
40	T1202	T1059.001	T1059.001, T1559.001, T1218.010, T1071.001
41	T1204.002	T1059.001	T1059.001, T1566.001, T1204.002, T1112, T1055, T1105, T1027, T1071.001
42	T1218.003	T1071.001	T1036.005, T1071.001, T1112, T1547.001
43	T1218.005	T1059.001	T1086, T1059.001, T1136.001, T1546.003, T1053.005
44	T1218.010	T1218.010	T1218.010, T1036.005, T1059.003
45	T1218.011	T1036.005	T1036.003, T1059.003, T1218.011, T1204.002
46	T1220	T1059.003	T1047, T1059.001, T1218.011, T1112, T1082, T1562.001
47	T1222.001	T1222.002	T1222.001, T1564.001, T1070.004
48	T1518	T1059.001	T1059.001, T1105, T1074.002
49	T1543.003	T1543.003	T1050, T1543.003, T1574.002

#	ATT&CK TTP	Mistral-7B FT (predicted IDs)	GPT-4o (predicted IDs)
50	T1546.001	T1546.013, T1059.001	T1059.001, T1546.001, T1562.001
51	T1546.003	T1546.003	T1053.005, T1047, T1059.001, T1546.003, T1220
52	T1546.008	T1546.008	T1546.008, T1055.002, T1547.001
53	T1546.012	T1546.013, T1546.014	T1059.001, T1059.003, T1112, T1546.012
54	T1547.001	T1112	T1547.001, T1546.001, T1055.001
55	T1547.004	T1546.015	T1546.010, T1546.011, T1059.001
56	T1547.008	T1112	T1059.001, T1547.001, T1112
57	T1547.009	T1059.003, T1547.009	T1059.001, T1547.009, T1204.002
58	T1557.001	T1546.013, T1059.001	T1036.005, T1059.001, T1071.001, T1105, T1112, T1547.001
59	T1560.001	T1059.001	T1059.001, T1027, T1560.001, T1074.001, T1486
60	T1562.001	T1562.001	T1562.001, T1562.004, T1059.001
61	T1562.002	T1546.013, T1059.001	T1059.001, T1105, T1562.006, T1027, T1021.001, T1055
62	T1562.003	T1059.001	T1562.001, T1059.001, T1105
63	T1562.004	T1569.002	T1059.001, T1021.004, T1562.004, T1112, T1053.003
64	T1562.006	T1562.001	T1562.001, T1562.006, T1070.006, T1569.002
65	T1564.001	T1564.001	T1059.001, T1112, T1564.001, T1547.001
66	T1564.002	T1204.001	T1059.001, T1059.003, T1087.001, T1136.001, T1112, T1547.009, T1078
67	T1564.003	T1059.003	T1059.001, T1059.003, T1218.005, T1071.001, T1106
68	T1566.001	T1059.003, T1105	T1566.001, T1059.001, T1105, T1204.002
69	T1566.002	T1059.003	T1059.001, T1055.001, T1121, T1204.001
70	T1574.001	T1543.003	T1059, T1547.001, T1543.003, T1543.002, T1055

Table B.1: Per-TTP detection under exact ATT&CK ID match. A checkmark indicates the model returned the ground-truth ID for that run.

#	ATT&CK TTP	Mistral-7B FT	GPT-4o	#	ATT&CK TTP	Mistral-7B FT	GPT-4o
1	T1003.001	–	✓	36	T1127.001	–	✓
2	T1005	–	✓	37	T1134.001	–	–
3	T1007	–	–	38	T1135	–	✓
4	T1010	–	–	39	T1140	✓	✓
5	T1012	–	✓	40	T1202	–	–
6	T1016	–	–	41	T1204.002	–	✓
7	T1021.001	–	✓	42	T1218.003	–	–
8	T1021.006	–	✓	43	T1218.005	–	–
9	T1033	✓	✓	44	T1218.010	✓	✓
10	T1036.003	–	✓	45	T1218.011	–	✓
11	T1036.004	–	–	46	T1220	–	–
12	T1036.005	–	–	47	T1222.001	–	✓
13	T1047	–	✓	48	T1518	–	–
14	T1049	–	–	49	T1543.003	✓	✓
15	T1053.005	–	–	50	T1546.001	–	✓
16	T1055.001	–	✓	51	T1546.003	✓	✓
17	T1055.002	–	–	52	T1546.008	✓	✓
18	T1055.003	–	–	53	T1546.012	–	✓
19	T1055.012	✓	–	54	T1547.001	–	✓
20	T1056.001	–	✓	55	T1547.004	–	–
21	T1057	–	–	56	T1547.008	–	–
22	T1059.001	✓	✓	57	T1547.009	✓	✓
23	T1059.003	✓	–	58	T1557.001	–	–
24	T1059.005	–	✓	59	T1560.001	–	✓
25	T1059.007	–	–	60	T1562.001	✓	✓
26	T1070.001	✓	✓	61	T1562.002	–	–
27	T1070.004	✓	✓	62	T1562.003	–	–
28	T1070.006	✓	✓	63	T1562.004	–	✓
29	T1071.001	✓	✓	64	T1562.006	–	✓
30	T1082	–	–	65	T1564.001	✓	✓
31	T1083	–	✓	66	T1564.002	–	–
32	T1110.004	–	–	67	T1564.003	–	–
33	T1112	✓	✓	68	T1566.001	–	✓
34	T1113	✓	✓	69	T1566.002	–	–
35	T1120	–	–	70	T1574.001	–	–

