HARESHUNTER: A Threat Hunting Automation Framework using Collaborative LLM Agents

Mohammed Machrouh*, Zakaria Yartaoui*, Mohammed Akallouch, Ismail Berrada

***Abstract*—Security Operations Center teams face growing challenges as cyber threats become increasingly complex, exceeding the capabilities of manual threat hunting. Current automation solutions provide only partial coverage or require expensive specialized systems, leaving organizations without comprehensive and accessible options for proactive threat detection. To address these limitations, we introduce HARESHUNTER, a novel framework that leverages Large Language Models and conversational agents to provide end-to-end automation of the Cyber Threat Hunting cycle through two primary modules: the Knowledge Fusion Module, which centralizes threat intelligence with security telemetry data, and the Threat Profiling & Mitigation Module, which deploys three specialized conversational agents (Hypothesis, Investigation, and Mitigation) to hunt and neutralize threats. Our prototype implements base versions of DeepSeek R1 and Codestral to drive our experiments, achieving an 84% hunt success rate across 26 attack scenarios drawn from a public dataset. HARESHUNTER demonstrates an end-to-end threat hunting automation solution with practical applicability in operationally realistic cybersecurity settings.**

***Index Terms*—Cyber Threat Hunting, Large Language Models, Autonomous agents, Cybersecurity.**

I. INTRODUCTION

THE cyber threat landscape has evolved dramatically in recent years, creating significant challenges for cybersecurity professionals. Organizations implementing digital transformation strategies now face increasingly sophisticated attack methodologies that have fundamentally shifted cybersecurity approaches. This shift has been further accelerated by the rapid spread of Artificial Intelligence (AI) tools, which are now accessible even to non specialists, increasing the variety of potential threats from basic phishing attempts to complex Advanced Persistent Threats (APTs) [1]. These developments have made proactive threat detection essential for any effective security strategy [2].

In this increasingly complex environment, analysts from the Security Operations Center (SOC) must identify cyber threats before they compromise information systems. This need has led to Cyber Threat Hunting (CTH), a proactive security approach that identifies, analyzes, and mitigates risks during the early stages of the attack kill chain [3]. Unlike reactive detection methodologies that rely on automated alerts from security devices such as Security Information and Event Management Systems (SIEM) or Intrusion Detection Systems (IDS), threat hunting follows an iterative approach that involves the formulation and continuous refinement of hypotheses about threats that may have bypassed existing detection mechanisms [4]. This approach is characterized by the working hypothesis of a threat actor's presence based on the correlation of various security events and Indicators of Compromise (IoCs) coming from Cyber Threat Intelligence (CTI) data, and internal alerts. These presumptions drive data collection and analysis, ultimately leading to the identification of existing or new attack Tactics, Techniques, and Procedures (TTP) that enhance future detection capabilities [5], [6].

Although threat hunting offers significant advantages over reactive methods, it heavily depends on analyst expertise to formulate hypotheses while managing overwhelming security alert volumes. This workload often results in cognitive fatigue and increase dwell time [7]. Recognizing these challenges, the industry has moved toward automating this process to enhance SOC capabilities. Commercial platforms (e.g. Splunk[1], IBM QRadar[2], CrowdStrike Falcon[3], and Microsoft Defender[4]) now include threat hunting automation features, beside SOAR automation. However, these solutions typically offer only partial automation at prohibitive costs, requiring investment in complex systems and specialized personnel, creating financial and operational barriers for many security teams.

From a research perspective, numerous studies have explored threat hunting automation [8]–[10], with some focusing specifically on hypothesis generation, which remains the biggest challenge in CTH. [4], [11]–[13]. These works have demonstrated promising results towards full automation, yet most proposed methods still require significant human intervention at certain stages of the process. The emergence of Large Language Models (LLMs) has further opened new opportunities for threat hunting automation by enhancing the contextualization of security events and enabling automatic generation of attack hypotheses. Recent studies have demonstrated that LLMs can effectively reason about monitored environments and generate hunting queries with minimal prompting [14]. Despite these advances, this area of research remains relatively unexplored.

The current state-of-the-art research can be categorized into three main automation strategies: Data-Driven approaches [15]–[19], partially automated methods [13], [20], [21], and LLM-based techniques [14], [22]–[26]. Thus, this paper addresses three fundamental research questions:

- **RQ1**. How have previous approaches automated the

* These authors contributed equally to this work
M.Machrouh, M.Akallouch, I.Berrada are with College of Computing; Z.Yartaoui is with Vanguard Center;
Both part of Mohammed VI Polytechnic University, Ben Guerir, Morocco
first.last@um6p.ma

[1]https://www.splunk.com/en_us/solutions/threat-hunting.html
[2]https://www.ibm.com/products/qradar
[3]https://www.crowdstrike.com/en-us/platform/
[4]https://www.microsoft.com/en-us/security/business/microsoft-defender

threat hunting process, and what limitations remain in the existing literature?
- **RQ2**. How can LLMs be leveraged to address the current gaps in threat hunting automation?
- **RQ3**. How efficient can an LLM-based framework identify and confirm cyber threats in real-world operational settings?

To address these questions, we introduce HARESHUNTER, a novel framework automating threat hunting cycle through LLMs and conversational agents. Our modular architecture integrates the Knowledge Fusion Module (KFM), centralizing threat-related knowledge and telemetry data, and the Threat Profiling & Mitigation Module (TP2M), managing key CTH phases through three autonomous agents: Hypothesis Agent for establishing hunting hypotheses, Investigation Agent for conducting playbook-based investigations, and Mitigation Agent for neutralizing threats and generating detection rules.

We implemented a prototype using collaborative LLM agents powered by Deepseek-R1 [27] and Codestral [28]. Evaluated on a public hunting dataset, HARESHUNTER achieved 84% success rate across 26 attack scenarios, demonstrating practical LLM-based automation applicability in operational cybersecurity. In summary, the main contributions of the paper are the following:

- We provide a comprehensive survey on the state-of-the-art methods and transition strategies for automating threat hunting, considering both partial and full automation approaches. This overview is further extended to include recent advances in LLMs and Generative AI applied to threat hunting.
- We introduce HARESHUNTER, a novel agent-based framework that achieves adaptive, end-to-end threat hunting automation. Our core architectural contributions include a **dynamic playbook generation** that creates investigation plans for novel threats, and a **closed-loop hypothesis refinement mechanism** that enables the framework to iteratively refine the hypothesis and self-correct its investigative path.
- We conduct a rigorous, two-part experimental evaluation to demonstrate the framework's effectiveness and resilience. We first establish a baseline performance on a variety of attack scenarios, achieving high success rate. We then conduct a resilience test using distorted initial hypotheses, showing that the framework successfully converges on the correct threat in nearly all cases.

The rest of this paper is structured as follows: Section II provides a discussion of related work in threat hunting and LLMs, examining current automation strategies. Section III presents an overview of the threat hunting process, its various phases, frameworks, approaches, and the challenges of automating it. Section IV introduces the HARESHUNTER framework, detailing its architecture, components, and workflow. Section V describes our implementation of HARESHUNTER through an illustrative example, while Section VI presents the evaluation experiments, and results. Finally, Section VII concludes with a summary of contributions and directions for future work. This paper uses various technical abbreviations and acronyms, which are compiled in Table I for easy reference.

II. Related Work

This section emphasizes recent studies focused on automating the CTH process, including emerging methods that take advantage of generative AI. Our state-of-the-art review (**RQ1**) revealed several automation-oriented strategies that can be grouped into three main categories, and summarized in Figure 1: (A) Data-Driven and Knowledge-Centric approaches, (B) Approaches that offer partial automation, and (C) more recent methods that use LLMs to partially or fully automate the CTH cycle. This section describes and critically analyzes the approaches within each category.

A. Data-Driven and Knowledge-Centric Approaches

Early CTH automation research focus on data-driven analytics with knowledge-based reasoning to generate testable hypotheses and detect threats automatically. Some approaches applied machine learning to large network data volumes for intrusion detection [16], while others implemented knowledge-centric frameworks using knowledge graphs to correlate IoCs, CTI feeds, and security alerts, providing comprehensive TTP insights particularly for APT attacks [29] [10] [26] [19]. Researchers also explored CTH automation in specific environments like ICS or SCADA systems [30]. Collectively, this body of work forms the foundation for automating threat hunting.

- **Traditional ML-based Threat Hunting Approaches**
  Many approaches rely on traditional machine learning and anomaly detection algorithms to automate threat hunting. Chen et al. [16] outline an end-to-end pipeline that ingests log data, trains ML classifiers, and surfaces suspicious activity for inspection. Villarreal-Vasquez et al. [15] employ LSTM models to detect anomalies in user behavior sequences, reducing manual triage load. Threat Trekker [31] demonstrates how ML-based pipelines can correlate large volumes of IoCs at scale. Although these solutions highlight ML-driven hunting effectiveness, they typically require substantial tuning for emerging attack vectors and often fail to capture deeper context or semantic meaning behind complex threats. Furthermore, ML-driven approaches often lack explainability, a crucial aspect in cybersecurity. While many efforts focus on explainable AI in cybersecurity [32], none target threat hunting specifically, to our knowledge.
- **Knowledge and Evidence-Based Systems**
  Another class of approaches focuses on hunting based on different forms of structured knowledge and reasoning techniques. ATHAFI [29] employs knowledge graphs to pivot across interconnected entities, such as hosts, processes, and IoCs, thus speeding up forensics investigations. Araujo et al. [33] and OntoHunt [26] push this further by infusing logic-based reasoning or ontologies, moving toward a fully automated mapping of adversarial behaviors. Similarly, ThreatScout [10] applies machine reasoning to orchestrate the hunting process from end to end, bridging data ingestion, correlation,

TABLE I
LIST OF ABBREVIATIONS AND ACRONYMS

Abbreviation	Full Form
AI	Artificial Intelligence
APT	Advanced Persistent Threat
ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge
C2	Command and Control
CTH	Cyber Threat Hunting
CTI	Cyber Threat Intelligence
CVE	Common Vulnerabilities and Exposures
EDR	Endpoint Detection and Response
ELK	Elasticsearch, Logstash, and Kibana
ICS	Industrial Control System
IDS	Intrusion Detection System
IoA	Indicator of Attack
IoC	Indicator of Compromise
ITDC	Internal Threat Data Collector
KFM	Knowledge Fusion Module
LLM	Large Language Model
LSTM	Long Short Term Memory

Abbreviation	Full Form
MISP	Malware Information Sharing Platform
ML	Machine Learning
NLP	Natural Language Processing
OSINT	Open-Source Intelligence
PLC	Programmable Logic Controller
RAG	Retrieval-Augmented Generation
SCADA	Supervisory Control and Data Acquisition
SD	Software Defined
SFT	Supervised Fine-Tuning
SIEM	Security Information and Event Management
SOC	Security Operations Center
TP2M	Threat Profiling and Mitigation Module
TIA	Threat Intelligence Aggregator
TTP	Tactics, Techniques, and Procedures
UEBA	User and Entity Behavior Analytics
XAI	Explainable Artificial Intelligence
XDR	Extended Detection and Response

