AutoPenGPT:Highly automated penetration testing framework based on LLM

Tianqi Jiang

Supervisors:
Aniket Mahanti
Ranesh Naha

School of Computer Science
The University of Auckland

A thesis submitted in fulfilment of the requirements for the degree of
Master of Science in Computer Science,
the University of Auckland, 2025.

Abstract

This thesis explores the integration of LLM to enhance automated penetration testing, addressing the shortcomings of previous models such as poor context memory, high susceptibility to hallucinations, and low automation levels. Existing approaches often fall short in dynamically complex cyber environments, leading to inefficient and error-prone testing processes. Our proposed framework utilizes external knowledge bases to enhance information coherence across multi-step tasks and integrates counterfactual analysis to significantly reduce hallucinations and errors in LLM outputs. The experimental results showcase notable advancements: AutoPenGPT attains a 20% task completion rate in complex scenarios, markedly outperforming traditional tools like Nessus and PentestGPT. Moreover, AutoPenGPT maintains 57.1% accuracy in contextual memory in complex tasks, and reduces hallucination rates to 12.7%, demonstrating its superior adaptability and effectiveness in addressing the critical gaps in automated penetration testing. These improvements highlight our approach's potential to significantly elevate the efficiency, accuracy, and scalability of security testing in evolving threat landscapes.

Acknowledgements

I would like to express my deepest gratitude to all those who directly or indirectly supported me throughout the course of this research. I am especially thankful to my advisors, Professor Aniket Mahanti and Professor Ranesh Naha, for their invaluable guidance, enthusiastic encouragement, and insightful critiques of my research work.

Professor Aniket Mahanti has been an exceptional mentor in technical as well as academic aspects, offering great assistance and insights that were crucial for the scientific exploration included in my work. His expertise and meticulous attention to detail have been a great motivator and helped me maintain my focus.

Similarly, Professor Ranesh Naha's guidance on research methodologies and data analysis has been profoundly insightful. His rigorous approach and attention to detail ensured the scientific integrity of my experimental design and the accuracy of my data analyses.

Lastly, my heartfelt thanks go to my family and friends for their understanding and support throughout my academic journey. Their constant encouragement has been a pillar of strength for me.

Tianqi Jiang

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cybersecurity is a key area that focuses on protecting computer systems, networks, and data through technical, policy, and regulatory measures against unscrupulous intrusions such as unauthorized access, data breaches, and malicious sabotage. This includes defending against not only external but also internal threats, and ensuring the integrity, confidentiality and availability of information to support the secure operations of individuals, businesses and governments in the digital world[65]. With the rapid development of technology and its increasing integration into every aspect of our lives, the demand for cybersecurity is also growing significantly. This area covers not only a wide range of practices, such as security policy development and risk management guidelines, but also the development and deployment of applications and safeguards for protecting data.

Nowadays, the challenges of cybersecurity are becoming increasingly complex. From data breaches and ransomware attacks to state-sponsored cyber espionage, a variety of cyber threats continue to challenge existing security defenses, underscoring the need for a robust and flexible cyber defense strategy. The scale of these challenges is accentuated by the significant financial implications[46]. The cybersecurity venture capital firm[1] predicts that global cybersecurity spending will exceed $1.75 trillion from 2021 to 2025, with an annual growth rate of 15 percent. However, these expenditures are still far from enough to offset the cost of cybercrime, which is expected to rise to $10.29 trillion by 2025 from $0.86 trillion in 2018[2]. According to IBM[3], the average cost of a data breach reached an all-time high of $4.45 million in 2023. This represents a 2.3% increase over the previous year, continuing a steady upward trend

[1] https://www.esentire.com/resources/library/2023-official-cybercrime-report
[2] https://www.statista.com/forecasts/1280009/cost-cybercrime-worldwide
[3] https://www.ibm.com/reports/data-breach

since 2020. As the number and cost of cybercrime continues to grow, cybersecurity has evolved from a mere technical necessity to a cornerstone of social stability, equity, and security[40]. At the same time, existing security measures are facing various challenges, especially in the application of emerging technologies such as large language models. We must re-examine the specific problems and challenges brought by these technologies. The integration of these technologies is not only about the security of data, but also about how to effectively use these advanced tools to enhance our cyber defenses.

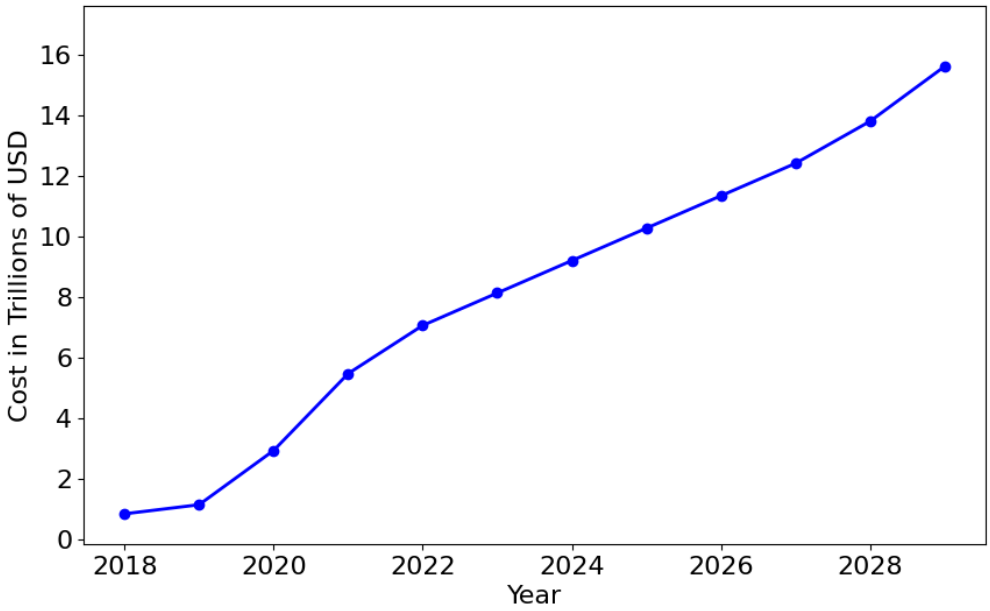

Figure 1.1: Estimated Cost of Cybercrime Worldwide 2018-2029

PT(Penetration Test), an important means of testing the effectiveness of network security protection, is used to assess the security of computer systems, networks or applications. By simulating the behavior of the attacker, the target system can be attacked without causing damage, and vulnerabilities, weaknesses, and security vulnerabilities in the system can be found, and suggestions for improving the security defense of the organization can be provided. PT application scenarios include, but are not limited to, IOT security, industrial security, mobile security, IOV security, Web security, and kernel security. Traditional PT typically includes the following steps:

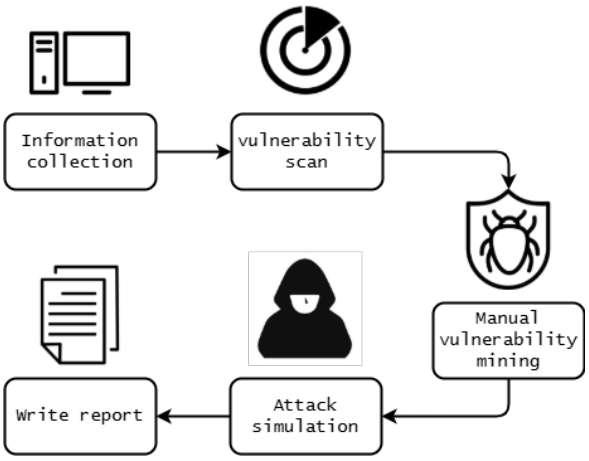

Figure 1.2: Penetration Test Flow

- `Information collection` : Collect information about the target system, network, or application, including IP addresses, domain names, and system architectures.
- `vulnerability scan` : Use automated tools to scan the system for known vulnerabilities and weaknesses.
- `Manual vulnerability mining` : Penetration testers use manual manipulation and techniques to dig deep into a system for possible unknown vulnerabilities.
- `Attack simulation` : Simulate the behavior of an attacker and try to exploit the discovered vulnerability to gain unauthorized access, permissions, or other attack targets.
- `Write report` : Write a PT report detailing the vulnerabilities found, attack paths, and recommended fixes.

During PT, security professionals simulate attacks using various techniques, such as network scanning, exploitation, and social engineering, to identify vulnerabilities in the system. In addition to identifying weaknesses, PT also assesses the system's responsiveness to different types of attacks and its ability to recover from potential damage. Regular security testing can help organizations patch vulnerabilities, enhance security measures, and reduce the risk of cyber attacks. The demand for skilled professionals in this field is growing rapidly. Cybersecurity venture capital firms report[4] a

[4]https://apnews.com/press-release/ein-presswire-newsmatics/technology-steve-morgan-ein-presswire-newsmatics-2c99c00b8673966bde5eca81f6535320

350 percent increase in cybersecurity job openings globally, reaching 3.5 million positions by 2021. The shortage is expected to continue. Further proving the importance of cybersecurity in today's digital environment. The talent shortage, coupled with the rising cost of cybersecurity, highlights the urgent need for automation. By automating PT processes, organizations can increase efficiency, reduce costs, and help bridge the cybersecurity workforce gap.

1.1 Motivation

Automated PT has become a fundamental requirement to defend against the latest digital systems, as cyber threats continue to evolve. With ML(Machine Learning) and LLM(Large Language Model)[62], security applications can improve their ability to effectively identify, resolve, and minimize security issues without human intervention. These operations leverage advanced technologies and platforms to perform duties typically performed by human analysts, such as identifying threats and responding to incidents. This automation not only speeds up the process of addressing threats, but also reduces the possibility of human error. Additionally, while humans cannot handle large amounts of data effectively, automated network operations can efficiently manage this challenge. Implementing automation in cybersecurity operations can also address the growing shortage of cybersecurity skills and lack of experience[5]. According to a report by Grand View Research, the security automation market is projected to grow significantly, driven by these demands for more efficient security solutions [6]. Furthermore, the Cyentia Institute's report highlights that automation in cybersecurity has led to a 30% reduction in response times to security incidents and a significant decrease in false positives, showcasing its increasing viability for repetitive tasks [7]. This progress allows security personnel to concentrate on more complex tasks, optimizing both effectiveness and resource allocation [30].

In terms of content, artificial PT needs to include: information collection, vulnerability detection, vulnerability exploitation, permission maintenance, post-penetration, report generation and other conventional steps, the general market automated PT products also contain these functions. Generally, automated tools need to answer two core questions:

[5]https://www.proquest.com/wire-feeds/global-penetration-testing-market-size-is/docview/2382032821/se-2

[6]https://www.grandviewresearch.com/industry-analysis/security-automation-market

[7]https://library.cyentia.com/reports/2023-state-of-cybersecurity-automation-adoption

- How effectively do automated tools report and manage non-critical risk points or false positives?

