Knowledge-Informed Self-Reflective Automated Penetration Testing Based on LLMs

Hanzheng Dai

A Thesis
in
Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of
Master of Applied Science in Information Systems Security at
Concordia University
Montréal, Québec, Canada

December 2025

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Hanzheng Dai**

Entitled: **Knowledge-Informed Self-Reflective Automated Penetration Testing Based on LLMs**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Amr Youssef

_____ Examiner
Dr. Honghao Fu

_____ Examiner
Dr. Amr Youssef

_____ Thesis Supervisor
Dr. Jun Yan

Approved by _____
, Graduate Program Director

December 1, 2025 _____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Knowledge-Informed Self-Reflective Automated Penetration Testing Based on LLMs

Hanzheng Dai

Automated penetration testing (AutoPT) plays a crucial role in identifying and mitigating cybersecurity vulnerabilities before they can be exploited. However, traditional AutoPT approaches remain limited in adaptability, contextual reasoning, and cross-stage coordination. Although recent advances in artificial intelligence (AI), including expert systems, reinforcement learning (RL), and large language models (LLMs), have improved penetration testing (PT) automation, existing AI-assisted AutoPT frameworks still suffer from poor generalization, context loss, hallucination, and an inability to incorporate prior trial-and-error experience, making it challenging to automate multi-stage PT tasks across different environments.

To address these limitations, we propose *RefPentester*, a knowledge-informed, self-reflective AutoPT framework built on LLMs that enables context-aware, adaptive, and interpretable automated workflows across multi-stage PT processes. RefPentester integrates three main components: 1) a *Process Navigator* that identifies the current PT stage and retrieves corresponding hierarchical knowledge via a retrieval-augmented generation (RAG) pipeline; 2) a *Generator* that produces actionable and context-aware step-by-step guidance; and 3) a *Reflector* that evaluates execution feedback and facilitates structured reflective learning. A comprehensive PT knowledge vector database (VDB) is built from public cybersecurity resources, including MITRE ATT&CK and the OWASP Testing Guide (OTG), forming a tree-structured repository of tactics, techniques, and abstract actions.

Experiments were conducted on the *Sau* machine from the Hack The Box (HTB) platform, which provides realistic, legally authorized, and reproducible virtual PT environments. Results

show that *RefPentester* consistently outperforms a GPT-4o baseline in both credential capture and stage transition success rates, demonstrating improved operational reliability. Beyond quantitative gains, log-based qualitative analysis reveals two key patterns: 1) the Reflector enhances decision stability by preventing repeated mistakes and supporting adaptive recovery across stages; and 2) the interaction between hierarchical knowledge retrieval and structured reflection yields clearer and more interpretable reasoning trajectories. These findings indicate that combining knowledge grounding with reflective optimization substantially strengthens robustness and interpretability in multi-stage AutoPT reasoning.

The study also acknowledges practical limitations, such as reliance on a single HTB environment, and it outlines several future research directions to further advance AI-assisted AutoPT.

Acknowledgments

I would like to express my heartfelt appreciation to everyone who has supported me and accompanied me throughout my master's journey.

First and foremost, I would like to extend my deepest appreciation to my supervisor, Professor Jun Yan, for his guidance, encouragement, and invaluable insights throughout my research. He accepted me as a research-based master's student despite my limited prior experience, and his patience and support have been instrumental to my development. One of the most important lessons I have learned from him is the philosophy of conducting research: *Being a researcher is fundamentally different from being an engineer*. This perspective has profoundly shaped my approach to problem-solving and knowledge creation. The insights and values I gained from Professor Yan will continue to influence my academic and professional journey for years to come.

I would also like to thank the members of my examination committee, Dr. Amr Youssef and Dr. Honghao Fu, for taking the time to review my work and for providing constructive feedback that greatly improved the quality of this thesis.

My gratitude also extends to the faculty, staff, and colleagues at CIISE, whose assistance, guidance, and support created a positive and collaborative environment for my studies. I would also like to express my appreciation to my colleagues at CREO Solutions. I am especially grateful to Tony Guan, Julian Conte, Amin Bayatpour, Guillaume Ah-ki, and Wen Bo Li for the great experience and memorable time we shared. My time at CREO not only broadened my professional perspective but also *fine-tuned* my ability to write clean, elegant, and reusable code.

I would also like to express my gratitude to my colleagues in the research lab at Concordia and

to the friends I met in Montréal for their valuable help, insights, and encouragement throughout my studies. I am especially thankful to Dr. Yuanliang Li, Pengyi Liao, Juanwei Chen, Yiheng Zhao, Chengming Hu, Hanlin Chen, Jianbing Lai, Xuesen Chen, Zhibo Zhang, and Kuaiyi Wang. Thank you for the extraordinary memories we shared together. I will always be grateful for that.

Finally, I would like to express my heartfelt thanks to my family for their unwavering love, understanding, and encouragement. Their continued support and inherent intelligence have been a constant source of strength and motivation throughout my academic journey.

To everyone who has supported this work, I extend my sincere thanks.

No mistakes in the tango, not like life.

It's simple.

That's what makes the tango so great.

If you make a mistake, get all tangled up, just tango on.

— Scent of a Woman

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we first introduce the background of the research in Section 1.1. The background contains four aspects: cybersecurity, penetration testing (PT), automated penetration testing (AutoPT), and Artificial Intelligence (AI) assisted AutoPT. Secondly, we describe the problem statement in Section 1.2. The objectives of this research are presented in section 1.3. The contributions of this work are described in section 1.4. Finally, the structure of this research is presented in Section 1.5.

1.1 Background

1.1.1 Cybersecurity

In an era characterized by rapid technological advancements and the exponential growth of digitalization, cybersecurity has become a critical and multifaceted domain that underpins the stability, integrity, and functionality of modern societies [1]. Cybersecurity refers to the practice of safeguarding computer systems, networks, and digital information from unauthorized access, use, disclosure, disruption, modification, or destruction [2]. The widespread adoption of interconnected technologies, including the Internet of Things, cloud computing, AI, and 5G networks, has not only revolutionized the way individuals and organizations operate but has also significantly expanded

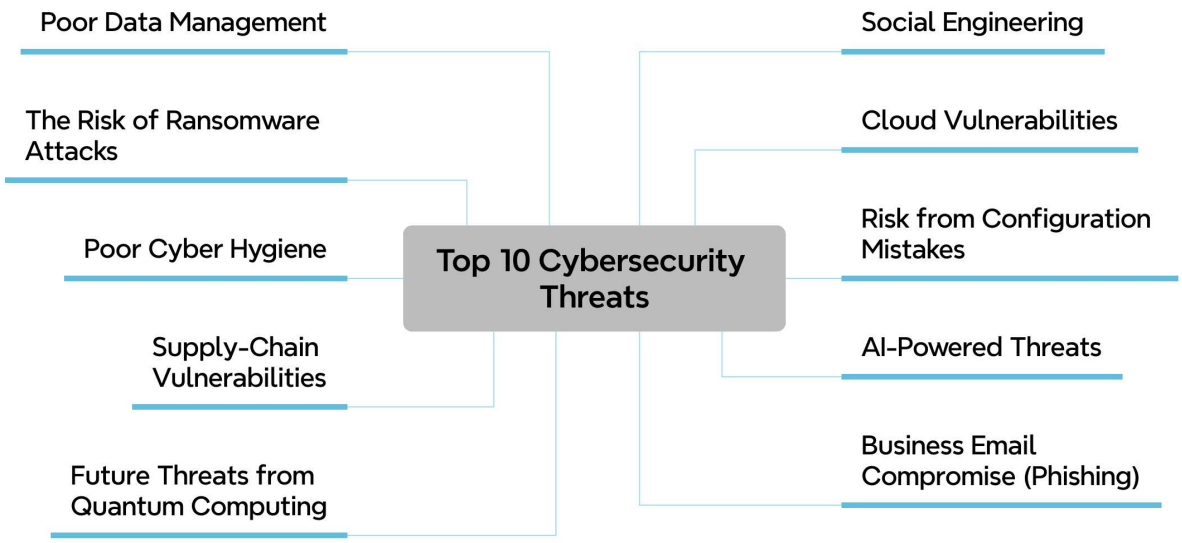

Figure 1: Top 10 Cybersecurity Threats in 2025

the attack surface, making cyberspace vulnerable to a diverse range of threats [3]. Fig. 1 illustrates the ten key threats in the field of cybersecurity, which are social engineering, cloud vulnerabilities, risk from configuration mistakes, AI-Powered threats, business email compromise (Phishing), future threats from quantum computing, supply-chain vulnerabilities, poor cyber hygiene, the risk of ransomware attacks, and poor data management [4]. Since the attack surface has expanded, individuals and organizations' network systems are more vulnerable to cyberattacks, and the consequences are even more severe.

On July 4, 2024, an internet user uploaded a file named "RockYou2024" to a notorious hacking forum, unleashing a cybersecurity nightmare. This extensive dataset introduced 1.5 billion additional plain-text passwords, augmenting the previous "RockYou2021" compilation. These newly leaked passwords significantly expand the arsenal available to threat actors. Malicious entities can leverage the "RockYou2024" password compilation to execute sophisticated brute-force attacks. By methodically testing the passwords against a multitude of online accounts, they aim to gain unauthorized access to personal email, financial, and social media profiles. This not only violates the privacy of countless individuals but also exposes them to risks such as identity theft, financial fraud, and the unauthorized dissemination of personal information [5]. A week later, the digital

realm experienced one of the most devastating cyber incidents in history, with 8.5 million computers falling prey. The root cause was a content update from security firm CrowdStrike for Windows hosts. This update, meant to boost security, instead harboured a critical "defect" that malicious actors exploited, infiltrating systems to steal data, disrupt operations, or seize control. The far-reaching impact spanned multiple industries. Flights were grounded globally, disrupting travel and causing financial losses to airlines. The healthcare sector faced risks to patient lives as access to crucial medical information was cut off, leading to postponed procedures. The financial sector also suffered, with payment services down, which harmed businesses, especially small ones, and caused economic instability [6].

Despite the increasing awareness of cybersecurity threats and the efforts made to develop countermeasures, the cybersecurity landscape continues to evolve at an alarming pace. Cybercriminals are constantly devising new attack techniques, leveraging emerging technologies, and exploiting vulnerabilities in software, hardware, and human behaviour. The rise of state-sponsored cyberattacks, ransomware campaigns, and advanced persistent threats (APTs) has further complicated and intensified the cybersecurity challenge. Fig. 2 shows the annual cost of global cybercrime. The data from 2018 to 2023 is actual data, and the data from 2024 to 2028 is forecasted data. What is clear is that the annual global cost of cybercrime is rising every year from 2018 to 2023 (from 0.86 trillion U.S. dollars to 8.15 trillion U.S. dollars) and is expected to rise to 13.82 trillion U.S. dollars in 2028 [7]. These incidents highlight the vulnerability of the digital world and underscore the urgent need for enhanced cybersecurity standards, techniques, and mitigation measures.

1.1.2 Penetration Testing

Penetration testing (PT) is a pivotal technique in the realm of cybersecurity, having already demonstrated its potential to mitigate cyber threats and address potential vulnerabilities. It involves the authorized simulation of real-world cyberattacks on a network system or application to identify vulnerabilities that adversaries could exploit [8]. Based on the Penetration Testing Execution Standard [9], the PT process typically consists of seven main stages:

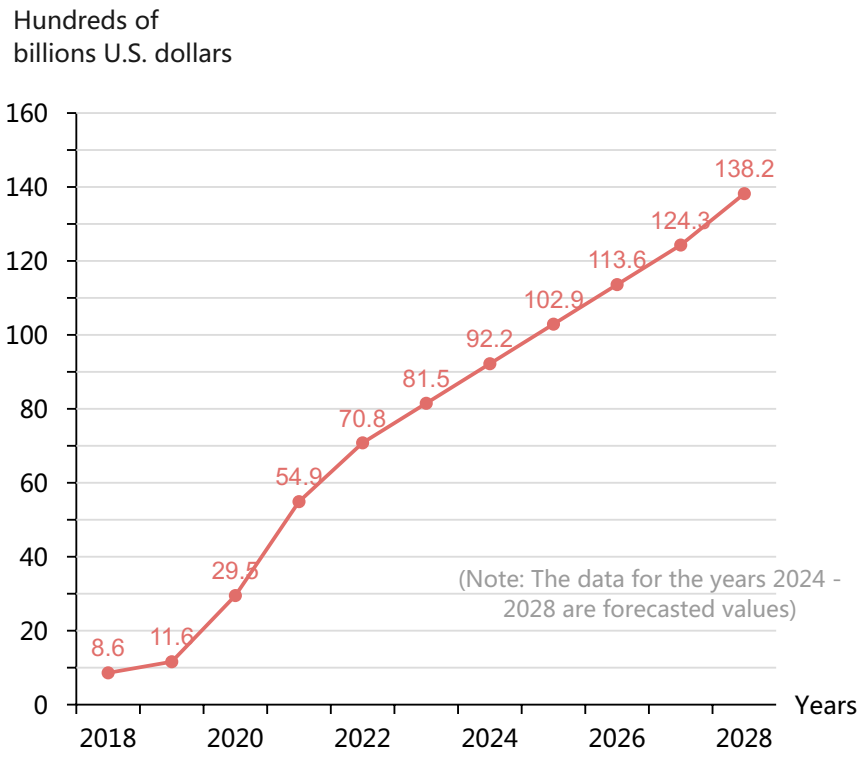

Figure 2: Annual Cost of Cybercrime Worldwide

- **Pre-engagement Interactions**: This is the initial stage where communication occurs between the pentester team and the client. It involves understanding the client's requirements, the scope of the test, and establishing the ground rules. This sets the foundation for the entire PT, ensuring both parties are on the same page regarding goals, boundaries, and expectations.

- **Intelligence Gathering**: Pentesters collect information about the target network system. This can include details like IP addresses, domain names, software versions, and employee information. The more data they gather, the better their understanding of the target's infrastructure becomes, which helps in identifying potential vulnerabilities.

- **Threat Modelling**: Based on the intelligence gathered, pentesters analyze and categorize potential threats. They determine what assets are at risk, identify potential adversaries, and assess the methods these adversaries might employ. This step helps prioritize security efforts and focus on the most critical areas.

- **Vulnerability Analysis**: Pentesters use various tools and techniques to scan for security

vulnerabilities in the target. This could involve identifying weaknesses in software, misconfigurations in network systems, or flaws in application logic. Identifying these vulnerabilities is crucial as they are the entry points that adversaries could exploit.

- **Exploitation**: If vulnerabilities are found, pentesters attempt to exploit them to gain unauthorized access or perform malicious actions. This is where the technical skills of the pentesters are put to the test, as they use techniques like buffer overflows or SQL injection to breach the system's defences. Since pentesters' actions are authorized, this is why PT is also referred to as ethical hacking.

- **Post-Exploitation**: After successfully exploiting a vulnerability, pentesters further explore the system. They might escalate their privileges, move laterally within the network, or extract sensitive information. This stage helps in understanding the full extent of the damage a real-world adversary could cause.

- **Reporting**: The final stage involves documenting the entire PT process. The report details the vulnerabilities found, how they were exploited, and the potential impact on the organization. It also provides recommendations for fixing the issues, helping the client improve their security posture.

Through the seven stages, the penetration team (red team) can provide organizations with a detailed understanding of their security posture, highlighting areas that need improvement. By fixing these vulnerabilities, companies can enhance their defences, protect sensitive data, and reduce the risk of cyberattacks, ensuring the integrity, confidentiality, and availability of their digital assets [10].

1.1.3 Automated Penetration Testing

PT serves as a vital safeguard in the field of cybersecurity, enabling organizations to identify and remediate vulnerabilities before adversaries can exploit them proactively. However, traditional

PT has several disadvantages that limit its ability. It is highly time-consuming, as each PT often requires meticulous manual execution, from initial reconnaissance to vulnerability exploitation and reporting, which prolongs system downtime and increases operational costs [11]. Additionally, it demands a high level of specialized knowledge and expertise [12]. Pentesters must be well-versed in a wide range of technologies, attack vectors, and security frameworks, which can be a significant barrier and also limit the scalability of testing efforts [13].

AutoPT emerges as an innovative solution to address these challenges. Leveraging advanced software tools and scripts, it automates many of the repetitive and time-consuming tasks inherent in traditional PT [14]. Compared to the traditional PT, AutoPT offers several distinct advantages. Firstly, it drastically improves efficiency. Tools like Nessus [15] and OpenVAS [16] can scan thousands of systems and applications within hours, whereas a manual team might take weeks to achieve the same coverage. Secondly, it reduces the reliance on highly specialized personnel. While some level of expertise is still required for setup and interpretation of results, the automated nature of the process means that organizations can conduct more frequent tests with less human intervention [17]. Thirdly, it ensures greater consistency [18]. Automated tools follow predefined procedures and algorithms, eliminating the variability that can occur in manual PT due to human error or fatigue. This consistency also aids in accurate trend analysis over time, enabling organizations to track the effectiveness of their security improvements [19].

Fig. 3 illustrates a comprehensive categorization of widely adopted AutoPT tools in the cybersecurity industry. These tools can be classified into ten distinct groups based on their primary functions. Vulnerability Scanning tools, such as Nessus [35] and OpenVAS [36], are essential for systematically identifying security weaknesses across systems, applications, and networks. Exploit Execution tools, including Metasploit [34], are used to exploit known vulnerabilities, enabling security professionals to assess the practical impact of discovered flaws.

Fuzz Testing tools, such as American Fuzzy Lop (AFL)[37, 32] and Peach Fuzzer[33], generate random or semi-random input data to trigger unexpected behaviour in applications, helping to uncover latent bugs and vulnerabilities. Credential Stuffing and Brute-Force tools, exemplified

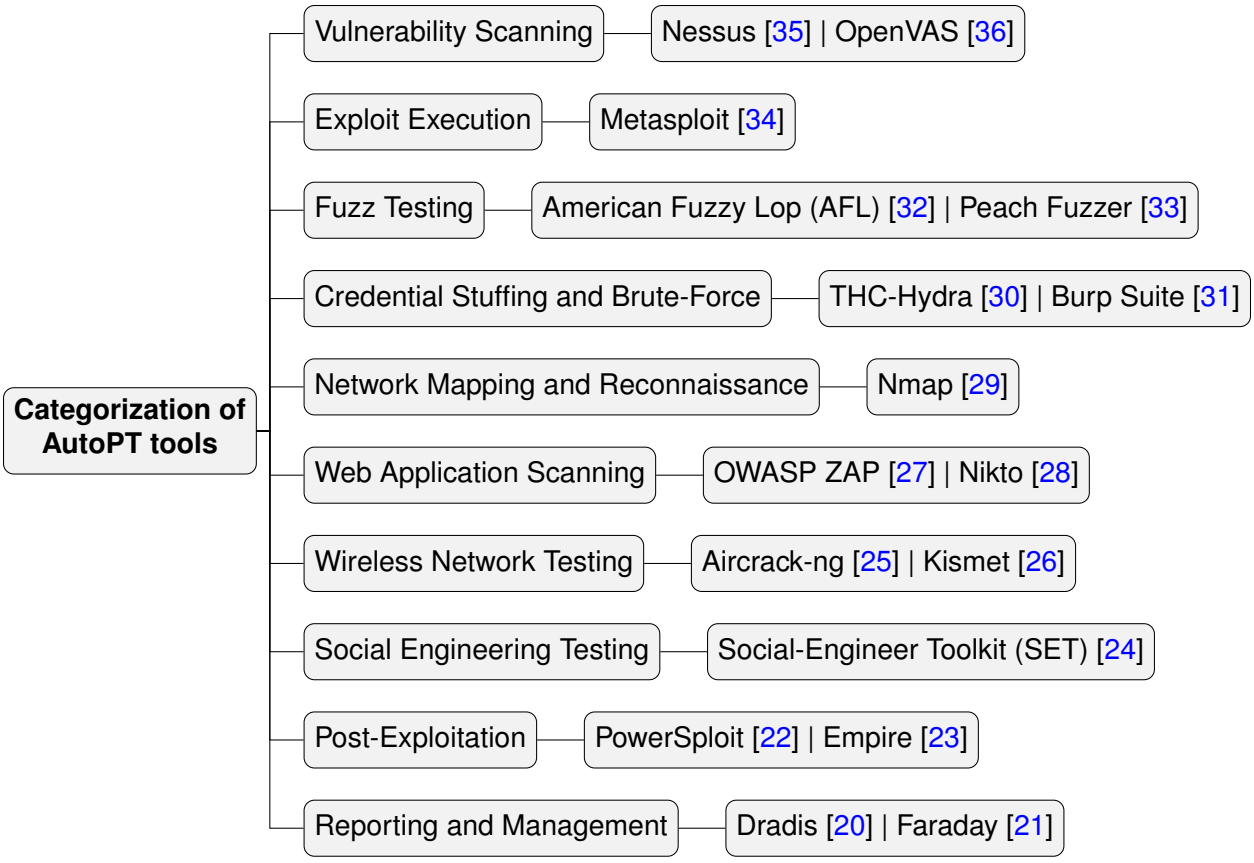

Figure 3: Categorization of Tools for AutoPT

by THC-Hydra [30] and Burp Suite [31], attempt unauthorized access by systematically testing passwords or using precompiled credential lists. Network Mapping and Reconnaissance tools, such as Nmap [29], play a crucial role in identifying hosts and services within a network, facilitating the mapping of network topology and potential targets.

Web Application Scanning tools, including OWASP ZAP [27] and Nikto [28], are used to detect vulnerabilities in web applications, such as SQL injection and cross-site scripting (XSS). Wireless Network Testing tools, such as Aircrack-ng [25] and Kismet [26], are used to assess the security of wireless networks by detecting unauthorized access points and attempting WPA/WPA2 password cracking.

Social Engineering tools, such as the Social-Engineer Toolkit (SET) [24, 38], simulate attacks like phishing to evaluate the human element of an organization's security posture. Post-exploitation tools, including PowerSploit [22] and Empire [23], are used after gaining access to a system to

escalate privileges, maintain persistence, and extract sensitive data. Reporting and Management tools, such as Dradis [20] and Faraday [21], facilitate the organization of testing results, report generation, and overall management of the PT workflow.

1.1.4 AI-Assisted AutoPT

AutoPT tools are widely used in the cybersecurity industry, which allows pentesters to cover a greater range of assessments and reduce organizations' costs in a fraction of the time [39]. Despite these advantages, traditional AutoPT tools often suffer from a key limitation: they are narrowly focused; each AutoPT tool automates only a specific task within the PT process. For instance, Nmap specialize in network scanning and host discovery. This siloed functionality requires human pentesters to manually coordinate tasks, make decisions, and perform contextual analysis to integrate the tools into the actual PT process effectively. As a result, automation of PT remains fragmented and rigid, preventing the complete automation of the entire process.

To address the limitation, some researchers integrate AI into the PT process and propose some AI-assisted AutoPT frameworks. Rather than automating isolated tasks, AI-assisted AutoPT frameworks aim to emulate human pentesters' reasoning, learning, and adaptation across the entire PT process. By leveraging AI, these frameworks can dynamically optimize attack paths, learn the optimal PT strategy and improve their ability through previous experience. This not only enhances the efficiency and accuracy of the PT process but also enables the detection of complex, multi-stage vulnerabilities that traditional tools may overlook.

AI-assisted AutoPT frameworks are commonly implemented using various AI paradigms, which can be broadly categorized based on their core techniques into three conceptual groups: reinforcement learning (RL)-based frameworks, expert system-based frameworks, and large language model (LLM)-based frameworks.