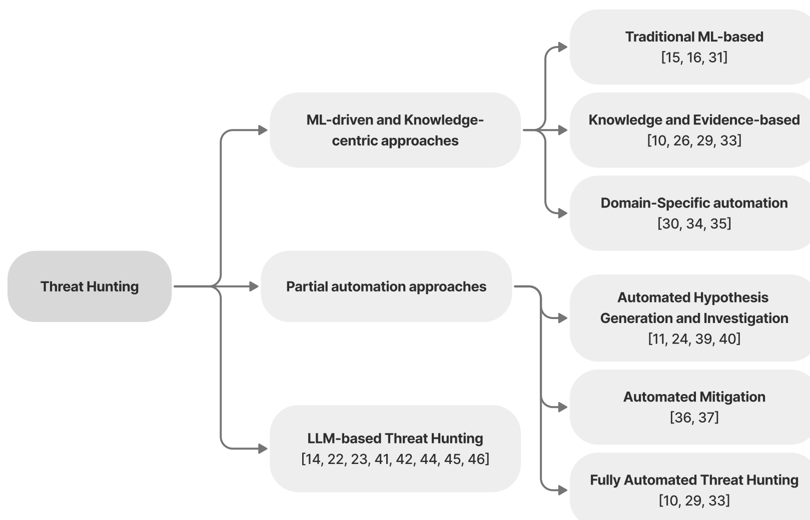

Fig. 1. Classification of Related Work by Category

and classification of events. Although these systems excel at structured analysis and interpretability, they can be constrained by solid knowledge engineering requirements or domain specificity.

- **Domain-Specific Automation**
Several studies have designed automated threat hunting frameworks for specific domains such as ICS, IoT, and cloud environments. Arafune et al. [30] focus on ICS communication protocols and safety demands, presenting a framework that automates hypothesis generation for PLC and SCADA environments. Kumar et al. [34] introduced a Deep Learning-based Threat Hunting Framework (DLTHF) for Software Defined-IoT architectures, integrating a Multi-head Self-attention Bidirectional GRNN-based system to identify binary and multi-vector attacks. Extending into telecommunications, Attieh et al. [35] propose a hunting methodology tailored to 5G networks, leveraging the MITRE FiGHT framework to map adversarial tactics on a Future Communications Testbed. Their approach integrates scenario-driven assessments, adversary emulation, and log analysis, thereby enhancing resilience against evolving 5G and emerging 6G cyber threats. These domain-specific frameworks demonstrate effective hunting in non-traditional IT environments; however, each solution is highly specialized and requires considerable adaptation for broader contexts.

B. Partial Automation Approaches

Another segment of research seeks to partially automate certain phases of the CTH cycle, namely, generating attack hypotheses, investigating potential compromises or mitigating threats. Automating these individual components substantially reduces the manual effort required from analysts. Meanwhile, other works focus on automating mitigation actions by, for instance, isolating infected hosts, blocking malicious domains, or updating firewall rules in near real-time [36] [37] [38]. These efforts represent a middle ground between purely manual hunting and fully autonomous solutions.

- **Automated Hypothesis Generation and Investigation**

Several contributions focus on automating attack hypothesis generation and validation. Alsaheel et al. [39] combine machine learning, NLP, and graph analysis to automatically connect related alerts and reconstruct attack narratives from system logs. Karuna et al. [24] employ NLP techniques to generate initial hunting queries and then apply evolutionary algorithms to iteratively mutate those queries, uncovering hidden attack traces that analysts might miss. AUTOMA [11] systematically enumerates possible adversary scenarios through knowledge discovery, significantly reducing reliance on analyst intuition for hypothesis generation. Complementing these, Threatify [40] leverages graph-based machine learning to generate APT threat variants, enabling the automated derivation of plausible adversarial scenarios by exploring variations within threat intelligence graphs. Together, these solutions demonstrate partial automation across multiple distinct phases of the CTH process, though each typically addresses only a single phase—highlighting the opportunity for more integrated end-to-end automation in future work.

- **Automated Mitigation**
  A complementary thread of research targets automated threat mitigation through defensive rule generation. LLM-CloudHunter [36], for example, uses a pretrained LLMs to derive detection rules from cloud-based CTI reports, while TTPXHunter [37] uses LLMs to extract crucial TTPs from threat reports for containment and eradication strategies. Each of these approaches introduce automation in threat mitigation but address discrete tasks rather than providing fully integrated hunting workflow. In practice, they would require to be combined with other tools and reviewed by analysts rather than serving as standalone solutions.
- **Fully Automated Threat Hunting Systems**
  A small body of work attempts to automate the entire threat hunting cycle from start to finish. For example, ATHAFI [29] combines knowledge graphs with reasoning engines to enable agile, fully automated threat hunting and forensic analysis. Araujo et al. [33] rely on evidential knowledge and reasoning to unify multiple stages of the process under a single framework. Recent work, ThreatScout [10] integrates machine reasoning to propose an automated and closed-loop solution for threat hunting with minimal human intervention.

C. LLM-based Approaches in Cybersecurity

Major advances in LLMs and generative AI have drawn significant interest from the cybersecurity research community due to their potential in addressing current limitations, notably in understanding unstructured data, improving anomaly detection, and generating actionable insights. Researchers are developing LLM-integrated solutions to enhance cybersecurity from various angles. Some approaches use interactive LLM-based chatbots to streamline tasks [41], [42], while others leverage LLMs for challenges such as event correlation.

LLMCloudHunter [36] and TTPXHunter [37] demonstrate that unstructured data such as threat reports or security logs can be processed by LLMs to produce detection rules or extracted TTPs, though each prototype remains limited to specific tasks. More recent systems incorporate LLMs more comprehensively. MCM-Llama [43] fine-tunes domain-specific LLMs for real-time security event correlation, acting as a smart correlation engine for attack detection. HuntGPT [44] and ChatIDS [45] merge traditional anomaly detection with explainable AI, using LLM-based chatbot interfaces to explain intrusion alerts and enhance analyst trust. Thelma [14] automates threat hunting using dual LLMs to prioritize hypotheses and generate hunting queries, though evaluation revealed significant accuracy limitations from hallucination problems. Further studies have explored multi-agent collaboration with LLMs. Building on the hypothesis-driven paradigm, HuntSmart [46] employs a hybrid approach that combines SecureBERT for TTP extraction from CTI data with an LLM for automated hypothesis generation. Further studies have explored multi-agent collaboration with LLMs. Jin et al. [22] and Song et al. [23] investigate how multiple AI agents can collaborate on complex security problems. Although these efforts represent significant progress, most still require human intervention for tuning or action approval. Achieving fully autonomous closed-loop hunting systems that can reconfigure logic and take definitive actions without human oversight remains an unsolved challenge.

While progress in automating specific threat hunting tasks is notable, a significant gap persists in achieving a fully autonomous, closed-loop cycle leveraging modern AI. The predominant focus on discrete phases (e.g., hypothesis generation, mitigation) and the heavy requirement for human oversight limit the adaptability and scalability of current systems. To address this gap, we propose HARESHUNTER, an end-to-end threat hunting automation framework. HARESHUNTER's primary contribution is its use of collaborative, LLM-based agents within an iterative reasoning framework, designed to enhance all stages of the threat hunting process and significantly reduce the need for manual intervention.

III. Threat Hunting Concepts and Frameworks

A. Threat Hunting Blueprint

Threat hunting is a proactive approach used by SOC analysts to uncover threats that evade existing detections, in contrast to reactive methods that focus on known attacks. It involves forming attack hypotheses and validating them through investigation, supported by data sources such as threat intelligence. The process follows a loop consisting of the following steps:

- **Preparation and Hypothesis.** In this step, SOC analysts formulate a threat hypothesis based on CTI knowledge or observables from telemetry data. This phase includes the definition of the hunt objectives. They rank IoCs with the "Pyramid of Pain" framework [47], focusing on those that force the attacker to expend the most effort if detected, as shown in Figure 2.
- **Data Collection.**: This step involves collecting all data required for the hunt. Logs, network flows, endpoint telemetry, and external threat feeds often arrive in diverse, unstructured formats. Security platforms (e.g., SIEM,

XDR, and UEBA) aggregate these sources into a unified repository, normalize fields, and time-align events, enabling hunters to analyze large volumes of data more effectively.
- **Investigation.** Threat hunters use the available tools and data collected to search for artifacts and suspicious patterns that would indicate a hidden threat and support the hypothesis.
- **Response and Remediation.** If a threat is uncovered, remediation actions are taken to mitigate it. This may include isolating compromised systems, removing malware, updating security policies, or implementing new detection rules.
- **Documentation.** Finally, the entire process is documented to extract insights and improve future hunting activities, while enriching threat intelligence.

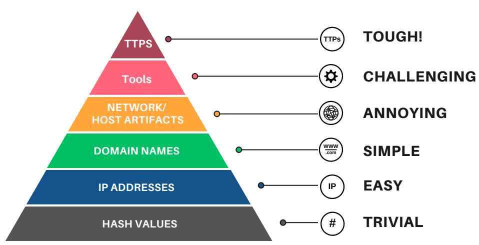

Fig. 2. The Pyramid of Pain

B. Threat Hunting Frameworks

Three important frameworks have emerged to structure CTH methodologies. **The Sqrrl Threat Hunting Reference Model** [48] provides a four-stage iterative approach encompassing Hypothesis Creation, Investigation via Tools and Techniques, Uncovering New TTPs, and Informing and Enriching Analytics. **Targeted Hunting integrating Threat Intelligence (TaHiTI)**[5], developed by financial institutions under the Dutch Payments Association, offers a standardized three-phase methodology covering Initialize, Hunt, and Finalize stages across six sequential steps. **The PEAK framework** [49] introduces a structured "**P**repare, **E**xecute, and **A**ct with **K**nowledge" approach that encompasses foundational concepts, execution processes, and evaluation metrics while distinguishing three hunt categories: Hypothesis-Based, Baseline, and Model-Assisted Threat Hunts.

C. Threat Hunting Challenges & Automation

Threat hunting extends beyond traditional security measures by actively identifying potential threats, yet faces significant challenges [50]. The vast volume and heterogeneity of organizational data complicates normalization and pattern extraction, while evolving cyber threats force analysts to continuously adapt defenses amid overwhelming false positives [20]. Resource constraints and persistent skills gaps compound these difficulties, as effective hunting demands specialized expertise that remains scarce. Consequently, organizations increasingly turn to automation to mitigate these limitations. Modern platforms integrate AI-powered automation to process data streams, normalize events, and identify potential threats, reducing dwell time while freeing experts for complex investigations [12]. Generative AI further enhances capabilities through alert summarization and hunt query creation, though current solutions typically automate only isolated components. Research on LLMs in threat hunting remains limited, representing a promising area for investigation [51].