- To what degree do automated tools improve the efficiency of PT?

The degree to which these two problems are solved determines the advantages and disadvantages of automated tools. Manual PT usually relies on a large number of PT tools to perform detailed and rigorous screening of the target system. At the same time, the reliability of the test results varies with the experience level of the PT engineer. For the same vulnerability, PT engineers will also be affected by factors such as business knowledge and vulnerabilities to a certain extent, and make different judgments about the degree of harm. Existing automated PT tools are still not smart enough. Most automated penetration products simply combine vulnerability scanning tools with related exploitation tools. In the process of PT, network security experts still need to make decisions with rich experience in key links, such as the configuration of payloads, the choice of attack methods.

And as described by security audit expert Pozdniakov [36], manual PT relies primarily on identifying and exploiting flaws in application software, such as executing code to trigger system vulnerabilities and deploying payloads. In practice, penetration testers often use a combination of tools and technical expertise to perform tests. However, these tools rely on the judgment of security experts and often do not automatically adapt to changing environments, so updating attack strategies is limited. In addition, as systems become more complex, conducting safety assessments manually can be time-consuming and laborious. In this regard, it is mentioned in the literature[12] that automated PT methods are gradually receiving attention.

LLMs have been shown to achieve impressive results on natural language tasks, and security researchers are beginning to use them in offensive and defensive systems[64][23]. In the field of cybersecurity, there has been a number of research efforts using LLMs to focus on the pre-disclosure stages of attacks, such as phishing and malware generation. However, until now, there has been a lack of comprehensive research on whether LLM-based systems can be utilized to simulate the compromised phase of a human-operated or ”hand-pressed keyboard” attack, which is typical in a variety of attack techniques and environments. As LLMs inevitably evolve, they may be able to automate both the pre - and post-intrusion attack phases. This shift can transform organizational attacks from rare, expert-led events to frequent, expert-free, automated operations performed at automated speed and scale. This risk has fundamentally changed global computer security with consequential significant economic implications, and one goal

of this work is to better understand these risks now so that we can better prepare for these inevitable and increasingly powerful LLMs.

However, despite the great potential of LLM, its application in automated PT faces many technical challenges and limitations. First, the LLM's limited understanding of context when dealing with long sequence conversations or complex interactions reduces its effectiveness in continuous PT tasks [7]. In addition, LLMs sometimes generate scripts that are not present in the environment, leading to a so-called "hallucination" problem[4], which not only reduces the accuracy of PT, but also misleads testers. "Hallucination" is the phenomenon that the generation system produces inaccurate or false output without exact data support. Another important challenge is the knowledge lag of LLMs, whose knowledge base is not updated in a timely manner to effectively identify and exploit emerging security vulnerabilities. The challenges of multi-step utilization scenarios and I/O processing issues also greatly limit the effectiveness of LLM applications in complex PT scenarios. Finally, built-in content filtering and usage policies limit the LLM's ability to generate security-related content, while environment and execution environment tracking challenges add complexity to its application in the dynamic PT environment [48].

1.2 Research Questions

In view of the shortcomings mentioned above, this thesis aims to address the following research questions:

- How to improve LLM information accuracy in automated PT tasks?
- How can context persistence be improved in automated PT tasks to mitigate information forgetting and context loss issues?

1.3 Research Objectives

The primary objective of this research is to enhance the efficiency, reliability, and adaptability of automated PT frameworks by integrating advanced LLM and RAG (Retrieval-Augmented Generation)technologies. In the context of existing literature, LLMs have been explored for various applications in cybersecurity, but their use in PT has often been limited by challenges related to the automation of complex workflows, the retention and utilization of contextual information, and the high incidence of erroneous or hallucinatory outputs [26] [29]. While frameworks like PentestGPT have

attempted to harness LLMs for PT, they typically struggle with maintaining context over extended operations and frequently require human intervention [8].

Our research aims to address these gaps by developing a sophisticated automated PT framework that not only leverages the capabilities of LLMs to process and synthesize vast amounts of information dynamically but also incorporates a novel context management system. This system uses external knowledge bases to maintain logical consistency and streamline information flow across multi-step testing processes, thus ensuring the framework's adaptability to the changing parameters of network environments [55].

To mitigate the common issue of hallucinations[19] in LLM outputs, our research introduces a counterfactual analysis strategy that enhances output reliability by systematically identifying and correcting potential inaccuracies before they affect the testing workflow [39]. Moreover, we explore the implementation of a modular multi-agent architecture that supports the autonomous generation and execution of testing tasks, alongside adaptive real-time adjustments, significantly reducing the need for manual oversight and intervention [41].

The effectiveness of this integrated framework will be rigorously evaluated against conventional tools like Nessus, Deep Exploit, and PentestGPT. We will utilize metrics that reflect task completion rates, accuracy of context memory retention, reduction of hallucination frequency, and overall improvements in operational efficiency. By systematically comparing these outcomes, this research seeks to substantiate the advantages of our approach and establish a new benchmark in the field of automated PT, advancing the state of the art in cybersecurity testing by proposing a system that not only meets current security demands but is also scalable and robust enough to anticipate and mitigate emerging threats [32].

1.4 Results Overview

In this thesis, the development and implementation of the AutoPenGPT framework is detailed, demonstrating an advanced tool designed for automation in PT tasks that integrates machine learning techniques to enhance efficiency, accuracy, and reliability in cybersecurity assessments. The framework capitalizes on a LLM to dynamically adjust tasks and reduce human intervention, thereby improving the overall PT process.

The evaluation of AutoPenGPT highlights its superior performance in automating complex multi-step cybersecurity tasks, clearly surpassing traditional tools such as Nessus, Deep Exploit, and PentestGPT. Key findings are summarized in the Table

1.1, which details the framework's enhancements in task completion rates, contextual memory capabilities, and reduced hallucination frequencies across varying levels of task complexities.

Table 1.1: Summary of Main Results for AutoPenGPT across Task Complexities

Metric	**Simple Tasks**	**Medium Tasks**	**Complex Tasks**
Task Completion Rate	100%	60%	20%
Contextual Memory Retention	91.3%	78.6%	57.1%
Contextual Memory Accuracy	86.4%	71.5%	63.8%
Hallucination Rate	7.3%	9.2%	12.7%
Severe Hallucination Rate	1.8%	3.2%	5.5%

These results confirm that AutoPenGPT significantly enhances the efficiency and reliability of PT, demonstrating a marked improvement in the automation of information gathering, vulnerability scanning, exploitation, and reporting phases. Through rigorous comparative analysis and detailed experimental setups, AutoPenGPT has been proven to offer considerable advancements over existing methods, establishing a new benchmark for automated PT frameworks. The framework's ability to meet and exceed the requirements of modern cybersecurity challenges showcases its potential to transform PT practices through automation and intelligent decision-making.

1.5 Contributions

This thesis contributes to the field of automated PT in the following ways:

1. **Advanced Context Management Framework:** We propose a context management framework that enhances LLM memory, improving context retention and utilization in multi-step PT. This framework significantly outperforms existing methods in terms of context persistence and accuracy.

2. **Hallucination Mitigation Strategies:** We introduce strategies to mitigate hallucinations in LLM-generated PT commands. These strategies improve reliability by reducing hallucination rates across both simple and complex tasks.

3. **Automated PT Framework:** We develop a novel automated PT framework that orchestrates task decision-making, vulnerability exploitation, and multi-agent collaboration. The framework enhances the automation of PT tasks, show-

ing clear improvements in task completion rates and efficiency when compared to tools like Nessus and PentestGPT.

1.6 Thesis organization

The rest of the paper is organized as depicted in Figure 1.3. Chapter 2 discusses previous work, provides a comprehensive comparison of existing PT frameworks, and presents solutions along with justifying the choice of the chosen method in the final section. Chapter 3 introduces the design of each module in the proposed framework in detail. In chapter 4, the effectiveness of the framework is evaluated by the experimental results. Finally, Chapter 5 concludes by highlighting the strengths and weaknesses of the framework and outlining the direction of future work.

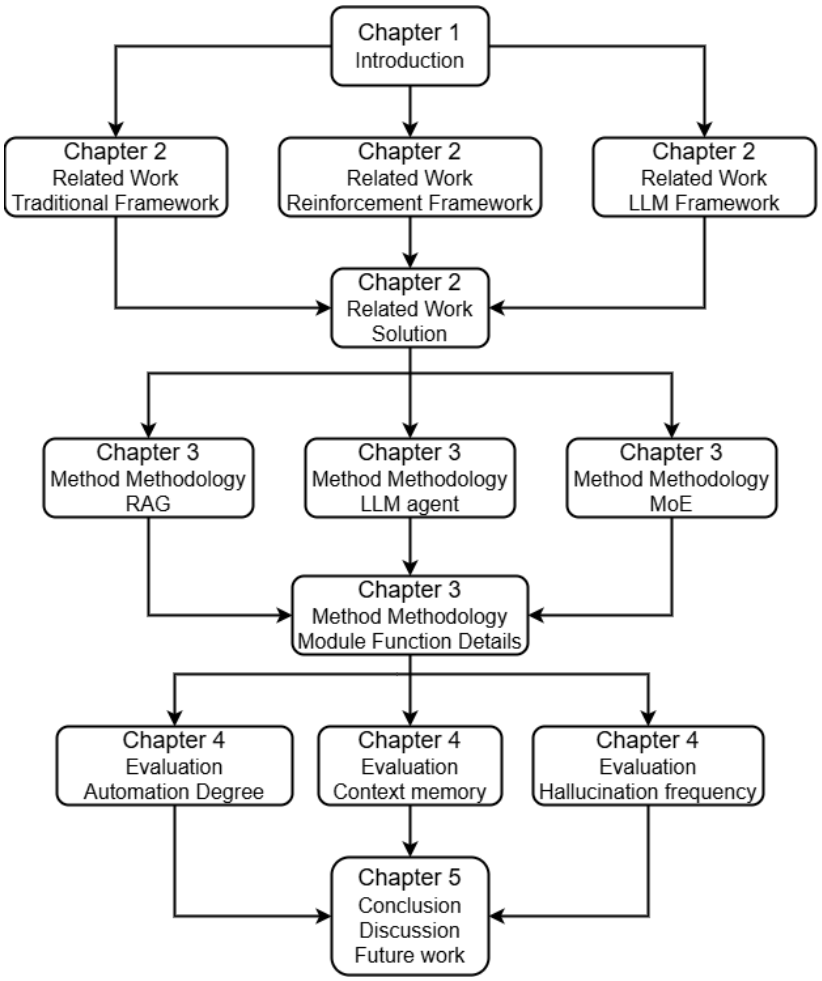

Figure 1.3: Thesis Structure Diagram

Chapter 2

Related Work

In this chapter, we conduct a detailed review of the various frameworks used in automated PT(Penetration Test), tracing their development as responses to the increasingly sophisticated challenges of cybersecurity. The objective of this chapter is to delineate the advantages and shortcomings of traditional, RL(Reinforcement Learning)-based, and LLM(Large Language Model)-based frameworks. By evaluating different studies and their outcomes, this chapter lays the groundwork for understanding and potentially enhancing these frameworks. We begin by examining traditional automated PT frameworks that rely on vulnerability scanning tools. Next, we analyze the incorporation of RL techniques in PT, and finally, we delve into the application of LLMs, discussing their capabilities and the specific challenges they face, such as memory limitations and the risk of generating inaccurate or misleading results.