As shown in Table 1, each category leverages distinct AI methodologies to address specific challenges in the PT process. Expert system-based AutoPT frameworks automate the PT decision-making process by encoding domain-specific knowledge into rule-based logic engines. These

Table 1: Categories of AI-Assisted AutoPT Frameworks

Category	Core Functionality	Example Use Case
Expert System-Based Frameworks	Encode human expertise into rule-based engines for decision-making	ESSecA [40] AlphaEXP [41]
Reinforcement Learning-Based Frameworks	Learn optimal PT strategies through environment interaction	HER-PT [42] PenGym [43]
Large Language Model-Based Frameworks	Understand and generate natural language for test planning, log interpretation, and script generation	PentestAgent [44] PentestGPT [45] AutoAttacker [46]

systems are particularly effective in scenarios that require deterministic behaviour or structured expert guidance, such as automating standard responses to known vulnerabilities. One example is ESSecA, an expert system designed to support PT in Internet of Things (IoT) environments using threat intelligence [40]. RL-based frameworks model the PT process as a sequential decision-making process. In these AutoPT frameworks, intelligent agents interact with the environment and learn optimal strategies through trial and error to ensure the agent can achieve a higher cumulative reward. For example, the HER-PT framework incorporates deep RL to enhance PT efficiency in evolving environments [42]. LLM-based frameworks utilize the inherent knowledge of LLMs to automate the PT process. They can combine natural language understanding ability with generative capabilities, enabling them to interpret complex textual inputs, such as vulnerability reports and logs, and generate the current stage of PT plans or executable scripts. For instance, PentestAgent uses LLMs to parse Common Vulnerabilities and Exposures (CVE) and produce tailored PT scripts [44].

1.2 Problem Statement

Traditional AutoPT tools have played a vital role in improving the efficiency, repeatability, and scalability of security assessments. They enable pentesters to automate routine tasks, such as

scanning, enumeration, and vulnerability detection, thereby significantly reducing manual workload and operational costs. However, despite these advantages, traditional AutoPT tools are often task-specific and fragmented, with each tool automating only a discrete phase of the PT process. They lack coordination, adaptability, and contextual awareness, requiring manual effort to interpret results, prioritize actions, and make decisions based on the testing environment. As a result, overall automation remains shallow and rigid, falling short of simulating the dynamic, multi-stage reasoning of human experts.

To overcome these limitations, AI-assisted AutoPT frameworks have gained increasing attention in recent years. Frameworks based on RL, expert systems, and LLMs have been developed to enhance decision-making, adaptability, and contextual understanding throughout the PT process. Each of these paradigms brings unique strengths: RL enables learning through interaction with the environment, expert systems encode structured human knowledge, and LLMs provide advanced natural language understanding and generation capabilities.

Despite these advancements, current AI-assisted AutoPT frameworks still exhibit limitations that hinder their effectiveness in real-world scenarios. RL- and expert system-based approaches face challenges such as poor generalization, limited adaptability, and rigid rule structures. LLMs represent a promising alternative to address these issues; however, LLM-based AutoPT frameworks also suffer from a distinct and underexplored set of limitations. Specifically, they struggle to maintain long-term context across multi-stage PT processes and lack mechanisms for learning from trial and error. Furthermore, due to the scarcity of PT content in general web corpora, LLMs often lack domain-specific knowledge, which undermines their reliability. Issues such as hallucinated command generation and short-sighted task planning further constrain their practical effectiveness.

1.3 Objectives

The main objective of this research is to develop and validate a knowledge-informed, self-reflective AutoPT framework that enhances reliability, adaptability, and reasoning continuity in multi-stage PT. Specifically, this study aims to: 1) Enhance long-term PT task planning by enabling LLM-based AutoPT agents to decompose multi-stage tasks, maintain reasoning continuity, and improve decision coherence; 2) Leverage structured and domain-grounded PT knowledge to reduce hallucination and improve interpretability via a hierarchical retrieval and reasoning pipeline; and 3) Establish a self-reflective learning mechanism that incorporates reflective optimization to improve framework performance through trial-and-error continuously. Based on the above objectives, this study contributes to the advancement of AI-assisted AutoPT in three primary aspects, including the framework design, module integration, and PT domain-specific knowledge base.

1.4 Contributions

The key contributions of this research can be summarized as follows:

1. **Framework design with functional modules.** We proposed an LLM-based AutoPT framework, called *RefPentester*, which introduces a modular architecture to enhance reasoning reliability, adaptability, and reduce hallucination across multi-stage PT workflows. The framework is built on three main components:

    - *Process Navigator*: This component models the PT process as a stage-based process guided by a PT Stage Machine. It determines the current PT stage and ensures that the hierarchical RAG pipeline can retrieve corresponding knowledge to narrow and support subsequent reasoning steps, which align with the overall PT objective. The RAG pipeline integrates cybersecurity knowledge from sources such as MITRE ATT&CK [47] and the OTG [48].

- *Generator*: This component transforms retrieved high-level PT knowledge and human instructions into executable operational guidance.
- *Reflector*: This component performs self-reflection and evaluation by analyzing prior trajectories and identifying failure causes. It provides structured feedback to improve reasoning consistency, adapt future decisions, and reduce repetitive or ineffective actions.

2. **Integration and customization of modules from existing paradigms.** We combined and refined mechanisms from expert system-based, RL-based, and LLM-based AutoPT paradigms to achieve structured reasoning and adaptive learning within a unified architecture. Specifically, the PT Stage Machine draws inspiration from rule-based expert systems to model the PT process as a seven-stage state machine, while the Reflector incorporates reinforcement-style linguistic feedback to guide self-correction and reasoning optimization. The RAG pipeline was customized into a hierarchical, tree-structured retrieval mechanism that aligns with the organization of PT knowledge in the vector database (VDB). Unlike flat RAG, the proposed hierarchical RAG provides multi-level knowledge retrieval, allowing the Generator to access contextually relevant information according to the current PT stage, from tactic-level to abstract action-level knowledge.

3. **Construction of a PT knowledge vector database.** We developed a three-tier tree-structured PT knowledge VDB using publicly available cybersecurity resources (i.e., MITRE ATT&CK [47], OTG [48]). The VDB organizes PT knowledge into tactic-level, technique-level, and abstract action-level entries, enabling stage-aware retrieval and domain-consistent generation during automated PT.

Experimental validation on the Hack The Box (HTB) platform demonstrates that *RefPentester* achieves higher credential capture and stage transition success rates than baseline LLM models (i.e., GPT-4o), confirming the effectiveness of our proposed framework.

1.5 Thesis Structure

The thesis is organized into four chapters. Chapter 1 introduces the motivation, background, objectives and contributions of this research, outlining the challenges of current AI-assisted AutoPT. Chapter 2 reviews the evolution of AutoPT across expert system-based, RL-based, and LLM-based paradigms, highlighting their respective strengths and limitations that motivate the proposed approach. Chapter 3 presents our prior research on RL-based AutoPT (*DRLRM-PT*) and the design and logic of our proposed knowledge-informed self-reflective AutoPT framework (*RefPentester*). We focus on its implementation, including architecture, components, and reasoning mechanisms for adaptive and interpretable PT automation. After that, in Chapter 4, we introduced the experimental setup, evaluation metrics, and comparative study against baseline models, followed by a discussion on key findings. Finally, Chapter 5 concludes the thesis by summarizing the main contributions of this work, limitations, and future research directions.

Chapter 2

Literature Review

This chapter provides a systematic review of AI-assisted AutoPT, including Expert system-based, RL-based and LLM-based AutoPT methods.

2.1 AI-Assisted Automated Penetration Testing

PT is the practice of simulating real cyberattacks on a network system to uncover security weaknesses before adversaries exploit them. It is widely regarded as one of the most effective ways to assess an organization's security posture [49]. Traditional PT is a manual, labour-intensive process, which requires multiple skilled pentesters to plan and execute multifaceted attack steps (reconnaissance, vulnerability scanning, exploitation, etc.), then interpret the results. This manual approach is time-consuming and costly, which causes significant system downtime. Its success heavily depends on the expertise of pentesters [49]. In an era of rapidly evolving threats and a shortage of cybersecurity professionals, relying solely on manual PT has become impractical to meet demand [50].

To address these challenges, the security industry has increasingly turned to automating the PT process (AutoPT). AutoPT refers to using tools and scripts to perform the PT process with minimal human intervention [51]. By automating repetitive tasks, these tools can increase the PT

process's efficiency, such as automating scanning networks and identifying common vulnerabilities, which is much more efficient than human pentesters [52]. AutoPT tools also bring consistency and repeatability, following the same procedures and reducing human error [52]. This leads to better scalability and allows organizations to test their defences more continuously [52]. In short, AutoPT tools can make the PT process more efficient, scalable, and cost-effective than manual efforts.

However, traditional AutoPT tools have several limitations. To the best of our knowledge, most AutoPT tools are designed to handle one specific task within the PT process. For instance, Nmap [29] is for network scanning, while Metasploit [34] is for exploiting vulnerabilities. These tools excel at their individual functions, but they operate in isolation without higher-level coordination or reasoning. Human pentesters need to manually string together the output of one tool into the input of another, which acts as the "brain" to guide the overall PT process. Scripted automation (e.g., running a fixed sequence of scans, exploits, reports, etc.) can be more efficient but lacks adaptability and intelligence. It means that if something unexpected occurs, the script cannot deviate from its predefined steps [49]. In practice, this means that traditional AutoPT tools lack high-level decision-making and situational awareness capabilities. This shortfall motivates the integration of AI into AutoPT solutions to enhance the intelligence and autonomy.

AI-assisted AutoPT refers to frameworks that incorporate AI techniques to enhance their ability, such as decision-making, planning, data analysis, etc. [53]. It encompasses both knowledge-driven systems that use predefined expert knowledge and learning-driven systems that improve through data-driven learning into the PT automation [50].

AI can equip an AutoPT framework with the ability to analyze complex data, provide corresponding knowledge, learn from prior experience and take optimal actions. For instance, Mohamed et al. [54] use machine learning (ML) models to detect subtle patterns or anomalies in scan results that might indicate hidden vulnerabilities. López-Montero et al. [55] illustrate that AI's planning and reasoning ability can evaluate different PT paths or exploit options and select those most likely to succeed, similar to how a human pentester focuses on high-impact targets.

Unlike rule-based systems, AI-driven approaches can adapt to new environments and evolving threats. This is crucial given the rapidly changing cybersecurity landscape; hard-coded PT scripts can quickly become outdated as network systems and attacker techniques evolve. AI can continually train on new scenarios and remain effective against emerging vulnerabilities [56], thereby achieving a more adaptive and resilient PT process than traditional rule-based automation tools. Moreover, certain AI advancements directly address the pain points of traditional tools. For instance, LLMs such as GPT-4 [57] have demonstrated an ability to parse and interpret large volumes of unstructured data, extract insights, and even generate actionable plans. Integrating such models into the AutoPT workflow can help in digesting the overwhelming output of scanners or logs and summarizing the security-relevant information (e.g., identifying which vulnerabilities are critical) [58]. AI can also aid in the decision-making process of PT, which historically required expert human judgment [59]. By embedding reasoning components, AI-assisted AutoPT systems can make autonomous planning about how to proceed during the PT process, giving them similar abilities to human pentesters.

Multiple AI paradigms have been applied to automate the PT process. This thesis identifies three key categories of AI-assisted AutoPT, each empowered by one different AI approach:

1. **Expert System–Based AutoPT.** These frameworks are built upon rule-based reasoning engines and explicitly predefined expert knowledge to automate the PT workflow. They typically implement logic inference, scripted decision trees, or production rules to select actions, such as vulnerability detection, exploit deployment, based on known patterns. By relying on human-defined PT knowledge bases or rule sets, expert system–based AutoPT frameworks deliver deterministic, transparent, and interpretable behaviour.

2. **RL-Based AutoPT.** These frameworks consider the PT process as a sequential decision-making problem, so the PT process is typically formulated as a Markov Decision Process (MDP) [60], where the pentester (agent) interacts with the target environment to learn optimal attack strategies. Each action corresponds to a PT operation (e.g., vulnerability scanning, privilege escalation, lateral movement), and yields a reward defined by predefined objectives

such as successful exploitation or credential capture. [56]. Through an iterative process of trial and error, the RL agent gradually refines its policy to achieve effective attack planning without requiring a predefined model of the environment [56]. This data-driven adaptability allows RL-based methods to autonomously explore and exploit the target environment, surpassing the rigidity of rule-based systems.

3. **LLM-Based AutoPT.** LLM–based AutoPT frameworks leverage the reasoning, generation, and contextual understanding capabilities of generative AI to automate PT processes. Since many PT processes involve unstructured information, such as network traffic, PT tool output, and exploit code, LLM will be ideal for serving as an engine to analyze this data and plan subsequent actions [61]. In an LLM-empowered PT workflow, an LLM model (e.g., GPT-4) can interpret tool outputs (e.g., enumeration, scan results), summarize key findings, and recommend subsequent actions by leveraging its pretrained domain knowledge [45].

In summary, the three paradigms represent the primary directions through which AI has been applied to AutoPT. To provide a comprehensive perspective, the following sections (Sections 2.2–2.4) introduce these paradigms in detail. Each section presents an overview of the corresponding AI paradigm, discusses representative applications, analyzes its architectural taxonomy, and concludes with a summary of key strengths and limitations. Section 2.5 summarizes the research gaps in the AI-assisted AutoPT domain, with a focus on issues such as sampling efficiency issues, the difficulty of reward specification, interpretability, context retention, hallucinated planning, insufficient experiential learning, and limited PT domain knowledge. The structured literature review builds a foundation for understanding the evolution of AI-assisted AutoPT and motivates us to develop our framework in subsequent chapters.

2.2 Expert System-Based AutoPT

2.2.1 Overview

Expert system-based AutoPT frameworks constitute one of the earliest AI approaches in AutoPT workflows. Grounded in symbolic AI, expert system-based AutoPT relies on explicit, human-predefined rules and logical inference engines to mimic the decision-making behaviour of human experts. These systems aim to codify human expertise into machine-readable formats, enabling the automation of specific PT tasks without requiring data-driven learning or real-time adaptation [40]. Traditional AutoPT tools are constrained by hard-coded workflows, which are mainly due to their dependence on static scripts or sequential logic. Expert systems-based AutoPT offer an improvement by introducing knowledge-driven reasoning, which allows for conditional branching, goal-oriented planning, and deterministic execution of actions based on predefined PT logic[50]. These systems' knowledge and rule base not only enhances transparency but also makes them highly interpretable, which is essential in many cybersecurity settings.

At a high level, expert-system-based AutoPT frameworks are typically built with three core components: (1) a knowledge base containing domain facts and production rules, (2) an inference engine responsible for evaluating and executing rules, and (3) a working memory that tracks current facts, intermediate decisions and system state [62]. This architecture allows rule-based frameworks to reason about components (e.g., network topology, host configurations, vulnerability profiles, potential attack paths) in a structured, deterministic manner. This section provides an in-depth review of expert system–based AutoPT, encompassing representative applications, a taxonomy of system paradigms and concludes with a comparative summary.

2.2.2 Applications

Expert system-based AutoPT frameworks have been widely adopted across various sub-tasks of PT due to their strong interpretability and rule-based decision-making capabilities. Fig. 4 shows the typical four categories of the applications of expert system-based AutoPT frameworks.

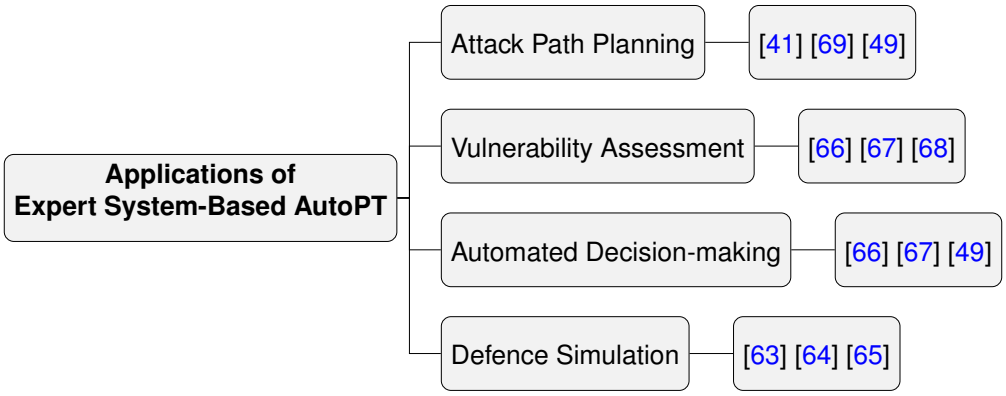

Figure 4: Expert System-Based AutoPT: Classification of Applications

(1) Attack Path Planning. Automated attack path planning represents one of the most significant applications of expert systems in AutoPT. These systems encode predefined rules to systematically generate attack chains to emulate the logical reasoning process of human experts. For example, Wang et al. [41] released AlphaEXP that implements a predefined expert knowledge-driven framework to identify kernel objects, enabling effective path inference in OS-level exploitation scenarios. Wang et al. [69] proposed planning-based approaches that apply formal logical reasoning to traverse segmented environments and uncover potential attack sequences. Wang et al. [49] proposed a framework for AutoPT that offers extensible rule-based representations to support both attack path generation and evaluation.

(2) Vulnerability Assessment. Expert systems have also shown their potential in assessing system vulnerabilities. These systems encode cybersecurity knowledge (e.g., vulnerability types, asset dependencies, and threat models) to evaluate risks without requiring large-scale datasets. For instance, Ghanem et al. [66] proposed ESASCF, which automates security compliance checks through knowledge extraction and rule formalization. Meanwhile, frameworks like Agent-based (BDI) modelling demonstrate how belief-desire-intention logic can model the behaviour of human experts and assess risk levels [67]. Autosploit goes further by integrating rule-based components to evaluate vulnerability exploitability across multiple vectors, which reduces manual workloads [68].

(3) Automated Decision-Making. In practical AutoPT workflows, expert systems play a critical role in high-level decision-making tasks, including test strategy selection and prioritization. By evaluating factors such as asset value, vulnerability severity, and system configuration, these

systems apply deterministic rule sets to guide the testing process systematically. Skandylas et al. present a formalized model for encoding such decision processes[50], while BDI-based agents demonstrate the capacity to simulate strategic behaviours through structured reasoning[67]. Additionally, the unified modelling framework offers decision-rule encoding capabilities for test execution planning, effectively linking high-level attack objectives with concrete, actionable steps[49].

Defence Simulation. Expert systems play a critical role in the cyber defence domain by matching observed adversary behaviours against predefined attack patterns to deliver real-time detection and responses. Enoch et al. proposed HARMer [63], which incorporates predefined rule modules to evaluate and classify simulated cyber attacks. Standen et al. [64] introduced Cyborg, a gym-like platform for training autonomous agents, where expert systems control environment behaviour to ensure realistic and repeatable exercises. Similarly, Applebaum et al. developed an intelligent, automated red team emulation framework in which rule-based components simulate attacker logic, thereby enhancing the realism and complexity of defence simulations[65].

2.2.3 Taxonomy

This subsection presents a taxonomy of expert-system paradigms for automating the PT process. These paradigms are categorized into three types: rule-based, case-based, and hybrid neural-symbolic, as illustrated in Fig. 5. These categories are distinguished by their inference mechanisms and knowledge representations.

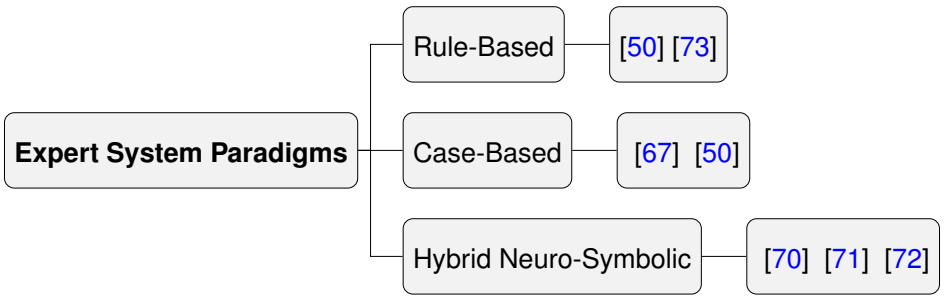

Figure 5: Expert System-Based AutoPT Paradigms

(1) Rule-Based Systems for AutoPT. Rule-based systems are the classical form of expert

systems, built upon IF–THEN logic structures and inference engines (e.g., forward or backward chaining). In AutoPT, such systems are used to encode PT logic as a set of explicit conditional rules triggered by observable states and asset configurations.

These systems excel in transparency and controllability. For instance, Huang et al. formalize the PT process by using rule-based architectures and instantiate it through the ADAPT framework[50]. Similarly, Zhao et al. introduce a tree structure based on a rule base to automate decision chains, reflecting clear logical progression[73]. These approaches demonstrate how predefined rule sets can encode deterministic testing strategies and drive the selection of actions across different PT scenarios.

However, although rule-based systems are effective for predefined tasks, they tend to suffer from rigid logic paths and lack contextual awareness. Chen et al. mentioned in their recent survey that these systems show high performance in constrained environments but struggle with scalability and generalization due to the absence of adaptive learning mechanisms[74].

(2) Case-based Reasoning Systems for AutoPT. In contrast to rule-based systems, case-based reasoning (CBR) systems rely on historical knowledge or trajectory. The CBR systems infer appropriate actions by comparing new situations to past instances based on similarity metrics. CBR systems are used in the AutoPT domain for PT path suggestion or adaptive scenario selection. For example, BDI-based agent models [67] simulate previous PT behaviours to decide future paths, showcasing features aligned with case-based reasoning. Additionally, the ADAPT framework also incorporates historical decision traces to complement rule logic with experience-based adjustments [50].

CBR-based AutoPT systems are advantageous when handling recurring patterns or previously encountered asset configurations. However, they require a well-maintained case database and often exhibit medium complexity in inference and system maintenance [75].

(3) Hybrid Neuro-Symbolic Systems for AutoPT. Hybrid neuro-symbolic-based AutoPT approaches combine symbolic reasoning with data-driven learning methods to balance interpretability and adaptability. These systems explicitly couple a symbolic layer (e.g., comprising rules,

logic programs, task constraints) with a learnable layer to optimize tactical decisions. The symbolic layer encodes PT knowledge, preconditions, and safety rules, while the neural component improves generalization and decision efficiency. Recent studies have demonstrated that hybrid neuro-symbolic-based AutoPT systems can integrate RL and symbolic constraints in various ways. For example, Lei et al. [70] proposed ADAPT, a rule-constrained RL framework for distributed adaptive PT that formalizes an attacker–defender meta-game and employs symbolic rules to guide exploration. Grov et al. [71] illustrated the benefits of neuro-symbolic integration in SOC operations through proof-of-concept studies that enhance the trustworthiness and explainability of ML outputs. Similarly, Kejriwal et al. [72] designed a hybrid neuro-symbolic framework for real-time adversarial-attack detection in autonomous systems, combining learned anomaly detection with rule-based reasoning to enforce safety invariants.

Overall, hybrid neuro-symbolic AutoPT systems act as a conceptual bridge between deterministic expert systems and adaptive learning-based paradigms, offering improved adaptability while maintaining interpretability. However, these integrations often have engineering overhead (e.g., knowledge formalization, interface alignment between symbolic neural modules) and limited scalability when applied to evolving PT environments [76].

2.2.4 Summary

This subsection systematically reviewed the main applications and paradigms of Expert system-based AutoPT. Expert system-based AutoPT paradigms, including rule-based, case-based, and hybrid neuro-symbolic approaches, represent the earliest efforts to formalize the PT process into deterministic, explainable automation frameworks. Despite their advantages, such as interpretability, these paradigms have several limitations. Their adaptability is constrained by static knowledge bases and the absence of self-learning mechanisms, which limit their performance in dynamic or unseen threats. Maintaining and updating rule or case repositories demands intensive expert intervention, which hinders scalability.

2.3 Reinforcement Learning-Based AutoPT

2.3.1 Overview

RL is a trial-and-error-based paradigm where an agent learns to make sequential decisions by interacting with an environment, aiming to maximize the cumulative reward. As shown in Fig. 6, RL involves an agent observing the current state of the environment, selecting an action according to its policy, and receiving a reward and its observation from a new state of the environment, then updating the policy to improve future decisions. This feedback loop makes RL particularly suitable for sequential decision-making tasks.