Bibliography

[1] "The essential role of SIEM in modern cybersecurity strategies," Jul. 2024. [Online]. Available: https://www.nomios.com/news-blog/the-essential-role-of-siem/

[2] gmcdouga, "Check Point Research Reports Highest Increase of Global Cyber Attacks seen in last two years – a 30% Increase in Q2 2024 Global Cyber Attacks," Jul. 2024. [Online]. Available: https://blog.checkpoint.com/research/check-point-research-reports-highest-increase-of-global-cyber-attacks-seen-in-last-two-years-a-30-increase-in-q2-2024-global-cyber-attacks/

[3] P. Tseng, Z. Yeh, X. Dai, and P. Liu, "Using LLMs to Automate Threat Intelligence Analysis Workflows in Security Operation Centers," Jul. 2024, arXiv:2407.13093 [cs]. [Online]. Available: http://arxiv.org/abs/2407.13093

[4] "What Is SIEM? | Microsoft Security." [Online]. Available: https://www.microsoft.com/en/security/business/security-101/what-is-siem

[5] R. Samson, "SIEM Best Practices for 2025: Ensuring Optimal Security Operations," Jun. 2025. [Online]. Available: https://www.clearnetwork.com/siem-best-practices-for-2025/

[6] "What Is NLP (Natural Language Processing)? | IBM," Aug. 2024. [Online]. Available: https://www.ibm.com/think/topics/natural-language-processing

[7] "What is Generative AI?" [Online]. Available: https://www.nvidia.com/en-us/glossary/generative-ai/

[8] "What Are Large Language Models (LLMs)? | IBM," Nov. 2023. [Online]. Available: https://www.ibm.com/think/topics/large-language-models

[9] iffort_admin, "GPT-3 vs GPT-3.5: Key Differences and Applications," Mar. 2023. [Online]. Available: https://www.iffort.com/blog/2023/03/31/gpt-3-vs-gpt-3-5/

[10] "GPT-4," Jan. 2024. [Online]. Available: https://openai.com/index/gpt-4-research/

[11] "Mistral 7B | Mistral AI." [Online]. Available: https://mistral.ai/news/announcing-mistral-7b

[12] P. Khadka, "Mistral 7B explained !" Dec. 2024. [Online]. Available: https://pub.towardsai.net/mistral-7b-explained-53720dceb81e

[13] "Prompt Engineering Method to Reduce AI Hallucinations - Kata.ai's Blog!" Oct. 2024, section: Prompt Engineering. [Online]. Available: https://kata.ai/blog/prompt-engineering-method-to-reduce-ai-hallucinations/

[14] "Few-Shot Prompting – Nextra," Jun. 2025. [Online]. Available: https://www.promptingguide.ai/techniques/fewshot

[15] B. Strom, "ATT&CK 101," Feb. 2024. [Online]. Available: https://medium.com/mitre-attack/att-ck-101-17074d3bc62

[16] "What Is Virtualization? | IBM," Apr. 2025. [Online]. Available: https://www.ibm.com/think/topics/virtualization

[17] S. M. T. I. Tonmoy, S. M. M. Zaman, V. Jain, A. Rani, V. Rawte, A. Chadha, and A. Das, "A Comprehensive Survey of Hallucination Mitigation Techniques

in Large Language Models," Jan. 2024, arXiv:2401.01313. [Online]. Available: http://arxiv.org/abs/2401.01313

[18] F. Cutitta, "How prompt engineering is reducing genAI hallucinations," Jul. 2024. [Online]. Available: https://dhinsights.org/blog/how-prompt-engineering-is-reducing-genai-hallucinations/

[19] H. Kang, J. Ni, and H. Yao, "Ever: Mitigating Hallucination in Large Language Models through Real-Time Verification and Rectification," Feb. 2024, arXiv:2311.09114 [cs]. [Online]. Available: http://arxiv.org/abs/2311.09114