2.1 Existing Framework

In the realm of automated PT, several frameworks have been established, each designed to address specific aspects of cybersecurity vulnerabilities through distinct methodologies. These frameworks can be broadly categorized into traditional, RL-based, and LLM-based frameworks. Traditional frameworks leverage established scanning tools and penetration techniques to identify and exploit known vulnerabilities efficiently. RL-based frameworks, on the other hand, employ adaptive algorithms to dynamically navigate through security systems, learning and optimizing strategies as they interact with the environment. Lastly, LLM-based frameworks utilize the advanced capabilities of generative AI to simulate attacks and generate complex penetration strategies. This section provides an in-depth examination of each framework

type, discussing their methodologies, operational nuances, and the specific challenges they aim to overcome, thereby offering insights into their effectiveness and limitations within the ever-evolving landscape of network security.

2.1.1 Traditional Framework

Traditional automated PT frameworks are typically based on scans generated by vulnerability scanning tools to further validate identified vulnerabilities[35] by incorporating specific PT tools such as Metasploit[1] or Nmap[2]. The main advantages for such frameworks are high test efficiency and the ability to cover a wide library of known vulnerabilities. However, traditional frameworks have limitations in dealing with complex and changing environments[44], especially emerging threats that have not yet been identified. In addition, traditional frameworks lack the ability to self-learn and adjust testing strategies independently according to dynamic changes in the environment[49], thus failing to reach the level of independent judgment of manual penetration testers.

These traditional frameworks also present significant disadvantages, particularly in terms of their complex configuration and the reliance on manual operation. This issue becomes particularly problematic in environments with frequent system updates, as these frameworks struggle to adapt to changes promptly, resulting in outdated configurations. Consequently, the inability to keep up with system updates leads to a high false positive rate, which in turn increases the workload of security analysts, who must manually review and filter out these false alarms.

To address this issue, there have been efforts in recent years to incorporate AI assistance into traditional frameworks, with the goal of reducing the false positive rate. However, these attempts have met with limited success and have not been widely adopted in practice.

Take Nessus[3] as an example. Prior to initiating a PT on a target system, Nessus requires a considerable number of plug-ins to be configured. This process can be cumbersome, and once the scan begins, the plug-ins often consume a significant amount of system resources. While Nessus is capable of identifying general vulnerabilities, it tends to struggle with more specific types, such as logical, business, and composite vulnerabilities.

[1] https://www.metasploit.com/
[2] https://nmap.org
[3] https://www.tenable.com/products/nessus

Logical vulnerabilities, for instance, stem from flaws in the business logic or process design of an application, causing the system to behave in unexpected ways—such as allowing negative value payment operations that may result in illegal refunds. Business logic vulnerabilities represent a more specific subset, often involving the abnormal use or circumvention of business processes, like bypassing payment processes to obtain goods or services without proper authorization. Composite vulnerabilities, on the other hand, arise from a combination or chain exploitation of multiple vulnerabilities. Individually, these vulnerabilities may not seem dangerous, but when exploited together, they can lead to severe security breaches, such as attackers first gaining partial user data through information disclosure vulnerabilities, then using privilege escalation flaws to elevate their access rights, and finally exploiting these rights to access or alter sensitive data.

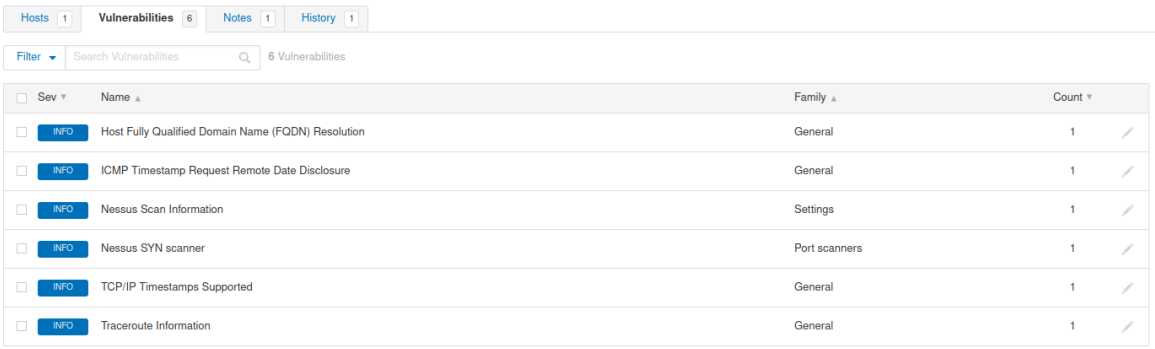

Figure 2.1: Nessus Result

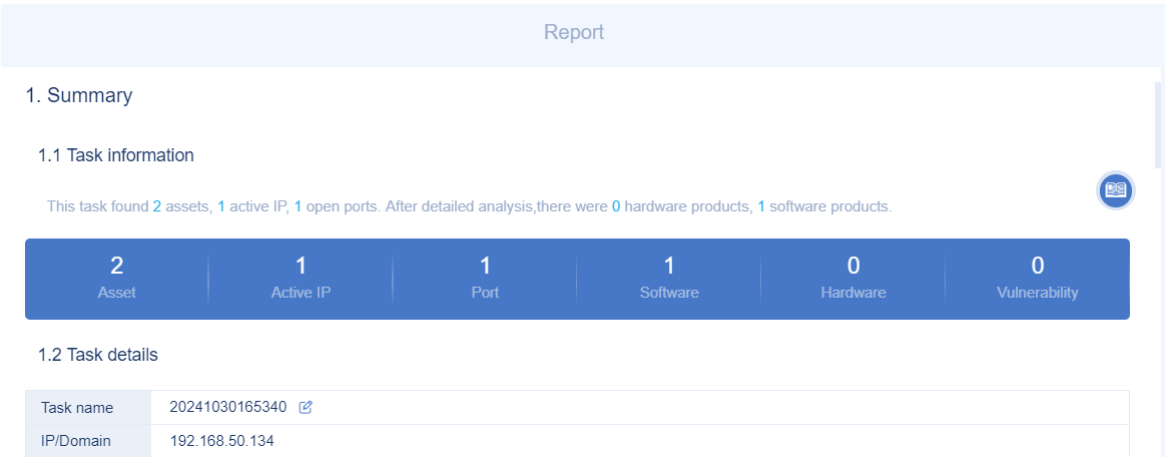

Figure 2.2: Goby Result

Figure 2.1 shows the result of scanning the target system using Nessus and Figure 2.2 shows the result of scanning the target system using Goby. Although the system had a file upload vulnerability, Nessus was unable to identify the vulnerability. Similarly, tests using other conventional vulnerability scanning tools did not identify the vulnerability. This shows that conventional automated PT tools have significant limitations in dealing with logic vulnerabilities, business vulnerabilities, complex vulnerabilities, and vulnerabilities that require complex operations.

2.1.2 Reinforcement Framework

In recent years, the application of RL in automated PT has garnered significant attention[51]. RL is a subfield of ML(Machine Learning) that focuses on how agents should act to maximize expected rewards based on interactions with an environment. It is one of the three primary paradigms of machine learning, alongside supervised and unsupervised learning. In the RL framework[52], an agent observes the state of the environment, takes an action that impacts the environment, and receives a corresponding reward. This feedback loop enables the agent to adjust its behavior in a way that maximizes long-term rewards. The essential components of an RL problem—state, action, and reward—are crucial for the effective application of RL algorithms in any domain. The basic process is illustrated in Figure 2.3.

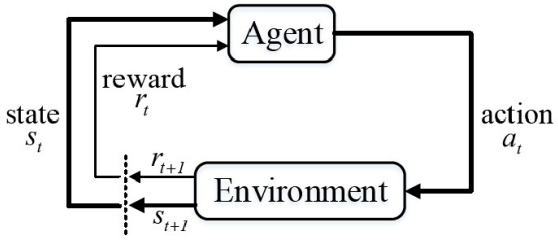

Figure 2.3: Reinforcement Flow

While RL offers significant advantages in adapting to dynamic and complex environments, its implementation in automated PT has faced certain challenges. Several RL-based automated PT frameworks, such as DeepExploit, AutoPentest-D RL, and GAIL-PT[5], have shown promising results in automating certain penetration tasks. However, these frameworks suffer from a critical shortcoming: the lack of effective correlation between states and actions, often resulting in pseudo-RL behavior[24]. This gap reduces their effectiveness in handling real-world penetration scenarios, where environments are often unpredictable and dynamic. Proper definition and application

of the three key RL elements—state, action, and reward—are critical to overcoming these challenges and improving the performance of RL-based PT frameworks.

Recent advancements in applying RL[59] to automated PT have shown promise, particularly in dynamic and complex network environments. Traditional automated PT frameworks, such as DeepExploit [47], leverage RL to enhance vulnerability detection and exploitation by automating decision-making processes through trial-and-error learning integrated with tools like the Metasploit framework. These frameworks use RL algorithms like asynchronous Actor-Critic Agents (A3C) to adaptively select the most effective penetration paths based on the current state of the system. However, a significant limitation is the static nature of state definitions in these frameworks, which often include immutable system parameters like operating system versions and port services. This static approach hampers the RL's ability to effectively learn and adapt, leading to actions that lack correlation with state transitions, thereby reducing the overall efficacy and resembling brute-force methods more than intelligent decision-making.

Furthermore, frameworks like GAIL-PT and the INNeS model proposed by Chen et al. [27] have attempted to refine the definition of states and actions within RL models to better suit the dynamic requirements of PT. These models aim to enhance the granularity of state representation and action relevance, thereby improving decision-making accuracy. However, despite these improvements, the practical application of these advanced RL models in large-scale network environments remains challenging due to high computational demands and limited scalability.