[5]https://www.betaalvereniging.nl/en/safety/tahiti

IV. HARESHUNTER

This section introduces HARESHUNTER, our novel framework for automating the threat hunting cycle through LLMs and conversational agents. At its core, HARESHUNTER operates on a closed-loop graph of LLM agents, leveraging their reasoning capabilities to autonomously conduct hunts in a collaborative way, without direct human intervention. It addresses the fragmentation and scalability limitations of partially automated systems by providing an end-to-end solution that enhances threat detection and mitigation speed and efficacy. We provide the high-level architecture of HARESHUNTER, including its main components and workflows.

A. Global Architecture

Figure 3 illustrates a comprehensive overview of HARESHUNTER. The architecture integrates two main modules: the Knowledge Fusion Module (KFM) and the Threat Profiling & Mitigation Module (TP2M). Each module serves a distinct function in optimizing the effectiveness of the threat hunting cycle.

1) Knowledge Fusion Module (KFM): KFM operates as a data management system that centralizes threat-related knowledge and telemetry data. Within this module, the Threat Intelligence Aggregator (TIA) collects CTI from diverse feeds, normalizes this information, and stores it in a central database. Complementing this function, the Internal Telemetry Data Collector (ITDC) continuously acquires telemetry data from SIEM, EDR and IDS, and maintains this information in a separate database. The Playbooks Store contains all defined playbooks which represent the investigation flow. The main purpose of this module is to consolidate all required data in a single accessible repository, thereby facilitating efficient data management and appropriate preparation for the hunting cycle.

2) Threat Profiling & Mitigation Module (TP2M): TP2M embodies the main phases of the threat hunting pipeline. Inspired by conventional frameworks (e.g., SQRRL [52], PEAK [49]), it consists of three separate agents that are the key actors in our workflow, each agent has a specialized role within the hunting cycle.

- **Hypothesis Agent.** This agent establishes hunting hypotheses based on observables and IoCs from telemetry data, SIEM alerts, or threat intelligence. It formulates testable hypotheses through analysis of unusual behavior and potential TTPs, clearly articulating expected threat types and where to look for evidence. The agent can be triggered automatically when new anomalous patterns or

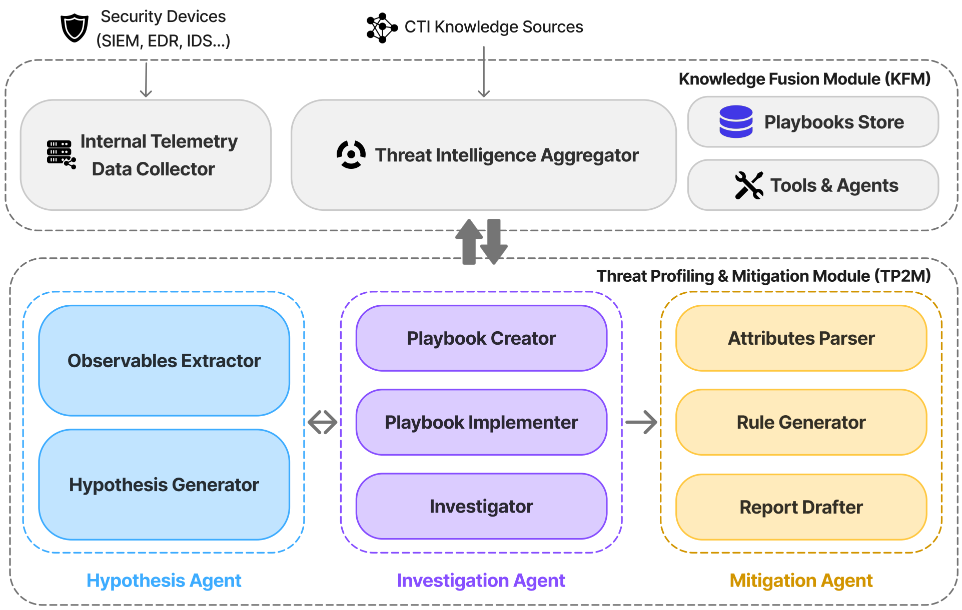

Fig. 3. HARESHUNTER High-level Architecture

intelligence emerge, which allows timely and proactive responses to threats.

- **Investigation Agent.** This agent conducts thorough investigations using playbooks that guide the threat hunting process. Its objective is to search for IoCs and other evidence to validate or disprove the established hypothesis. Once evidence is collected, the agent consolidates the findings to assess the hypothesis validity. A hypothesis is considered valid if sufficient and strong supporting evidence is found. However, a lack of evidence does not necessarily refute the hypothesis. In such cases, the hypothesis is refined based on the available evidence, and the investigation process restarts.
- **Mitigation Agent.** Acting as the final stage, this agent neutralizes confirmed threats through isolation actions such as blocking malicious connections, terminating suspicious processes, and creating new detection rules, depending on the complexity of the threat. The agent also prepares detailed reports summarizing findings and actions taken. These reports are then communicated to the SOC team for further analysis and response if necessary.

B. HARESHUNTER *Components*

1) Graph of agents using CoALA architecture and ReAct paradigm: Agents represent the core component of our framework. An agent is an autonomous actor that uses large language models as their backbone to perform usually complex tasks. In HARESHUNTER, the TP2M module comprises a graph of three sequential agents: Hypothesis, Investigation, and Mitigation. Each specializes in specific threat hunting stages. We used these framework to build our agents, each for its advantages.

- **CoALA Architecture.** HARESHUNTER agents are implemented on top of the CoALA architecture [53], a cognitive framework inspired by cognitive science and symbolic AI. Each agent features working memory (short-term, context-bound) and long-term memory (persistent storage). An agent also has its own action space for internal reasoning and external actions via tool calls.
- **ReAct Paradigm.** HARESHUNTER agents employ the ReAct paradigm for their decision-making [54], iteratively merging reasoning and action. Agents break complex tasks into manageable sequences of thoughts and actions through three main stages: **Thought** (planning the next step), **Action** (executing an operation or tool invocation), and **Observation** (evaluating outcomes). This iterative cycle continues until sufficient information is obtained, concluding with a final response.

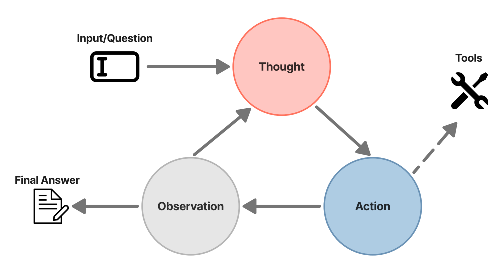

Fig. 4. ReAct Agent Flow

2) LLMs: LLMs serve as the engine of our agents, driving the entire hunting workflow from hypothesis generation to reporting. These large-scale, pretrained transformer models are fine-tuned on extensive datasets, including cybersecurity-specific corpora, which makes them able to handle complex inputs, detect patterns, and produce context-aware outputs. Within HARESHUNTER, we integrated LLMs at multiple stages: hypothesis generation, playbook creation, investigation (to drive our agents), crafting final detection rules, and drafting threat reports.

3) Tools: Tools enhance LLM agent capabilities beyond text generation, providing HARESHUNTER with essential threat hunting functionalities. These tools primarily manage data retrieval from the KFM module and perform IoC verification through platforms such as VirusTotal [6]. Agents interact with the Knowledge Fusion Module through **function calling** [7], enabling interaction with external systems and supplementing context with actual data rather than relying solely on pre-trained knowledge. This tool-augmented approach ensures hunting workflows remain grounded in real data while leveraging language model analytical capabilities. Each tool follows a standardized interface defining the tool name, comprehensive description of purpose and capabilities, specific function to call, and configuration settings for result handling.

4) Memory: For memory management, each agent maintains its internal memory to store tasks history and thought traces. Additionally, all agents access a shared memory repository that stores key information between agents, such as generated hypotheses and investigation findings. This shared memory improves workflow consistency and avoids information loss between hunting phases. It also ensures that hunting results and findings can easily be integrated into the final report.

C. Playbooks

We automate investigations using *playbooks*, structured sets of threat hunting steps designed to guide analysts and validate hypotheses. Playbooks encapsulate typical analyst logic for specific attack types, tailored to available tools and intelligence sources.

In our framework, a playbook P, is modeled as a tuple $P = (M, \tau)$, where M represents the playbook's structured metadata and $\tau = \langle \text{task}_1, \ldots, \text{task}_k \rangle$ is an ordered sequence of tasks executed by dedicated agents. The metadata M explicitly declares the playbook's capabilities by specifying the entities it covers, the data sources it requires (e.g., Sysmon, ...), and the operating systems it targets. This metadata helps choosing the right playbook based on the provided hypothesis. A task, task_j, specifies an objective, an agent $a \in A$ to be invoked, its required inputs as well as the expected output. The execution agent leverages an LLM for reasoning and tool interaction to accomplish the task's objective, and store its output as evidence $e \in E$. An example of a complete playbook, structured in human-readable YAML for parsing and storage within our Playbook Store (P_{Store}), is detailed in Appendix D.

D. HARESHUNTER *Workflow*

The workflow of HARESHUNTER follows a structured and highly adaptive approach to threat hunting, organized into a cycle of hypothesis, investigation, and mitigation. This iterative process, depicted in Figure 5, is designed to mirror and enhance the cognitive flow of an expert analyst.

- The cycle begins in the KFM module, where the Observables Extractor continuously pulls and consolidates data from diverse sources (logs, cti feeds...) 1. These observables are passed to the Hypothesis Agent 2, which uses an LLM to analyze the data, formulate a testable Threat Hypothesis H, and generate contextual Hunting Tips 3.
- The generated context (hypothesis, tips) are then sent to the Playbook Picker, which first queries the Playbook Store 4 to retrieve candidate playbooks based on similarity. The candidates are checked whether they are suitable based on their metadata M. If there is no matching playbook, the Playbook Generator is triggered. This latter composes a new, custom playbook, constrained by the available system tools, and sends it to the Playbook Initializer 5b.
- The Investigator then executes the playbook through which the designed agents perform tasks and collect evidence. The collected evidence E is then fed to an LLM to verify or disprove the initial hypothesis H_i (step 5). This investigation is not a linear process but an iterative loop. If the collected evidence confirms the hypothesis, the findings are sent to the Mitigation Agent to initiate remediation 7a. However, if the evidence is ambiguous or contradicts the active hypothesis, the Investigator sends this information back to the Hypothesis Agent in a Refinement Loop 7b, triggering the regeneration of a revised hypothesis H_{i+1}. This revised hypothesis re-initiates the playbook orchestration cycle, allowing the system to self-correct and adapt its hunting strategy.
- In case a threat is confirmed, the Incident Attribute Parser extracts key IoCs and artifacts from the investigation findings 8. This structured data is used by the Incident Signature Generator to create new, targeted detection rules 9. These rules are then applied by the Threat Neutralizer to contain the threat 10. Finally, the Report Drafter compiles a comprehensive technical summary of the entire hunt and delivers it to the SOC Team for review 11.
- Finally, the Report Drafter documents the entire process in a technical report (step 9) that is communicated to the SOC team for review and feedback.