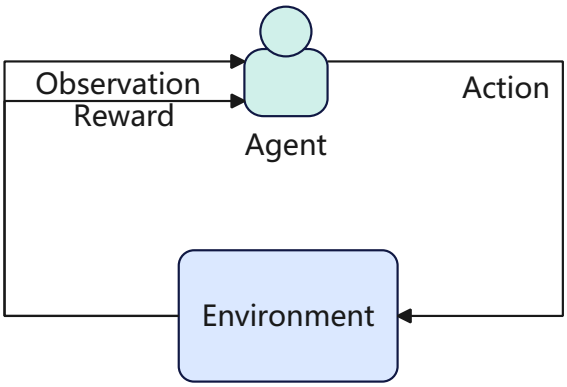

Figure 6: General RL Workflow

In the AutoPT domain, the primary objectives, such as system exploration, vulnerability discovery, and attack path optimization, can be naturally formulated as sequential decision-making problems. The agent can perform an action, the target environment will change accordingly, and provide feedback to the agent. Unlike expert system-based AutoPT approaches that rely on predefined logic and expert predefined rules, RL-based approaches can autonomously learn optimal strategies through interaction with the testing environment. This enables a more adaptive, generalized, and intelligent automation mechanism in PT.

Recently, RL has gained attention in the AutoPT. Prior work based on abstract simulation

environments, such as Microsoft's CyberBattleSim [77], to explore RL-based attack simulations in controlled environments, enabling agents to learn exploitation patterns through interaction with the target environment. Subsequent research extended these concepts toward more complex and realistic PT scenarios, where agents autonomously optimize attack paths, prioritize vulnerabilities, or participate in adversarial red–blue co-training.

Compared with the expert system-based paradigms discussed in the previous section, RL introduces a paradigm shift from static, rule-based decision logic to dynamic, data-driven self-learning. While expert systems emphasize interpretability and deterministic reasoning, their adaptability is limited in novel or evolving network conditions. In contrast, RL agents can generalize across dynamic and uncertain environments but still face challenges such as high training cost, sample inefficiency, and limited transparency, which may hinder real-world deployment.

The following subsections outline two complementary perspectives of RL-based AutoPT research. First, we examine application domains that leverage RL for various PT tasks, such as attack path planning and adversarial training. Then, we present a taxonomy of RL paradigms commonly adopted in the AutoPT domain, analyzing their respective strengths and limitations. The section concludes with a summary that synthesizes key insights and highlights open challenges for future research.

2.3.2 Applications

RL has been increasingly applied to various aspects of AutoPT, enabling agents to autonomously plan, explore, and adapt within complex and dynamic environments. Existing research in RL-based AutoPT can be broadly categorized into four main application domains, including *attack path planning*, *adversarial training*, *multi-agent collaboration*, and *hybrid RL frameworks*, as illustrated in Fig. 7. Each category emphasizes a different application of the RL-based approach to the autonomous PT process.

(1) Attack Path Planning. Previous research on RL-based attack path planning formulated the PT process as a sequential decision-making process, in which agents autonomously explore

the network environment to identify exploitation chains and learn the optimal policy to maximize the cumulative reward. Hu et al. [78] demonstrated the feasibility of using Q-learning to model privilege escalation and lateral movement, showing that agents can learn effective attack sequences without explicit rule sets. Building on this foundation, Kim et al. [79] introduced a CVSS-guided RL framework that prioritizes vulnerabilities according to their severity and exploitability, which improves efficiency compared with purely random and exhaustive search methods.

Recent advancements have emphasized scalability and generalization in more complex environments. Lopez et al. [55] proposed a hybrid RL model integrating Deep Q-Networks (DQN) with environment state abstraction, which enables efficient path optimization under dynamic network conditions. These studies collectively highlight RL's potential by offering adaptive, data-driven exploration and decision-making capabilities.

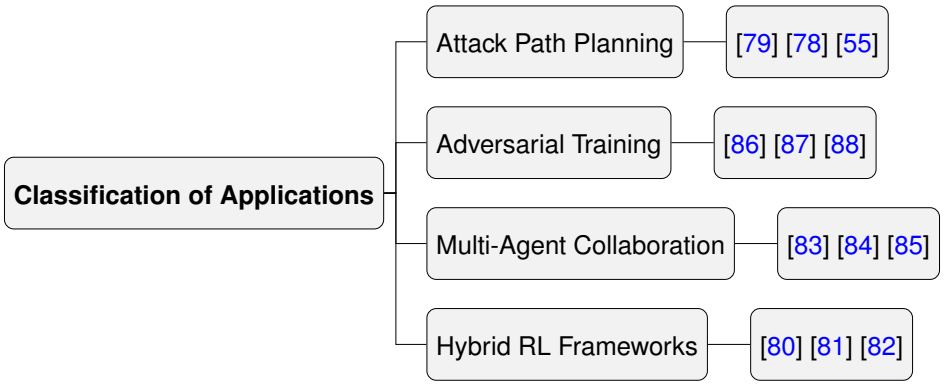

Figure 7: RL-Based AutoPT: Classification of Applications

(2) Adversarial Training. Adversarial training introduces RL-based AutoPT frameworks into the red-blue team paradigm, where an attacker agent and a defender agent are co-trained through adversarial RL. The attacker learns to maximize exploitation success, while the defender seeks to minimize the attacker's reward by strengthening system defences and dynamically adapting policies. This paradigm enables both agents to iteratively improve their strategy, which leads to more robust and realistic PT simulations.

Previous researchers, such as Chen et al. [87], analyzed the vulnerability of RL agents under adversarial perturbations and proposed defensive training strategies to enhance policy robustness.

Building upon this foundation, Piplai et al. [86] applied adversarial RL to incorporate domain knowledge to guide both attacker and defender agents in exploring complex network topologies. Their framework demonstrated that co-evolutionary training can improve the adaptability of both agents. Farooq et al. [88] introduced a hybrid adversarial-defensive learning scheme that combines supervised and RL objectives to accelerate convergence and enhance defence generalization across dynamic and unseen scenarios. These studies illustrate that adversarial RL-based methods provide a promising direction for AutoPT. By simulating a realistic red–blue team, these agents can identify target systems' weaknesses while learning resilient defence strategies.

(3) Multi-Agent Collaboration. Recent advancements have extended RL-based AutoPT frameworks from single-agent to multi-agent architectures, enabling distributed and scalable PT strategies across complex network systems. In such systems, multiple agents operate simultaneously, each agent responsible for one specific PT subtask (e.g., reconnaissance, privilege escalation) and coordinate through shared rewards to achieve one objective.

Finistrella et al. [83] proposed a cooperative multi-agent reinforcement learning (MARL) framework for large-scale enterprise networks, where multiple agents learn complementary behaviours via hierarchical policy decomposition to efficiently manage overlapping attack surfaces. Tran et al. [84] introduced a cascaded RL architecture that decomposes attack decision-making into multiple learning layers, improving the adaptability and coordination among agents operating at different abstraction levels of the environment. Pagano et al. [85] expanded this paradigm by incorporating communication-aware MARL mechanisms, such as shared replay buffers and inter-agent message passing, which significantly accelerate policy convergence and enhance coverage of network states. These studies collectively demonstrate that multi-agent collaboration offers substantial benefits in exploration efficiency, scalability, and robustness for AutoPT.

(4) Hybrid RL Frameworks. Hybrid RL-based AutoPT frameworks integrate RL with other methods, such as expert systems and knowledge graphs, to achieve a balance between autonomous adaptability and rule-based interpretability. These systems aim to overcome the limitations of data-driven approaches by combining the exploration capability of RL with structured domain

knowledge or human feedback to enhance the training efficiency or generalization.

Zhou et al. [80] proposed a framework that incorporates symbolic reasoning and human-in-the-loop supervision into the RL training process, enabling adaptive policy refinement while preserving interpretability in PT decision-making. Piplai et al. [81] introduced an Offline RL model augmented with a cybersecurity knowledge graph (CKG), which guides agent exploration by embedding domain-specific constraints and prior knowledge into the policy learning process, thereby improving sample efficiency and safety in network environments. Similarly, Liu et al. [82] developed a hierarchical hybrid RL framework for AutoPT, which combines hierarchical task decomposition with RL's optimization to achieve more structured and goal-oriented attack planning. Previous studies demonstrate that hybrid RL frameworks enhance AutoPT systems by integrating human knowledge with an adaptive RL policy. They provide a promising method to balance explainability, data efficiency, and generalization ability.

2.3.3 Taxonomy

This subsection presents a taxonomy of RL paradigms that have been widely applied to the AutoPT process. These paradigms are categorized into three classes, including *Value-Based*, *Policy Optimization*, and *Advanced Extensions*, as shown in Fig. 8. The first two categories represent the two different agent learning architectures, while the third category encompasses extended paradigms, including multi-agent RL, adversarial RL, and knowledge-integrated RL.

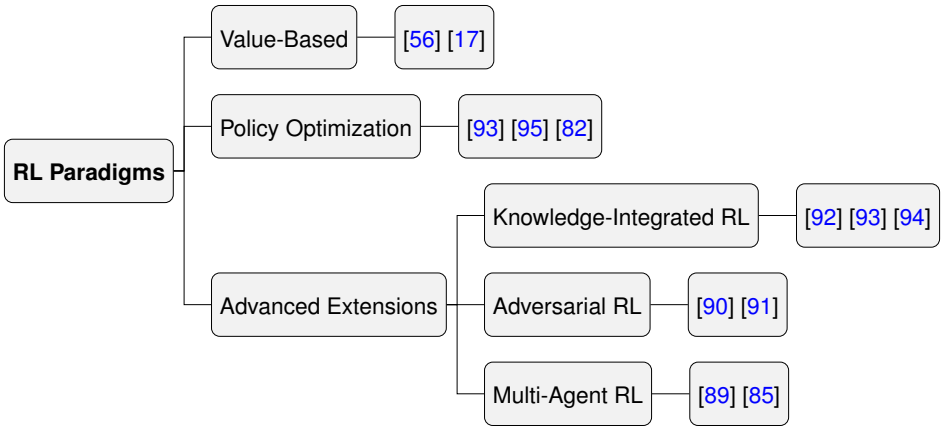

Figure 8: RL-Based AutoPT Paradigms

(1) Value-Based RL paradigm for AutoPT. Value-based RL methods for AutoPT learn an action-value function $Q(s, a)$ that estimates the expected cumulative reward of one action a in a given network state s. By iteratively updating $Q(s, a)$ through interaction with s, the RL agent gradually learns an optimal attack policy without relying on predefined rules [56].

Value-based approaches are typically implemented using Deep Q-Networks (DQNs). Schwartz et al. [56] introduced DQN-based AutoPT frameworks that formalize the PT process as a Markov Decision Process (MDP), enabling the agent to explore and exploit target networks autonomously. This work demonstrated that a value-based RL agent can effectively generalize over significant and partially observable environments by incorporating reward shaping and state abstraction. Building on this foundation, Samad et al. [17] improved the DQN paradigm by integrating hierarchical state encoding and prioritized experience replay to enhance exploration efficiency and mitigate convergence instability. These improvements address challenges such as sparse rewards and redundant exploration, which are essential in a large-scale network environment.

Value-based AutoPT frameworks have advantages in learn optimal policies through direct interaction with the target environment without predefined human knowledge. However, they are limited to discrete action spaces and deterministic environments, which limit their generalization ability.

(2) Policy Optimization-based RL paradigm for AutoPT. Policy Optimization methods for AutoPT aim to optimize a parameterized policy $\pi_\theta(a|s)$, which defines the probability of selecting an action a given an environment state s. Instead of indirectly inferring optimal a through $Q(s, a)$, these methods optimize the policy parameters θ via gradient-based updates to maximize the expected cumulative reward. Compared with value-based approaches, policy optimization-based AutoPT frameworks achieve smoother convergence and demonstrate stronger adaptability to large-scale action spaces, which is essential for handling real-world PT tasks.

Li et al. [93] proposed an efficient policy gradient framework for AutoPT, in which the RL agent dynamically adjusts attack strategies through stochastic gradient updates to maximize the

expected reward of successful exploitation. This approach demonstrates that the policy optimization method yields more stable exploration compared to value-based methods, such as Q-learning, when facing partially observable environments. Nguyen et al. [95] designed an Actor–Critic-based AutoPT framework, which formalizes the attack planning process as an MDP where the actor network generates candidate attack actions and the critic evaluates their expected reward. Their results confirm that Actor–Critic optimization enhances adaptability and convergence speed in high-dimensional network environments. Recently, Liu et al. [82] developed a hierarchical RL framework for automated PT in which policy optimization is performed at two levels—network selection and host-level exploitation. Each level maintains an independent policy model optimized through reward feedback, enabling modular decision-making without explicit coordination among agents. This hierarchical RL structure allows efficient decomposition of large-scale attack graphs and improves training stability across different environments.

These studies exemplify the growing adoption of policy optimization techniques in AutoPT. By directly updating the policy parameters through gradient-based methods, policy optimization-based frameworks achieve greater flexibility and generalization, especially when handling high-dimensional, dynamic and complex PT environments.

(3) Advanced RL paradigm for AutoPT. While value-based and policy optimization methods have established the foundations of RL-based AutoPT, recent advancements have expanded towards more sophisticated paradigms that integrate prior knowledge, adversarial dynamics, and multi-agent collaboration. These paradigms are collected and categories in *Advanced Extensions* category in Fig. 8, aim to improve the efficiency, generalization, and scalability of RL-based AutoPT.

Knowledge-Integrated RL-based AutoPT. Knowledge-Integrated RL frameworks enhance the learning process by embedding human-predefined PT domain knowledge (e.g., vulnerability databases, attack ontologies)into the RL pipeline. This integration allows the agent to leverage prior knowledge to guide its action choosing, which can improve the training efficiency. Piplai et al. [92] introduced a knowledge-informed RL model that embeds structured cybersecurity

knowledge graphs to guide agent exploration in the target environment. Li et al. [93] proposed an efficient AutoPT framework that combines state abstraction with hierarchical reward shaping, achieving higher sample efficiency in large-scale network environments. Similarly, Moreno et al. [94] incorporated ontology-based vulnerability mapping into RL decision-making, improving interpretability and convergence stability. The Knowledge-Integrated RL-based AutoPT provides a hybrid learning paradigm that bridges the gap between predefined reasoning and autonomous exploration, which can enhance the efficiency and interpretability.

Adversarial RL-based AutoPT. Adversarial RL paradigms model AutoPT as a dynamic interaction between competing agents (e.g., attacker vs. defender), thereby capturing the inherently adversarial nature of cybersecurity environments. Elderman et al. [90] formulated a Markov game–based framework in which red-team (attacker) and blue-team (defender) RL agents co-evolve attack and defence strategies, enhancing robustness and adaptability under uncertainty. Oh et al. [91] applied adversarial training to simulate network defenders that actively adapt against attacker policies, leading to more realistic and resilient AutoPT scenarios. These studies demonstrate that adversarial RL can significantly improve the robustness and transferability of learned attack strategies. However, this paradigm also introduces challenges in maintaining training stability and balancing co-evolutionary dynamics, which remain active research areas.

Multi-Agent RL-based AutoPT. MARL extends AutoPT to distributed and cooperative environments, where multiple agents collaborate to achieve complex PT objectives. Ghanem et al. [89] proposed a hierarchical MARL architecture in which specialized agents coordinate across different PT stages via shared reward signals. Pagano et al. [85] further introduced a communication-aware MARL framework that integrates inter-agent message passing and shared replay buffers to improve coordination and convergence in large-scale enterprise networks. Compared to single-agent approaches, MARL frameworks offer improved scalability, exploration diversity, and robustness in heterogeneous environments.

Overall, advanced RL paradigms extend AutoPT beyond traditional single-agent learning by

introducing structured knowledge integration, adversarial co-evolution, and multi-agent coordination. These developments collectively enhance the adaptability, efficiency, and realism of AutoPT.

2.3.4 Summary

This subsection systematically reviewed the four main applications and paradigms of RL-based AutoPT. The RL-based AutoPT methods have progressed from value-based and policy optimization methods toward more advanced paradigms that integrate knowledge, adversarial dynamics, and multi-agent collaboration. However, despite these advantages, RL-based frameworks still face several challenges, including sparse reward design, inefficient credit assignment, limited generalization to unseen topologies, and the lack of interpretability in policy reasoning. Recent advances in LLMs offer a promising avenue to address these limitations by introducing powerful natural language reasoning, contextual memory, and tool-using abilities into AutoPT systems. The following section focuses on LLM-based AutoPT, which leverages language-driven reasoning and instruction-following capabilities to enable higher-level automation, explainability, and human–AI collaboration in the PT domain.

2.4 Large Language Model-Based AutoPT

2.4.1 Overview

LLMs-empowered AutoPT enabling frameworks to perform complex reasoning, contextual understanding, and decision-making based on natural language knowledge. In the AutoPT domain, the PT stages (e.g., reconnaissance, vulnerability identification) can be reformulated as MDP/POMDP reasoning tasks. Given a target context of the environment, an LLM-based agent can parse the input, infer actionable insights, and autonomously generate or adapt testing procedures. Compared with expert system-based and RL-based paradigms that rely on handcrafted rules or explicit environment modelling, LLM-based AutoPT frameworks rely on structured reasoning and

knowledge grounding across multiple domains of training data, which enable them to achieve high adaptability and generalization.

The following section classifies current LLM-based AutoPT research into two major application domains and introduces the taxonomy of LLM-based AutoPT systems' paradigm, as well as a summary of open challenges for future research.

2.4.2 Applications

(1) Task Planning, Execution and Validation. LLM-based PT task planning and execution extend RL-based AutoPT systems by introducing reasoning, contextual memory, and dynamic adaptation. These models act as autonomous agents that can interpret user intents, analyze the current environment state, make high-level planning to decompose the PT process into multiple subtasks, and generate executable actions aligned with the testing environment.

Happe et al. [96] demonstrate that LLMs are capable of autonomously identifying potential attack vectors and planning exploitation steps without explicit supervision. Their study highlighted that LLMs can simulate multi-step PT reasoning processes similar to those of experienced human experts, establishing the foundation for autonomous task decomposition. Building upon this direction, Deng et al. [45] proposed PentestGPT, an AutoPT framework where GPT-based agents interact with command-line tools and knowledge bases to perform task planning, vulnerability enumeration, and exploit execution. The system maintains an internal state memory and reasoning trace, enabling iterative validation and correction across PT stages. This hybrid design integrates both reasoning-driven and tool-driven capabilities, significantly improving task coherence and reproducibility.

Xu et al. [46] introduced AutoAttacker, which reformulates the entire PT workflow as a goal-conditioned reasoning process. Their framework leverages LLMs to translate environment feedback (e.g., scan results, CVE matches) into structured task representations and dynamically adapt exploitation actions. The validation loop ensures that each generated payload or command is tested, logged, and refined through feedback alignment, thereby establishing a closed

reasoning–execution–validation loop. Huang et al. [97] proposed a framework called PenHeal, a dual-phase framework coupling vulnerability detection and automated repair generation. PenHeal demonstrates the potential of LLMs in bidirectional task reasoning, including testing, validation, and patch recommendation, and integrates them into one continuous workflow. This bidirectionality enhances PT task traceability and fosters explainability across multiple PT stages. These studies mark a transition from static rule execution to adaptive, reflective, and context-aware PT automation.

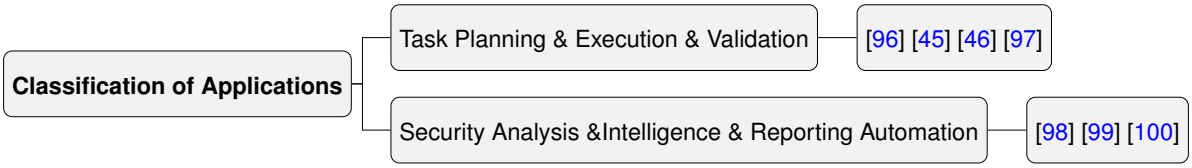

Figure 9: LLM-Based AutoPT: Classification of Applications

(2) Security Analysis, Intelligence and Reporting Automation. LLM-based security intelligence frameworks mainly focus on proactive vulnerability reasoning, contextual threat analysis, and automated reporting. While traditional vulnerability assessment systems rely on signature matching and manually curated CVE databases, LLMs enable semantic understanding of security data, such as system logs, which allows comprehensive and explainable analysis.

Palma et al. [98] introduced a multi-agent LLM framework for Cyber Threat Intelligence (CTI) automation. One agent performs threat extraction from unstructured data (e.g., CISA, NVD, and OSINT sources), while another agent conducts reasoning-based correlation across indicators, tactics, and impacted assets. This framework allows the system to generate multi-source intelligence reports as well as contextualized attack-path hypotheses. Kravsovec et al. [99] proposed an LLM-powered security reporting and risk assessment assistant that automates the documentation phase of PT. They use RAG to integrate PT logs, CVE evidence, and mitigation recommendations into natural language reports aligned with professional templates (e.g., OWASP[48], MITRE ATT&CK[47]). By reasoning over structured and unstructured data, the system enhances report consistency, reduces analyst workload, and mitigates human bias in risk prioritization. Ren et al. [100] explored the use of LLMs for constructing Security Knowledge Graphs (SKGs) that

unify vulnerability descriptions, exploit codes, and dependency chains. This framework employs LLM-based entity linking and relation extraction to convert unstructured vulnerability text into machine-readable triples, which can be queried to support vulnerability propagation analysis and cross-system risk reasoning. This work highlights LLMs' ability to act as semantic intermediaries between human-readable reports and automated reasoning pipelines, enabling continuous intelligence enrichment for AutoPT systems.

These studies indicate that LLM-based Security Analysis and Intelligence systems are evolving from reactive information retrieval toward contextualized, explainable, and automated threat interpretation. Moreover, by integrating reporting automation into intelligence pipelines, these frameworks can support traditional AutoPT tools from vulnerability discovery into a complete, reasoning-driven security intelligence ecosystem.

2.4.3 Taxonomy

(1) Knowledge Integration. Knowledge integration in LLM-based AutoPT frameworks focuses on embedding structured cybersecurity domain knowledge into LLM pipelines. Since LLM is trained on a large amount of data across different domains, and the cybersecurity data on the internet is in a small portion, this may limit its performance on PT tasks. This paradigm uses LLMs' reasoning ability with retrieval, predefined and structured knowledge and data to maintain factual accuracy throughout the PT process. Such integration enables the model to not only recall specific cybersecurity knowledge but also dynamically adapt to specific PT environments.

There is some prior research that focuses on this domain. Shen et al. [61] proposed PentestAgent, a knowledge-enhanced AutoPT framework that leverages LLMs as central controllers orchestrating multi-stage PT workflows. The system integrates RAG pipelines with structured knowledge bases, such as MITRE ATT&CK[47], allowing the LLM agent to retrieve environment-specific and vulnerability patterns knowledge in real time. This hybrid design bridges language reasoning and factual retrieval, thereby mitigating hallucination and improving exploit reproducibility.

Similarly, Wu et al. [101] introduced CurriculumPT, an LLM-based AutoPT framework that combines knowledge integration with curriculum-guided task learning. This framework employs a hierarchical memory and task scheduler to structure knowledge from previous PT sessions into an Experience Knowledge Base (EKB), which is continuously queried and expanded as agents perform reconnaissance, exploitation, and reporting tasks. The EKB stores contextualized vulnerability data and reasoning traces, allowing LLM agents to generalize learned patterns to unseen targets and maintain long-term reasoning. This design demonstrates that knowledge integration not only enhances factual consistency but also improves the sample efficiency and adaptability of LLM-based AutoPT.