Additionally, Ghanem, Chen, and Nepomuceno's introduction of the IAPTF framework [11], which treats PT tasks as Partially Observable Markov Decision Processes (POMDPs) [42], represents a significant step forward. By employing hierarchical network representations and focusing on cluster-based attack strategies, this framework aims to manage the complexity of large networks more effectively. However, this approach can potentially overlook intricate attack vectors that a human hacker might exploit, highlighting a gap in the framework's ability to cover all possible attack paths.

In the realm of automated PT, my research shifts the focus toward utilizing Large Language Models (LLMs) to overcome some of these challenges. Unlike RL-based frameworks, LLMs offer potential in generating sophisticated attack scripts and managing multi-step penetration paths through advanced natural language processing capabilities. This shift is driven by LLMs' ability to process vast amounts of textual data and generate contextual outputs, which is crucial for identifying and exploiting com-

plex security vulnerabilities that are not typically covered by traditional or RL-based systems.

2.1.3 LLM Framework

In the area of automated PT, the integration of LLM like GPT-4 has introduced significant advancements, particularly in generating complex attack scripts and navigating multi-step penetration paths. LLMs, including models like GPT, BERT, CTRL, and BART, leverage extensive text data training to enhance natural language understanding and generation, making them adept at producing nuanced and contextually relevant outputs for cybersecurity applications [10][50][28]. This application, while innovative, reveals significant gaps in current technology, particularly in automation, memory capacity, and the management of erroneous outputs, commonly known as hallucinations.

The research surrounding PentestGPT, as conducted by Deng, provides a clear example of these challenges. PentestGPT utilizes Large Language Model (LLM) technology to improve its ability to comprehend and adapt to various penetration testing (PT) scenarios by analyzing contextual information within the testing framework. However, the framework's effectiveness is hampered by the LLM's limited memory for lengthy task sequences, which leads to high error rates and a dependency on frequent human intervention to correct and guide the testing process, as illustrated in Deng's findings [8].

Furthermore, Hape and Cito's evaluation of GPT-3.5 highlights its feasibility in initial task planning and vulnerability detection but points out its inadequacies in handling more complex, multi-step tasks, which reduces its automation capabilities and overall effectiveness in real-world applications [63]. Similarly, Xu's development of AutoAttack aims to enhance the accuracy of LLMs by optimizing their task execution processes. Despite improvements, the model still faces significant challenges with situational amnesia and hallucinations, showing that even with advancements, the models cannot fully rely on their generated data without human oversight [54].

Hilario's research into generative AI's role in PT showcases the potential for creating adaptable and creative testing environments. Nonetheless, this approach still encounters hallucination issues, where the LLMs generate misleading or incorrect outputs that could potentially jeopardize the integrity of PT results [18].

A particularly illustrative case by Fang et al. explores the autonomous exploitation capabilities of LLMs against real-world vulnerabilities. Their study evaluated GPT-4 and other models against a set of one-day vulnerabilities, revealing that while GPT-

4 could successfully exploit 87% of these when provided with CVE descriptions, its effectiveness drastically reduced without them. This underscores the model's current limitations in identifying vulnerabilities without explicit prior information, highlighting its ineffectiveness in more dynamic scenarios [15].

Lastly, Wu et al.'s PTGroup framework, which implements a multi-agent and cue-chain approach, showcases the adaptability and extensibility of LLMs in handling various testing environments. Despite its innovative approach, the model's reliance on complex cue chains highlights ongoing challenges with autonomy and efficient task management in real-world scenarios [53].

Collectively, these studies underscore a critical gap in the current deployment of LLMs within automated PT frameworks: while they can generate detailed and complex instructions, their practical application is severely limited by their inability to operate autonomously in dynamic environments. The dependency on human intervention to correct errors arising from limited memory and hallucinations highlights the necessity for further research and development. The goal of my research is to address these gaps by enhancing the LLM's memory capabilities and reducing hallucinations to improve the autonomy and reliability of automated PT frameworks, aiming for a solution that can adapt more fluidly to complex security environments without extensive manual oversight.

2.2 Solution

This section presents the solutions integrated into our framework to directly address the challenges of contextual memory and hallucinations in automated PT frameworks. These solutions are designed to fill the gaps identified in prior methodologies, offering enhanced autonomy and reliability for LLMs in complex security contexts.

Retrieval-Augmented Generation (RAG): To combat the issue of hallucinations and contextual memory loss, we have integrated the RAG approach, which leverages external knowledge bases to complement the LLM's responses. The LLM is connected to a vector database, where tokens are grouped by semantic proximity. This allows the model's output to be enriched with factual information retrieved from the database [26][3][14]. Unlike previous approaches that rely solely on the LLM's internal training data, our method ensures that the LLM has access to up-to-date, relevant data, reducing hallucinations and improving the accuracy of responses. In the context of security testing, where precision and context-aware answers are crucial, RAG plays a key role in maintaining the accuracy and consistency of responses over long interac-

tions. This solution directly addresses the issue of contextual forgetting [38], which is common in earlier models that struggle to maintain coherence over extended tasks.

Mixture of Experts (MoE): To improve the specialization of our LLM framework, we have incorporated the MoE architecture. This architecture divides the model into several expert sub-models, each trained to handle different facets of PT. Inputs are routed to the most relevant expert, which allows the model to provide more accurate and context-specific responses. This contrasts with the traditional single LLM approach, which often leads to inefficiencies and hallucinations, especially when dealing with complex or niche tasks. The MoE architecture offers a scalable solution to these challenges by improving task-specific performance and reducing the need for manual intervention. This feature is a significant departure from prior work, where models struggled to adapt to varying levels of complexity in PT scenarios [58].

These two enhancements—RAG and MoE—have been carefully selected to address the shortcomings in prior LLM-based PT frameworks. They directly tackle the issues of low automation and poor memory capabilities, offering a more autonomous and reliable approach to automated security testing. By incorporating these methods, our solution provides significant improvements in both the accuracy of responses and the ability of the system to maintain contextual continuity, reducing human oversight in the testing process.

2.3 Summary

This chapter has thoroughly reviewed various frameworks in automated PT, highlighting their evolution in response to complex cybersecurity challenges. We explored traditional, RL-based, and LLM-based frameworks, each offering unique benefits and facing distinct challenges. Traditional frameworks, although efficient in identifying known vulnerabilities, struggle with adaptability and dynamic threat landscapes. RL-based frameworks offer dynamic path optimization but often suffer from inadequate state-action definitions that hinder their practical application. LLM-based frameworks show promise in generating sophisticated attack strategies yet are limited by issues related to automation, memory, and hallucinations.

The contributions and limitations of the relevant work of each framework are set out in Table 2.1.

As we transition to the next chapter, we will delve deeper into the methodologies that underpin our proposed frameworks, focusing on the technical specifics of the RAG and MoE systems, which aim to address the shortcomings identified in this

chapter. By enhancing the LLM framework's memory and reducing hallucinations, these innovations seek to elevate the automation and efficacy of PT, providing a more robust defense against evolving security threats. This discussion will set the stage for a deeper understanding of how our proposed solutions operate within a comprehensive PT framework, aiming to overcome the limitations identified in previous systems.

Table 2.1: Contributions and Limitations of Related LLM Works

Work	**Contribution**	**Limitations**
PentestGPT [8]	Demonstrated LLM utility in multi-step attack generation; improved contextual script generation.	Low automation and memory constraints; hallucination issues persisted in complex scenarios.
Hape and Cito (LLM Study)[63]	Evaluated GPT-3.5 for task planning; provided insights on LLM adaptability to penetration tasks.	Lacked multi-step task effectiveness; highlighted automation limitations in GPT-3.5.
AutoAttack [54]	Optimized execution processes in GPT-4; reduced error rates in multi-step tasks.	Relied heavily on GPT-4; did not fully resolve context and memory issues.
GENAI [18]	Integrated GENAI for adaptive test generation; enhanced environment customization and reporting.	Encountered hallucination and scalability issues; limited complex scenario testing.
PTGroup [53]	Developed multi-agent systems with prompt chaining; improved task execution and adaptability.	Limited by hallucination problems; still required manual intervention in complex tasks.
Fang et al. (LLM Agents) [15]	Showcased LLM agents' abilities in real-world vulnerabilities; emphasized potential for autonomous tasks.	Dependent on CVE descriptions for success; struggled with discovering new vulnerabilities.

Chapter 3

Methodology

In this chapter, we introduce the innovative automated PT(Penetration Test) framework, AutoPenGPT, which integrates cutting-edge technologies including RAG(Retrieval-Augmented Generation), LLM(Large Language Model) Agents, and a Mixture of Experts (MoE) system. The purpose of this chapter is to delineate the architecture and operational dynamics of AutoPenGPT, highlighting its potential to revolutionize traditional PT methodologies by enhancing the accuracy, adaptability, and efficiency of security testing. This chapter is structured to first elucidate the core technologies and their synergistic integration within the AutoPenGPT framework. We begin by outlining the enhanced capabilities brought by the RAG module, followed by a detailed discussion on the role and functionalities of the LLM Agent in task decomposition and dynamic response generation. Subsequently, the implementation and advantages of the MoE system are explored to demonstrate how specialized expertise is leveraged within the framework. Finally, we examine the collaborative mechanism of these modules which underpins the framework's operational efficacy and its ability to address complex PT scenarios with reduced human oversight.

Through this structured approach, the chapter aims to provide a clear understanding of each component's contribution to the overarching functionality of the AutoPenGPT framework, setting a robust foundation for subsequent chapters that delve into specific case studies and empirical validations.

3.1 System architecture

AutoPenGPT runs on the Kali Linux environment and integrates several PT tools such as Nmap and Metasploit. The system architecture consists of the following main modules:

1. Decision Module: As a strategic decision-making center, it receives user-defined objectives (such as IP address, domain name) and test requirements (such as vulnerability scanning or vulnerability exploitation), generates dynamic task tree and manages task status. By analyzing task priorities and dependencies, the module develops optimal test strategies and assigns tasks to downstream modules.

2. Expert Module: Responsible for handling specific tasks, assigning tasks to the most appropriate experts through LLM intelligent analysis, and supporting multi-threading parallel execution of infiltration tasks. Experts in this module perform specific types of PT tasks based on their expertise, such as information gathering, vulnerability exploitation or post-analysis.

3. Analyser Module: Analyze the task results output by the expert module, use RAG technology to extract valuable information and feed it back to the Decision module to optimize the follow-up task planning. The analysis module is also used to evaluate mission performance, identify potential vulnerabilities, and generate recommendations for follow-up action.

4. Summarizer Module: Summarizer Module: Summarizes the results of a penetration test and generates a structured report (such as JSON format) to support follow-up audit and trend analysis. The summary module ensures traceability of test results and provides detailed test logs and safety assessments.

5. Util Module: Provides log recording, configuration management, and token compression functions to ensure efficient system operation and security. The tool module is also responsible for managing interfaces with external tools, such as penetration test scripts and scanning tools, to ensure seamless collaboration between components during testing.