V. FRAMEWORK IMPLEMENTATION

This section goes into the finer details of our framework implementation, focusing on the technical architecture and its specific components. We demonstrate the hunting workflow through a practical hunting scenario that illustrates the operational capabilities of the system.

A. Choice of LLMs

HARESHUNTER incorporates two specialized LLMs, each selected for specific capabilities.

- For both hypothesis generation and validation, we utilize a distilled version of Deepseek-R1[8], a reasoning-enabled

[6] https://www.virustotal.com/

[7] https://platform.openai.com/docs/guides/function-calling

[8] https://www.deepseek.com/

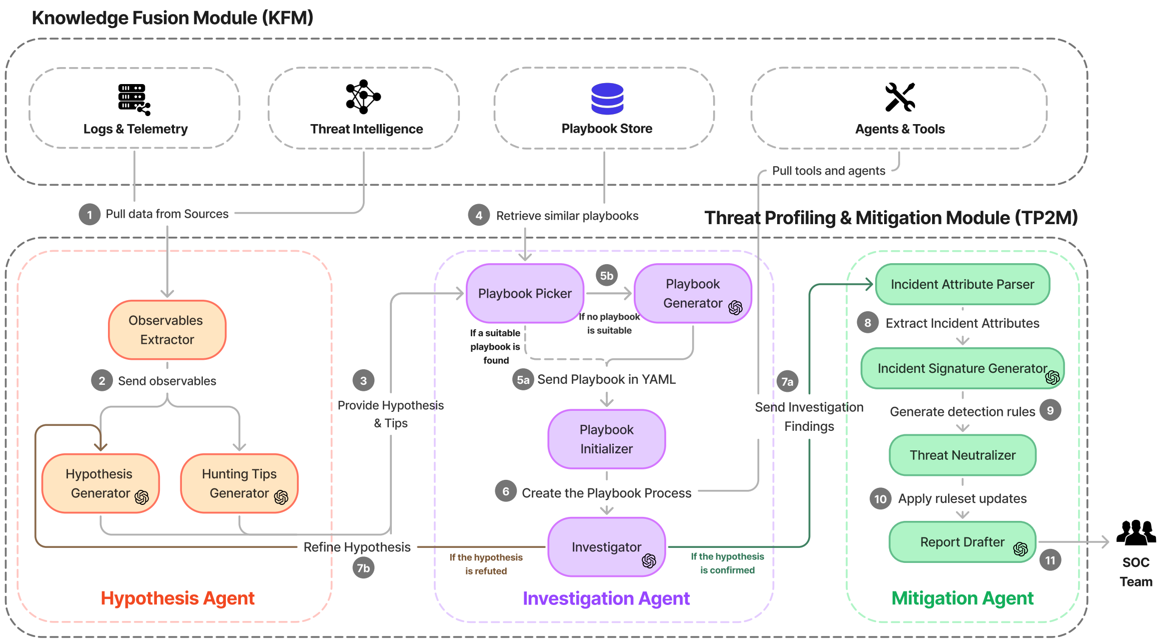

Fig. 5. HARESHUNTER detailed workflow, blocs with [LLM icon] are using an LLM in their core

model with 7 billion parameters. This model was selected for its reasoning and analytical capabilities. As Large Reasoning Models emerge they demonstrate exceptional potential for reasoning-intensive tasks in cybersecurity, this helps the agents to thoroughly analyze information before delivering conclusions.

- For investigation and mitigation, we use Codestral[9] with 22B parameters. This model is used to create playbooks, drives investigation agents and generates detection rules at mitigation stage. It is selected for its superior performance with code-like text which is essential for analyzing raw logs, and its extensive 32K token context length, suitable of large code segments. For the rest of the paper we will refer to them as $LLM_{Reasoning}$ and LLM_{Code} respectively.

Through rigorous experimentation, we determined this model combination provides optimal performance while maintaining efficient framework operation. While many research initiatives use proprietary closed-source LLMs [10][11], we chose open-source alternatives for their comparable capabilities, full customization control, specialized optimization for reasoning and cybersecurity analysis, and complete data privacy through local deployment [55], which is essential in our context.

B. Choice of Tools & Investigation Agents

In our implementation, we developed a set of tools to assist the agents during the investigation phase. Most of these tools interface with our KFM databases or query a cloud-based CTI platform (e.g. VirusTotal). To maximize efficiency, we defined two generic agents: one responsible for querying Elasticsearch (for accessing KFM), and the other for retrieving CTI data for a given indicator. A comprehensive inventory of the tools and agents incorporated into our implementation is provided in Appendix B.

C. Implemetation Workflow

1) Hypothesis Generation: The first step in hypothesis-driven threat hunting is to formulate a plausible hypothesis. In HARESHUNTER, the approach involves parsing information from SIEM alerts, logs, and system abstractions into a structured and unified format, which is then provided to the reasoning model ($LLM_{Reasoning}$) as context. This model is tasked with analyzing the given observables through a sequential reasoning process and proposing a hypothesis that aligns with them. Appendix A1 presents an example of the thinking process through which the $LLM_{Reasoning}$ evaluates the observables and formulates the corresponding hypothesis.

2) Playbook Selection & Generation: In this step, the Playbook Picker receives the hypothesis along with a set of hunting tips generated during the hypothesis formulation phase. It uses this context to retrieve semantically similar playbooks from the playbook store. To do this, we used the SecureBERT+ [56] embedding model to convert both the (hypothesis, tips) and playbook descriptions P_M into vector representations, and then retrieve the most similar playbooks. Mathematical details are provided in Appendix C. Once the top $k = 5$ playbooks are retrieved, their metadata (OS, data sources, keywords) is compared against the hunting tips for an exact match. To achieve this, we used a rule-based parser with regular expressions to extract relevant keywords from the context and cross-match them with playbook metadata. If no suitable playbook is found, the Playbook Generator employs LLM_{Code} to produce a new playbook in a parsable YAML format that specifically addresses the given hypothesis.

[9] https://huggingface.co/mistralai/Codestral-22B-v0.1
[10] https://platform.openai.com/docs/models
[11] https://www.anthropic.com/claude

The reason behind this hybrid approach is to avoid playbook mismatches, since semantic similarity alone can be misleading. Playbooks may appear similar in text but differ significantly in practice (e.g. *"PowerShell"* and *"CMD"*).

3) Investigation: The investigation phase begins after the Playbook Initializer transforms the YAML file into executable Python functions. The Investigator then carries out the tasks as specified in the playbook. For each task, the assigned agent processes the inputs and invokes appropriate tools. The agent uses the tool outputs as contextual information to achieve the task goal, as illustrated in Appendix E. Upon completing each task, the agent stores the findings in a shared investigation memory that persists throughout the entire hunt.

4) Hypothesis Validation & Refinement: Once all tasks are completed, the collected evidence is sent to $LLM_{Reasoning}$, which performs in-depth analysis and correlation to determine whether the threat can be confirmed, as shown in Appendix A2. If the hypothesis is confirmed, it triggers the Mitigation Agent. If the evidence does not support the hypothesis, the model generates feedback that is passed back to the Hypothesis Generator. The hypothesis is then refined based on this feedback, and the investigation process starts over. This cycle repeats until a hypothesis is confirmed.

5) Detection Rule Generation: Once a threat is confirmed during investigation, findings are translated into detection rules to identify similar malicious activity in the future. The generation process follows a structured pipeline detailed in Appendix F. The process extracts core artifacts and IoCs from investigation context, feeds them to the LLM to generate a structured JSON object containing rule logic, metadata, and platform-specific details, then renders the rule into its final form (e.g., Sigma, YARA) using platform-specific Jinja2 templates. Jinja2 was chosen for its flexibility in rendering structured data and ensuring syntactically correct, consistent rules across multiple platforms while preventing malformed output.

6) Neutralization & Report Drafting: This step involves applying the previously generated rule to the affected targets. The rule may be a firewall rule that blocks a specific malicious connection or a detection rule designed to identify the threat pattern in the future. The final stage of the workflow is to generate a technical report documenting the entire hunt, from hypothesis generation to investigation traces and mitigation steps. This report is then shared with the SOC team for further analysis.

VI. Experiments & Evaluation

This section provides a detailed overview of the HaresHunter evaluation, with a focus on the experimental setup, dataset, and evaluation metrics.

A. Setup

Our evaluation framework is configured as follows:

- **Test Bed.** Our evaluation environment utilizes two NVIDIA RTX A6000 GPU with 48GB VRAM for LLM inference. For our hunting platform, we used HELK[12], an open-source research project designed for threat hunting. It is equipped with advanced analytics capabilities such as SQL declarative language, graphing, structured streaming, and machine learning. For our evaluation process, we used only the ELK stack serving as our SIEM system, running Elasticsearch version 7.16 on Ubuntu 18.04, as recommended in the project documentation.
- **Dataset.** We use the Threat Hunter Playbook[13][14], an open-source, community-driven project that shares detection logic, adversary tradecraft, and resources to improve detection development efficiency. The dataset includes 26 guided hunting playbooks focused on detecting attacks targeting Windows hosts, each accompanied by prerecorded logs and detection rules as outputs. We selected this dataset because it combines hunting playbooks with corresponding logs for each attack scenario, making it especially well-suited to our use case.
- **Models Used.** As mentioned in the previous section, we used the Deepseek-R1-7B model for hypothesis generation and validation. Investigation and mitigation tasks were handled by the Codestral-22B model. For both models, we used their 4-bit quantized versions (`Q4_K_M`). Both models were configured with a temperature of **0** to ensure near deterministic outputs and were hosted locally via Ollama[15].

B. Experiments

Our evaluation is structured into two distinct experiments designed to test different facets of the framework's capabilities.

- **Experiment 1 (E1): Baseline Performance.** The objective of this experiment is to establish the baseline effectiveness of HaresHunter framework under ideal conditions. The methodology consists of executing all 26 hunting scenarios from the dataset. For each scenario, the framework is provided with the correct, corresponding hypothesis from our pre-defined set. The Playbook Picker in this case is expected to deterministically select the correct, existing playbook from the Playbook Store to conduct the investigation. This experiment measures the performance of the core components, including hypothesis validation and playbook execution.
- **Experiment 2 (E2): Resilience and Refinement Test.** The objective of this experiment is to stress-test the framework's adaptability and the efficacy of the hypothesis refinement loop. The methodology involves executing the same hunting scenarios as E1, but for each of them the framework is intentionally initiated with a distorted hypothesis (a plausible but incorrect version of the ground truth). This either forces the initial playbook selection to fail, or the hypothesis to be refuted, thereby triggering the iterative refinement. This experiment is designed to measure whether and how quickly the framework can self-correct from an initial wrong hypothesis and converge on

[12]https://thehelk.com/intro.html

[13]https://threathunterplaybook.com

[14]https://securitydatasets.com

[15]https://ollama.com/

the correct threat. A few examples of how hypotheses are mutated is shown in Appendix G.