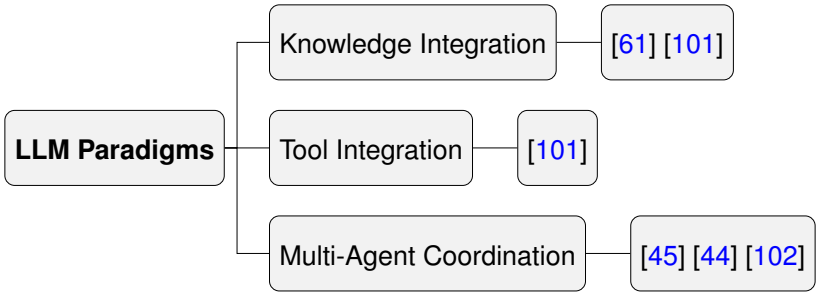

Figure 10: LLM-Based AutoPT Paradigms

(2) Tool Integration. LLM-based tool integration methods use LLMs' reasoning ability with external PT utilities, enabling them to perform system-level actions. Instead of acting as purely linguistic planners, LLMs in this paradigm function as autonomous controllers that translate high-level tactics into executable commands, interpret tool outputs, and iteratively refine their reasoning based on feedback.

Wu et al. [101] proposed CurriculumPT with a dedicated tool integration layer that bridges the LLM reasoning core and standard PT tools such as Nmap[29], Metasploit[34], and Burp Suite[31]. The framework enables bidirectional communication between the LLM and external tools via standardized APIs and structured output parsing. Specifically, the LLM agent generates commands aligned with its task plan, triggers external scans or exploitation routines, and parses returned results to update its internal reasoning state. This design empowers the agent to continuously validate

hypotheses, verify discovered vulnerabilities, and autonomously adapt its strategy in dynamic environments. CurriculumPT demonstrates how LLM-based AutoPT frameworks can move beyond static command generation toward interactive, tool-grounded intelligence. By embedding execution feedback into the reasoning loop, the LLM becomes capable of reflecting on the success or failure of actions and adjusting subsequent steps accordingly.

(3) Multi-Agent Coordination.

LLM-based multi-agent AutoPT frameworks aim to decompose the complex PT workflow into multiple subtasks and assign each subtask to one specific agent, agents that operate under a shared reasoning context, instead of relying on a single agent to plan and execute all testing stages. This paradigm coordination mechanisms ensure synchronized task scheduling and reasoning consistency while enhancing scalability and reducing hallucination during the PT process.

Some researchers did prior research on this domain. Deng et al. [45] introduced PentestGPT, a multi-agent framework in which the main LLM agent coordinates subordinate task-specific models to handle scanning, exploitation, and result validation. The system employs a hierarchical control loop, where the controller agent interprets overall PT objectives, delegates subtasks to execution agents, and validates outcomes through iterative feedback. Similarly, Shen et al. [44] proposed PentestAgent, a multi-agent AutoPT framework where specialized LLM agents collaborate via a shared memory and reflection channel. The coordinator agent maintains global context—tracking vulnerabilities, system states, and intermediate findings, while domain agents (e.g., network scanner, exploit verifier, and report generator) communicate asynchronously through structured message passing. This architecture enables concurrent task execution and reduces cognitive overload on any single agent, thus achieving adaptive reasoning and cooperative planning under partial observability. Kong et al. [102] proposed VulnBot, an LLM-based agent ecosystem that combines autonomous vulnerability scanning, exploit generation, and patch recommendation. Each agent is governed by a meta-controller that ensures consistency across knowledge retrieval, tool invocation, and policy adaptation. The framework also integrates feedback loops for inter-agent negotiation, where conflicting assessments are resolved through voting and consensus reasoning. This design

demonstrates the potential of multi-agent LLM coordination to produce more interpretable, reliable, and reproducible PT outcomes.

The LLM-based multi-agent paradigm enables systems to scale across distributed workflows while maintaining reasoning coherence and task specialization. By leveraging role differentiation, shared context, and reflective communication, these frameworks transform PT from sequential task execution into a collaborative, goal-oriented reasoning.

2.4.4 Summary

This subsection systematically reviewed the main applications and paradigms of LLM-based AutoPT. These frameworks introduce natural language reasoning, contextual understanding, and self-reflective decision-making into PT. However, challenges remain in the sustainability of high-level task planning, hallucination, and limited domain knowledge.

2.5 Research Gap

Despite recent advancements in AI-assisted AutoPT, several critical research gaps remain unresolved. As shown in Table 2, expert system-based AutoPT frameworks mainly rely on predefined rules and encoded knowledge, which provides transparency but results in limited adaptability and high maintenance overhead. To keep its performance on new network architectures, traffic or systems, experts need to update the rules and knowledge regularly. RL-based AutoPT frameworks are capable of learning strategies through interaction with the environment; however, they often suffer from poor sample efficiency, sparse reward specification, limited interpretability, and difficulty in generalizing across different network topologies. These limitations constrain their effectiveness in real-world PT scenarios.

LLM-based AutoPT frameworks, which represent an emerging paradigm, also exhibit distinct shortcomings that restrict their real-world applicability. First, since PT is a multi-stage long-term process, current LLMs struggle with long-term context retention, which is essential for maintaining

Table 2: Research Gaps of AI-assisted AutoPT

Paradigm	Research Gap
Expert system-based	1) Limited adaptability 2) High maintenance costs
RL-based	1) Poor sampling efficiency 2) Sparse reward issue 3) Interpretability 4) The gap between simulation and real-world scenario
LLM-based	1) Short-sighted planning 2) Hallucinations 3) Hard to learn from multiple trial-and-error 4) PT domain knowledge imbalance

coherence and making a high-level plan across PT processes. Second, since in the current stage, LLMs' hallucination is unavoidable, and PT requires multiple trial-and-error, current LLM-based AutoPT frameworks lack a structured mechanism to learn, adapt and reflect on previous failed actions, which can cause repeated errors and limit their decision-making ability. Third, compared to other public knowledge from the internet, such as history and geography, PT knowledge is in a small portion, which can potentially limit the performance of an LLM-based system to solve PT tasks. These systems suffer from insufficient domain knowledge, which often leads to hallucinated command generation or short-sighted planning. These challenges highlight the need to develop a new AutoPT framework that addresses these limitations and can be applied in real-world PT scenarios.

2.6 Summary

In this chapter, we reviewed the evolution of AI-assisted AutoPT frameworks across three main paradigms: expert systems, RL and LLMs. Expert system-based AutoPT relies on predefined rules and encoded knowledge, providing transparency and deterministic reasoning but lacking adaptability to dynamic environments. RL-based AutoPT introduces autonomous decision-making through trial-and-error learning, enabling adaptive attack strategy generation. However, challenges such

as sparse rewards, limited interpretability, and poor generalization persist. Recent advancements in LLM-based AutoPT further extend automation by introducing natural language reasoning, contextual understanding, and self-reflective planning. The literature reveals a clear progression from static rule execution to adaptive and reflective intelligence in AutoPT. Nevertheless, existing LLM-based AutoPT frameworks still have limitations, such as PT task planning, hallucination and limited PT domain knowledge. To address these issues, the next chapter introduces our prior work and the proposed *RefPentester* framework, which integrates structured cybersecurity knowledge, self-reflective feedback loops, PT stage machine and contextual memory to achieve coherent, adaptive, and interpretable PT automation.

Chapter 3

Knowledge-Informed Self-Reflective PT Framework Based on LLMs

3.1 Overview

In this chapter, first, we introduced our previous work, which attempts to address some of the limitations of using RL-based AutoPT methods. Based on that, we introduced the design and implementation of our proposed knowledge-informed AutoPT framework based on LLM with a self-reflection mechanism, referred to as *RefPentester*, which enables the framework to solve real-world multi-stage PT problems in a more coherent, adaptive, and context-aware manner.

This chapter is organized as follows. Section 3.2 reviews our prior study on an RL-based AutoPT framework enhanced with reward machines, which serves as the foundational work for addressing sampling inefficiency, sparse reward, and interpretability issues. Section 3.3 discusses the design rationale, objectives, and logical structure of the proposed knowledge-informed and self-reflective AutoPT framework. Section 3.4 details the implementation of the *RefPentester* framework, including its core components: the Process Navigator, Generator, Reflector, Success Log, and Failure Log, as well as the mechanisms for knowledge retrieval and self-reflective optimization.

3.2 Preliminaries

To address the limitations identified in Section 2.5, we previously conducted a study that explored RL as a foundation for AutoPT. The key motivation behind this work was to address three fundamental challenges mentioned in the previous section. 1) In PT scenario, the action space is typically huge, and PT process requires a large amount of trial-and-error, which can limit RL-based methods' sampling efficiency; 2) The RL-based agent will get rewards when it reveals a vulnerability or credential, but it will not get any feedback before that, which causing the sparse reward problem, and limit the training efficiency; 3) Traditional RL-based AutoPT methods often lack interpretability, a critical factor in cybersecurity for specialist analysis.

To mitigate these limitations, in our previous work, we proposed a knowledge-informed AutoPT framework based on RL with reward machines [103], which we called DRLRM-PT, as shown in Fig. 11. This work aimed to integrate prior PT domain knowledge, which is encoded in RMs as guidance for more efficient PT policy training, provide more detailed rewards and also have interpretability for specialists to analyze.

Since PT is a sequential decision-making problem, the next state only relies on the current state and the action, while the inherent state of the network or computer system is not fully observable so that it can be modelled as a partially observable Markov decision-making process (POMDP), and the optimal PT policy can be obtained through using RL/Deep Reinforcement Learning (DRL) algorithms.

The DRLRM-PT framework utilizes an agent acting as a pen-tester within a target network composed of hosts, firewalls, routers, and communication channels. The agent selects actions from the action space and gathers observations, such as discovered vulnerabilities or leaked credentials, then obtains rewards through the reward machine. Rewards measure the usefulness of actions, with positive rewards for gaining control of critical resources and negative ones otherwise. The agent's goal is to learn an optimal policy that maximizes cumulative rewards through repeated environment interactions. To address the large action and observation spaces, the policy is represented by deep neural networks.

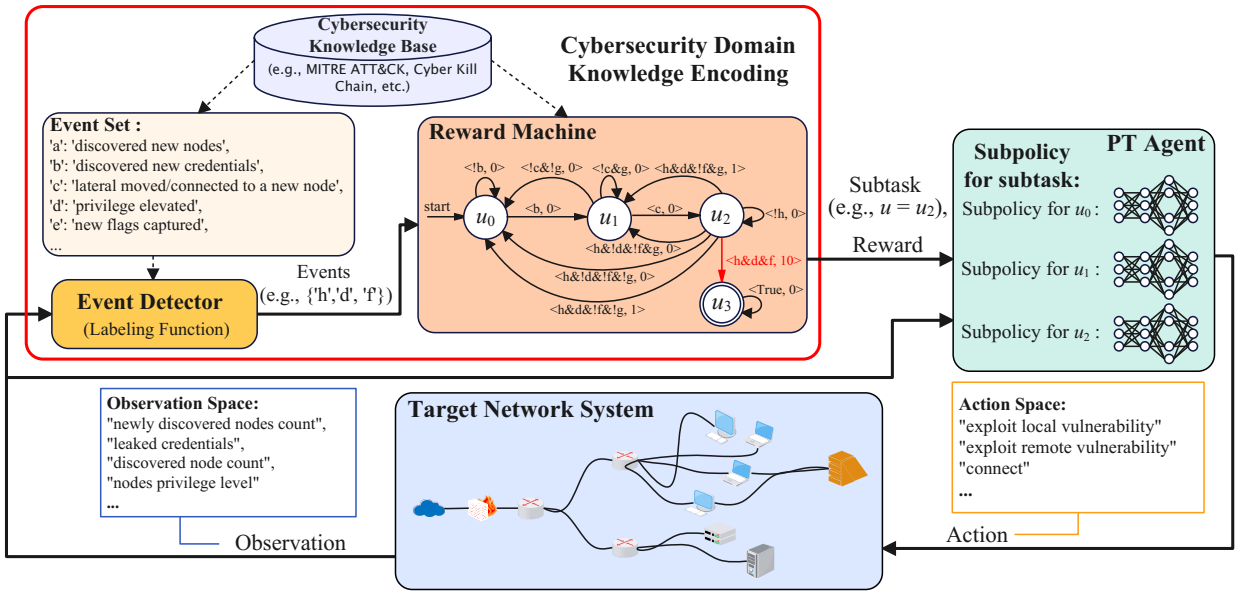

Figure 11: The Proposed DRLRM-PT Framework

The reward machine (RM) is a state machine [104]. Basically, it has two essential functions. First is to decompose the PT process into multiple subtasks, such as "discover credentials", and organize it using a state machine's structure. Second, we use the reward machine to serve as a reward function to provide rewards for each state transition.

The input of RMs is events detected by the Event Detector. The function of the Event Detector is to identify occurring events. We refer to public cybersecurity knowledge bases, such as MITRE ATT&CK [47], to design a PT event set, where each event from the event set represents a successful execution of adversary tactics. As shown in Table 3, we defined eight types of events. The events are used by RMs to identify current PT subtasks and provide rewards accordingly. For example, 'a' for "discovered new nodes" and 'b' for "discovered new credentials".

Each state in the RM, such as u_0 and u_1, represents a subtask in the PT process, respectively. For each subtask, we trained a subpolicy to maximize the cumulative rewards. We called the overall algorithm deep Q-learning with RM algorithm (DQRM), which can decompose the PT policy into multiple subpolicies for each subtask, and train all the subpolicies simultaneously. We use the DQRM to optimize the overall policy.

In the experimental study, we compared our proposed algorithm with the base algorithm (DQN)

on a simulation environment, i.e., CyberBattleSim [77]. We evaluate our algorithm with two different RMs on two different network topologies. We found that the DQRM algorithm has better training efficiency compared to DQN in two network topologies, which means DQRM can have a higher cumulative reward in fewer steps, and different RM designs have a significant influence on the performance of the DQRM algorithm.

While the prior work addressed several of the limitations discussed in Section 2.5, such as the sparse reward issue, poor sampling efficiency and interpretability, it still exhibits notable shortcomings. First, the learned policies often lack strong generalization, limiting their applicability beyond the specific network topologies used in training. Second, the simulation environment (e.g., CyberBattleSim [77]) remains highly abstracted and oversimplified compared to real-world enterprise networks, creating a gap between experimental results and practical deployment.

To overcome these challenges, our current research introduces LLMs as a foundational method for AutoPT. LLMs, trained on vast amounts of diverse data, offer strong reasoning capabilities and improved generalization across varied scenarios, which makes it has the potential to serve as a 'brain' for AutoPT. Nevertheless, LLMs also bring their own limitations. As mentioned in Section 2.5, they often suffer from short-sighted planning and may produce hallucinations, which undermine their reliability. Furthermore, LLMs are not inherently optimized for iterative trial-and-error learning, making it difficult for them to adapt efficiently in dynamic environments. Finally, their knowledge of the PT domain is uneven, leading to imbalances between general reasoning ability and domain-specific expertise. These limitations highlight the need for a hybrid framework that can leverage the strengths of LLMs while mitigating their weaknesses, forming the foundation for the approach proposed in this thesis.

These limitations indicate that neither traditional RL approaches nor LLM-based methods alone are sufficient for achieving robust and reliable AutoPT. This motivates the design of our proposed framework, which aims to integrate the reasoning and generalization strengths of LLMs with structured mechanisms that address their limitations, thereby advancing the state of AutoPT. The details of our proposed framework will be introduced in the following section.

Table 3: Event Set of PT ($\mathcal{P}$)

Event Symbol	Event Description
'a'	Discovered new nodes
'b'	Discovered new credentials
'c'	Lateral moved (connected to a new node)
'd'	Privilege elevated
'e'	New flags captured (new target nodes owned)
'f'	Achieved the PT goal
'g'	Has unused credentials
'h'	Taken action to elevate the privilege

3.3 Framework Design and Logic

The motivation for designing the *RefPentester* framework stems from the limitations observed in our previous work on RL-based AutoPT. As discussed in Section 3.2, RL-based methods such as DRLRM-PT improved automation through reward decomposition but still faced challenges, such as limited generalization and the gap between the simulation platform and the real-world PT scenario. Meanwhile, recent LLM-based systems introduced general reasoning ability; however, they suffered from hallucinations, inconsistent task planning, and a lack of PT domain knowledge.

3.3.1 Design Rationale

To address the limitations mentioned in the previous section, the proposed *RefPentester* framework is designed following a knowledge-informed and self-reflective paradigm. The overall framework aims to ensure coherent reasoning across multi-stage PT workflows while maintaining interpretability and adaptability to real-world PT scenarios, as well as reducing hallucinations. Specifically, each component in the framework fulfills a distinct and complementary role within a unified decision cycle: the *Process Navigator* establishes stage awareness and retrieves high-level PT knowledge; the *Generator* produces context-grounded operational guidance; the *Reflector* performs self-evaluation and strategy refinement; and the *Success/Failure Logs* maintain memory for

long-term trajectory and short-term trial-and-error evaluation.

3.3.2 Design Objectives

To overcome these issues, the proposed framework is designed with three core objectives:

- **Coherent multi-stage reasoning:** ensuring that the agent's decisions remain aligned with the current PT process, following a clear and state-aware workflow.
- **Knowledge-grounded decision-making:** leveraging structured cybersecurity knowledge to minimize hallucinations and improve domain awareness.
- **Self-reflective learning:** enabling the model to evaluate its own reasoning trajectories, identify incorrect actions, and iteratively refine its strategies from previous trial-and-error.

3.3.3 Design Logic

To fulfill these objectives, the framework integrates five synergistic components, each addressing a specific aspect of PT automation:

- *Process Navigator:* models the PT procedure as a seven-stage state machine that enforces global goal awareness and sequential logic across stages. It also encodes a hierarchical RAG pipeline, which serves as the knowledge interface that retrieves relevant information from the PT Knowledge VDB in a coarse-to-fine manner, supporting stage-dependent reasoning granularity.
- *Generator:* performs contextual reasoning and generates operational guidance based on human instruction, previous trajectory, retrieved knowledge and the current PT stage, ensuring that the operational guidance is detailed and grounded.
- *Reflector:* monitors and evaluates the executed operation and its corresponding feedback from the target machine, analyzing the potential failure reasons to refine future decision-making.

- *Success/Failure Logs:* store historical trajectory traces to maintain the current PT process and enable reflection-based adaptation.

In summary, Section 3.3 elaborates the theoretical rationale and logical design of the RefPentester framework, defining how each component conceptually interacts to fulfill the core objectives. Building upon this design foundation, the next section (3.4) details the concrete methodology and implementation of each module, including the framework architecture, knowledge preparation, and agent workflow.

3.4 Methodology

3.4.1 Framework Overview

The proposed *RefPentester* framework consists of five components: **Process Navigator**, **Generator**, **Reflector**, **Success Log**, and **Failure Log**, as illustrated in Fig. 12. It is a human-in-the-loop paradigm, which involves human operators to execute operational guidance provided by *RefPentester*. The notations included in the framework are illustrated in Table 4.

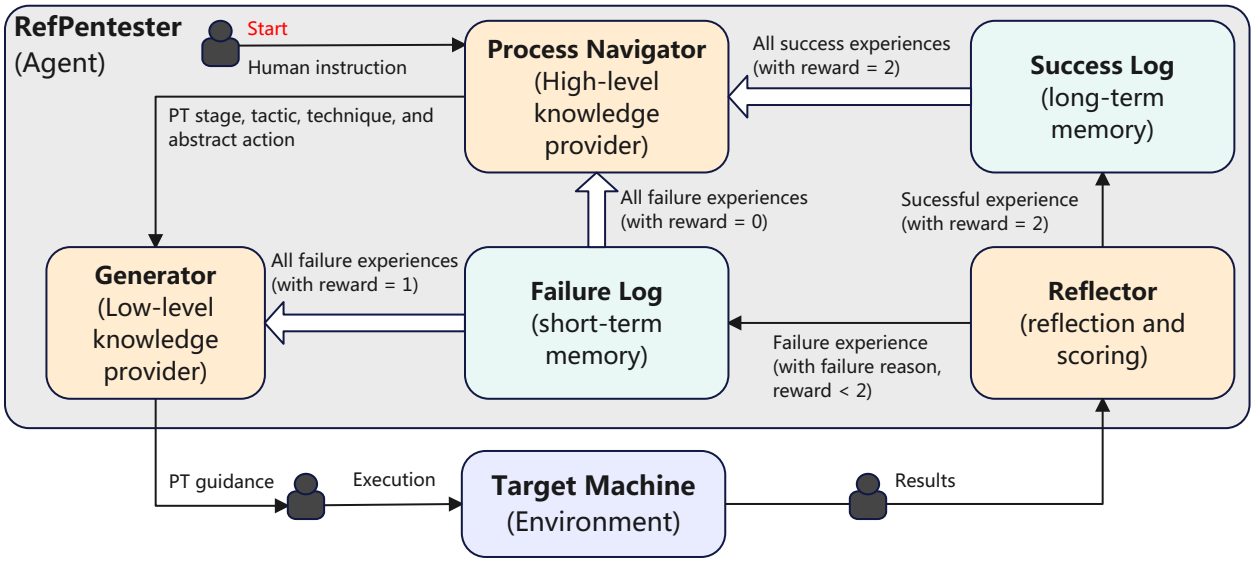

Figure 12: The Proposed RefPentester Framework

The **Process Navigator** is a high-level PT knowledge provider powered by an RAG pipeline,

which means it provides knowledge which is abstract and non-actionable. It determines the current PT process stage (τ) based on human input instructions (q), and provides tactic (c), technique (u) and abstract action (a) accordingly. The **Generator** is a low-level knowledge provider, which means it can provide step-by-step actionable guidance to help human operators execute actions on the target machine. It generates more detailed PT guidance (g) for the human operator to perform executions. The target machine will respond with results (o) to the human operator, who will then collect them and input them into the Reflector. The **Reflector** is used to evaluate and reflect on the results of executed PT actions by assigning rewards (r) and identifying potential failure reasons for the generated PT guidance and the provided high-level PT knowledge based on o. The **Success Log** (Y) is a list of successful experiences in temporal order. Each successful experience y is a seven-element tuple, that is: human instruction, PT stage, tactic, technique, abstract action, reward, results (q, τ, c, u, a, r, o), which will be kept until the termination of the PT process. The **Failure Log** (F) is a list of failure experiences in temporal order. Each failure experience f is a nine-element tuple, that is: human instruction, PT stage, tactic, technique, abstract action, reward, PT guidance, results and potential failure reasons ($q, \tau, c, u, a, r, g, o, \phi$). F will be reset if the executions corresponding to a are executed successfully on the target machine.

Each component applies multiple LLM sessions (LLM sessions with a certain predefined system prompt) to ensure that each session is responsible for a dedicated task during the PT process. All LLM sessions in each component are interconnected through a chaining structure, where one session's output serves as the input of the subsequent session. The design philosophy of the Ref-Pentester framework is: we decomposed the PT task into multiple subtasks, utilizing one specific LLM session with a predefined system prompt to deal with each subtask, respectively. By doing so, when an error occurs, we can easily identify the corresponding session and optimize it accordingly.

Initially, human operators provide q to the Process Navigator, which will generate high-level PT knowledge (τ, c, u, a) based on q, then provide the high-level knowledge to the Generator. The Generator will generate low-level PT guidance g to the human operator based on it, and the human operator will follow the g to execute executions on the target machine. The target machine will

Table 4: List of Notations

Symbol	Description	Symbol	Description
t	Iteration	q	Human Instruction
τ	PT Stage	h	PT Event
$\mathcal{M}$	PT Stage Machine	Π	LLM Session
c	Tactic	$\vec{c}$	Vectorized Tactic
u	Technique	$\vec{u}$	Vectorized Technique
A	Potential Actions	$\vec{A}$	Vectorized Potential Actions
a	Abstract Action	$\vec{a}$	Vectorized Abstract Action
$\mathcal{C}$	Tactic Set	$\mathbb{V}_{\mathcal{C}}$	Vectorized Tactic Set
$\mathcal{U}$	Technique Set	$\mathbb{V}_{\mathcal{U}}$	Vectorized Technique Set
$\mathcal{A}$	Abstract Action Set	$\mathbb{V}_{\mathcal{A}}$	Vectorized Abstract Action Set
g	PT Guidance	o	Results
r	Reward	ϕ	Failure Reason
Y	Success Log	F	Failure Log
y	Success Experience	f	Failure Experience
$\mathcal{K}$	RAG Pipeline		

provide results o to the Reflector. The Reflector will reflect and score high-level PT knowledge and low-level PT guidance based on o. If the value of r is two, it means the PT knowledge and guidance are correct, recording the (q, τ, c, u, a, r, o) as a successful experience to the Y. If the r is less than two, that means that the execution based on PT guidance has failed. The Reflector records the $(q, \tau, c, u, a, r, g, o, \phi)$ to the F, and evaluates if the high-level PT knowledge (c, u, a) is correct. If it is correct, the value of the reward is one; we will proceed to the Generator to reflect on the previous g. If it is not, the value of the reward is zero, and we will proceed to the Process Navigator to reflect on the previous high-level PT knowledge.