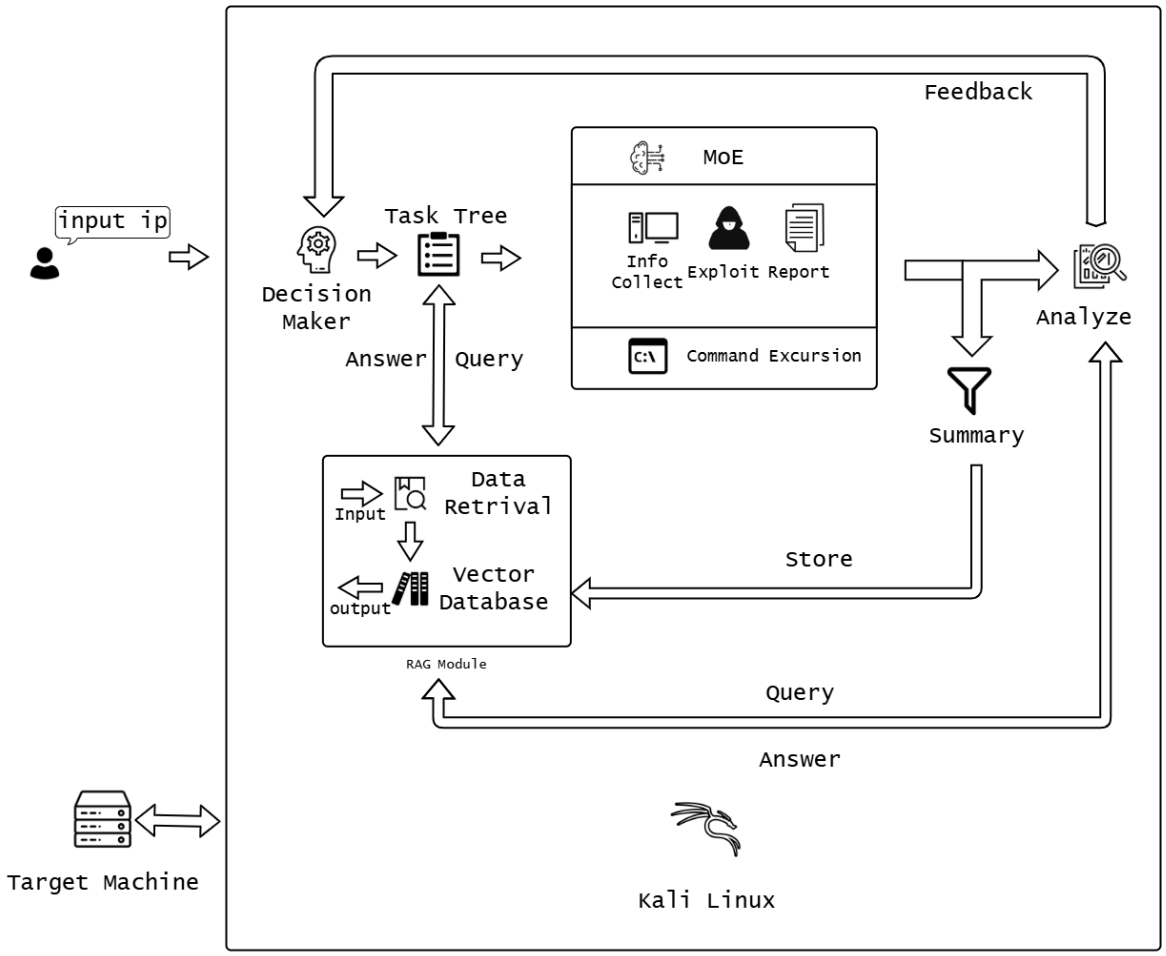

Figure 3.1: AutoPenGPT Flow

Figure 3.1 shows the architecture of AutoPenGPT. The user-defined objective is first translated into a task tree by the Decision module. As the task progresses, the Expert module executes specific PT actions, while the Analyser module analyzes the results. Feedback from the Analyser module allows the Decision module to dynamically adjust the task tree. Then, the Summarizer module organizes and stores the results in a vector database for future analysis.

Algorithm 1 Framework Execution Flow

Require: T_{user}: User-defined objective
Ensure: R: Final results

function AUTOPENGPT_FLOW(T_{user})
    $T_0 \leftarrow$ Interpret input with LLM(T_{user})
    $T_{\text{tree}} \leftarrow$ Generate task tree(T_0)
    for $T_i \in T_{\text{tree}}$ **do**
        $R_i \leftarrow$ Execute task via Expert Module(T_i)
        $A_i \leftarrow$ Analyze results(R_i)
        if A_i.status = "failed" **then**
            if Task can be retried(T_i) **then**
                $T_i \leftarrow$ Adjust for retry(T_i)
                Retry(T_i)
            else
                Skip and remove from tree(T_i)
                $T_{\text{tree}} \leftarrow$ Remove node(T_i, T_{tree})
            end if
        else if A_i.status = "completed" **then**
            Mark task as completed(T_i)
            $T_{\text{tree}} \leftarrow$ Update node as completed(T_i)
        end if
        $JSON_i \leftarrow$ Format into JSON(R_i)
        Store JSON in database($JSON_i$)
    end for
    if Task tree is incomplete(T_{tree}) **then**
        Adjust incomplete tree(T_{tree})
    end if
    return R
end function

3.2 Technical details of the RAG module

RAG[26] is a method proposed in 2020 for optimizing the output of LLMs. This technique cleverly combines the powerful reasoning power of the generated model with the flexibility of the retrieval module to significantly improve the accuracy and relevance of the model's answers through dynamic access to external knowledge sources[55]. The

process is similar to an open-book test in humans, which focuses on reasoning skills rather than relying solely on memorizing information. The core advantage of this approach is to separate factual knowledge from the reasoning power of LLM, storing knowledge in an external knowledge source for easy access and updating. Through this mechanism, RAG not only significantly improves the adaptability of the model, but also expands its application potential in rapidly changing fields such as cybersecurity.

In order to better understand the implementation of RAG, we must distinguish between parametric knowledge and non-parametric knowledge[22]. Parameter knowledge is the tacit knowledge that the model learns from massive data in the training process, and it is stored in the weight of the neural network. This knowledge forms the basis of model generation capabilities, but updating it requires retraining the model, which is expensive and inflexible. Non-parametric knowledge is stored in external knowledge sources, such as vector databases. This knowledge can be dynamically retrieved and directly applied without changing the parameters of the model, thus providing the model with up-to-date, domain-specific information support.

In the AutoPenGPT framework, the RAG module contains two knowledge bases to meet the requirements of different scenarios. The first is a temporary knowledge base to record information generated during PT, such as the network environment and test configuration. This knowledge base is emptied after each penetration test is completed to prevent information contamination between different tests, thus ensuring data independence and accuracy. The second is a persistent knowledge base based on the MITRE ATT&CK framework, which details the behavior patterns, commonly used techniques and attack tools of cyber attackers, providing an authoritative reference for PT tasks. The combination of these two knowledge bases enables AutoPenGPT to efficiently perform PT tasks while maintaining the reliability and usefulness of the results.

RAG supports a variety of work types to meet the task needs in different scenarios[60]:

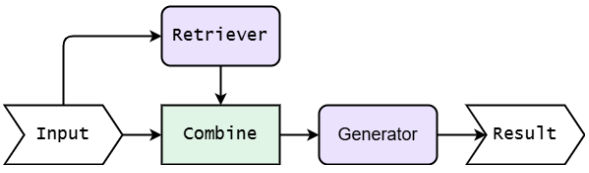

Figure 3.2: Query-based RAG Flow

Query-based RAG: Query-based RAG[45] is also known as prompt enhancement. It integrates user queries and information retrieved from files directly into the initial stage of language model input. This pattern is widely adopted in RAG

applications[45]. Once the documents are retrieved, their contents are merged with the user's original query to create a combined input sequence. This sequence of enhancements is then fed into a pre-trained language model to generate responses.

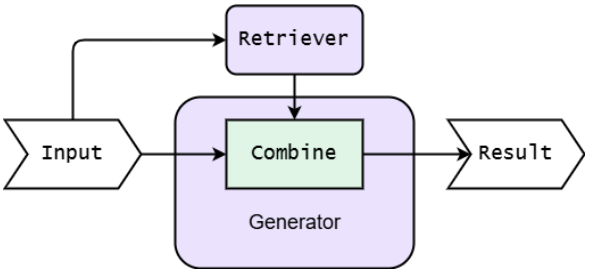

Figure 3.3: Latent Representation-based RAG Flow

Latent Representation-based RAG: In a RAG framework based on implicit Representation, the retrieved objects are integrated into the generation model as implicit representations, thereby improving the model's understanding and the quality of the generated content. In this framework, the generative model interacts with the underlying representation of the retrieved object, improving the accuracy of the generated content. This approach shows great potential and adaptability in dealing with code, structured knowledge, and multimodal data.

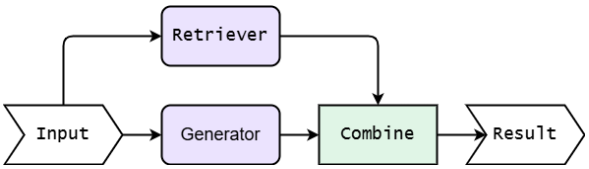

Figure 3.4: Logit-based RAG Flow

Logit-based RAG: In log-likelihood based RAG[1], the generated model retrieves information through logarithmic fusion during decoding. Typically, logarithms are summed or combined by models to produce probabilities that are generated step by step. In a code-to-text conversion task, Rencos generates multiple digest candidates for the retrieved code in parallel, then normalizes them using the edit distance, calculating the final probability to select the digest output that best matches the original code. In the code summary task, EDITSUM improves the quality of summary generation by integrating prototype summaries at a probabilistic level. For text-to-code tasks, the kNN-TRANX model combines confidence networks and meta-knowledge to merge the retrieved code fragments. It uses the seq2tree structure to generate object code that closely matches the input query, improving the accuracy and relevance of code

generation. This approach is particularly suitable for sequence generation tasks. It focuses on the training of generators and can devise novel ways to efficiently utilize the acquired probability distribution to suit subsequent tasks.