C. Evaluation Metrics

To evaluate our framework, we need to assess how accurately our agents can conduct threat investigations from the findings collection through their analysis and final decision. For this, we employed the following metrics.

- **Success Rate (SR).** This metric determines how far our agents were able to progress with the investigation, using the following formula:

$$\text{SR} = \frac{|E_{\text{found}}|}{|E|} \tag{1}$$

where E_{found} is the set of evidence items successfully collected by the agent, and $|E|$ denotes the total number of evidence items expected (i.e., the optimal set).

- **Accuracy.** This metric evaluates how accurately the playbook is executed, accounting for both failed and repeated steps:

$$\text{Accuracy} = \frac{s_{\text{success}}}{s_{\text{success}} + s_{\text{fail}}} \tag{2}$$

where s_{success} is the number of successfully executed steps, and s_{fail} is the number of failed steps.

- **Refinement Efficiency (RE).** A metric specific to **E2**, defined as the number of refinement iterations (cycles of Hypothesis $\rightarrow$ Investigation $\rightarrow$ Feedback) required for the framework to converge on the correct hypothesis after starting from a distorted one. A lower value indicates higher efficiency. In our experiment, we considered a failed convergence if **RE** $>$ **10**.

D. Results

The comprehensive results of our evaluation are presented in Table II, with detailed metrics for each playbook across both Experiment 1 (E1) and Experiment 2 (E2). The findings demonstrate that HARESHUNTER is not only effective under baseline conditions but also shows remarkable resilience and adaptability when faced with initial uncertainty. Under the baseline conditions of **E1**, the framework achieved a Detection Rate of **84%**, successfully confirming **22 of the 26 scenarios**. The average Success Rate (SR) of **88%** shows that the investigation agents were consistently able to gather the vast majority of required evidence. The average Accuracy of **81%** reflects a high degree of efficiency in task execution, though it also suggests that more complex playbooks *(e.g. Local Powershell Execution)*, occasionally encountered failed or repeated steps. The four failures in E1 were mainly attributed to **reasoning errors** in the final validation stage, where the LLM incorrectly deemed the collected evidence insufficient to confirm the hypothesis, as shown in Table III.

The Resilience and Refinement Test (**E2**) revealed the impact of our iterative, self-correcting architecture. With the average SR slightly increased to **92%** and successfully confirming **24 of the 26 scenarios**. This improvement demonstrate that forcing the framework to re-evaluate the problem with new context from failed investigations, helps it overcome the

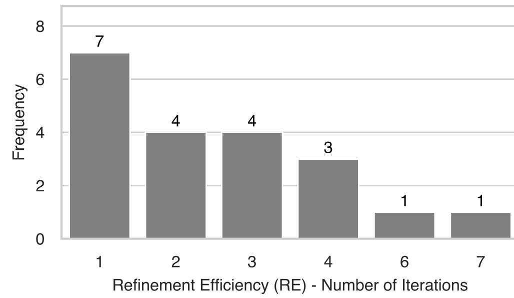

Fig. 6. Refinement Efficiency (RE) Distribution.

initial validation errors that led to failures in E1. Illustrated by scenarios like *Local Powershell Exec* and *Remote WMI ActiveScriptEventConsumers*, which were incorrectly refuted in E1 but were successfully confirmed in E2 after 2 and 4 refinement iterations, respectively. We considered the remaining two playbooks as failed, since they exceeded 10 refinements and could not confirm a threat.

The Refinement Efficiency (RE) metric provides insight into the complexity of this self-correction process. While many scenarios converged in just 1 or 2 iterations, more ambiguous cases like *AD Object Access via Replication Services* (RE=6) and *WMI Eventing* (RE=7) required multiple cycles to resolve. This reflects the challenging nature of distinguishing between similar attack techniques based on subtle evidentiary differences. While our results demonstrate the framework's powerful adaptive capabilities, they also highlight key trade-offs. The average Accuracy saw a minor decrease from 81% in E1 to 77% in E2, an expected consequence of the trial-and-error process inherent in the refinement loop, while also accounting for failed playbook executions. This indicates a small trade-off in step-level efficiency for a significant gain in overall hunt-level success and resilience against uncertainty.

E. Discussion

Regarding **RQ3**, our results indicate that HARESHUNTER is suitable for operationally realistic settings where senior analysts shift from being *"in the loop"* to being *"on the loop"*. Instead of manually executing each hunt step, the analyst would only supervise the agents execution, inspects their intermediate artifacts, and signs off on high confidence reports. This preserves human oversight while scaling coverage, and it aligns the framework with typical SOC review and approval practices. Also as the refinement loop exposes the full reasoning path, including missteps and recoveries, the analyst can audit how the system corrected an error rather than accepting a black box conclusion. In practice, this visibility is as important as raw accuracy, because it enables easier validation of outcomes and faster diagnosis when a hunt goes wrong.

A limitation of this phase is that poorly worded feedback can significantly prolong investigations, potentially sending the Hypothesis Agent down an incorrect path, as observed in the two failed examples from E2. Nevertheless, in most cases the findings help steer the system back toward convergence.

TABLE II
HARESHUNTER BENCHMARK RESULTS; (✓): CONFIRMED (✗): REFUTED

Attack/Playbook	Tasks	Accuracy		SR		Decision		RE
		E1	E2	E1	E2	E1	E2	
Access to Microphone Device	2	1.00	1.00	1.00	1.00	✓	✓	1
AD Object Access via Replication Services	2	1.00	1.00	1.00	1.00	✓	✓	6
AD Root Domain Modification for Replication Services	2	1.00	1.00	1.00	1.00	✓	✓	3
Alternate Powershell Hosts	3	0.66	0.60	0.66	1.00	✓	✓	1
DLL Process Injection via CreateRemoteThread	2	0.50	1.00	0.50	1.00	✓	✓	4
Domain DPAPI Backup Key Extraction	4	0.25	0.50	0.25	1.00	✗	✓	2
Local Powershell Execution	6	0.55	0.57	0.83	1.00	✓	✓	2
Local Service Installation	1	1.00	1.00	1.00	1.00	✓	✓	1
LSASS Memory Read Access	4	0.50	0.50	0.50	1.00	✓	✓	1
Reg Modification for Extended NetNTLM Downgrade	3	0.75	1.00	1.00	1.00	✓	✓	3
Reg Modification to Enable Remote Desktop Connections	1	1.00	1.00	1.00	1.00	✓	✓	1
Remote Service Control Manager Handle	5	0.71	1.00	1.00	1.00	✓	✓	4
Remote DCOM IErtUtil DLL Hijack	3	1.00	0.50	1.00	1.00	✓	✓	2
Remote Interactive Task Manager LSASS Dump	3	1.00	1.00	1.00	1.00	✓	✓	4
Remote Powershell Session	5	1.00	1.00	1.00	1.00	✓	✓	3
Remote Service creation	1	1.00	1.00	1.00	1.00	✗	✓	2
Remote WMI ActiveScriptEventConsumers	6	0.63	0.57	0.83	1.00	✗	✓	4
Remote WMI Wbemcomn DLL Hijack	3	1.00	0.50	1.00	1.00	✓	✓	2
SAM Registry Hive Handle Request	1	1.00	1.00	1.00	1.00	✓	✓	1
SMB Create Remote File	3	0.75	0.50	1.00	1.00	✓	✓	2
SysKey Registry Keys Access	1	1.00	–	1.00	–	✓	✗	–
WDigest Downgrade	1	1.00	1.00	1.00	1.00	✓	✓	1
WMI Eventing	4	0.75	1.00	0.75	1.00	✓	✓	7
WMI Module Load	1	0.50	–	1.00	–	✓	✗	–
WMI Win32 Process C&C Method for Remote Execution	3	0.50	1.00	0.66	1.00	✗	✓	3
Wuauclt CreateRemoteThread Execution	3	1.00	1.00	1.00	1.00	✓	✓	1
Average	-	0.81	0.77	0.88	0.92	0.84	0.92	-

TABLE III
REASONS BEHIND FAILED RESULTS IN E1

Scenario	Enough Evidence	Valid Analysis
Domain DPAPI Backup Key Extraction	✗	✓
Remote Service creation	✓	✗
Remote WMI ActiveScriptEventConsumers	✓	✗
WMI Win32 Process C&C	✓	✗

VII. CONCLUSION & FUTURE WORK

In this article, we explored the key challenges associated with cyber threat hunting and outlined several automation strategies designed to address them. We also introduced HARESHUNTER, a novel framework that delivers end-to-end cyber threat hunting workflow automation via an adaptive, agent-based architecture with minimal human approval. HARESHUNTER leverages a set of collaborative LLM agents within an iterative loop to autonomously formulate hypotheses, establish investigation plans, and analyze findings. Its closed loop hypothesis refinement mechanism enables recovery from inconclusive hunts. Our experiments, consisting of a baseline (E1) and an adaptive resilience test (E2) across 26 attack scenarios, achieved an 84% hunt success rate in E1 and 92% in E2, demonstrating the efficacy of this approach. Future work will focus on enhancing the efficiency and accuracy of HARESHUNTER by fine-tuning smaller, expert LLMs on extensive threat hunting corpora, and on expanding its capabilities to multimodal analysis over non-text telemetry. Finally, we will develop more robust memory architectures, such as knowledge graphs, to further mitigate reasoning errors in complex hunts.

ACKNOWLEDGMENTS

This work was carried out using the Toubkal Supercomputer HPC service, supported by Mohammed VI Polytechnic University and College of Computing.

REFERENCES

[1] NordLayer, "Cybersecurity statistics of 2024," https://nordlayer.com/blog/cybersecurity-statistics-of-2024/, 2024, accessed: April 5, 2025. [Online]. Available: https://nordlayer.com/blog/cybersecurity-statistics-of-2024/

[2] N. Sun, M. Ding, J. Jiang, W. Xu, X. Mo, Y. Tai, and J. Zhang, "Cyber threat intelligence mining for proactive cybersecurity defense: A survey and new perspectives," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 3, pp. 1748–1774, 2023.

[3] Lockheed Martin, "Cyber kill chain," n.d., accessed: February 17, 2025. [Online]. Available: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[4] B. Nour, M. Pourzandi, and M. Debbabi, "A survey on threat hunting in enterprise networks," *IEEE Communications Surveys & Tutorials*, 2023.