In the following sections, we introduce the PT knowledge preparation workflow for building the VDB and engineering processes for each component used in RefPentester.

3.4.2 Knowledge Preparation

The PT knowledge preparation workflow, depicted in Fig. 13, creates a VDB that embeds PT knowledge through knowledge collection, processing, and embedding steps. For this purpose, we

collect PT tactics, techniques, and abstract actions from various public cybersecurity resources, including the MITRE ATT&CK [47] and the OTG [48], ensuring that the VDB provides comprehensive knowledge ranging from abstract tactics to detailed procedures.

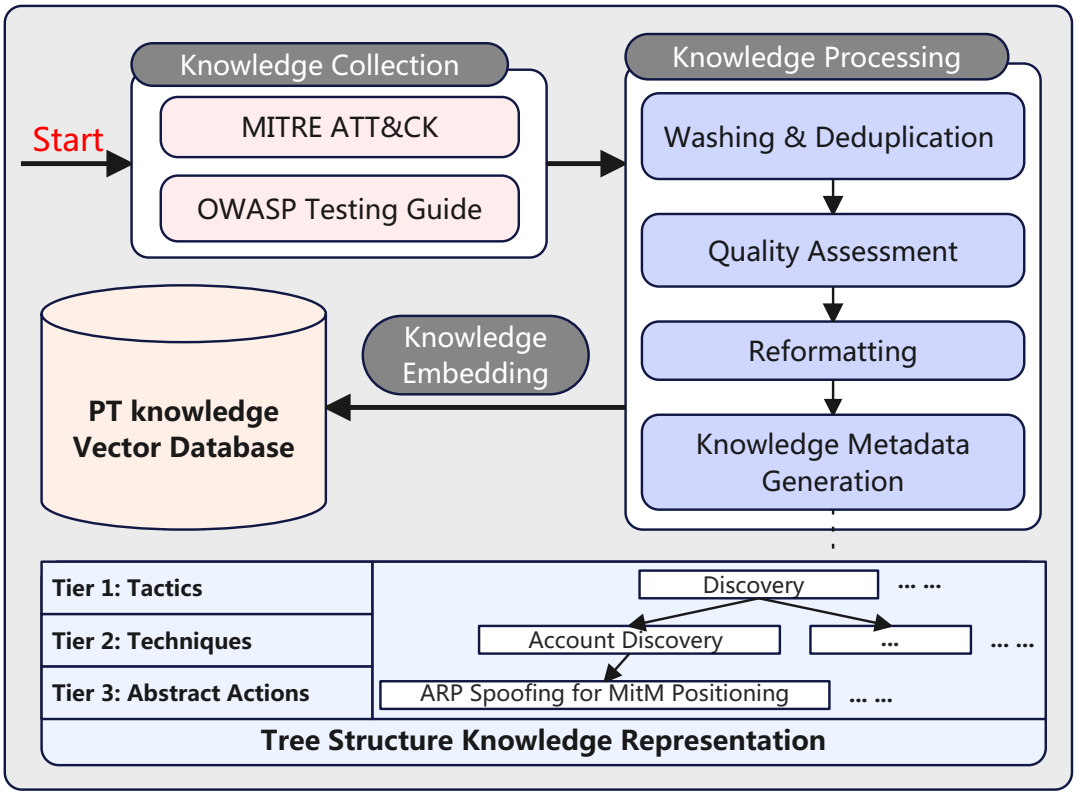

Figure 13: PT Knowledge Preparation Workflow for Building a VDB

MITRE ATT&CK [47] is a globally recognized knowledge base of adversary tactics, techniques, and procedures based on real-world observations. Tactics represent the high-level goals or objectives that an adversary aims to achieve during the PT process; techniques are the methods employed by an adversary to achieve a specific tactical goal. Consequently, each tactic may leverage multiple techniques to achieve its intended outcome. However, for security reasons, the procedures defined in MITRE ATT&CK [47] are non-actionable real-world examples, which we cannot directly leverage as more detailed PT action knowledge.

OTG is a key resource for web server security testing [48]. Using a black-box testing approach, OTG details security assessments across web servers and their deployment stacks, covering multiple aspects, including information gathering and configuration testing. We collect actionable and

detailed PT knowledge from OTG and map it onto MITRE ATT&CK techniques. For example, we collect OTG's Enumerate Applications on Webserver (OTG-INFO-004), then map it onto MITRE ATT&CK's Gather Victim Host Information (T1592). This ensures that our PT knowledge is capable of supplying the LLM session with the action-level knowledge, thereby enabling the generation of the PT guidance for human operators to execute.

Based on these resources, our PT knowledge preparation workflow to build a VDB is illustrated in Fig. 13. Our PT knowledge representation is designed using a three-tiered tree structure. The tiers represent sets of tactics ($\mathcal{C}$), techniques ($\mathcal{U}$), and abstract actions ($\mathcal{A}$), respectively. $\mathcal{C}$ and $\mathcal{U}$ are defined under the MITRE ATT&CK Matrix for Enterprise. The definition of $\mathcal{A}$ is based on OTG. We adopted the following practices to collect PT knowledge:

1. Provide precise guidance that offers meaningful and actionable insights.

2. Refrain from providing overly specific details to ensure applicability across various contexts or scenarios.

3. Adopt standard terminology within the MITRE ATT&CK and the OTG to guarantee consistency and facilitate understanding throughout the security community.

The example of collected knowledge is shown in the Table 5. It illustrates that for each tactic, such as 'Reconnaissance', it has a corresponding tactic name and description, as well as multiple techniques, such as 'Active Scanning', to achieve the specific tactic. And for each technique, there are a number of potential actions, such as 'OTG-INFO-001', that can provide detailed actionable knowledge to execute the technique.

After the knowledge collection phase, we proceed to the knowledge processing phase. We first remove any redundant or inaccurate knowledge through washing and de-duplication. Then, a quality assessment is performed to evaluate the integrity and relevance of the knowledge. After that, we reform the collected knowledge by adding the features of tactics and techniques to the knowledge of techniques and potential actions, respectively. Since multiple child-level knowledge items correspond to each parent-level knowledge item, we combine the features of the parent-level

Table 5: An Example of Original PT Knowledge


```
[{   "tactics": [{
        "name": "Reconnaissance",
        "description": "The penetration tester is attempting to gather
   ↪ information that can be used to plan future assessments.\n\
   ↪ nReconnaissance consists of techniques that involve penetration
   ↪ testers actively or passively gathering information to support
   ↪ testing efforts. Such information may include details about the
   ↪ target organization, infrastructure, or personnel. This
   ↪ information can be leveraged by the penetration tester to aid in
   ↪ other phases of the assessment lifecycle, such as using gathered
   ↪ information to plan and execute initial access tests, scope and
   ↪ prioritize post-access objectives, or guide further
   ↪ reconnaissance activities.",
        "techniques": [{
            "name": "Active Scanning",
            "id": "T1595",
            "description": "Penetration testers may execute active
   ↪ reconnaissance scans to gather information that can be used
   ↪ during the assessment. Active scans involve probing the target
   ↪ infrastructure via network traffic, as opposed to other forms of
   ↪ reconnaissance that do not involve direct interaction.\n\
   ↪ nPenetration testers may perform different forms of active
   ↪ scanning depending on the information they seek to gather. These
   ↪ scans can also be conducted using various methods, including
   ↪ leveraging native features of network protocols such as ICMP.[1][
   ↪ 2] Information from these scans may reveal opportunities for
   ↪ other forms of reconnaissance (e.g., searching open websites/
   ↪ domains or open technical databases), establishing operational
   ↪ resources (e.g., developing or obtaining capabilities), and/or
   ↪ testing for initial access (e.g., external remote services or
   ↪ exploiting public-facing applications).",
            "PotentialActionList": [
              { "id": "OTG-INFO-001",
                "description": "Conduct Search Engine Discovery and
   ↪ Reconnaissance for Information Leakage\n To understand what
   ↪ sensitive design and configuration information of the application
   ↪ /system/organization is exposed both directly (on the
   ↪ organization's website) or indirectly (on a third party website).
   ↪  Use a search engine to search for:\n\nNetwork diagrams and
   ↪ configurations\nArchived posts and emails by administrators and
   ↪ other key staff\nLog on procedures and username formats\
   ↪ nUsernames and passwords\nError message content\nDevelopment,
   ↪ test, UAT and staging versions of the website"}]}]}]}]
```

knowledge item, such as tactic name, with those of the child-level knowledge items. So we have the knowledge metadata with a three-tiered tree structure. The knowledge metadata is shown in Table 6. We can see that the knowledge data structure is reformatted as a pair of 'id' and 'text'. Each ID is unique and readable to humans. For knowledge data with an ID value of 'tactic_x', that means the knowledge corresponds to a tactic, and the tactic number is x. If the knowledge data's ID is 'technique_x_y', that means the text is for a technique, which corresponds to a tactic x, and the technique number is y. In the text for 'technique_x_y', we add the feature of its corresponding tactic, which is the tactic name, to ensure that once the tactic is determined, we can use the tactic to retrieve the most relevant technique. If the knowledge data's ID is 'action_x_y_z', that means the text is for an abstract action, which corresponds to a tactic x and technique y, and the action number is z. We add the feature of its related tactic and technique to the text of 'action_x_y_z' as well, that is, the tactic and the technique's name, to make sure that once the tactic and technique are determined, we can retrieve the most relevant abstraction action.

Finally, we embed the knowledge metadata as vectors and store them in one VDB. By doing so, we have the vectorized tactic, technique and abstract action set, which is notated as $\mathbb{V}_{\mathcal{C}}$, $\mathbb{V}_{\mathcal{U}}$ and $\mathbb{V}_{\mathcal{A}}$, respectively.

3.4.3 Process Navigator

The Process Navigator determines the high-level PT knowledge, which is informed by the PT Stage Machine ($\mathcal{M}$) and retrieved knowledge using the RAG pipeline ($\mathcal{K}$). First, it uses Penetration Event Analyst (Π_1) to identify the current PT event; the system prompt assigned to Π_1 is shown in Table 7. We employed role-based prompting techniques to provide the system prompt, considering two different scenarios. If it is the first iteration, that means we do not have a previous trajectory of the current PT task, Π_1 only needs to analyze the human operator's instruction (q); If it is not the first iteration, Π_1 will need to investigate the q as well as the previous trajectory.

By providing the $\mathcal{M}$ with the notation of an identified PT event, $\mathcal{M}$ identifies the PT state (τ_t). t stands for iteration. Then, it applies the $\mathcal{K}$ to identify c_t, u_t, and potential abstract actions A_t.

Table 6: An Example of PT Knowledge Metadata


```
[{      "id": "tactic_1",
        "text": "Tactic: Reconnaissance\n Description: The penetration
   ↪ tester is attempting to gather information that can be used to
   ↪ plan future assessments.\n\nReconnaissance consists of techniques
   ↪  that involve penetration testers actively or passively gathering
   ↪  information to support testing efforts. Such information may
   ↪ include details about the target organization, infrastructure, or
   ↪  personnel. This information can be leveraged by the penetration
   ↪ tester to aid in other phases of the assessment lifecycle, such
   ↪ as using gathered information to plan and execute initial access
   ↪ tests, scope and prioritize post-access objectives, or guide
   ↪ further reconnaissance activities."},
    {   "id": "technique_1_1",
        "text": "Technique: Active Scanning\n Id: T1595\n Description:
   ↪ Active Scanning is a technique to achieve Reconnaissance tactic.
   ↪ Penetration testers may execute active reconnaissance scans to
   ↪ gather information that can be used during the assessment. Active
   ↪  scans involve probing the target infrastructure via network
   ↪ traffic, as opposed to other forms of reconnaissance that do not
   ↪ involve direct interaction.\n\nPenetration testers may perform
   ↪ different forms of active scanning depending on the information
   ↪ they seek to gather. These scans can also be conducted using
   ↪ various methods, including leveraging native features of network
   ↪ protocols such as ICMP.[1][2] Information from these scans may
   ↪ reveal opportunities for other forms of reconnaissance (e.g.,
   ↪ searching open websites/domains or open technical databases),
   ↪ establishing operational resources (e.g., developing or obtaining
   ↪  capabilities), and/or testing for initial access (e.g., external
   ↪  remote services or exploiting public-facing applications)."},
    {   "id": "action_1_1_1",
        "text": "Action id: OTG-INFO-001\n Description: 'OTG-INFO-001'
   ↪ is an action for technique 'Active Scanning' to achieve tactic '
   ↪ Reconnaissance'. In detail: Conduct Search Engine Discovery and
   ↪ Reconnaissance for Information Leakage\n To understand what
   ↪ sensitive design and configuration information of the application
   ↪ /system/organization is exposed both directly (on the
   ↪ organization's website) or indirectly (on a third party website).
   ↪  Use a search engine to search for:\n\nNetwork diagrams and
   ↪ configurations\nArchived posts and emails by administrators and
   ↪ other key staff\nLog on procedures and username formats\
   ↪ nUsernames and passwords\nError message content\nDevelopment,
   ↪ test, UAT and staging versions of the website"}]
```

Table 7: Prompt Assignment for Penetration Event Analyst Session (Process Navigator)

```
You are a penetration testing assistant. You will be provided with the
   ↪ following two scenarios and your task is to analyze the penetration
   ↪ testing stage using the Penetration Stage Set provided.
There are two situations for input:
1. The assistant will be provided with a penetration tester's prompt and a
   ↪ trajectory prompt if there exists a trajectory.
2. The assistant will be provided with a penetration tester's prompt and "It
   ↪ seems we don't have previous trajectory" if there is no trajectory
   ↪ available.
Penetration Event Set:
"
            'α': 'Information Gathering',
            'β': 'Vulnerability Identification',
            'γ': 'Exploited'
            'δ': 'Post-Exploitation',
            'ϵ': 'Capture the Flag',
            'ζ': 'Documentation'
"
Please adhere to the following rules:
- Select the current penetration testing event from the Penetration Event Set
   ↪ based on the provided inputs.
- The penetration testing event may not change based on the provided prompt
   ↪ and trajectory; if the event didn't change, just output the same event.
- Your return should only include one notation from the Penetration Event Set,
   ↪  nothing else. Not explain, just the notation. For example, if the stage
   ↪  is Information Gathering, return:
  α
# Input Format
Pentester's prompt is:
What should I do next?
The previous trajectory's information is:
[{'PT_stage': 'Information Gathering', 'tactic': 'reconnaissance', 'technique
   ↪ ': 'Gather Victim Network Information', 'action': 'Enumeration find
   ↪ ports. The action goal is to find all open ports on the target IP and
   ↪ then gather detailed service and version information about those open
   ↪ ports to figure out potential ways to attack the target machine.', '
   ↪ operation_and_feedback': [{'current_operation': 'STEP 1: Open the
   ↪ terminal on your Linux-based machine.', 'current_feedback': '\n Opened
   ↪ '}]}]
# Output Format
- A single character notation from the Penetration Event Set corresponding to
   ↪ the current event of penetration testing.
```

Finally, it uses the Action Chooser session (Π_2) to determine the action a_t that will be executed. The system prompt assigned to Π_2 is illustrated in Table 8. In the Π_2, we considered two different scenarios, one of which is if Π_2 determines that one action from the selected potential actions (A_t) is correct, it will return the selected action (a_t) and set the indicator value equal to 0; Another

situation occurs when the action chooser module (Π_2) determines that the suggested actions (A_t) are not suitable for the current trajectory. In this case, Π_2 will generate a new action based on its knowledge and the provided tactic (c) and technique (u), and set the indicator k to 1.

Table 8: Prompt Assignment for Action Chooser Session (Process Navigator)

```
Assume you are a penetration testing assistant. You will be provided with the
    ↪ current human pentester's instructions and potential action list. Please
    ↪  select the most appropriate action for a penetration testing case from
    ↪ the provided list. Each action is detailed in a dictionary format,
    ↪ including the pentester's instruction, current PT stage, action ID, and
    ↪ description. If you think these provided potential actions are not
    ↪ correct, please use the best of your knowledge to generate the action
    ↪ that you think is most suitable for the current situation.
# Task
- Review the list of actions.
- Analyze which action best fits the pentester's instruction and current
    ↪ penetration testing stage.
- Clearly choose and justify your selection.
# Output Format
- If you think an action in the potential action list is correct, your output
    ↪ should only include the content of the action description and set the
    ↪ indicator 'k' equal to 0, nothing else. DO NOT add an explanation.
- If you think these actions inside the potential action list are incorrect,
    ↪ please output the action that you think is most suitable for the current
    ↪  PT situation, and set the indicator 'k' equal to 1.
# Examples
**Example Input:**
Pentester's instruction: help me test IP address xx.xx.xx.xx.  Potential
    ↪ action list: [ 'pt stage': 'Stage xxx', 'action id': 'OTG-INFO-xxx', '
    ↪ action description': 'zzz', 'pt stage': 'Stage xxx', 'action id': 'OTG-
    ↪ INFO-yyy', 'action description': 'kkk' ]
**Example Output:**
{
'chosen action': 'xxx',
'k': 0
}
```

The Process Navigator workflow is presented by Algorithm 1. The inputs include human instruction (q_t), Success Log (Y_{t-1}), and Failure Log (F_{t-1}). q_t tells what the human operator intends to do during the current PT iteration. Y_{t-1} records a list of successful experiences. These records are ranked in temporal sequence. By recording the Success Log of PT, our framework mitigates the risk of context loss inherent to LLMs, thereby preserving the continuity of the PT process. F_{t-1} is a list of failure experiences which includes low-level and high-level PT knowledge corresponding

to the unsuccessfully executed executions. The failure experiences follow a temporal sequence. F shall be reset if one PT action is successfully executed. The reason is PT in practice, the trial-and-error in different PT stages might be different, which means previous failure experience might not be insightful in the current PT stage. By maintaining F associated with the current failure execution, our framework can effectively leverage previous failure experiences to facilitate learning and improvement.

Algorithm 1 Process Navigator Workflow

Require: q_t, Y_{t-1}, F_{t-1} (**Note:** Either Y_{t-1} or F_{t-1} should be empty)

Ensure: τ_t, c_t, u_t, a_t

if Y_{t-1} is not empty **then**
    $P_{t-1} \leftarrow Y_{t-1}$
else
    $P_{t-1} \leftarrow F_{t-1}$
end if
$h_t \leftarrow \Pi_1(P_{t-1})$
$\tau_t \leftarrow \mathcal{M}(h_t)$
$c_t, u_t, A_t \leftarrow \mathcal{K}(q_t, \tau_t)$
$a_t \leftarrow \begin{cases} \Pi_2(q_t \oplus P_{t-1} \oplus A_t) & \text{if } k = 0 \\ \Pi_2(q_t \oplus P_{t-1}) & \text{if } k = 1 \end{cases}$
return τ_t, c_t, u_t, a_t

Firstly, the Process Navigator applied Π_1 to identify the PT event (h_t). PT Stage Machine ($\mathcal{M}$) will determine the current PT stage (τ_t) by h_t. $\mathcal{M}$ is a state machine and will be introduced in the following subsection. In the second step, the RAG pipeline ($\mathcal{K}$) will be applied to retrieve the current tactic (c_t), technique (u_t), and A_t based on q_t and the determined τ_t. The detailed RAG pipeline will be introduced in the next part of the section. Finally, Π_2 will be used to determine whether to pick an abstract action from A_t or generate an action based on its inherent knowledge. Since accumulative reward is included in the logs (Y_{t-1} and F_{t-1}), Π_2 will be leveraged to maximize the accumulative reward. k is an indicator that indicates Π_2 will choose an action from A_t or generate a new action. If the k is 0, that means that Π_2 will choose an action from A_t; if the k is 1, that means that Π_2 determines the actions inside A_t is not suitable for current q_t and Log (Y_{t-1}

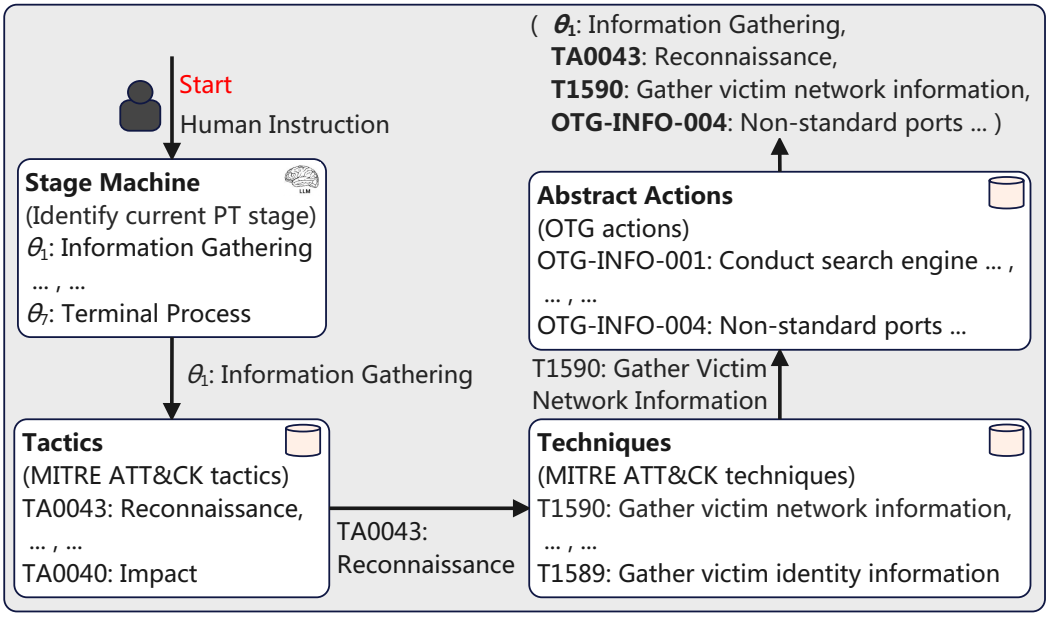

Figure 14: An Illustrative Example of Process Navigator Knowledge Retrieval

or F_{t-1}), so Π_2 will leverage its inherent knowledge to generate a new abstract action. The workflow finalizes the operation by returning the τ_t, c_t, u_t and a_t. Fig. 14 is an illustrative example of knowledge retrieval in Process Navigator. Initially, the PT Stage Machine will be used to identify the PT stage based on the selected PT event, which is determined by Π_1. In this case, the PT stage is θ_1: information gathering. The human instruction and θ_1 will be applied to retrieve tactic knowledge (TA0043: Reconnaissance) from VDB. After that, the Process Navigator will use the current PT stage, tactic knowledge, as well as the human instruction to retrieve the technique knowledge (T1590: Gather Victim Network Information), thus bridging the gap between high-level tactics and specific technical implementations. Finally, we will combine the human instruction, PT stage, tactic and technique knowledge to retrieve multiple potential actions (OTG-INFO-001, etc), and apply Π_2 to select the action (OTG-INFO-004: Non-standard ports) that is most suitable for the current circumstance, ensuring a targeted and effective response, as well as reducing hallucinations from Π_2. This iterative process allows for dynamic adaptation and precise execution within the framework.

3.4.3.1 PT Stage Machine

The PT Stage Machine is a state machine, as shown in Fig. 15. We modelled the PT process into seven stages. Each stage represents a sub-goal of the PT process. The input of the PT Stage Machine is PT events. PT events are used to identify if a sub-goal has been achieved. The stage transfer depends on the current stage as well as the identified PT event. The Stage Machine can help to identify the current PT stage and provide the stage description to help the RAG pipeline retrieve the most suitable high-level knowledge. We defined the PT stages by referring to the PT Execution Standard [9], a comprehensive framework that outlines practices for conducting penetration tests.