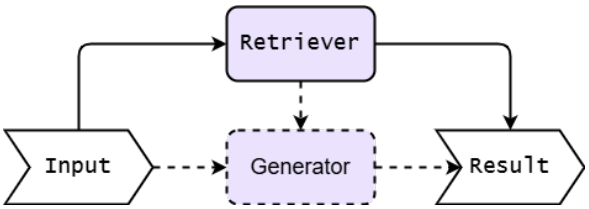

Figure 3.5: Speculative RAG Flow

Speculative RAG: Speculative RAG[13] is designed to save resources and speed up responses by utilizing retrieval rather than pure generation. REST technology enables draft generation by substituting retrieval for small models in conjectural decoding. GPTCache solves the high latency problem when using the LLM API by building a semantic cache to store responses from LLMs. At present, speculative RAG is mainly applied to serial data. It decouples the generator and the searcher so that the pre-trained model can be used directly as a component. Within this paradigm, a wider range of strategies can be explored to make effective use of the retrieved content.

Query-based RAG is the most widely used type of work in the AutoPenGPT framework. Its efficient retrieval and enhancement capabilities enable the system to quickly integrate task-related knowledge and provide targeted operational recommendations.

The function of RAG module is mainly divided into three key stages: retrieval, enhancement and generation. The process flow of these stages can be represented by the following pseudocode:

Algorithm 2 RAG Module Execution Flow

User query Q, External Knowledge Base K, Pre-trained LLM Model M Generated Response R

Step 1: Retrieval Phase

Query vector: $\mathbf{v_Q} = \text{Embed}(Q)$

Retrieve top-K relevant knowledge items: $\{K_1, K_2, \ldots, K_K\} = \text{Retrieve}(\mathbf{v_Q}, K)$

Step 2: Enhancement Phase

Filter and sort knowledge items based on relevance: $\{K_1, K_2, \ldots, K_K\} = \text{FilterSort}(\{K_1, K_2, \ldots, K_K\})$

Construct enhanced input sequence: $I_{enhanced} = \text{Merge}(Q, \{K_1, K_2, \ldots, K_K\})$

Step 3: Generation Phase

Generate response: $R = M(I_{enhanced})$

In the retrieval phase, the RAG module obtains information highly relevant to the current task from the external knowledge base by converting the user's input into a query vector. Specifically, a user's query first generates a high-dimensional vector representation through an embedded model (such as Sentence-BERT[31]) that is used to match a pre-built vector index in the knowledge base. In order to achieve efficient semantic similarity retrieval, the system adopts FAISS vector database[37] technology, whose efficient indexing and retrieval algorithms enable the model to quickly find the most relevant knowledge items. The results of the retrieval are usually the first K pieces of information, sorted by similarity, which are considered to be the most valuable supporting data for the task at hand.

In the enhancement phase, the retrieved information is integrated into the context of user input to build task-relevant prompt templates. The RAG module will further filter and sort the search results in this process to ensure that the final selected content has high relevance and high confidence. This information is organized into a structured format that works with the user's original input to form an enhanced input sequence. This sequence not only contains the initial requirements of the user, but also combines the background knowledge related to the task, providing powerful context support for the subsequent generation of LLMs.

In the generation phase, the enhanced input is passed to a pre-trained large language model (such as GPT-4) that generates the specific responses required for PT. These responses may include PT instructions, tool call recommendations, or an ex-

ploitation plan for the vulnerability. By combining the retrieved authoritative knowledge, the RAG module is able to significantly reduce the possibility of model generation hallucinations[21], ensuring the reliability and accuracy of the generated content within a specific domain.

RAG modules are designed to address several known challenges in the application of LLMs, including the generation of false information, lack of domain-specific knowledge, and the generation of outdated or generic content. By accessing an external knowledge base, RAG modules excel at improving the domain relevance of generated content, especially when dealing with complex queries or tasks that require support from the latest knowledge. It not only retrieves key information from authoritative sources, but also extends the breadth of generated content through long-tail semantic knowledge, further improving the depth and quality of responses. In addition, the RAG module is designed to reduce over-reliance on training data, so that the system does not need to undergo time-consuming retraining when knowledge needs to be updated.

In the AutoPenGPT framework, the RAG module has a wide range of application scenarios. First of all, RAG provides support for the retrieval and utilization of vulnerability information, and the vulnerability description and attack method retrieved by RAG can provide a solid basis for subsequent task planning. Second, RAG provides real-time advice for the operation of the PT tool, by combining the tool usage documentation and real-world scenarios, making the test execution more efficient and accurate. Finally, in complex attack tasks, RAG module provides comprehensive knowledge support for task execution by combining dynamic knowledge retrieval and context enhancement, ensuring the flexibility of the framework in dealing with the changing network environment.

Through the combination of temporary knowledge base and persistent knowledge base, RAG module not only improves the security and accuracy of data input, but also provides a reliable knowledge verification and support mechanism for the entire process of automated PT. Throughout the task execution, the RAG module's efficient retrieval and intelligent enhancement capabilities are the core components of the AutoPenGPT framework to successfully achieve accurate PT.

3.3 Technical details of the LLM Agent module

This section presents the technical details of the LLM Agent module, which serves as the intelligent core of AutoPenGPT. The LLM Agent integrates advanced LLM technology to facilitate automated PT through task planning, memory management,

tool invocation, and dynamic action execution. Key components, including the Planning, Memory, Tools, and Action modules, are described in detail, highlighting their roles in supporting intelligent task execution. Furthermore, the ReAcT (Reasoning, Acting, and Observing) framework is introduced, which governs task execution cycles and ensures dynamic adaptation based on real-time feedback. The section also covers task execution strategies, multi-agent collaboration, and error handling mechanisms that contribute to the flexibility, efficiency, and reliability of the system in complex testing scenarios.

3.3.1 Core framework

LLM Agent[17] is the intelligent core of AutoPenGPT. It integrates advanced LLM technology to realize task planning, tool invocation and dynamic feedback processing in automated PT. The design of LLM Agent consists of four core parts, namely Planning, Memory, Tools and Action, which support the intelligent execution of Agent in complex tasks.

Planning is the core of the Agent's thinking, responsible for breaking down complex tasks into executable sub-tasks and evaluating strategies at each step. In terms of technical implementation, through the prompt engineering of LLMs (such as REACT framework and chain thinking reasoning mode), agents can generate clear task dismantling paths and optimize execution strategies according to priorities. For example, in PT, the planning module generates a multi-phase test plan from asset scanning to vulnerability exploitation based on the target environment, ensuring that the task is systematic and efficient.

The Memory module supports the short-term and long-term information storage and invocation of agents. Short-term memory is used to store the current task context and support complex multi-round task interaction. Long-term memory stores important domain knowledge, user characteristics, and historical task data. Long-term memory is realized through vector database technology to ensure efficient access and correlation of data. For example, when an Agent processes multiple rounds of interactive tasks, it can invoke historical data in long-term memory as context support to optimize the continuity and accuracy of task execution.

Tools module is the key ability of Agent to perceive and interact with the external environment. By dynamically invoking external tools, such as APIs or client interfaces, agents are able to extend their capabilities to suit various task requirements. For example, an Agent can call Nmap for port scanning, or Metasploit for vulnerability exploitation. In addition, the tool module also supports plug-in extensions, giving

agents the flexibility to handle new tools or scenarios, such as parsing documents or generating reports.

The Action module converts the planning and memory of the Agent into a specific execution process, including the interaction with external tools and real-time monitoring of operations. In PT, the action module generates specific commands and executes tasks according to the plan, while adjusting subsequent plans through feedback mechanisms. For example, when a vulnerability fails to be exploited, the action module triggers a rollback mechanism to re-generate commands or adjust parameters to ensure that the task is completed.

3.3.2 ReAcT Workflow

The task execution within the AutoPenGPT framework is governed by the ReAcT (Reasoning, Acting, and Observing) [56]framework, which facilitates intelligent task cycles to enhance the automated penetration testing process. This methodical approach is integral to the system's ability to perform complex security assessments with high efficiency and adaptability.

Within the ReAcT workflow, tasks are initiated by the Decision module, which assesses the scope and requirements of the task. At the onset of a new task, such as a network vulnerability assessment, the LLM first analyzes the task characteristics, extracts relevant features, and determines the most appropriate Expert Module for execution. For ongoing tasks, the LLM evaluates the results to determine if the objectives have been met and whether to proceed with further actions or conclude the cycle.

For instance, consider a typical penetration testing process involving a port scan followed by vulnerability exploitation. In the "Thought" phase, the LLM Agent extracts task features based on the input information of the Decision module. It might classify tasks as information-gathering tasks. It then assigns the task to the corresponding expert module for further solving.

Following the planning, the "Act" phase commences where the designated Expert Module formulates a strategic plan and executes the task. It might decide to initiate a port scan to identify open ports and potential vulnerabilities. The plan includes specifying the scope of the scan and setting the parameters for the tools to be used, such as Nmap with commands like *nmap -sV -p 80,443 <target IP>*. Upon completion, the results are fed back to the LLM Agent.

In the "Obs" phase, the LLM Agent reviews the feedback from the action phase. If the scan indicates open ports, the Agent analyzes the service details to identify

vulnerabilities and plans subsequent exploitation tasks. This might involve configuring and launching an attack using tools like Metasploit based on the vulnerabilities identified during the scan.

The ReAcT cycle ensures that each task, from the initial input to the final execution, is dynamically adjusted based on real-time data and feedback. This continuous interaction cycle enhances the framework's ability to adapt to changing network environments and ensures tasks are performed accurately and efficiently.

With the introduction of the LLM Agent and the ReAcT framework, AutoPenGPT has significantly improved its capabilities in planning and executing complex penetration tests. The framework allows for the flexible disassembly of tasks into manageable sub-tasks, aligning each with the appropriate expert modules for execution, and refining strategies based on ongoing feedback. This leads to an efficient and reliable solution for network security testing that dynamically adapts to task requirements and optimizes the penetration testing process.

3.3.3 Task Execution

LLM Agent ensures efficient completion of complex tests through highly flexible process management, multi-round task iteration and intelligent error handling during task execution. First, each step of the task is based on real-time analysis and inference, combining user input, contextual knowledge, and intermediate results to generate a dynamic action path. For example, in a multi-phase penetration test, the Agent first invokes the RAG module to retrieve target environment information, and then plans and performs specific tasks such as port scanning, service identification, and vulnerability exploitation.

Multi-round task iteration is a key feature in Agent execution. For complex scenarios, the Agent optimizes the test results in a step-by-step manner. For example, when an open high-risk port is found, the Agent will further analyze the service information and try to obtain the exploitation code of the relevant vulnerability. If the initial vulnerability fails to be exploited, the Agent generates an alternate operation path and finally achieves the task goal by adjusting tool parameters or switching attack policies.