[5] Z. Wang, "A systematic literature review on cyber threat hunting," *arXiv preprint arXiv:2212.05310*, 2022.

[6] S. Mansfield-Devine, "Threat hunting: assuming the worst to strengthen resilience," *Network Security*, vol. 2017, no. 5, pp. 13–17, 2017.

[7] B. Gelman, S. Taoufiq, T. Vörös, and K. Berlin, "That escalated quickly: An ml framework for alert prioritization," *arXiv preprint arXiv:2302.06648*, 2023.

[8] F. K. Kaiser, U. Dardik, A. Elitzur, P. Zilberman, N. Daniel, M. Wiens, F. Schultmann, Y. Elovici, and R. Puzis, "Attack hypotheses generation based on threat intelligence knowledge graph," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 4793–4809, 2023.

[9] A. Elitzur, R. Puzis, and P. Zilberman, "Attack hypothesis generation," in *2019 European Intelligence and Security Informatics Conference (EISIC)*. IEEE, 2019, pp. 40–47.

[10] Z. M. Ferdjouni, B. Nour, M. Pourzandi, and M. Debbabi, "Threatscout: Automated threat hunting solution using machine reasoning," *IEEE Security & Privacy*, 2024.

[11] B. Nour, M. Pourzandi, R. K. Qureshi, and M. Debbabi, "Automa: Automated generation of attack hypotheses and their variants for threat hunting using knowledge discovery," *IEEE Transactions on Network and Service Management*, 2024.

[12] A. Mahboubi, K. Luong, H. Aboutorab, H. T. Bui, G. Jarrad, M. Bahutair, S. Camtepe, G. Pogrebna, E. Ahmed, B. Barry *et al.*, "Evolving techniques in cyber threat hunting: A systematic review," *Journal of Network and Computer Applications*, p. 104004, 2024.

[13] S. R. Sindiramutty, "Autonomous threat hunting: A future paradigm for ai-driven threat intelligence," *arXiv preprint arXiv:2401.00286*, 2023.

[14] R. Lin, "Thelma: Threat hunting enhanced language models for hunt automation," in *2024 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2024, pp. 1–9.

[15] M. Villarreal-Vasquez, G. Modelo-Howard, S. Dube, and B. Bhargava, "Hunting for insider threats using lstm-based anomaly detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 451–462, 2021.

[16] C.-K. Chen, S.-C. Lin, S.-C. Huang, Y.-T. Chu, C.-L. Lei, and C.-Y. Huang, "Building machine learning-based threat hunting system from scratch," *Digital Threats: Research and Practice (DTRAP)*, vol. 3, no. 3, pp. 1–21, 2022.

[17] S. Schmitt, F. I. Kandah, and D. Brownell, "Intelligent threat hunting in software-defined networking," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2019, pp. 1–5.

[18] M. Dehghan, B. Sadeghiyan, E. Khosravian, A. S. Moghaddam, and F. Nooshi, "Proapt: Projection of apt threats with deep reinforcement learning," *arXiv preprint arXiv:2209.07215*, 2022.

[19] M. Beechey, K. G. Kyriakopoulos, and S. Lambotharan, "Evidential classification and feature selection for cyber-threat hunting," *Knowledge-Based Systems*, vol. 226, p. 107120, 2021.

[20] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song, "Enabling efficient cyber threat hunting with cyber threat intelligence," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 193–204.

[21] A. Bolla and F. Talentino, "Threat hunting driven by cyber threat intelligence," Ph.D. dissertation, Politecnico di Torino, 2022.

[22] J. Jin, B. Tang, M. Ma, X. Liu, Y. Wang, Q. Lai, J. Yang, and C. Zhou, "Crimson: Empowering strategic reasoning in cybersecurity through large language models," *arXiv preprint arXiv:2403.00878*, 2024.

[23] C. Song, L. Ma, J. Zheng, J. Liao, H. Kuang, and L. Yang, "Audit-llm: Multi-agent collaboration for log-based insider threat detection," *arXiv preprint arXiv:2408.08902*, 2024.

[24] P. Karuna, E. Hemberg, U.-M. O'Reilly, and N. Rutar, "Automating cyber threat hunting using nlp, automated query generation, and genetic perturbation," *arXiv preprint arXiv:2104.11576*, 2021.

[25] S. Rajapaksha, R. Rani, and E. Karafili, "A rag-based question-answering solution for cyber-attack investigation and attribution," *arXiv preprint arXiv:2408.06272*, 2024.

[26] R. A. Chetwyn, M. Eian, and A. Jøsang, "Onto hunt-a semantic reasoning approach to cyber threat hunting with indicators of behaviour," in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2024, pp. 853–859.

[27] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[28] M. AI, "mistralai/codestral-22b-v0.1 - huggingface," 2024, accessed: April 5, 2025. [Online]. Available: https://huggingface.co/mistralai/Codestral-22B-v0.1

[29] R. Puzis, P. Zilberman, and Y. Elovici, "Athafi: Agile threat hunting and forensic investigation," *arXiv preprint arXiv:2003.03663*, 2020.

[30] M. Arafune, S. Rajalakshmi, L. Jaldon, Z. Jadidi, S. Pal, E. Foo, and N. Venkatachalam, "Design and development of automated threat hunting in industrial control systems," in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 2022, pp. 618–623.

[31] Á. C. Bienzobas and A. Sánchez-Macián, "Threat trekker: An approach to cyber threat hunting," *arXiv preprint arXiv:2310.04197*, 2023.

[32] X.-H. Li, C. C. Cao, Y. Shi, W. Bai, H. Gao, L. Qiu, C. Wang, Y. Gao, S. Zhang, X. Xue, and L. Chen, "A survey of data-driven and knowledge-aware explainable ai," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 29–49, 2022.

[33] F. Araujo, D. Kirat, X. Shu, T. Taylor, and J. Jang, "Evidential cyber threat hunting," *arXiv preprint arXiv:2104.10319*, 2021.

[34] P. Kumar, A. Jolfaei, and A. N. Islam, "An enhanced deep-learning empowered threat-hunting framework for software-defined internet of things," *Computers & Security*, vol. 148, p. 104109, 2025.

[35] H. Attieh, A. A. X. Tong, L. Yeo, N. J. Kai, L. T. Y. Teck, I. N. S. Mun, and P. M. Mohan, "Threat hunting on 5g future communication testbed using mitre fight framework," in *TENCON 2024 - 2024 IEEE Region 10 Conference (TENCON)*, Dec. 2024, pp. 1–5.

[36] Y. Schwartz, L. Benshimol, D. Mimran, Y. Elovici, and A. Shabtai, "Llmcloudhunter: Harnessing llms for automated extraction of detection rules from cloud-based cti," *arXiv preprint arXiv:2407.05194*, 2024.

[37] N. Rani, B. Saha, V. Maurya, and S. K. Shukla, "Ttpxhunter: Actionable threat intelligence extraction as ttps from finished cyber threat reports," *Digital Threats: Research and Practice*, vol. 5, no. 4, pp. 1–19, 2024.

[38] T. McIntosh, T. Liu, T. Susnjak, H. Alavizadeh, A. Ng, R. Nowrozy, and P. Watters, "Harnessing gpt-4 for generation of cybersecurity grc policies: A focus on ransomware attack mitigation," *Computers & security*, vol. 134, p. 103424, 2023.

[39] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "{ATLAS}: A sequence-based learning approach for attack investigation," in *30th USENIX security symposium (USENIX security 21)*, 2021, pp. 3005–3022.

[40] B. Nour, M. Pourzandi, and M. Debbabi, "Threatify: Apt threat variant generation using graph-based machine learning," *IEEE Transactions on Network and Service Management*, vol. PP, pp. 1–1, 01 2025.

[41] P. Balasubramanian, J. Seby, and P. Kostakos, "Cygent: A cybersecurity conversational agent with log summarization powered by gpt-3," in *2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*. IEEE, 2024, pp. 1–6.

[42] M. Kaheh, D. K. Kholgh, and P. Kostakos, "Cyber sentinel: Exploring conversational agents in streamlining security tasks with gpt-4," *arXiv preprint arXiv:2309.16422*, 2023.

[43] M. L. Diakhame, C. Diallo, and M. Mejri, "Mcm-llama: A fine-tuned large language model for real-time threat detection through security event correlation," in *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*. IEEE, 2024, pp. 1–6.

[44] T. Ali and P. Kostakos, "Huntgpt: Integrating machine learning-based anomaly detection and explainable ai with large language models (llms)," *arXiv preprint arXiv:2309.16021*, 2023.

[45] V. Jüttner, M. Grimmer, and E. Buchmann, "Chatids: Explainable cybersecurity using generative ai," *arXiv preprint arXiv:2306.14504*, 2023.

[46] R. Alghanmi, S. Kulaibi, J. Alghamdi, E. Bahashwan, and A. Aeshmawi, "Huntsmart: Hypothesis-driven threat hunting using genai," in *2025 International Conference on Innovation in Artificial Intelligence and Internet of Things (AIIT)*, 2025, pp. 1–12.

[47] D. J. Bianco, "The pyramid of pain," 2013, accessed: April 6, 2025. [Online]. Available: https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html

[48] A. M. P. Milani, A. Starr, S. Hill, C. Curtis, N. Anderson, D. Moreno-Lumbreras, and M.-A. Storey, "Fuzzy to clear: Elucidating the threat hunter cognitive process and cognitive support needs," *arXiv preprint arXiv:2408.04348*, 2024.

[49] D. Bianco, "Introducing the peak threat hunting framework," 2023, accessed: April 5, 2025. [Online]. Available: https://www.splunk.com/en_us/blog/security/peak-threat-hunting-framework.html

[50] F. Aldauiji, O. Batarfi, and M. Bayousef, "Utilizing cyber threat hunting techniques to find ransomware attacks: A survey of the state of the art," *IEEE Access*, vol. 10, pp. 61 695–61 706, 2022.

[51] W. French, "Enhancing threat hunting automation with large language models," Master's thesis, The University of North Carolina at Charlotte, 2024.

[52] H. Sqrrl Data, Gurgaon, "A framework for cyber threat hunting," 2018, accessed: April 5, 2025. [Online]. Available: https://www.threathunting.net/files/framework-for-threat-hunting-whitepaper.pdf

[53] T. Sumers, S. Yao, K. Narasimhan, and T. Griffiths, "Cognitive architectures for language agents," *Transactions on Machine Learning Research*, 2023.

[54] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023. [Online]. Available: https://arxiv.org/abs/2210.03629

[55] J. Sallou, T. Durieux, and A. Panichella, "Breaking the silence: the threats of using llms in software engineering," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER'24. ACM, Apr. 2024, p. 102–106. [Online]. Available: http://dx.doi.org/10.1145/3639476.3639764

[56] E. Aghaei, X. Niu, W. Shadid, and E. Al-Shaer, "Securebert: A domain-specific language model for cybersecurity," in *Security and Privacy in Communication Networks: 18th EAI International Conference, SecureComm 2022, Virtual Event, October 2022, Proceedings*. Springer, 2023, pp. 39–56.