[20] "Reducing hallucinations in large language models with custom intervention using Amazon Bedrock Agents | Artificial Intelligence," Nov. 2024, section: Amazon Bedrock. [Online]. Available: https://aws.amazon.com/blogs/machine-learning/reducing-hallucinations-in-large-language-models-with-custom-intervention-using-amazon-bedrock-agents/

[21] "What is LLM Temperature? - Hopsworks." [Online]. Available: https://www.hopsworks.ai/dictionary/llm-temperature

[22] D. Pantielieiev, "Stop AI Hallucinations: A Developer's Guide to Prompt Engineering," Feb. 2025. [Online]. Available: https://shelf.io/blog/stop-ai-hallucinations-a-developers-guide-to-prompt-engineering/

[23] K. He, "Detect Hallucinations Using LLM Metrics | Fiddler AI Blog." [Online]. Available: https://www.fiddler.ai/blog/detect-hallucinations-using-llm-metrics

[24] G. Michelet and F. Breitinger, "ChatGPT, Llama, can you write my report? An experiment on assisted digital forensics reports written using (Local) Large Language Models," Dec. 2023, arXiv:2312.14607 [cs]. [Online]. Available: http://arxiv.org/abs/2312.14607

[25] Y. Ji, Y. Liu, F. Yao, M. He, S. Tao, X. Zhao, S. Chang, X. Yang, W. Meng, Y. Xie, B. Chen, and H. Yang, "Adapting Large Language Models to Log Analysis with Interpretable Domain Knowledge," Dec. 2024, arXiv:2412.01377 [cs]. [Online]. Available: http://arxiv.org/abs/2412.01377

[26] Y. Liu, S. Tao, W. Meng, J. Wang, W. Ma, Y. Zhao, Y. Chen, H. Yang, Y. Jiang, and X. Chen, "Interpretable Online Log Analysis Using Large Language Models with Prompt Strategies," Jan. 2024, arXiv:2308.07610 [cs]. [Online]. Available: http://arxiv.org/abs/2308.07610

[27] M. Boffa, R. V. Valentim, L. Vassio, D. Giordano, I. Drago, M. Mellia, and Z. B. Houidi, "LogPrécis: Unleashing Language Models for Automated Malicious Log Analysis," *Computers & Security*, vol. 141, p. 103805, Jun. 2024, arXiv:2307.08309 [cs]. [Online]. Available: http://arxiv.org/abs/2307.08309

[28] R. Karanjai, Y. Lu, D. Alsagheer, K. Kasichainula, L. Xu, W. Shi, and S.-H. S. Huang, "LogBabylon: A Unified Framework for Cross-Log File Integration and Analysis," Dec. 2024, arXiv:2412.12364 [cs]. [Online]. Available: http://arxiv.org/abs/2412.12364

[29] B. Al-Sada, A. Sadighian, and G. Oligeri, "MITRE ATT&CK: State of the Art and Way Forward," Aug. 2023, arXiv:2308.14016 [cs]. [Online]. Available: http://arxiv.org/abs/2308.14016

[30] P. N. Wudali, M. Kravchik, E. Malul, P. A. Gandhi, Y. Elovici, and A. Shabtai, "Rule-ATT&CK Mapper (RAM): Mapping SIEM Rules to TTPs Using LLMs," Feb. 2025, arXiv:2502.02337 [cs]. [Online]. Available: http://arxiv.org/abs/2502.02337

[31] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," Mar. 2023, arXiv:2210.03629 [cs]. [Online]. Available: http://arxiv.org/abs/2210.03629

[32] "Fine-Tune Mistral-7B with LoRA A Quickstart Guide | DigitalOcean." [Online]. Available: https://www.digitalocean.com/community/tutorials/mistral-7b-fine-tun

ing
[33] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7B," Oct. 2023, arXiv:2310.06825 [cs]. [Online]. Available: http://arxiv.org/abs/2310.06825
[34] O. Hartong, "olafhartong/sysmon-modular," Aug. 2025, original-date: 2018-01-13T21:20:59Z. [Online]. Available: https://github.com/olafhartong/sysmon-modular
[35] markruss, "Sysmon - Sysinternals." [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon
[36] T. Nguyen, N. Šrndić, and A. Neth, "Noise contrastive estimation-based matching framework for low-resource security attack pattern recognition," in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Mar. 2024.
[37] T. Contributors, "Security-ttp-mapping dataset," 2024. [Online]. Available: https://huggingface.co/datasets/tumeteor/Security-TTP-Mapping
[38] OpenAI, "Gpt-4o technical report," 2024. [Online]. Available: https://openai.com/index/gpt-4o-system-card/
[39] R. Canary, "Invoke-atomicredteam," 2023. [Online]. Available: https://github.com/redcanaryco/atomic-red-team
[40] dknappettmsft, "regsvr32." [Online]. Available: https://learn.microsoft.com/it-it/windows-server/administration/windows-commands/regsvr32