Error handling is another important function of the LLM Agent. With the built-in rollback mechanism, agents can quickly adjust their policies if a task fails. For example, if a tool call fails, the system will regenerate the command or replace the standby tool. If the external tool returns abnormal data, the Agent attempts to re-parse the result

Algorithm 3 LLM Agent Task Execution with Mathematical Expressions

`Initialize` Task with goal G and context data C
`Planning:`
Generate task decomposition path π based on reasoning
Evaluate strategies at each step: $S = \{s_1, s_2, \ldots, s_n\}$, where s_i represents a subtask
Define the best execution plan: $\arg\max S$
`Memory:`
Store short-term task context C_t
Retrieve long-term memory L (historical data, domain knowledge)
Correlate context information: $C_{total} = C_t \cup L$
`Tools:`
Select appropriate tools T based on task requirements Q: $T = \text{Select}(Q)$
Dynamically invoke external tools $t_1, t_2, \ldots, t_m$ (e.g., Nmap, Metasploit)
Extend tool capabilities via plug-in support
`Action:`
Convert planning and memory into executable actions
Execute tools and generate commands: $A = \{a_1, a_2, \ldots, a_k\}$ where a_i represents an action
Monitor task execution in real-time
Adjust subsequent task plans based on feedback: $F_{feedback} = \text{Update}(F)$
If task fails, trigger rollback mechanism, regenerate commands or adjust parameters:
    If $P_{fail} \in \{P_{task}, P_{tool}\}$, re-generate commands A', adjust parameters θ
`ReAct Cycle:`
while task not completed **do**
    `Thought:`
    Generate reasoning path: $\mathcal{R} = \text{Reason}(G, C)$
    Define task objectives and execution steps: $O = \text{Define}(G)$
    Generate task execution plan based on user input and context data: $P = \text{Generate}(O, C)$
    `Act:`
    Select specific actions based on reasoning results: $A_i = \text{Select}(P)$
    Execute tool invocation and configure parameters: $T_i = \text{Invoke}(A_i)$
    `Obs:`
    Update task environment based on feedback: $F = \text{Feedback}(T_i)$
    Adjust subsequent task plans based on feedback information: $P' = \text{Update}(P, F)$
    If task has new progress, continue task execution
end while
`End Task` =0

or request additional information from the user to avoid task interruption. In addition, the system records all errors for subsequent audit and analysis.

3.3.4 Multi-agent Collaboration

In order to support more complex task scenarios, AutoPenGPT adopts the multi-agent collaboration mode[41], which enables different agent modules to process their tasks in parallel and maintain collaboration consistency through shared context. These agents include information collection agents, vulnerability exploitation agents, and summary agents, each responsible for a specific task scenario. For example, the information collection agent focuses on asset scanning, subdomain enumeration, and service identification, the exploit agent executes specific attack operations, and the summary agent integrates all results to generate reports.

At the heart of the multi-agent architecture is a continuous interaction and feedback mechanism based on the REACT loop. During testing, agents maintain data consistency through a shared context. For example, when an information collection agent completes a port scan, the results are immediately passed on to an exploit agent to generate an attack path against the target. In addition, resource allocation can be dynamically adjusted between agents. For example, when a task at a certain stage encounters a bottleneck, the system will prioritize the allocation of computing resources to solve the critical problem through a decision agent.

This modular design not only improves the efficiency of task execution, but also enhances the adaptability and expansibility of the system. Through multi-agent collaboration, AutoPenGPT enables comprehensive PT capabilities for complex network environments to provide users with more efficient and reliable cybersecurity solutions.

3.4 Technical Details of the MoE System

This section outlines the technical details of the MoE System, a key component of the AutoPenGPT framework that enhances task allocation and optimization in automated PT. The MoE module leverages a dynamic system where expert models, each specialized in handling specific types of tasks, collaborate under the control of a LLM. The design of the MoE system enables efficient resource utilization and task execution through expert selection, parallel processing, and dynamic feedback integration. The section also describes the workflow of the MoE system, including task reception, expert selection, parallel task processing, and result integration, as well as the

technological advantages of the MoE module, such as dynamic adaptability, efficient resource usage, and task hierarchical processing, which collectively contribute to the overall performance improvement of AutoPenGPT.

3.4.1 Core Design

The MoE[61] module is a key component of task allocation and optimization in AutoPenGPT, and the design concept stems from the replacement of the FFN in the traditional Transformer model with a gated network and a MoE layer composed of several experts. The module significantly improves the efficiency and accuracy of task execution through dynamic task allocation and resource optimization in PT.

The core of the MoE module is the collaboration of experts and a gated network. Expert models are specifically designed to handle specific types of tasks (such as vulnerability scanning, asset analysis, and attack simulation) through efficient neural networks or other algorithms that optimize a specific subset of data. As the decision-making core, the gated network dynamically selects the most suitable experts to participate in task processing according to the characteristics of the input task, and adjusts the assigned weights to ensure that each task is executed by the most professional expert model.

In AutoPenGPT, the traditional gate control mechanism is transformed into a dynamic task management system based on a LLM, which further enhances the module's adaptability and resource utilization efficiency. The LLM is directly responsible for task assignment and expert model scheduling, eliminating the need for additional fixed gated networks to more efficiently adapt to diverse and dynamic PT scenarios.

3.4.2 Work Flow

In the MoE module, the processing of tasks, from the reception of tasks to the integration of results, is strictly designed to ensure efficient execution. The process flow is represented by the following pseudocode:

Algorithm 4 MoE System Execution Flow

Task T, LLM Agent LLM, Expert Models $\{E_1, E_2, \ldots, E_N\}$ Final Task Report R

Step 1: Task Reception and Analysis

Parse task T to extract key features: $\{target, priority, resources\} = \text{Parse}(T)$

Step 2: Expert Selection and Task Distribution

Analyze task features: $features = \text{Analyze}(T)$

Select suitable expert E_i based on task features: $E_i = \text{SelectExpert}(features)$

Assign task to expert E_i: $E_i(T) \rightarrow \text{Execute}(T)$

Step 3: Parallel Task Processing

Execute tasks in parallel for multiple experts: $E_1(T), E_2(T), \ldots, E_N(T)$

Step 4: Feedback and Result Integration

Collect outputs from experts: $\{O_1, O_2, \ldots, O_N\} = \text{CollectResults}(\{E_1, E_2, \ldots, E_N\})$

Pass outputs to Analyzer Module for further processing: $O_i \rightarrow \text{Analyze}(O_i)$

Summarize results: $R = \text{Summarize}(\{O_1, O_2, \ldots, O_N\})$

- Task reception and resolution: After a task is distributed from the LLM Agent or Decision module to the MoE module, the LLM parses the task and extracts the key features of the task, such as the target type, priority, and required resources.
- Expert selection and task distribution: Based on task analysis results, the LLM module plays the role of a gated network and dynamically selects the most suitable experts to participate in task execution based on context characteristics. After an expert is selected, the system assigns the task to the expert according to the nature of the task.
- Feedback and integration: When the expert model completes its task, the output is passed to the Analyzer Module for further analysis and optimization, and summarized by the Summarizer Module into a structured report to provide users with the final test results.

3.4.3 Technological Advantage

Dynamic adaptability is a core advantage of MoE modules. Through the dynamic allocation mechanism of LLM, the system can adjust the resource allocation according

to the real-time characteristics of the task, ensuring the accuracy and flexibility of the task execution.

Efficient resource utilization is another notable feature of MoE modules. Because each expert model focuses on a specific task, it is much more optimized than the generic model, which can significantly reduce ineffective calculations and repetitive operations. In addition, the parallel execution strategy enables the module to make full use of the computing power of the system to handle complex and diverse tasks.

Task hierarchical processing is also an important feature of MoE module. By breaking down complex tasks into multiple independently executable subtasks and assigning them to different experts, the system can gradually achieve efficient task execution and goal achievement.

In the AutoPenGPT framework, the MoE module provides an efficient solution for complex PT tasks through its flexible expert system and dynamic allocation capabilities, while achieving an optimal balance in terms of resource utilization and task execution efficiency. Compared with the traditional fixed gate control mechanism, the dynamic task assignment based on LLM is more adaptable and can meet diverse network security test requirements, providing strong support for the overall performance improvement of AutoPenGPT.

3.5 System Module Function Details

This section provides a detailed overview of the various system modules that collaborate to enable the efficient execution of automated penetration testing (PT) within the AutoPenGPT framework. The modules described in this section include the Decision, Expert, Analyser, Summarizer, and Util modules, each with distinct responsibilities and functions that contribute to the smooth operation of the system. The Decision module handles strategic task management and ensures efficient task flow, while the Expert module is responsible for assigning and executing tasks through specialized experts. The Analyser module provides data analysis and feedback for continuous optimization, and the Summarizer module consolidates and stores execution results for further analysis. Finally, the Util module serves as the infrastructure backbone, supporting the system with functions such as logging, configuration management, and token compression. Together, these modules form the core of the AutoPenGPT framework, enabling automated and efficient penetration testing with enhanced security and resource management.

3.5.1 Decision Module

As the strategic decision-making center of the system, the Decision module plays the role of commander of the whole PT process. The main responsibility of this module is to receive user-defined objectives, generate and maintain a dynamic task tree based on these objectives, and each task node contains detailed test strategies and expected actions.

The task tree is the heart of PT, providing structure and guidance for the entire testing process. The Decision module develops specific test strategies for each node in the task tree, which are customized based on the nature of the task and the priority of the target. These policies are then passed to the Expert module to perform specific PT actions.

Task state management is another important responsibility of the Decision module. Each task is initially marked as "to-do", which means that the task has been defined but not yet executed. Once the task is started and completed, its status is updated to Completed. If the task fails due to a fault or other reason, it is marked as failed. Based on feedback from the Analyser module, the Decision module makes the necessary adjustments to the failed task, which may include retries, redefining task parameters or policies, or canceling the task entirely.

In order to prevent the Decision module from falling into a loop when dealing with some complicated or difficult problems, the Decision module adopts an anti-reality strategy. When it detects that a task has cycled through the automated PT framework several times without being solved, the module uses counterfactual strategies to determine that the task is unsolvable, and then logs and skips the task. This strategy effectively prevents the waste of resources and improves the overall efficiency of PT. Through many practical tests, we determined that the maximum number of cycles is five; Beyond this threshold, the automated PT framework is considered unable to complete the task.

Through these efficient strategies and mechanisms, the Decision module ensures the smooth flow of PT and provides reliable decision support for the operation of the entire system. This not only speeds up task execution, but also improves the system's ability to cope with complex situations.

3.5.2 Expert Module

The Expert module plays a key gating model role in the MoE system and is responsible for handling the task assignment issued by the Decision module. This module utilizes

advanced LLM agent technology to analyze incoming tasks and then delegate them to the most suitable experts. In this process, experts are selected based on their specific skills and the correspondence of task needs, ensuring that each task can be handled with maximum efficiency by the most suitable experts.

In all stages of automated PT, such as information gathering, vulnerability exploitation, report generation, etc., the Expert module is equipped with specialized experts to perform the corresponding tasks. Each expert is designed to meet the needs of a specific test process, and these experts are not only proficient in their field, but also able to execute complex sequences of commands based on test requirements.