APPENDIX

A. Reasoning Examples

The following subsections demonstrate how the LLM performs reasoning during both the hypothesis generation and validation stages. The examples illustrate a hunt for a Powershell remote session. Both examples are generated with `Deepseek-R1-7B` model.

1) Hypothesis Generation: This subsection demonstrates the reasoning process during the hypothesis generation phase. Table IV presents an example of how LLM processes observables related to suspicious PowerShell activity, analyzes potential attacker behaviors, and forms a coherent attack hypothesis.

TABLE IV
EXAMPLE OF HYPOTHESIS GENERATION THROUGH REASONING, CASE OF A POWERSHELL REMOTE SESSION HUNTING

<think> Okay, so I need to figure out a plausible attack hypothesis based on the provided observables. Let me break down each part step by step.
First, looking at the logs, there are multiple PowerShell commands executed with the -EncodedCommand flag. That's interesting because I know that this is often used to hide malicious activities from detection tools like SIEM (Security Information and Event Manager). So, someone might be trying to run commands without leaving a trail, making it harder for security systems to catch them.
Next, the system logs show remote WinRM sessions started by accounts that aren't typically used for remote administration. Hmm, WinRM is Windows Remote Management, which allows users to manage remote devices from their local machine. If these sessions are coming from accounts I don't usually use, it could indicate unauthorized access or a script running in the background without my knowledge.
Then there's network monitoring data with spikes on PowerShell remoting ports 5985/5986 at around 04:01:59 am. I remember that these ports are used by PowerShell to establish remote connections. Port 5985 is for local Windows PowerShell, and 5986 is for Windows Server PowerShell. A spike in traffic here suggests that someone is actively using PowerShell over a remote connection.
Putting this together: the presence of encoded commands might be an attempt to hide malicious activity, but the fact that these are coming from untrusted accounts (non-remote admin) combined with increased network traffic on known WinRM ports points towards a possible script or automation tool running in the background. Maybe someone is trying to automate some kind of attack, like credential dumping or file exfiltration.
I should also consider potential actors. It could be an unknown threat actor, maybe state-sponsored or a cybercriminal group. Their goal might be to gather sensitive information or disrupt operations without being detected immediately.
So, the behavior here seems to involve using PowerShell scripts with encoded commands to perform remote actions on untrusted accounts, which is facilitated by increased network activity on WinRM ports. The evidence would be the logs showing these activities and the spikes in network traffic.
I think I've got a good hypothesis now: **someone might be running a script that uses PowerShell with encoded commands to execute malicious operations on untrusted accounts, using remote WinRM sessions, which is detected by increased network activity on ports 5985/5986.** </think>

2) Hypothesis Validation: After the playbook execution, the findings are analyzed to decide on the hypothesis confirmation. Table V shows the detailed reasoning process used to evaluate findings related to PowerShell remote sessions, and reach a final determination about the attack hypothesis.

TABLE V
HYPOTHESIS VALIDATION REASONING PROCESS

<think> Okay, so I'm trying to figure out whether the attack hypothesis that adversaries are using remote PowerShell sessions to execute code in the environment is confirmed or refuted based on the findings provided. Let me go through each finding step by step.
First, looking at the initial Elasticsearch query results, there were several log entries related to PowerShell and WMI activity. The first few events (Event IDs 400, 5860, 5858, and 600) don't seem directly related to wsmprovhost or indicate lateral movement. So, those might not be too concerning on their own.
But then, there are entries where 'wsmprovhost.exe' is spawning PowerShell processes. That's interesting because wsmprovhost is a legitimate Windows process used for WMI remote management. However, if it's spawning PowerShell, especially in an unusual way or at odd times, that could be suspicious. The fact that it spawned powershell.exe twice within the same timeframe might indicate some activity, but I'm not sure yet.
Next, there are network connections to port 5985 from a specific IP address. Port 5985 is typically used for WinRM (Windows Remote Management) over HTTP, which is often used by PowerShell Remoting. The fact that these connections are happening could be normal if the environment uses remote management, but it's something to watch because adversaries might use this port for malicious activities.
Another finding mentions 'powershell.exe' being executed by 'wscript.exe'. That's a red flag because wscript is often used in scripts and can be abused. Also, attempts to communicate with an external IP on ports 5985 and 5986 suggest possible command-and-control activity, which is a hallmark of tools like PowerShell Empire.
Putting this together: while some events are normal, the spawning of PowerShell by wsmprovhost and network connections to known remote management ports could indicate that adversaries are using these legitimate tools for malicious purposes. The lack of direct evidence in the initial logs might mean we need more data or deeper analysis, but the indicators present do support the hypothesis.
I think there's enough here to suggest that the hypothesis should be confirmed because the findings point towards potential misuse of PowerShell and remote sessions, even if some details are missing. </think>

Final Decision: CONFIRMED
The attack hypothesis that adversaries might be leveraging remote PowerShell sessions is supported by the investigation findings. Key indicators include 'wsmprovhost.exe' spawning PowerShell processes and network connections to ports 5985 and 5986, which are commonly used for malicious activities such as command-and-control communication. While some events are normal, the suspicious activity detected suggests potential misuse of legitimate tools for adversarial purposes. Therefore, the hypothesis is confirmed.

B. Tools List

This section provides a comprehensive inventory of the tools implemented in our framework. These tools mainly search logs, query threat intelligence, and interact with external services. Table VI shows the inventory of tools available at the time of implementation. All tools are written in Python.

TABLE VI
TOOL FUNCTIONS BY AGENT

Agent	Tool	Description
Internal Logs Search Specialist	`search_es_by_query(query)`	Searches in elasticsearch indexes for a given query
	`retrieve_latest_logs(n, index)`	Retrieves n latest logs from a specific index
	`search_by_keyword(keyword)`	Searches in all indexes for a specific keyword (wildcard match)
CTI Search Specialist	`search_cti(keyword)`	Searches in CTI indexes for a specific keyword or IoC
VirusTotal Search Specialist (Through API)	`get_related_files(ioc)`	Retrieves files/items in relationship with a specific IoC
	`scan(ioc)`	Performs a VirusTotal scan for a specific IoC and returns scan results

C. Playbook Similarity Score

To automatically select the most relevant playbooks for a given investigation, we compute the semantic similarity between the generated *hypothesis* (h) and the associated *tips* (t), and the metadata description of each candidate playbook (p_M).

We first concatenate h and t into a single text query:

$$q = \text{concat}(h,\, t), \tag{3}$$

and then transform it into a dense vector representation $\mathbf{q} \in \mathbb{R}^{768}$ using SecureBERT+ model [56], finetuned on cybersecurity corpora. Each playbook's metadata p_M is similarly encoded into a vector $\mathbf{m}_p \in \mathbb{R}^{768}$.

The resulting vectors are normalized to unit length:

$$\hat{\mathbf{q}} = \frac{\mathbf{q}}{\|\mathbf{q}\|}, \quad \hat{\mathbf{m}}_p = \frac{\mathbf{m}_p}{\|\mathbf{m}_p\|}, \tag{4}$$

and the cosine similarity between them is computed as:

$$s(p; h, t) = \langle \hat{\mathbf{q}},\, \hat{\mathbf{m}}_p \rangle, \tag{5}$$

where $s \in [-1, 1]$ measures the semantic closeness between the (hypothesis,tips) pair and the playbook metadata.

Finally, the system retrieves the top $k = 5$ playbooks with the highest similarity scores:

$$\mathcal{P}^{\star} = \text{TopK}_{p \in \mathcal{P}}\ s(p; h, t), \quad k = 5. \tag{6}$$

D. Playbook Structure

Figure 7 depicts an example of a playbook, defining both its tasks and agents.

E. Playbook Execution Example

Figure 8 illustrates the execution flow of an automated playbook for investigating potential malicious PowerShell activities, particularly a remote PowerShell session. The figure shows the sequence of actions, decision points, and data gathering steps that occur during the investigation process, demonstrating how the system coordinates multiple components to analyze and respond to security incidents.

F. Rule Generation Process

Figure 9 depicts how a detection rule is generated. Each step is associated with a code example as part of generating a detection rule for Remote PowerShell sessions.

G. Hypotheses Distortion Examples

In the second experiment, we used three methods to alter the original hypotheses and confuse the framework.

- **Technique Confusion.** Here we alter the specific technique in the hypothesis while keeping the overall goal intact. This misleads the investigation by obscuring exactly where to look. In the initial iterations, the system may fail to identify the precise technique, but it will notice hints that gradually steer it toward the correct hypothesis.
- **Specificity Mismatch.** In this case, the goal remains the same, but a detail in the hypothesis is either removed or replaced with an incorrect one. As a result, the system searches for the wrong evidence, fails to find it, and instead receives feedback that points to the correct evidence. It then refines the hypothesis to the right technique.
- **Component/Tool Swapping.** Similar to **Technique Confusion**, here we substitute the main component of the hypothesis with a different but related one, introducing further misdirection (e.g. using *taskmgr.exe* to dump LSASS $\rightarrow$ using *procdump.exe*)

A few examples of each category are illustrated in Table VII.