Table 9: PT Stages and Events

Stage	Description	Event	Description
θ_1	Information Gathering	α	Gathered Information
θ_2	Vulnerability Identification	β	Identified Vulnerability
θ_3	Exploitation	γ	Exploited
θ_4	Post-Exploitation	δ	Post-Exploited
θ_5	Capture the Flag	ϵ	Flag Captured
θ_6	Documentation	ζ	Documented
θ_7	Terminal Process		

For each PT stage θ_i, we define a corresponding sub-goal to characterize its operational intent and functional boundary. Specifically, stage θ_1 corresponds to **Information Gathering**, where the pentester collects environmental intelligence and target-related data to support subsequent actions. Stage θ_2 represents **Vulnerability Identification**, focusing on discovering and validating potential security weaknesses within the target system. Stage θ_3 is defined as **Exploitation**, during which the tester actively leverages the identified vulnerabilities to gain unauthorized access or control. Stage θ_4 corresponds to **Post-Exploitation**, emphasizing the maintenance of access, privilege escalation, and further lateral movement within the compromised environment. Stage θ_5 denotes **Capture the Flag**, where the tester aims to achieve the predefined mission objective, such as retrieving a specific flag or sensitive data. Stage θ_6 refers to **Documentation**, which involves

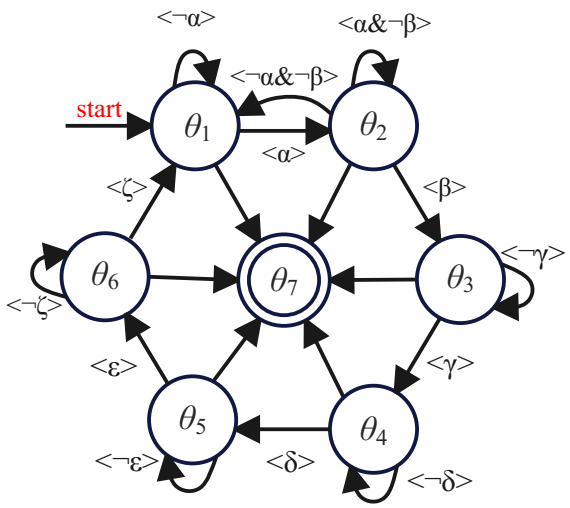

Figure 15: The PT Stage Machine

recording the attack process, findings, and evidence to facilitate reproducibility and reporting. Finally, stage θ_7 corresponds to the **Terminal Process**, representing the formal conclusion of the PT task, including environment cleanup and session termination.

We further define PT events to describe the accomplishment of each stage-specific sub-goal. As shown in Table 9, each stage (except the terminal stage θ_7) is associated with a corresponding event that signifies its successful completion. We denote these events as α, β, γ, δ, ε, and ζ, representing **Gathered Information**, **Identified Vulnerability**, **Exploited**, **Post-Exploited**, **Flag Captured**, and **Documented**, respectively. These events serve as observable outcomes that indicate the agent's progression through the PT process.

As illustrated in Fig. 15, the PT process is formalized as a finite-state machine $\mathcal{M}$ composed of a sequence of stages $\{\theta_1, \theta_2, \ldots, \theta_7\}$ and their corresponding events $\{\alpha, \beta, \gamma, \delta, \varepsilon, \zeta\}$, as summarized in Table 9. Each stage represents a distinct sub-goal within the PT process, and the occurrence of an event indicates the successful completion of that sub-goal. The transition of $\mathcal{M}$ is thus determined by the detected event at each iteration.

Initially, $\mathcal{M}$ starts from θ_1 (Information Gathering). When event α is detected, indicating that information has been successfully gathered, $\mathcal{M}$ transitions to θ_2 (Vulnerability Identification). Conversely, if the input is $\neg\alpha$, meaning that no useful information has been collected, the state of

$\mathcal{M}$ remains at θ_1. This design ensures that the agent cannot progress to the next phase without completing the prerequisite reconnaissance task.

At stage θ_2, if event β occurs, implying that a vulnerability has been successfully identified, $\mathcal{M}$ transitions to θ_3 (Exploitation). If the input is $\alpha\&\neg\beta$, the pentester can still reuse previously gathered information, but since no vulnerability has been confirmed, $\mathcal{M}$ remains in θ_2. In contrast, if the input is $\neg\alpha\&\neg\beta$, meaning that both reconnaissance information and vulnerabilities are unavailable, $\mathcal{M}$ reverts to θ_1 to reinitiate the information-gathering process. This cyclic transition reflects the iterative nature of real-world PT workflows.

At stage θ_3, an input of γ indicates that the vulnerability has been successfully exploited, prompting $\mathcal{M}$ to transition to θ_4 (Post-Exploitation). If the input is $\neg\gamma$, the exploitation attempt fails, and the state remains unchanged. In this stage, repeated attempts may occur until a successful exploit triggers the transition.

Subsequent transitions follow a similar logic. When $\mathcal{M}$ is in θ_4, θ_5, or θ_6, the transitions to θ_5, θ_6, and θ_1 occur only upon receiving events δ, ε, and ζ, respectively. Specifically, event δ denotes successful post-exploitation (e.g., privilege escalation or persistence establishment), event ε signifies the successful completion of the capture-the-flag objective, and event ζ represents the documentation of testing results. If the input to any of these stages is $\neg\delta$, $\neg\varepsilon$, or $\neg\zeta$, the state remains at the current stage, reflecting that the corresponding sub-goal has not been achieved.

Finally, $\mathcal{M}$ includes a termination mechanism to handle non-progressive or completed processes. If $\mathcal{M}$ receives the maximum consecutive iterations without a state change, the process is considered stagnant, and $\mathcal{M}$ automatically transitions to θ_7, the **Terminal Process**. Similarly, if all flags have been successfully captured or all sub-goals are completed, $\mathcal{M}$ also transitions to θ_7 to terminate the PT task. The state θ_7 therefore represents both natural and forced termination conditions, ensuring the boundedness and convergence of the PT process within the PT Stage Machine.

3.4.3.2 RAG Pipeline

The overall workflow of the RAG pipeline is illustrated in Algorithm 2, which is designed to retrieve hierarchical PT knowledge from the VDB. The input of the pipeline consists of the current human instruction q_t and the identified PT stage τ_t, while the output includes the selected tactic c_t, technique u_t, and a set of candidate abstract actions A_t. The final executable action a_t will later be determined by the Action Chooser session from the set A_t.

At the beginning of the process, the pipeline employs an embedding function $\mathrm{Embed}(\cdot)$ to encode the concatenation of the human instruction q_t and the current stage τ_t (where $\oplus$ denotes string concatenation) into a dense vector representation. A cosine similarity function $\mathrm{Cosim}(\cdot)$ is then used to measure the semantic proximity between this embedded query and all candidate tactic vectors $\vec{c} \in \mathbb{V}_{\mathcal{C}}$ stored in the tactic space. The tactic vector $\vec{c}_t$ that yields the maximum cosine similarity is selected as the most relevant tactic representation. This vectorized form is then decoded by a reverse mapping function $\psi(\cdot)$ to obtain the textual tactic c_t.

Algorithm 2 RAG Pipeline (i.e., $\mathcal{K}(\cdot)$)

Require: q_t, τ_t

Ensure: c_t, u_t, A_t

$\vec{c}_t \leftarrow \arg\max_{\vec{c} \in \mathbb{V}_{\mathcal{C}}} \mathrm{Cosim}(\mathrm{Embed}(q_t \oplus \tau_t), \vec{c})$

$c_t \leftarrow \psi(\vec{c}_t)$

$\vec{u}_t \leftarrow \arg\max_{\vec{u} \in \mathbb{V}_{\mathcal{U}}} \mathrm{Cosim}(\mathrm{Embed}(q_t \oplus \tau_t \oplus c_t), \vec{u})$

$u_t \leftarrow \psi(\vec{u}_t)$

$\vec{x}_t \leftarrow \mathrm{Embed}(q_t \oplus \tau_t \oplus c_t \oplus u_t)$

$A_t \leftarrow \{\psi(\vec{a}) \mid \mathrm{Cosim}(\vec{x}_t, \vec{a}) > \lambda, \vec{a} \in \mathbb{V}_{\mathcal{A}}\}$

return c_t, u_t, A_t

Subsequently, the pipeline proceeds to identify the most relevant technique associated with the selected tactic. Specifically, the model embeds the concatenation of $(q_t \oplus \tau_t \oplus c_t)$ and searches the technique vector space $\mathbb{V}_{\mathcal{U}}$ for the technique vector $\vec{u}_t$ that maximizes the cosine similarity with the embedded query. This step ensures contextual consistency between the current task, the identified

stage, and the retrieved tactic. The selected vectorized technique $\vec{u}_t$ is then transformed back to its symbolic form u_t through the mapping function $\psi(\cdot)$.

After both tactic and technique are determined, the pipeline jointly embeds all contextual information, forming a composite vector $\vec{x}_t = \mathrm{Embed}(q_t \oplus \tau_t \oplus c_t \oplus u_t)$ in a d-dimensional semantic space. The purpose of this step is to generate a comprehensive latent representation that integrates user intent, stage context, and retrieved PT knowledge. Using this representation, the algorithm searches within the action vector space $\mathbb{V}_{\mathcal{A}}$ to identify potential abstract actions $\vec{a}$ that exhibit a cosine similarity with $\vec{x}_t$ greater than a predefined threshold λ. Formally, the candidate action set is defined as:

$$A_t = \{\psi(\vec{a}) \mid \mathrm{Cosim}(\vec{x}_t, \vec{a}) > \lambda, \vec{a} \in \mathbb{V}_{\mathcal{A}}\} \tag{1}$$

The threshold λ thus serves as a filtering criterion to ensure semantic relevance between the retrieved actions and the current operational context.

Finally, the RAG pipeline outputs the selected tactic c_t, the corresponding technique u_t, and the filtered set of abstract actions A_t. These outputs form a hierarchical reasoning chain that reflects how high-level human intent (q_t) is progressively grounded into executable PT actions through contextual retrieval and semantic alignment. This structured retrieval process provides interpretability and ensures that each decision step in the pipeline is guided by semantically coherent PT knowledge.

The example retrieved data through the RAG pipeline is illustrated in Table 10, which demonstrates the hierarchical retrieval structure of the knowledge base. Specifically, the retrieved results contain three-layer knowledge: tactic, technique, and a corresponding list of potential actions. The first layer, represented by the *Reconnaissance* tactic (c_t), describes the overall strategic objective of the PT task, emphasizing information gathering activities that support subsequent assessments. The second layer corresponds to the retrieved technique *Gather Victim Network Information (T1590)* (u_t), which specifies a concrete operational method for achieving the reconnaissance goal. This technique description explains how a penetration tester can collect network-related intelligence, such as IP ranges, domain names, or topology details, through both active scanning and

passive observation. At the final layer, the retrieved *Potential Action List* (A_t) provides abstract action knowledge that the agent can utilize. For example, the action *OTG-INFO-004.2* describes the process of identifying non-standard TCP ports using tools such as `nmap` with the `-sV` option to recognize web services running on arbitrary ports.

This hierarchical retrieval result exemplifies how the RAG pipeline effectively decomposes complex PT knowledge into multi-level representations, enabling interpretable reasoning and structured knowledge provided for the following component. By aligning human-readable cybersecurity knowledge with vectorized embeddings, the framework ensures that each retrieved component maintains both semantic coherence and operational relevance throughout the various stages of the PT process.

3.4.4 Generator

The Generator module is responsible for producing PT operational guidance by synthesizing semantic and contextual elements derived from the input parameters q_t, τ_t, c_t, u_t, and a_t. As illustrated in Algorithm 3, this component functions as a reasoning component that transforms structured task information into executable textual guidance g_t. The additional variable F_{t-1} denotes the Failure Log from the previous step, illustrating previous executed steps and potential failure reasons, which is important for adaptive generation when the last action execution was unsuccessful.

In the standard scenario, when F_{t-1} is not provided (i.e., the previous operation was executed successfully or no reflective feedback is available), the Generator activates the Operation Generator Session Π_3 to produce a new piece of operational guidance g_t.

Table 10: RAG Example of Retrieved Data

```
 [   "Tactic": "Tactic Name: Reconnaissance\n Description: The
↪ penetration tester is attempting to gather information that can
↪ be used to plan future assessments.\n\nReconnaissance consists of
↪  techniques that involve penetration testers actively or
↪ passively gathering information to support testing efforts. Such
↪ information may include details about the target organization,
↪ infrastructure, or personnel. This information can be leveraged
↪ by the penetration tester to aid in other phases of the
↪ assessment lifecycle, such as using gathered information to plan
↪ and execute initial access tests, scope and prioritize post-
↪ access objectives, or guide further reconnaissance activities.",

    "Technique": "Technique Name: Gather Victim Network Information
↪ .\n Id: T1590\n Description: Gather Victim Network Information is
↪  a technique to achieve Reconnaissance tactic. Penetration
↪ testers may gather information about the target's networks to
↪ support assessment activities. Information about networks may
↪ include various details, such as administrative data (e.g., IP
↪ ranges, domain names) as well as specifics regarding network
↪ topology and operations.\n Penetration testers may collect this
↪ information using methods like active scanning or simulated
↪ phishing to gather data. Network information may also be publicly
↪  accessible through online data sets or technical databases (e.g
↪ ., searching open technical databases).[1][2][3] Gathering this
↪ information can uncover opportunities for further reconnaissance
↪ (e.g., active scanning or searching open websites/domains),
↪ preparing resources (e.g., acquiring or testing infrastructure),
↪ or identifying initial access points (e.g., exploiting trusted
↪ relationships).",

    "PotentialActionList":
    ["action id: OTG-INFO-004.2\n action description: Non-standard
↪ ports\n While web applications usually live on port 80 (http) and
↪  443 (https), there is nothing magic about these port numbers. In
↪  fact, web applications may be associated with arbitrary TCP
↪ ports, and can be referenced by specifying the port number as
↪ follows: http[s]://www.example.com:port/. For example, http://www
↪ .example.com:20000/.\n It is easy to check for the existence of
↪ web applications on non-standard ports. A port scanner such as
↪ nmap [2] is capable of performing service recognition by means of
↪  the -sV option, and will identify http[s] services on arbitrary
↪ ports. What is required is a full scan of the whole 64k TCP port
↪ address space."    ]]
```

Algorithm 3 Generator

Require: q_t, τ_t, c_t, u_t, a_t, F_{t-1}

Ensure: g_t

1: **if** F_{t-1} is empty **then**

2: $\quad g_t \leftarrow \Pi_3(q_t \oplus \tau_t \oplus c_t \oplus u_t \oplus a_t)$

3: **else**

4: $\quad g_t \leftarrow \Pi_4(F_{t-1})$

5: **end if**

6: **return** g_t

As illustrated in Algorithm 3, Π_3 synthesizes actionable instructions by jointly embedding the task-specific context, including the current human query q_t, the identified PT stage τ_t, the retrieved tactic c_t, technique u_t, and the selected abstract action a_t. This composition $(q_t \oplus \tau_t \oplus c_t \oplus u_t \oplus a_t)$ forms the semantic input sequence for Π_3, where the operator $\oplus$ denotes the string concatenation used for context splicing.

The system prompt assigned for Π_3 is detailed in Table 11 and Table 12. It defines the Π_3's operational persona and constrains its contextual reasoning. Specifically, the prompt positions Π_3 as a PT assistant capable of generating structured, context-aware, and executable guidance for authorized PT environments. The input to Π_3 consists of structured PT metadata, including the current PT stage τ_t, tactic c_t, technique u_t, and a candidate abstract action a_t retrieved from the RAG pipeline, as well as the human instruction q_t. Based on this structured context, Π_3 produces a set of step-by-step operational procedures expressed in JSON format. Each generated JSON object contains an array of "`Operational Guidance`" that explicitly enumerates the terminal commands or procedural actions required to execute a_t under the current PT scenario.

The system prompt enforces three key principles during generation:

1. **Action Execution:** Ensure that every command required for task execution is concretely specified;

Table 11: Prompt Assignment for Operation Generator Session (Generator) (1)

````
You are a penetration testing assistant for an authorized web penetration
    ↪ testing platform where the target machine is Linux-based. You will
    ↪ receive structured information and provide detailed guidance based on
    ↪ the specified action, stage, tactic, and technique related to
    ↪ penetration testing.

Here's an example of the provided information:

```
I'm doing penetration testing on Hack The Box platform's Sau. It's a Linux
    ↪ machines running on the intranet. The IP address is 10.10.11.224.
Penetration Testing Stage: Information Gathering
Tactic Name: reconnaissance
Technique: Technique: Gather Victim Network Information
Action:
 ['Non-standard ports\n ...']
```
Your task is to utilize the provided current action, its description, and
    ↪ contextual details to generate a comprehensive, step-by-step guideline.
    ↪ This guide should account for the user's limited knowledge in
    ↪ penetration testing.
# Guidelines for Providing Guidance
- **Action Execution:** Determine if terminal operations are required. If so,
    ↪ list every necessary command detailing the actions to be performed.
````

2. **Clarity:** Each instruction must be written in concise, human-readable language suitable for novice users;

3. **Contextual Awareness:** The generated guidance must align with the given penetration stage τ_t, tactic c_t, and technique u_t.

This structured prompting strategy ensures that the output g_t is both semantically aligned with the PT context and operationally executable within a Linux-based testing environment.

An example of such a generation process is shown in Table[11, 12], where the system receives an input specifying the *Information Gathering* stage with the *Reconnaissance* tactic and *Gather Victim Network Information* technique. The Generator then formulates an operational guidance to perform port scanning on non-standard TCP ports, providing clear, stepwise terminal commands (e.g., `sudo apt-get install fping`, `fping -a -g 192.168.1.0/24`)

alongside descriptive instructions. Through this process, Π_3 bridges high-level semantic knowledge and low-level executable procedures, thereby transforming abstract PT knowledge into actionable, interpretable operational guidance.

Table 12: Prompt Assignment for Operation Generator Session (Generator) (2)

```
- **Clarity:** Ensure each step is clear and straightforward to aid
   ↪ understanding.
- **Contextual Awareness:** Use the information corresponding to the current
   ↪ penetration stage, tactic, and technique to support the action.
# Output Format
- Your output should be a JSON object.
- The JSON should include:
  - "operations": an array with step-by-step instructions, using clear and
    ↪ direct language. Include necessary "commands" within the operations if
    ↪ command-line inputs are required.
# Example Output
```json
[
    "STEP 1: Open the terminal in your Linux-based machine.",
    "STEP 2: Ensure you have 'fping' installed. You can install it using: sudo
   ↪  apt-get install fping",
    "STEP 3: ...
]
```
Notes
- Tailor each guide to align with the current penetration stage, technique,
   ↪ and tactic.
- Assume the user is familiar with operating a Linux-based terminal but lacks
   ↪ experience in complex penetration testing tasks.
```

Alternatively, when the previous guidance g_{t-1} associated with action a_{t-1} fails to achieve the expected outcome, the Reflector component produces a structured Failure Log F_{t-1}. This feedback encapsulates the diagnostic information from the failed attempt, including the previous human instruction q_{t-1}, tactic c_{t-1}, technique u_{t-1}, selected abstract action a_{t-1}, executed operations g_{t-1}, corresponding outputs, execution results o_{t-1}, and potential failure reasons ϕ_t. Under this reflective scenario, the Generator activates the Operation Optimizer Session Π_4, which aims to refine or regenerate a new operational guidance g_t based on the Failure Log encoded in F_{t-1}.

As illustrated in Algorithm 3, Π_4 conditions its generation process on the Failure Log F_{t-1}, allowing the system to perform corrective reasoning rather than simply reissuing prior instructions.

The system prompt assigned for Π_4 is provided in Table[13, 14]. It defines the input structure: a failure experience record that logs the previously executed operations and feedback messages, as well as its corresponding tactic, technique and abstract action. This design provides the model with explicit traces of failure, enabling it to analyze what went wrong and to adapt its next guidance accordingly.

Table 13: Prompt Assignment for Operation Optimizer Session (Generator) (1)

````
You are a penetration testing assistant. You will be provided with the
   ↪ following information:

A failure experience containing details about the current tactic,
   ↪ technique, selected action, operations, and potential failure
   ↪ reasons. The reference will look as follows:
```json
{
   "human instruction": "the current human instruction",
   "pt stage": "the current pt stage",
   "tactic": "tactic name",
   "technique": "technique description",
   "selected_action": "action description",
   "executed_operation_and_feedback": [
      {
         "executed_operation": "xxx",
         "feedback": "xxx"
      }
      ...
   ],
   "potential_failure_reasons": "xxx"
}
```
````

Within the Π_4 session, the Generator is instructed to first analyze the Failure Log by examining potential failure causes and operational feedback, and then to generate an optimal operational guidance that avoids repeating the prior mistakes. The generated output maintains the same JSON schema as in Π_3, where each element in the `"Operational Guidance"` array specifies a clear and executable step, often accompanied by corresponding terminal commands (e.g., `sudo`

`apt-get install fping`, `fping -a -g 192.168.1.0/24`). In contrast to Π_3, however, the Π_4 process incorporates reasoning over failure patterns, such as invalid command parameters, permission errors, or unreachable targets, to produce improved, contextually adaptive guidance.

To ensure robustness and interpretability, the system prompt of Π_4 enforces three operational principles:

1. **Failure Aware Adaptation:** The session Π_4 must explicitly consider the failure reasons provided in F_{t-1} before generating new steps;

2. **Context Preservation:** The refined guidance should remain consistent with the current PT stage, tactic, and technique;

3. **Operation Guidance Optimization:** The regenerated instructions must correct the operational flaws identified in prior attempts while preserving task objectives.

As shown in Tables 13–14, Π_4 thereby functions as a reflective optimization mechanism: it transforms feedback into actionable learning signals, enabling the Generator to iteratively refine its decision policy and produce context-aware, failure-resilient operational guidance.

Overall, the Generator dynamically balances proactive generation (via Π_3) and reflective adaptation (via Π_4). By leveraging structured PT knowledge and contextualized feedback, it transforms high-level semantic representation knowledge into actionable PT steps. This mechanism allows the system to maintain operational consistency, minimize redundant exploration, and improve cumulative rewards in multi-round PT scenarios, ultimately contributing to the framework's goal of self-improving autonomous decision-making.

3.4.5 Reflector

The Reflector serves as the evaluative and introspective component of the framework, leveraging multiple LLM sessions to analyze the effectiveness of previously generated guidance and to

Table 14: Prompt Assignment for Operation Optimizer Session (Generator) (2)

```
**Your task** is to analyze the current tactic and technique description from
   ↪ the reference and select, or refine a more appropriate operational
   ↪ guidance, as the current step-by-step guidance in the reference might
   ↪ not be correct.
# Steps
1. **Analyze**:
    - Review the potential failure reasons include in the reference
2. **Generate Optimal Operational Guidance**:
    - Based on current tactic, technique, executed action and failure reasons,
   ↪  generate an optimal operational guidance.
# Output Format
- Your response should be in the following JSON format:
```json
{
    "STEP 1": "xxx",
    "STEP 2": "yyy"
    ... ...
}
```
Examples
```json
[
    "STEP 1: Open the terminal in your Linux-based machine.",
    "STEP 2: Ensure you have 'fping' installed. You can install it using: sudo
   ↪  apt-get install fping",
    "STEP 3: Run the ICMP Ping Sweep command: fping -a -g 192.168.1.0/24",
    "STEP 4: Observe the output for live hosts within the specified IP range."
]
```
Notes
- Ensure that the refined action guidance applies to the given tactic and
   ↪ technique and is suitable for the current situation.
- Consider the feedback from executed operations if relevant in decision-
   ↪ making.
```

provide reward-based feedback for policy optimization. Its core objective is twofold: (1) to assess the correctness of the tactic c_t, technique u_t, abstract action a_t, as well as the operational guidance g_t executed on the target environment, and (2) to construct reflective feedback when the operation fails, thereby enabling the system to improve its decision-making process iteratively. The complete workflow of the Reflector is presented in Algorithm 4.