TABLE VII
HYPOTHESIS PERTURBATIONS ANALYSIS

Perturbation Type	Scenarios	Original Hypothesis	Distorted Hypothesis
Technique Confusion	Access to Microphone Device	Adversaries might be accessing the microphone in endpoints over the network.	Adversaries might be capturing user keystrokes via a keylogger to steal sensitive information.
Technique Confusion	AD Object Access via Replication Services	Adversaries might attempt to pull the NTLM hash of a user via active directory replication apis from a non-domain-controller account with permissions to do so.	Adversaries might be attempting a Kerberoasting attack to harvest service account password hashes.
Technique Confusion	Local Service Installation	Adversaries might be creating new services to execute code on a compromised endpoint in my environment	Adversaries might be creating a new scheduled task to establish persistence and execute code on a compromised endpoint.
Technique Confusion	Reg Modification to Enable Remote Desktop Conections	Adversaries might be modifying registry key values to enable remote desktop connections in my environment	Adversaries might be modifying registry keys to enable WinRM, allowing for remote PowerShell execution.
Technique Confusion	LSASS Memory Read Access	Adversaries might be accessing LSASS and extract credentials from memory.	Adversaries might be getting a handle to the SAM database via the registry to extract credentials in my environment.
Technique Confusion	Remote WMI ActiveScriptEvent-Consumers	Adversaries might be leveraging WMI ActiveScriptEventConsumers remotely to move laterally in my network.	Adversaries might be creating new services remotely to execute code and move laterally in my environment.
Technique Confusion	SysKey Registry Keys Access	Adversaries might be calculating the SysKey from registry key values to decrypt SAM entries.	Adversaries might be using Volume Shadow Copy to create a backup of the NTDS.dit file for offline hash extraction.
Technique Confusion	WMI Eventing	Adversaries might be leveraging WMI eventing for persistence in my environment.	Adversaries might be creating a new scheduled task to establish persistence in my environment.
Specificity Mismatch	Alternate Powershell Hosts	Adversaries might be leveraging alternate PowerShell Hosts to execute PowerShell evading traditional PowerShell detections that look for powershell.exe in my environment.	Adversaries might be using heavily obfuscated and base64-encoded commands within standard powershell.exe processes to evade signature-based detections.
Specificity Mismatch	DLL Process Injection via CreateRemoteThread	Adversaries might be injecting a dll to another process to execute code via CreateRemoteThread and LoadLibrary functions.	Adversaries might be injecting a DLL into another process to establish persistence via the AppCertDlls registry key.
Specificity Mismatch	Local Powershell Execution	Adversaries might be leveraging PowerShell to execute code within my environment	Adversaries might be using local PowerShell execution specifically to run reconnaissance scripts like PowerView to enumerate domain objects.
Specificity Mismatch	Remote WMI Wbemcomn DLL Hijack	Threat actors might be copying files remotely to abuse a DLL hijack opportunity found on the WMI provider host (wmiprvse.exe).	Threat actors might be performing a generic DLL search order hijacking by dropping a malicious DLL in a system directory.
Component Swapping	Remote DCOM IErtUtil DLL Hijack	Threat actors might be copying files remotely to abuse a DLL hijack opportunity found on the DCOM InternetExplorer.Application Class.	Threat actors might be abusing the MMC20.Application DCOM object for lateral movement or persistence.
Component Swapping	Remote Interactive Task Manager LSASS Dump	Adversaries might be RDPing to computers in my environment and interactively dumping the memory contents of LSASS with task manager.	Adversaries might be RDPing to computers and using the Sysinternals tool procdump.exe to create a memory dump of the LSASS process.
Component Swapping	SMB Create Remote File	Adversaries might be creating a file remotely via the Server Message Block (SMB) Protocol.	Adversaries might be creating a malicious VBScript file (.vbs) remotely via SMB in the C:\\PerfLogs directory.

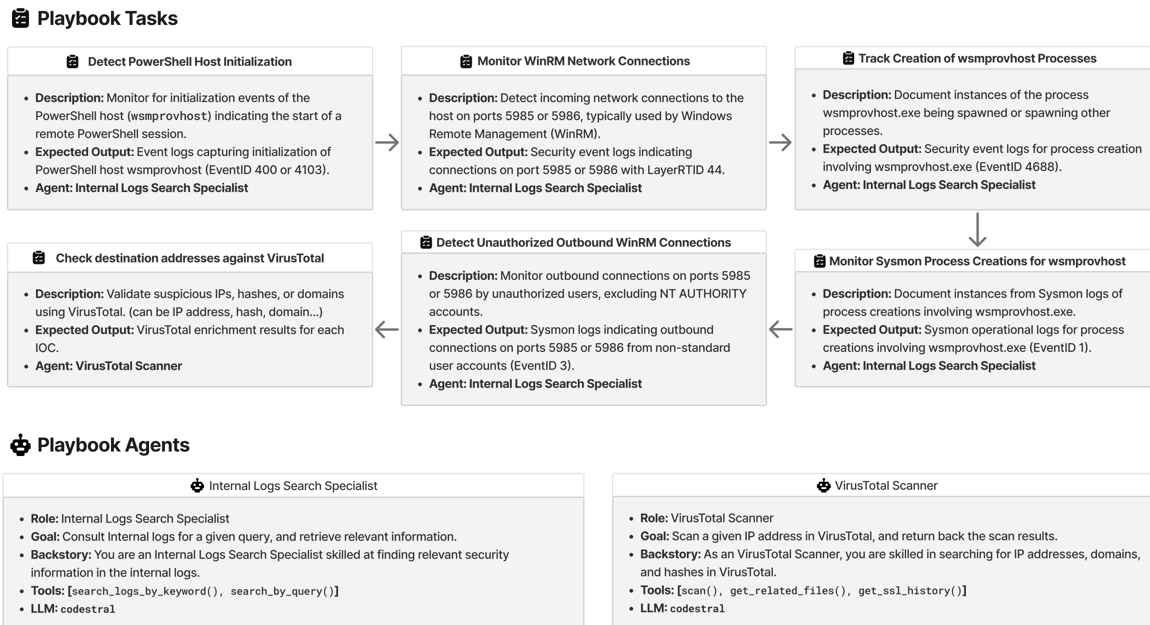

Fig. 7. Example of a playbook for investigating a remote PowerShell session threat

```
# Agent: Threat Hunting Specialist
## Task: Your task is to Monitor for the initialization of the PowerShell host wsmprovhost. In order to do so, you should
look for Event IDs 400 or 4103 (winlog.event_id). Filter for messages (winlog.event_data.Message) for values containing
"wsmprovhost". This helps pinpoint remote PowerShell engine activity associated with lateral movement.
# Agent: Threat Hunting Specialist
## Thought: I will start by executing an Elasticsearch query to search for Event IDs 400 or 4103 in the winlog.event_id
field that contain "wsmprovhost" in the winlog.event_data.Message field. This will help us identify any initialization of
the PowerShell host wsmprovhost, which is often associated with lateral movement.
## Using tool: search_elasticsearch
## Tool Input:
"{\"query\": {\"bool\": {\"should\": [{\"match\": {\"winlog.event_id\": 400}}, {\"match\": {\"winlog.event_id\": 4103}}],
 \"must\": [{\"wildcard\": {\"winlog.event_data.Message\": \"*wsmprovhost*\"}}]}}, \"source\": [\"@timestamp\",
\"winlog.event_id\", \"winlog.channel\"] }"
## Tool Output:
[{'@timestamp': '2020-09-20T21:09:11.871Z', 'winlog': {'event_id': 400, 'channel': 'Windows PowerShell'}},
{''@timestamp': '2020-09-20T21:09:11.871Z', 'winlog': {'event_id': 400, 'channel': 'Windows PowerShell'}},
{'@timestamp': '2020-10-09T22:30:36.805Z', 'winlog': {'event_id': 5860, 'channel': 'Microsoft-Windows-WMI-Activity/
Operational'}}, {'@timestamp': '2020-10-09T22:32:37.338Z', 'winlog': {'event_id': 5858, 'channel': 'Microsoft-Windows-
WMI-Activity/Operational'}}, {'@timestamp': '2020-09-20T21:09:11.869Z', 'winlog': {'event_id': 600, 'channel':
'Windows PowerShell'}}, {'@timestamp': '2020-09-20T21:09:11.869Z', 'winlog': {'event_id': 600,
'channel':'WindowsPowerShell'}}, {'@timestamp': '2020-09-20T21:09:11.869Z', 'winlog': {'event_id': 600, 'channel':
'Windows PowerShell'}}, {'@timestamp': '2020-09-20T21:09:11.870Z', 'winlog': {'event_id': 600, 'channel': 'Windows
PowerShell'}}, {'@timestamp': '2020-09-20T21:09:11.870Z', 'winlog': {'event_id': 600, 'channel': 'Windows PowerShell'}},
{'@timestamp': '2020-09-20T21:09:11.871Z', ''winlog': {'event_id': 600, 'channel': 'Windows PowerShell'}}]
# Agent: Threat Hunting Specialist
## Final Answer:
The Elasticsearch query returned several log entries that match our criteria for monitoring the initialization of the
PowerShell host wsmprovhost.
1. Event ID 400 occurred at '2020-09-20T21:09:11.871Z' in the 'Windows PowerShell' channel, which indicates that an
instance of PowerShell was started.
2. Another Event ID 400 occurred at '2020-09-20T21:09:11.871Z' in the 'Windows PowerShell' channel, similar to the first
entry.
3. Event ID 5860 occurred at '2020-10-09T22:30:36.805Z' in the 'Microsoft-Windows-WMI-Activity/Operational' channel.
4. Event ID 5858 occurred at '2020-10-09T22:32:37.338Z' in the 'Microsoft-Windows-WMI-Activity/Operational' channel,
similar to the previous entry.
5. Several Event ID 600 entries occurred at '2020-09-20T21:09:11.869Z' and '2020-09-20T21:09:11.870Z' in the 'Windows
PowerShell' channel.
```

Fig. 8. An example of a playbook task execution: case of a remote PowerShell hunting

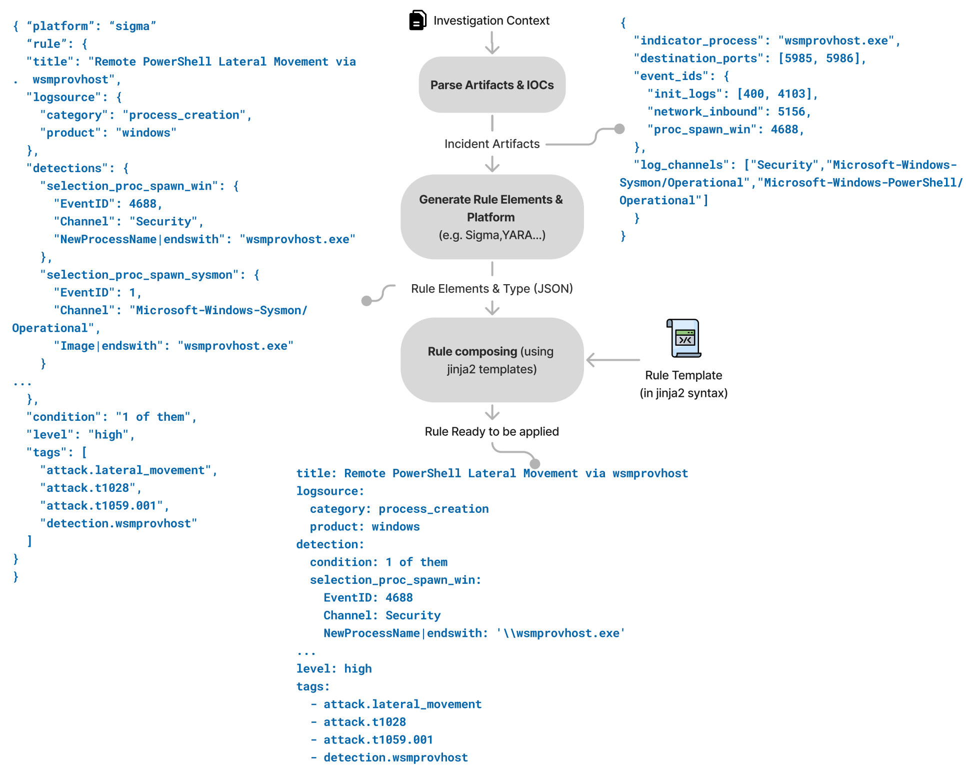

Fig. 9. Rule Generation Process