The Reflector operates on a contextual input set $\{q_t, \tau_t, c_t, u_t, a_t, g_t, o_t, Y_{t-1}, F_{t-1}\}$, where o_t denotes the observed execution outcome collected by a human operator from the target environment.

Algorithm 4 Reflector Workflow

Require: q_t, τ_t, c_t, u_t, a_t, g_t, o_t, Y_{t-1}, F_{t-1}
(**Note:** Either Y_{t-1} or F_{t-1} should be empty.)
Ensure: Y_t, F_t
(**Note:** Either Y_t or F_t should be return.)
$r_t \leftarrow R(\tau_t, c_t, u_t, a_t, g_t, o_t)$
if $r_t = 2$ **then**
  $Y_t \leftarrow Y_{t-1} \cup \{(q_t, \tau_t, c_t, u_t, a_t, r_t, o_t)\}$
  return Y_t
  {PT guidance executed successfully, go to Process Navigator}
else
  $\phi_t \leftarrow \Pi_5(q_t \oplus a_t \oplus o_t)$
  $F_t \leftarrow F_{t-1} \cup \{(q_t, \tau_t, c_t, u_t, a_t, r_t, g_t, o_t, \phi_t)\}$
  if $r_t = 1$ **then**
    return F_t
    {PT knowledge c_t, u_t, a_t is correct, go to Generator}
  else
    return F_t
    {c_t, u_t, a_t is incorrect, go to Process Navigator to retrieve reflected knowledge}
  end if
end if

Initially, the Reflector invokes an LLM-based reward function $\mathcal{R}(\cdot)$ to produce a scalar reward r_t based on both the high-level knowledge elements (τ_t, c_t, u_t, a_t) and the low-level operational outcomes (g_t, o_t).

The session follows the instruction schema summarized in Table 15. It analyzes the semantic and functional alignment between the provided tactic–technique–abstract action triplet and the current PT stage and situation, producing a scalar reward $r_t \in \{0, 1, 2\}$ that quantifies the outcome:

- $r_t = 2$: Both high-level knowledge and low-level guidance are correct; the operation succeeded.
- $r_t = 1$: High-level knowledge is valid, but the execution was partially unsuccessful.
- $r_t = 0$: Neither the knowledge hierarchy nor the operation fits the current situation.

If $r_t = 2$, the guidance is deemed fully successful. A new success experience

$$y_t = (q_t, \tau_t, c_t, u_t, a_t, r_t, o_t) \tag{2}$$

Table 15: Prompt Assignment for Reward Function Session (Reflector)

```
# Tasks
Evaluate the correctness of the selected tactic, technique and abstract action
   ↪  in a given penetration testing stage on a Linux-based web penetration
   ↪ testing system, and give the corresponding rewards.

You are a penetration testing reward function tasked to determine if the
   ↪ selected tactic, technique and action are appropriate for the given
   ↪ penetration testing stage in a verified Linux-based web penetration
   ↪ testing target system. You will receive the following information:
1. Current penetration testing stage.
2. Current tactic name and description.
3. Current technique name and description.
4. Current action name and description.
5. Executed operations and their corresponding results.

# Steps
1. Review the provided penetration testing stage.
2. Analyze the appropriateness of the tactic in the given stage.
3. Assess the relevance of the technique under the current tactic.
4. Analyze the appropriateness of the abstract action in the given stage,
   ↪ tactic and technique.
5. Analyze if the operational guidance is correct based on its feedback (
   ↪ observations)

# Output Format
- Your output should only be numbers.
- If the operations are executed successfully, and we meet the expected
   ↪ outcome, the rewards would be 2.
- If the high-level knowledge, including tactic, technique and abstract action
   ↪ , is correct, but the operational guidance has errors, the rewards would
   ↪  be 1.
- If the high-level knowledge is not suitable for the current situation, the
   ↪ rewards would be 0.
- Do not provide any explanation or additional comments.
```

is appended to the prior Success Log Y_{t-1}, producing the updated Y_t. The framework then transitions to the *Process Navigator* for subsequent knowledge retrieval and stage progression.

When the outcome is suboptimal ($r_t < 2$), the Reflector invokes the Operation Reflector Session Π_5 to conduct a fine-grained failure analysis. This session's system prompt is defined in Tables 16 and 17, accepts structured feedback in JSON-like format, including the current action, executed operations, and their feedback logs. It examines each operation to identify where failures occurred and produces a structured reasoning object ϕ_t describing the likely causes of failure.

Table 16: Prompt Assignment for Operation Reflector Session (Reflector) (1)

```
Analyze penetration testing operations and explain reasons for action failures
   ↪  utilizing structured feedback.
Given the structured information in a JSON-like format regarding web
   ↪ penetration testing operations, evaluate each operation and its feedback
   ↪  to determine the possible causes of failures in successfully executing
   ↪ the action.
# Steps
1. Examine the structured information provided in JSON-like format, which
   ↪ includes:
   - `action_description`: A description of the current action.
   - A list of operations, each with a `current_operation` and corresponding `
   ↪ current_feedback`.
2. Identify the operations where feedback indicates failure.
3. Analyze the feedback for these operations to deduce possible reasons for
   ↪ failure.
4. Summarize the reasons for failure for each problematic operation.
# Output Format
```

These may include incorrect parameters, environmental constraints, or privilege limitations detected during the command execution phase. The resulting failure experience

$$f_t = (q_t, \tau_t, c_t, u_t, a_t, r_t, g_t, o_t, \phi_t) \tag{3}$$

is appended to the Failure Log F_{t-1} to form F_t.

Depending on the reflective evaluation, the next decision branch is as follows:

- If $r_t = 1$, the high-level knowledge (c_t, u_t, a_t) remains semantically correct but the operational guidance (g_t) requires refinement. In this case, F_t is passed to the *Generator*, which reactivates the Operation Optimizer Session (Π_4) to produce an improved version of g_t.

- If $r_t = 0$, both the high-level and low-level components are determined to be invalid. The framework then forwards F_t to the *Process Navigator* to retrieve new, stage-appropriate penetration knowledge for reinitialization.

Through this dual-session reflective mechanism, the Reflector achieves a hybrid evaluation process combining quantitative assessment and qualitative reasoning. The reward function $\mathcal{R}(\cdot)$

Table 17: Prompt Assignment for Operation Reflector Session (Reflector) (2)

````
Provide a JSON object detailing the operations that encountered failure along
   ↪ with potential reasons:
```json
{
  "failed_operations": [
    {
      "current_operation": "Description of the operation",
      "potential_reasons": ["reason1", "reason2", ...]
    },
    ...
  ]
}
```
Ensure the language used in the output is brief and precise.
# Examples
### Example Input
```json
{
  "action_description": "Attempt to access restricted directory",
  "operations": [
    {
      "current_operation": "Use default credentials",
      "current_feedback": "Access denied due to invalid credentials",
      "potential_reasons": ["Default credentials are invalid"]
    },
    {
      "current_operation": "Brute-force login",
      "current_feedback": "Account locked after multiple failed attempts",
      "potential_reasons": ["Account locking mechanism is active", "Brute-
   ↪ force approach detected"]
    }
  ]
}
```
# Notes
- Focus on identifying clear, actionable reasons for operation failures.
- Consider system-specific feedback nuances in your analysis.
````

provides discrete numerical signals that guide reinforcement updates, while the operation reflector Π_5 converts textual feedback into structured diagnostic knowledge. Together, they enable the RefPentester to evolve continuously: successful experiences are accumulated in Y_t, while failure experiences F_t provide rich learning signals for iterative adaptation. This design ensures that the framework not only measures success but also learns from failure, transforming raw execution traces into actionable intelligence for robust autonomous PT.

3.5 Summary

This chapter presented the design rationale, architecture, and implementation details of the proposed *RefPentester* framework. The framework was developed to address the limitations of existing AutoPT systems, including task planning consistency, hallucination, and adaptive learning. We achieved these goals by integrating a seven-stage state machine mechanism, a hierarchical RAG pipeline, and a reflective learning module.

By integrating predefined knowledge structures with LLMs' reasoning ability, the proposed *RefPentester* framework establishes a balance between interpretability and adaptability in AutoPT. The following chapter presents comprehensive experiments designed to examine the effectiveness of our proposed framework.

Chapter 4

Experiments and Evaluation

This chapter evaluates the proposed *RefPentester* framework to validate its effectiveness, robustness, and interpretability in AutoPT. Building upon the design principles introduced in Chapter 3, the experiments aim to answer three key research questions:

- **RQ1:** Does the proposed *RefPentester* framework outperform the baseline LLM-based AutoPT system in penetration task completion and credential capture success?
- **RQ2:** How does the Reflector component's reflective reasoning influence *RefPentester's* decision stability and error recovery across iterative testing cycles?
- **RQ3:** What qualitative patterns emerge from the interaction between knowledge grounding and reflection, and how do they contribute to interpretable reasoning in AutoPT?

These RQs aim to evaluate the efficiency, reasoning stability, and interpretability of the *RefPentester*, which is achieved by the Process Navigator and the Reflector modules.

To address these questions, we conducted a series of experiments on HTB's *Sau* machine. The evaluation focuses on two quantitative metrics: the *State Transition Success Rate* (STSR) and the *Credential Capture Success Rate* (CCSR), to measure reasoning reliability and operational completeness jointly.

Beyond quantitative analysis, qualitative observations and failure case discussions are also presented to uncover the underlying dynamics of reflection-driven reasoning and to provide insights for future improvements. The remainder of this chapter is organized as follows: Section 4.1 introduces the experimental setup and target environment; Section 4.2 defines the evaluation metrics; Section 4.3 reports comparative results; and Section 4.4 provides in-depth discussion.

4.1 Experimental Setup

This section describes the experimental environment, implementation configuration, and rationale used to evaluate the proposed *RefPentester* framework. The goal of this setup is to ensure a reproducible and controlled testing environment that can reflect both the reasoning capability and operational stability of the framework under real-world PT scenarios. All experiments were conducted on the *Sau* machine from the HTB platform [105], which provides legally authorized, reproducible, and virtualized PT environments with inherent vulnerabilities. The *Sau* machine was selected due to its difficulty and the operating system's type (Linux), to align with the human-predefined PT knowledge base, which allows an objective evaluation of reasoning depth and action precision. To ensure consistency across trials, each experiment started with a clean reset of the target machine to its default state.

Implementation Environment. The *RefPentester* framework and the base model framework were implemented and executed on a MacBook Pro (M3 Pro, 18 GB RAM) running macOS Sequoia 15.6.1. Development was performed in PyCharm 2025.1 (Professional Edition) using Python 3.10. All experiments were executed through a secure OpenVPN [106] connection to the *Sau* machine, which ensures a safe, reproducible and isolated environment.

Framework Configuration. The configuration of hyperparameters of the framework is summarized in Table 19. The *RefPentester* employs OpenAI's GPT-4o [107] as the reasoning

Table 18: Experimental Environment

Category	Configuration
System Environment	Macbook Pro m3 pro 18 RAM
Operating System	macOS Sequoia 15.6.1
Integrated Development Environment (IDE)	PyCharm 2025.1 (Professional Edition)
Python	Python 3.10
Experinment Environment	HackTheBox[105] Sau Machine

backbone, configured with deterministic decoding (temperature=0.01, top-p=1) to guarantee reproducibility. For PT knowledge retrieval, the `llama-text-embedding-v2` model was employed to encode human-predefined PT knowledge into 1024-dimensional dense vectors [108]. The dimensionality of 1024 was chosen as it provides a balanced trade-off between representational richness and computational efficiency. Higher-dimensional embeddings (e.g., 4096) may improve expressiveness but incur greater memory and latency costs, whereas lower-dimensional ones (e.g., 512) can lead to information compression and reduced semantic granularity. These embeddings were stored and queried via a Pinecone VDB (*pt-knowledge-index*) [109], enabling cosine-similarity search within a serverless AWS (us-east-1) instance. The database contained 508 knowledge records derived from the MITRE ATT&CK [47] and OTG [48], forming a unified knowledge base for RAG retrieval. For each experimental episode, we limited the maximum number of attempts per PT stage to five. If no state transition occurred within five consecutive attempts, the episode was terminated and marked as failed. Each framework was executed for three independent episodes on the *Sau* machine to ensure statistical consistency.

Knowledge and Embedding Rationale. To ensure comprehensive reasoning coverage, the knowledge corpus combined adversary-oriented ATT&CK tactics and techniques with procedural OTG testing actions. Each entry was reformulated from a penetration tester's perspective before embedding, aligning the vector space semantics with operational PT contexts. This hybrid structure allowed *RefPentester* to retrieve both abstract attack intents and fine-grained actions within a single reasoning cycle.

Session Management and Logging. Each experimental iteration began with a cold-start reset to mitigate LLM memory carryover. Outputs from one module (e.g., Process Navigator) were structured and fed into subsequent modules (e.g., Generator or Reflector), ensuring coherent multi-stage reasoning. All outcomes were recorded into *Success Logs* or *Failure Logs*, establishing a reinforcement-style feedback loop for behaviour analysis and iterative improvement.

Table 19: Experimental Configuration

Hyperparameter	**Configuration**
Foundation Model	OpenAI GPT-4o
Model Access	OpenAI API
Temperature	0.01
Top-p	1
Embedding Model	llama-text-embedding-v2
Embedding Dimension	1024
Query Embedding Model	llama-text-embedding-v2
Vector Database Engine	Pinecone
Index Type	Cosine Similarity (Dense)
Vector Database	pt-knowledge-index
Maximum number of attempts	5 (per stage)
Number of experiments	3 (per framework)

Ethical Considerations. All experiments were conducted within the officially authorized HTB environment. No external or unauthorized systems were accessed. All operations followed ethical research practices concerning responsible AI use, user privacy, and reproducibility.

4.2 Evaluation Metrics

To quantitatively evaluate the effectiveness of the proposed *RefPentester* framework, two evaluation metrics were defined to capture different aspects of AutoPT performance: *CCSR* and *STSR*. While CCSR quantifies the operational completeness of task execution, STSR reflects the continuity and stability of reasoning across multi-stage penetration processes. Together, they jointly

assess both the practical success and the reliability of the reasoning behind the framework.

- **Credential Capture Success Rate (CCSR):** This metric quantifies the framework's ability to complete the penetration objective by successfully obtaining all valid credentials or flags. Formally, it is defined as:

$$\text{CCSR} = \frac{N_{\text{success}}}{N_{\text{total}}} \times 100\%, \tag{4}$$

  where N_{success} denotes the number of credentials that successfully captured, and N_{total} is the total number of credentials. A higher CCSR indicates a more complete and efficient exploitation process under real-world PT scenarios.

- **Stage Transition Success Rate (STSR):** This metric measures the system's capability to autonomously progress across PT stages. It is expressed as:

$$\text{STSR}_i = \frac{S_i}{T_i} \times 100\%, \tag{5}$$

  where S_i represents the number of successful transitions achieved in stage i, and T_i is the total number of attempts at that stage. A higher STSR reflects stronger situational awareness and continuity of reasoning between each PT stage.

Both metrics were evaluated across three independent episodes for each framework configuration, comparing the performance of *RefPentester* with the baseline framework under identical experimental conditions. During each episode, both systems received the same initial environment state, target context, and operator prompts to ensure a fair comparison. The CCSR emphasizes tangible task outcomes, while the STSR emphasizes reasoning robustness. Together, they offer a comprehensive perspective on the integration of reasoning and execution within the framework, as well as its overall system reliability.

4.3 Comparative Studies

To address **RQ1**, we conducted controlled comparative experiments on the *Sau* machine. The goal of this section is solely to quantify whether the proposed framework delivers measurable performance improvements over a GPT-4o baseline under identical operational conditions.

We first extracted the complete set of required PT actions from the official walk-through of the *Sau* machine. These actions were mapped to the five PT stages (i.e., information gathering, vulnerability identification, exploitation, post-exploitation, and credential capture), forming a ground-truth transition sequence. For each stage, both systems were allowed up to **five** reasoning iterations to advance before entering the terminal state. Failure to obtain the required credentials or flags within this limit resulted in a failed episode.

Two system configurations were evaluated: 1) **RefPentester**, the full proposed framework integrating the Process Navigator, Generator, and Reflector module; 2) **Baseline GPT-4o**, A direct application of a GPT-4o model, employing the identical configuration of *RefPentester* and a fundamental system prompt (i.e., You are a powerful penetration testing assistant...). Both systems received the same prompts, environmental states, and target contexts across three independent episodes to ensure fairness.

Table 20 summarizes the end-to-end CCSR achieved by both systems. *RefPentester* successfully captured all credentials in all three episodes, achieving a perfect CCSR of 100%, which means in each stage of each episode, the *RefPentester* can successfully move to the next stage within five tries, and finally reveal all the credentials (Each episode has two credentials that need to be discovered). In contrast, the baseline configuration failed to retrieve all the credentials in the experiment, resulting in an 83.3% success rate. This 16.7% performance gap indicates that *RefPentester* provides stronger operational completeness in multi-step PT workflows.

Table 20: PT Credential Capture Success Rate on *Sau* Machine

System	CCSR
RefPentester	**100%**
GPT-4o Baseline	83.3%

To further evaluate overall PT task progression, Table 21 compares the STSR across the four major stages during the PT process. Across all stages, *RefPentester* consistently achieved higher transition success rates. The most significant improvement occurs in *Vulnerability Identification*, where *RefPentester*'s 87.5% success rate starkly exceeds the baseline's 35.7%. This suggests that incorporating structured PT knowledge substantially improves the system's ability to identify viable attack vectors.

Table 21: PT Stage Transition Success Rate

Stage	**RefPentester**	**GPT-4o**
Information Gathering	80%	61.5%
Vulnerability Identification	87.5%	35.7%
Exploitation	52.9%	36.7%
Post-Exploitation	71.4%	29.1%

The experimental findings provide strong evidence that *RefPentester* substantially outperforms the GPT-4o baseline in both end-to-end credential capture and intermediate stage progression. The improvements stem from its ability to maintain coherent, multi-stage reasoning and retrieve relevant PT knowledge based on the identified PT stage, thereby reducing hallucination and reflecting based on previous failure attempts across the trajectory. Therefore, **RQ1** is answered affirmatively. *RefPentester* delivers significantly better AutoPT performance than an LLM-based baseline under identical PT conditions.

4.4 Discussions

Building on the quantitative findings reported in Section 4.3, this section provides a discussion of the mechanisms driving *RefPentester*'s performance and addresses the questions in RQ2 and RQ3. While Section 4.3 establishes that *RefPentester* achieves higher CCSR and consistently superior STSR across multiple PT stages, these numbers alone do not explain why the improvements occur. To provide a comprehensive interpretation, this section examines the system's behaviour

using qualitative evidence from **experiment logs**, revealing how knowledge structuring and reflective optimization influence *RefPentester*'s decision-making ability. Although no ablation studies were conducted to isolate individual components, the consistent patterns exhibited across three independent experimental episodes allow for a mechanism-level interpretation of *RefPentester*'s reasoning process. Specifically, three behavioural mechanisms emerge from the experiment logs: 1) improved recovery from failed operations; 2) suppression of hallucination through the reflection mechanism; and 3) enhanced consistency between PT stages. These mechanisms jointly explain the substantial STSR improvements and the framework's ability to maintain coherent multi-stage reasoning.

Reflection improves recovery from failed operations. One of the clearest behavioural differences between *RefPentester* and the baseline GPT-4o is the ability to recover from execution failures. Across multiple episodes, the baseline model frequently repeated invalid commands, such as referencing incorrect file paths, attempting operations without sufficient privileges, or invoking tools that did not exist in the target environment. Experiment logs show that GPT-4o often reissued the same incorrect command without adaptation, leading to stagnation within a PT stage.

In contrast, *RefPentester*'s Reflector module consistently generated explicit diagnostic information following a failed attempt (e.g., identifying missing dependencies, recognizing incorrect working directories, or revising misinterpreted tool outputs). This diagnostic information served as a corrective feedback mechanism: each reflection introduced a revised plan, often accompanied by target machine feedback specific adjustments. This iterative refinement stabilized the progression through complex PT stages such as Vulnerability Identification and Post-Exploitation, preventing error accumulation that would otherwise halt task flow.

Mechanistically, this behaviour can be interpreted as a form of implicit reward shaping. The Reflector analyze the failure operations and generates structured, human-readable explanations that assist the Generator in updating its next-step reasoning. This creates a feedback loop analogous to RL, but achieved entirely within an LLM-driven architecture. The effect is reflected directly in the STSR metrics, specifically in stages with high failure sensitivity. Most notably, Vulnerability

Identification shows the largest performance gap (87.5% vs. 35.7%). These findings support **RQ2** by demonstrating that reflective reasoning significantly improves failure recovery and reduces stage stagnation.

Reflection Suppresses Hallucination. Beyond error recovery, qualitative experiment logs suggest a second behavioural mechanism: reflection reduces hallucination by prompting the system to reevaluate uncertain or speculative operational guidance and its corresponding feedback. Baseline GPT-4o frequently produced hallucination-like operational guidance, such as assuming the presence of services not observed in prior scans or extrapolating exploitation paths based on nonexistent system components. These hallucinations emerged primarily in early-stage enumeration tasks, where ambiguous outputs from tools like `nmap` invite speculative reasoning. With the Reflector module, *RefPentester* consistently flagged inconsistencies between feedback from the target machine and prior assumptions. A typical reflective message is "The result contradicts the earlier scan; re-evaluating the...". The reflector module grounded the reasoning process with the retrieved PT knowledge and the target machine's feedback, reducing irresponsible extrapolations. Reflection, therefore, functioned as a mechanism for uncertainty calibration. Instead of moving forward with unverified assumptions, the system paused, reconsidered earlier steps with corresponding feedback, and sought additional context.

This behaviour explains why *RefPentester* achieves significantly higher STSR in Information Gathering and Vulnerability Identification, stages where ambiguity and partial information are common. The improved stability observed in these stages directly addresses **RQ2** by showing that reflective feedback not only corrects errors but also constrains speculative reasoning, thereby reducing hallucination.

Reflection Stabilizes Multi-Stage PT Reasoning. A third mechanism supported by the experiment logs concerns the stability of cross-stage PT reasoning. As defined in Table 9, *RefPentester* operates over a Stage Machine comprising ordered PT stages $\theta_1 \rightarrow \theta_7$, where each transition is triggered by its corresponding event (e.g., α, β, γ). Maintaining coherent reasoning across these stages is particularly important in PT workflows: inconsistent or irrelevant operations can break

the event sequence and prevent the system from advancing to the next stage.

The baseline GPT-4o frequently exhibited such inconsistencies. In several episodes, it produced operational guidance that contradicted earlier decisions or were incompatible with the expected following operations. These divergences caused the trajectory to stall, consuming all five allowed attempts for a stage without triggering the required event, which directly resulted in lower STSR scores.

RefPentester mitigates this issue through the joint effect of the predefined PT Stage Machine and the Reflector module. Building on the PT Stage Machine, the Reflector maintains a long-term memory (Success Log) that continually checks whether the current operational guidance remains compatible with the expected event of the active stage.

After each operation, the Reflector compares the newly observed feedback against both the retrieved PT knowledge, the current PT stage and the operational guidance. Suppose the system's proposed next operations deviate from the PT Stage Machine's prescribed transition. For example, attempting a post-exploitation action when the event γ (Exploited) has not been triggered, the Reflector will provide failure reasons to the Generator module to regenerate the operational guidance to realign the trajectory. This correction mechanism prevents stage drift and preserves coherent multi-stage reasoning throughout the PT workflow.

This mechanism explains the substantial improvements observed in Exploitation and Post-Exploitation STSR (52.9% vs. 36.7%, and 71.4% vs. 29.1%, respectively). By stabilizing the event-driven progression across PT stages, reflection enables *RefPentester* to maintain coherent multi-stage reasoning and sustain end-to-end task flow, thereby providing direct support for **RQ2**.

Implications for LLM-based AutoPT. The observed mechanisms also reveal qualitative interaction patterns between knowledge grounding and reflection, providing insight into **RQ3**. These patterns illustrate how *RefPentester* achieves interpretable and controllable multi-stage PT reasoning.

1. *Knowledge grounding narrows the reasoning space.* Across all experiment episodes, the identified PT stage and retrieved PT knowledge consistently served as the structural anchor

for operation selection. By grounding each operation in high-level knowledge, the system avoided speculative paths that were common in the baseline. This grounding effect is visible in the experiment logs where *RefPentester* references ATT&CK and OTG-derived guidance when generating enumeration, exploitation, or post-exploitation operational guidance. The mechanism improves interpretability because each operation can be traced back to a semantically meaningful knowledge entry, rather than being an opaque model-generated guess.

2. *Reflection dynamically refines operations when the environment feedback contradicts prior assumptions.* When the previous operations yielded no impact, the Reflector generated explicit diagnostic feedback to each operation, which allowed the Generator to revise the following operational guidance without abandoning the original PT intent. This demonstrates that Reflector operates as an adjustment layer on top of grounded knowledge rather than a replacement for it, creating a two-level reasoning structure that is inherently interpretable.

3. *Knowledge–reflection interaction yields stable reasoning trajectories across PT stages.* Experiment logs show that grounded knowledge defines *what* should be done within each PT stage, while reflection determines *how* to proceed when unexpected feedback is encountered. This interaction prevents trajectory drift. When the system proposed operations incompatible with the current stage (e.g., post-exploitation actions before γ was triggered), reflection redirected the reasoning back to the appropriate knowledge-aligned sequence. The resulting behaviour is interpretable because the system's corrections are explicitly justified by both the retrieved knowledge and the observed machine state.

These qualitative patterns demonstrate that *RefPentester*'s interpretability arises not from a single component, but from the coordinated interaction between knowledge grounding and reflective refinement. **Process Navigator** provides structural clarity, while **Reflector** supplies context-aware reasoning updates, enabling the system to maintain transparent and logically defensible decisions throughout multi-stage PT workflows. This analysis directly answers **RQ3**.

Chapter 5

Conclusions

This thesis presented a comprehensive study on advancing the automation and intelligence of PT through AI techniques. The research traced the evolution of AI-assisted AutoPT across three main paradigms, including expert systems, RL, and LLMs, and identified the key limitations that motivated the design of a new hybrid framework. Building upon these findings, we proposed a knowledge-informed, self-reflective AutoPT Framework, named *RefPentester*, which introduces a unified architecture that integrates knowledge retrieval, reflective optimization, and multi-stage reasoning to achieve adaptive, interpretable, and context-aware PT automation.

RefPentester introduces a unified architecture composed of the *Process Navigator*, the *Generator*, and the *Reflector*. The Process Navigator models the PT process through a predefined PT Stage Machine and retrieves stage-aligned knowledge via a hierarchical RAG pipeline. The Generator transforms this knowledge into actionable operational guidance, while the Reflector evaluates execution feedback, identifies potential reasoning failures, and provides corrective suggestions that stabilize multi-stage PT trajectories. Together, these components form a reasoning–feedback loop that refines decisions over time and preserves interpretability by grounding each operation in explicit knowledge and reflective justification.

The experimental evaluation of the HTB *Sau* machine demonstrates that *RefPentester* substantially outperforms the base GPT-4o model across multiple PT stages, achieving higher stage transition and credential capture success rates. Qualitative analysis further reveals three behavioural mechanisms: Improved recovery from failed operations, suppression of hallucination, and enhanced cross-stage consistency. These collectively explain the observed performance gains. These findings affirm the value of combining knowledge grounding with reflective optimization to support interpretable multi-stage PT reasoning.

Despite these promising outcomes, several limitations remain. The evaluation was conducted on a single HTB machine, which restricts claims regarding cross-environment generalization. Testing across diverse operating systems, service configurations, and PT scenarios is needed to validate robustness. Additionally, the current multi-session architecture introduces interdependence between modules, where reasoning errors in early stages may propagate across the pipeline. Future work should explore fault-tolerant orchestration mechanisms, such as rollback strategies, partial trajectory repair, or confidence-based gating, to reduce cascading errors. Finally, *RefPentester* does not yet incorporate adversarial or self-play training. An automated red–blue team setup could generate self-supervised adversarial trajectories, enabling the system to improve its reasoning robustness and strategic planning over time progressively.

Overall, this thesis demonstrates that integrating structured knowledge with reflective reasoning enables more interpretable, adaptive, and consistent AutoPT workflows. The proposed *RefPentester* framework provides a foundation for next-generation LLM-driven AutoPT systems, opening pathways toward scalable, self-improving, and security-aware AutoPT agents that can operate in increasingly complex real-world environments.

Bibliography

[1] P. A. Williams and A. J. Woodward, "Cybersecurity vulnerabilities in medical devices: a complex environment and multifaceted problem," *Medical Devices: Evidence and Research*, pp. 305–316, 2015.

[2] D. Craigen, N. Diakun-Thibault, and R. Purse, "Defining cybersecurity," *Technology innovation management review*, vol. 4, no. 10, 2014.

[3] W. Dimitrov, "The impact of the advanced technologies over the cyber attacks surface," in *Artificial Intelligence and Bioinspired Computational Methods: Proceedings of the 9th Computer Science On-line Conference 2020, Vol. 2 9*. Springer, 2020, pp. 509–518.

[4] Top 10 cybersecurity threats in 2025. [Online]. Available: https://certpro.com/cybersecurity-threats/

[5] C. R. Team. Rockyou2024: 10 billion passwords leaked in the largest compilation of all time. [Online]. Available: https://cybernews.com/security/rockyou2024-largest-password-compilation-leak/

[6] Technical details: Falcon content update for windows hosts. [Online]. Available: https://www.crowdstrike.com/en-us/blog/falcon-update-for-windows-hosts-technical-details/

[7] Content update causes global it outage, and other cybersecurity news to know this month. [Online]. Available: https://www.weforum.org/stories/2024/07/crowdstrike-global-it-outage-cybersecurity-news-july-2024/

[8] R. Baloch, *Ethical hacking and penetration testing guide*. Auerbach Publications, 2017.

[9] Penetration testing execution standard main page. [Online]. Available: http://www.pentest-standard.org/index.php

[10] A. Whitaker and D. P. Newman, *Penetration testing and network defense*. Cisco Press, 2005.

[11] M. Denis, C. Zena, and T. Hayajneh, "Penetration testing: Concepts, attack methods, and defense strategies," in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. IEEE, 2016, pp. 1–6.

[12] T. Wilhelm, *Professional penetration testing: Creating and learning in a hacking lab*. Elsevier, 2025.

[13] M. E. Armstrong, K. S. Jones, A. S. Namin, and D. C. Newton, "The knowledge, skills, and abilities used by penetration testers: Results of interviews with cybersecurity professionals in vulnerability assessment and management," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 62, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2018, pp. 709–713.

[14] F. Abu-Dabaseh and E. Alshammari, "Automated penetration testing: An overview," in *The 4th international conference on natural language computing, Copenhagen, Denmark*, 2018, pp. 121–129.

[15] H. Anderson, "Introduction to nessus," *Retrieved from Symantec*, vol. 16, 2003.

[16] S. Rahalkar, "Openvas," in *Quick Start Guide to Penetration Testing: With NMAP, OpenVAS and Metasploit*. Springer, 2018, pp. 47–71.

[17] A. Samad, S. Altaf, and M. J. Arshad, "Advancements in automated penetration testing for iot security by leveraging reinforcement learning," *evaluation*, vol. 8, p. 9, 2024.

[18] K. Abdulghaffar, N. Elmrabit, and M. Yousefi, "Enhancing web application security through automated penetration testing with multiple vulnerability scanners," *Computers*, vol. 12, no. 11, p. 235, 2023.

[19] S. Pargaonkar, "Advancements in security testing: A comprehensive review of methodologies and emerging trends in software quality engineering," *International Journal of Science and Research (IJSR)*, vol. 12, no. 9, pp. 61–66, 2023.

[20] Dradis. Dradis: Your cybersecurity collaboration platform. [Online]. Available: https://dradis.com/

[21] F. Security. Penetration testing reporting by faraday. [Online]. Available: https://faradaysec.com/penetration-testing-reporting/

[22] PowerShellMafia. Powersploit. [Online]. Available: https://github.com/PowerShellMafia/PowerSploit

[23] Empire: A powerful post-exploitation tool. [Online]. Available: https://www.ciso.inc/blog-posts/empire-powerful-post-exploitation-tool/

[24] TrustedSec. The social-engineer toolkit (set). [Online]. Available: https://github.com/trustedsec/social-engineer-toolkit

[25] T. A. ng Team. Aircrack-ng - wireless network penetration testing tool. [Online]. Available: https://www.aircrack-ng.org/

[26] K. D. Team. Kismet wireless network tool. [Online]. Available: https://www.kismetwireless.net/

[27] T. Z. Team. Owasp zap - the web application security scanner. [Online]. Available: https://www.zaproxy.org/

[28] Sullo. Nikto - web server scanner. [Online]. Available: https://github.com/sullo/nikto

[29] Insecure.org. Nmap - the network mapper. [Online]. Available: https://nmap.org/

[30] van Hauser / THC. thc-hydra. [Online]. Available: https://github.com/vanhauser-thc/thc-hydra

[31] PortSwigger. Burp suite - web security testing tool. [Online]. Available: https://portswigger.net/burp

[32] Google. american fuzzy lop - a security-oriented fuzzer. Version 2.57b released on 2020-06-30. [Online]. Available: https://github.com/google/AFL

[33] M. Eddington. Peach fuzzer. [Online]. Available: https://peachtech.gitlab.io/peach-fuzzer-community/

[34] Rapid7. Metasploit | penetration testing software, pen testing security. [Online]. Available: https://www.metasploit.com/

[35] Tenable. Nessus® - the first tool in your cybersecurity toolbox. [Online]. Available: https://www.tenable.com/products/nessus

[36] Greenbone. Greenbone openvas. [Online]. Available: https://www.openvas.org/

[37] A. Fioraldi, A. Mantovani, D. Maier, and D. Balzarotti, "Dissecting american fuzzy lop: a fuzzbench evaluation," *ACM transactions on software engineering and methodology*, vol. 32, no. 2, pp. 1–26, 2023.

[38] TrustedSec. The social-engineer toolkit (set) - trustedsec resources. [Online]. Available: https://trustedsec.com/resources/tools/the-social-engineer-toolkit-set

[39] H. M. Z. Al Shebli and B. D. Beheshti, "A study on penetration testing process and tools," in *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. IEEE, 2018, pp. 1–7.

[40] M. Rak, G. Salzillo, and D. Granata, "Esseca: An automated expert system for threat modelling and penetration testing for iot ecosystems," *Computers and Electrical Engineering*, vol. 99, p. 107721, 2022.

[41] R. Wang, K. Chen, C. Zhang, Z. Pan, Q. Li, S. Qin, S. Xu, M. Zhang, and Y. Li, "{AlphaEXP}: An expert system for identifying {Security-Sensitive} kernel objects," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4229–4246.

[42] M. Li, T. Zhu, H. Yan, T. Chen, and M. Lv, "Her-pt: An intelligent penetration testing framework with hindsight experience replay," *Computers & Security*, p. 104357, 2025.

[43] H. P. T. Nguyen, Z. Chen, K. Hasegawa, K. Fukushima, and R. Beuran, "Pengym: Pentesting training framework for reinforcement learning agents." in *ICISSP*, 2024, pp. 498–509.

[44] X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, "Pentestagent: Incorporating llm agents to automated penetration testing," *arXiv preprint arXiv:2411.05185*, 2024.

[45] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "{PentestGPT}: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 847–864.

[46] J. Xu, J. W. Stokes, G. McDonald, X. Bai, D. Marshall, S. Wang, A. Swaminathan, and Z. Li, "Autoattacker: A large language model guided system to implement automatic cyber-attacks," *arXiv preprint arXiv:2403.01038*, 2024.

[47] MITRE. Mitre att&ck® matrix. [Online]. Available: https://attack.mitre.org/matrices/enterprise/

[48] OWASP. Owasp testing guide. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/

[49] Y. Wang, S. Liu, W. Wang, C. Zhou, C. Zhang, J. Jin, and C. Zhu, "A unified modeling framework for automated penetration testing," *arXiv preprint arXiv:2502.11588*, 2025.

[50] C. Skandylas and M. Asplund, "Automated penetration testing: Formalization and realization," *Computers & Security*, vol. 155, p. 104454, 2025.

[51] Picus Security, "What is automated penetration testing?" n.d., accessed: 2025-05-19. [Online]. Available: https://www.picussecurity.com/resource/glossary/what-is-automated-penetration-testing

[52] CyCognito, "Automated pentesting: Pros/cons, key features & 5 best practices," https://www.cycognito.com/learn/exposure-management/automated-pentesting.php#:~:text=,option%20for%20frequent%20security%20assessments, 2025, accessed on: May 20, 2025.

[53] S. O. Alwabisi, "Ai in penetration testing: A systematic mapping study," 2025.

[54] N. Mohamed, "Artificial intelligence and machine learning in cybersecurity: a deep dive into state-of-the-art techniques and future paradigms," *Knowledge and Information Systems*, pp. 1–87, 2025.

[55] D. López-Montero, J. L. Álvarez-Aldana, A. Morales-Martínez, M. Gil-López, and J. M. A. García, "Reinforcement learning for automated cybersecurity penetration testing," *arXiv preprint arXiv:2507.02969*, 2025.

[56] J. Schwartz and H. Kurniawati, "Autonomous penetration testing using reinforcement learning," *arXiv preprint arXiv:1905.05965*, 2019.

[57] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[58] H. S. Al-Sinani and C. J. Mitchell, "Pentest++: Elevating ethical hacking with ai and automation," *arXiv preprint arXiv:2502.09484*, 2025.

[59] S. Thapaliya and S. Dhital, "Ai-augmented penetration testing: A new frontier in ethical hacking," *International Journal of Atharva*, vol. 3, no. 2, pp. 28–37, 2025.

[60] J. Yi and X. Liu, "Deep reinforcement learning for intelligent penetration testing path design," *Applied Sciences*, vol. 13, no. 16, p. 9467, 2023.

[61] X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, "Pentestagent: Incorporating llm agents to automated penetration testing," in *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, 2025, pp. 375–391.

[62] A. Abraham, "130: rule-based expert systems," *Handbook of Measuring System Design, edited by Peter H. Sydenham and Richard Thorn, John Wiley & Sons, Ltd. ISBN: 0-470-02143*, vol. 8, pp. 909–919, 2005.

[63] S. Y. Enoch, Z. Huang, C. Y. Moon, D. Lee, M. K. Ahn, and D. S. Kim, "Harmer: Cyber-attacks automation and evaluation," *IEEE Access*, vol. 8, pp. 129 397–129 414, 2020.

[64] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. Marriott, "Cyborg: A gym for the development of autonomous cyber agents," *arXiv preprint arXiv:2108.09118*, 2021.

[65] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent, automated red team emulation," in *Proceedings of the 32nd annual conference on computer security applications*, 2016, pp. 363–373.

[66] M. C. Ghanem, T. M. Chen, M. A. Ferrag, and M. E. Kettouche, "Esascf: expertise extraction, generalization and reply framework for optimized automation of network security compliance," *IEEE Access*, vol. 11, pp. 129 840–129 853, 2023.

[67] G. Chu and A. Lisitsa, "Agent-based (bdi) modeling for automation of penetration testing," *arXiv preprint arXiv:1908.06970*, 2019.

[68] N. Moscovich, R. Bitton, Y. Mallah, M. Inokuchi, T. Yagyu, M. Kalech, Y. Elovici, and A. Shabtai, "Autosploit: A fully automated framework for evaluating the exploitability of security vulnerabilities," *arXiv preprint arXiv:2007.00059*, 2020.

[69] Z. Wang, Y. Zhang, Z. Liu, X. Wei, Y. Chen, and B. Wang, "An automatic planning-based attack path discovery approach from it to ot networks," *Security and Communication Networks*, vol. 2021, no. 1, p. 1444182, 2021.

[70] H. Lei, Y. Ge, and Q. Zhu, "Adapt: A game-theoretic and neuro-symbolic framework for automated distributed adaptive penetration testing," in *MILCOM 2024-2024 IEEE Military Communications Conference (MILCOM)*. IEEE, 2024, pp. 7–12.

[71] G. Grov, J. Halvorsen, M. W. Eckhoff, B. J. Hansen, M. Eian, and V. Mavroeidis, "On the use of neurosymbolic ai for defending against cyber attacks," in *International Conference on Neural-Symbolic Learning and Reasoning*. Springer, 2024, pp. 119–140.

[72] D. K. Kejriwal and A. Sharma, "A hybrid neuro-symbolic framework for real-time detection of adversarial attacks in autonomous systems," *IRE Journals. https://www. irejournals. com/paper-details/1706618*, 2024.

[73] J. Zhao, W. Shang, M. Wan, and P. Zeng, "Penetration testing automation assessment method based on rule tree," in *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE, 2015, pp. 1829–1833.

[74] Z. Chen, F. Kang, X. Xiong, and H. Shu, "A survey on penetration path planning in automated penetration testing," *Applied Sciences*, vol. 14, no. 18, p. 8355, 2024.

[75] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI communications*, vol. 7, no. 1, pp. 39–59, 1994.

[76] C. Michel-Delétie and M. K. Sarker, "Neuro-symbolic methods for trustworthy ai: a systematic review," *Neurosymbolic Artificial Intelligence*, 2024.

[77] Microsoft Research / CyberBattleSim Team, "Cyberbattlesim," https://github.com/microsoft/CyberBattleSim, 2021, gitHub repository.

[78] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 2–10.

[79] B.-S. Kim, H.-W. Suk, Y.-H. Choi, D.-S. Moon, and M.-S. Kim, "Optimal cyber attack strategy using reinforcement learning based on common vulnerability scoring system." *CMES-Computer Modeling in Engineering & Sciences*, vol. 141, no. 2, 2024.

[80] S. Zhou, J. Liu, D. Hou, X. Zhong, and Y. Zhang, "Autonomous penetration testing based on improved deep q-network," *Applied Sciences*, vol. 11, no. 19, p. 8823, 2021.

[81] A. Piplai, A. Joshi, and T. Finin, "Offline rl+ ckg: A hybrid ai model for cybersecurity tasks," in *Proceedings of the AAAI 2023 Spring Symposium on Challenges Requiring the Combination of Machine Learning and Knowledge Engineering (AAAI-MAKE 2023)*, 2023.

[82] H. Liu, C. Liu, X. Wu, Y. Qu, and H. Liu, "An automated penetration testing framework based on hierarchical reinforcement learning." *Electronics (2079-9292)*, vol. 13, no. 21, 2024.

[83] S. Finistrella, S. Mariani, F. Zambonelli *et al.*, "Multi-agent reinforcement learning for cybersecurity: Approaches and challenges," in *Proceedings of the 25th workshop "from objects to agents", Bard (Aosta), Italy, July 8-10, 2024*, vol. 3735. CEUR-WS. org, 2024, pp. 103–118.

[84] K. Tran, M. Standen, J. Kim, D. Bowman, T. Richer, A. Akella, and C.-T. Lin, "Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing," *Applied Sciences*, vol. 12, no. 21, p. 11265, 2022.

[85] F. Pagano, M. Ceccato, A. Merlo, and P. Tonella, "Multi-agent deep reinforcement learning for penetration testing of iot devices through their mobile companion app," *Authorea Preprints*, 2025.

[86] A. Piplai, M. Anoruo, K. Fasaye, A. Joshi, T. Finin, and A. Ridley, "Knowledge guided two-player reinforcement learning for cyber attacks and defenses," in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2022, pp. 1342–1349.

[87] T. Chen, J. Liu, Y. Xiang, W. Niu, E. Tong, and Z. Han, "Adversarial attack and defense in reinforcement learning-from ai security view," *Cybersecurity*, vol. 2, no. 1, p. 11, 2019.

[88] M. O. Farooq and T. Kunz, "Combining supervised and reinforcement learning to build a generic defensive cyber agent," *Journal of Cybersecurity and Privacy*, vol. 5, no. 2, p. 23, 2025.

[89] M. C. Ghanem, T. M. Chen, and E. G. Nepomuceno, "Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks," *Journal of Intelligent Information Systems*, vol. 60, no. 2, pp. 281–303, 2023.

[90] R. Elderman, L. J. Pater, A. S. Thie, M. M. Drugan, and M. A. Wiering, "Adversarial reinforcement learning in a cyber security simulation," in *9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*. SciTePress Digital Library, 2017, pp. 559–566.

[91] S. H. Oh, M. K. Jeong, H. C. Kim, and J. Park, "Applying reinforcement learning for enhanced cybersecurity against adversarial simulation," *Sensors*, vol. 23, no. 6, p. 3000, 2023.

[92] A. Piplai, "Knowledge graphs and reinforcement learning: A hybrid approach for cybersecurity problems," Ph.D. dissertation, University of Maryland, Baltimore County, 2023.

[93] Q. Li and J. Wu, "Efficient network attack path optimization method based on prior knowledge-based ppo algorithm," *Cybersecurity*, vol. 8, no. 1, pp. 1–14, 2025.

[94] A. C. Moreno, A. Hernandez-Suarez, G. Sanchez-Perez, L. K. Toscano-Medina, H. Perez-Meana, J. Portillo-Portillo, J. Olivares-Mercado, and L. J. García Villalba, "Analysis of autonomous penetration testing through reinforcement learning and recommender systems," *Sensors*, vol. 25, no. 1, p. 211, 2025.

[95] H. V. Nguyen, S. Teerakanok, A. Inomata, and T. Uehara, "The proposal of double agent architecture using actor-critic algorithm for penetration testing." in *ICISSP*, 2021, pp. 440–449.

[96] A. Happe and J. Cito, "Getting pwn'd by ai: Penetration testing with large language models," in *Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering*, 2023, pp. 2082–2086.

[97] J. Huang and Q. Zhu, "Penheal: A two-stage llm framework for automated pentesting and optimal remediation," in *Proceedings of the workshop on autonomous cybersecurity*, 2023, pp. 11–22.

[98] G. Palma, G. Cecchi, M. Caronna, and A. Rizzo, "Leveraging large language models for scalable and explainable cybersecurity log analysis," *Journal of Cybersecurity and Privacy*, vol. 5, no. 3, p. 55, 2025.

[99] A. Krašovec, G. Steri, G. Karopoulos, and M. Trapani, "Large language models for cyber threat intelligence: Extracting mitre with llms," in *International Conference on Availability, Reliability and Security*. Springer, 2025, pp. 80–89.

[100] S. Ren and S. Chen, "Large language models for cybersecurity intelligence, threat hunting, and decision support," *Computer Life*, vol. 13, no. 3, pp. 39–47, 2025.

[101] X. Wu, Y. Tian, Y. Chen, P. Ye, X. Cui, J. Jia, S. Li, J. Liu, and W. Niu, "Curriculumpt: Llm-based multi-agent autonomous penetration testing with curriculum-guided task scheduling," *Applied Sciences*, vol. 15, no. 16, p. 9096, 2025.

[102] H. Kong, D. Hu, J. Ge, L. Li, T. Li, and B. Wu, "Vulnbot: Autonomous penetration testing for a multi-agent collaborative framework," *arXiv preprint arXiv:2501.13411*, 2025.

[103] Y. Li, H. Dai, and J. Yan, "Knowledge-informed auto-penetration testing based on reinforcement learning with reward machine," *arXiv preprint arXiv:2405.15908*, 2024.

[104] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 73, pp. 173–208, 2022.

[105] HackTheBox. Active machines list. [Online]. Available: https://app.hackthebox.com/machines/list/active

[106] Openvpn website. [Online]. Available: https://openvpn.net/

[107] OpenAI. Hello gpt-4o. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[108] Pinecone. llama-text-embed-v2 nvidia. [Online]. Available: https://docs.pinecone.io/models/llama-text-embed-v2

[109] Pinecone. Pinecone database quickstart. [Online]. Available: https://docs.pinecone.io/guides/get-started/quickstart