A big advantage of the Expert module is its multi-threaded execution capability, which allows it to handle multiple tasks simultaneously. This means that when an expert needs to execute multiple commands, these commands can be executed in parallel without interfering with each other, greatly improving the speed and efficiency of task processing. For example, when performing port scanning, subdomain blasting and website information query can be performed simultaneously, greatly reducing the overall test time.

After completing a task, an expert records the command execution result and outputs it to the Analyser module and Summarizer module. These results include all data output during execution, security vulnerabilities that may have been discovered, and reports of any anomalies. The Analyser module is responsible for further analyzing the results and extracting valuable information, while the Summarizer module organizes the results into reports for subsequent review and reference.

Through this efficient task processing and result feedback mechanism, the Expert module ensures a smooth and efficient PT process, while ensuring the accuracy and practicality of the results. This not only speeds up the process of PT, but also improves the quality and reliability of testing.

3.5.3 Analyser Module

The Analyser module plays the role of data analysis and feedback in automated PT framework. The core task of the module is to deeply analyze the data output after execution by the Expert module, and utilize advanced RAG technology to optimize this analysis process.

RAG technology combines the advantages of information retrieval and generation models, allowing the Analyser module to invoke the external knowledge base during analysis, enhancing the depth and breadth of analysis. This makes the data derived from executed commands not limited to direct output, but also includes extended

insights gained by associating external information. This approach increases the accuracy and valuable information of the analysis, especially when dealing with complex cybersecurity issues, and is able to provide a more comprehensive perspective and in-depth understanding.

The analysis results of the Analyser module are critical to the overall PT framework. Through detailed data analysis, this module not only validates the effect of executing commands, but also identifies possible vulnerabilities, misconfigurations, or other critical security risks. These analysis results are fed back to the Decision module to help it optimize the existing task tree and adjust future test strategies based on actual conditions. This feedback mechanism ensures continuity and adaptability of PT, allowing the framework to dynamically adapt to changing network environments and security threats.

The analysis results also provide data support and decision-making basis for the execution of the next round of tasks. By mining and interpreting data, the Analyser module reveals patterns and trends in PT, information that is critical to developing an effective PT strategy. For example, if a particular type of vulnerability is found to recur over a series of tests, the Analyser module recommends tweaking the strategy to specifically test for that type of vulnerability in more depth.

3.5.4 Summarizer Module

The Summarizer module is responsible for data consolidation and recording in an automated PT framework. This module is responsible for summarizing and formatting the execution output from the Expert module to ensure that the data is consistent and structured for subsequent analysis and long-term archiving.

The Summarizer module converts raw command execution results from the Expert module to a structured JSON format. This formatting process not only makes the data more regular and easy to parse, but also facilitates the subsequent module to access and process the data quickly. The JSON format was chosen because of its wide compatibility and ease of human-machine reading, so that both internal and external auditors and analysts can quickly understand the data content.

The data converted to JSON format is then stored in a dedicated vector database. The vector database can maintain the vector characteristics of data during storage, that is, maintain the relationships and dimensions between the data, which is especially critical for performing complex queries and data analysis.

The data stored in the vector database not only supports the immediate analysis needs, but also facilitates the comparison and trend analysis of historical data. The

Summarizer module ensures that all collected data is properly formatted and stored categorically, making it easily accessible in the future to detect long-term trends in security breaches or to assess the effectiveness of specific policies. In addition, structured storage greatly improves the accessibility and availability of data, enabling PT teams to make decisions based on reliable data.

With these features, the Summarizer module greatly enhances the data management capabilities of the PT framework, providing a solid data foundation to support efficient data analysis and accurate security decisions. The work of this module not only guarantees the integrity and consistency of the data, but also improves the transparency and traceability of the entire PT process, providing strong support for continuous monitoring and response in a cybersecurity environment.

3.5.5 Util Module

As an auxiliary module to support the operation of the entire framework, Util module provides logging, configuration file management, command execution, token compression and other functions. These tools are essential for maintaining system stability, tracking operation history, and optimizing resource usage, ensuring that the framework is efficient and secure in complex operating environments.

Through the close collaboration of these modules, AutoPenGPT is able to effectively perform automated PT in the Kali Linux environment, not only increasing the level of test automation, but also enhancing the safety and accuracy of the test process.

The Util module acts as an infrastructure support in the automated PT framework, providing critical back-office service functions for the entire system. This module ensures the stable operation and efficiency of the framework by providing functions such as logging, configuration file management, command execution and token compression.

Logging is one of the core functions of the Util module, which is responsible for recording detailed data about each operation, including time stamps, operation details, execution results, and so on. These logs are not only critical for problem diagnosis and system monitoring, but also an essential part of complying with audit and compliance requirements. With detailed logging, system administrators can track any exceptions or security events and take swift action.

The Util module also manages all the configuration files required for the system to run, ensuring that each test takes place under controlled and expected environment Settings. Configuration management includes but is not limited to API management, model management, anti-reality policy management, etc. The centralized management

and regular review of these configuration files help to maintain the consistency and security of the system configuration.

This module is also responsible for executing system-level commands, which are important for automating tasks and maintaining system state.

In addition to that, when a penetration test is conducted, a large amount of temporary data and tokens may be generated. The token compression function of the Util module can effectively manage these data, reduce storage space usage, improve data processing speed, and reduce overhead. This not only optimizes resource usage, but also enhances security during data transmission.

Through the integration of these functions, the Util module provides the necessary support for AutoPenGPT, ensuring the stability and efficiency of the system when performing complex automated PTs. In addition, the module's operational support enables AutoPenGPT to efficiently perform PTs in Kali Linux environments, increasing the level of test automation.

3.6 Summary

In this chapter, we have laid the groundwork for the innovative AutoPenGPT framework, which is designed to overcome traditional limitations in automated penetration testing by integrating cutting-edge technologies such as RAG, LLM Agents, and the MoE system. These components work synergistically to enhance the framework's adaptability, precision, and efficiency, making it a robust solution for dynamic and complex security environments. By detailing the functionalities of each module within AutoPenGPT, we have demonstrated how they collectively contribute to the system's capability to perform comprehensive and accurate penetration tests with minimal human intervention.

Moving forward, the subsequent chapters will delve deeper into the empirical evaluation of AutoPenGPT, exploring its performance through a series of experiments and case studies. We will assess the framework's automation degree, contextual memory capabilities, and hallucination frequency under various conditions and compare these with traditional penetration testing tools. The detailed evaluation and case studies outlined will not only validate the framework's practical benefits but also highlight areas for future enhancements.

Chapter 4

Evaluation

This chapter evaluates the AutoPenGPT framework for automated penetration testing. We first detail the experimental setup, including hardware and software configurations. We then assess AutoPenGPT using key metrics: degree of automation, contextual memory ability, and hallucination frequency. Finally, we compare its performance against existing tools like Nessus, Deep Exploit, and PentestGPT.

4.1 Experiment Settings

This section outlines the experimental setup and evaluation methodology used to assess the performance and capabilities of the proposed automated PT(Penetration Test) framework. It provides a detailed description of the experimental environment, including the hardware, software, and target environment configurations. Additionally, this section introduces the comparison tools—NESSUS, Deep Exploit, and PentestGPT—and discusses their roles as benchmarks in the comparative analysis. Lastly, the test methods and evaluation criteria are presented, ensuring a rigorous and standardized approach to measuring the effectiveness of the PT framework against existing solutions.

4.1.1 Experimental Environment

The experimental environment of this study is designed to ensure uniformity, repeatability and reliability of experimental conditions, and adopts comprehensive hardware and software configuration and carefully selected test target environment.

In terms of operating systems for attack machines, this study selected Kali Linux 2024.2, a Debian-based Linux[1] distribution designed for PT and cybersecurity research. Kali Linux has a wide range of built-in security testing and digital forensics tools, such as Nmap, Metasploit Framework and Burp Suite, which can meet the needs of complex security testing and attack simulation, providing a solid foundation environment for the test task in this study.

The experiment running platform is based on VirtualBox 7.0.20 virtual machine manager, which has efficient virtualization performance and flexible network configuration functions, and can be used to simulate a variety of network topologies and experimental environments to ensure the controllability of the experiment process and the consistency of test conditions. VirtualBox provides network modes (such as NAT, bridging, and host-only networking) that enable the communication environment between attack and targets to accurately simulate the conditions in the real world network, providing a real network interaction experience for PT.

The hardware in the experiment was configured with an Intel Core i5-10400 processor, clocked at 2.90GHz, and equipped with 32GB of RAM, of which 31.8GB of available memory was enough to support multiple virtual machines running simultaneously without performance bottlenecks. The graphics card uses the NVIDIA GeForce RTX 3060 Laptop GPU to support possible GPU acceleration and improve the efficiency of running resource-intensive tools such as deep learning inference tasks or graphical interface analysis tools during experiments.

[1]https://www.kali.org/

Table 4.1: Experimental Environment Specifications

Component	**Specification**
Operating System	Kali Linux 2024.2, Debian-based Linux distribution
Virtual Machine Manager	VirtualBox 7.0.20
Processor	Intel Core i5-10400, 2.90 GHz
RAM	32GB, 31.8GB available
Graphics Card	NVIDIA GeForce RTX 3060 Laptop GPU
Network Modes	NAT, Bridging, Host-only Networking

In the choice of target environment, the experimental targets in this study comes from VulnHub[2] platform. VulnHub is a shared resource platform focused on PT targets designed to simulate real-world cybersecurity challenges, covering multiple vulnerability and attack scenarios. In this study, 15 targets from VulnHub platform were selected, which were divided into three categories according to difficulty: simple, medium and difficult, with 5 targets in each category. These targets cover a variety of vulnerabilities, including, but not limited to, Local File Inclusion (LFI), Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), SQL Injection, brute-force attacks, file upload vulnerabilities, security configuration errors, permission and access control issues, insecure design, the use of vulnerable and outdated components, and authentication and session management failures. The specific vulnerabilities are described in Table 4.1. The comprehensive coverage of these vulnerabilities[2] [20] ensures a multifaceted assessment of the capabilities of the PT framework and provides a wealth of experimental data for research.

[2]https://www.vulnhub.com/

Table 4.2: Common Web Vulnerabilities and Their Descriptions

Vulnerability	**Description**
LFI	Local File Inclusion, a type of attack where attackers manipulate web applications to include files on a server.
CSRF	Cross-Site Request Forgery, an attack that tricks the victim into submitting a malicious request.
XSS	Cross-Site Scripting, allows attackers to inject client-side scripts into web pages viewed by other users.
SQL Injection	Malicious SQL statements are inserted into an entry field for execution, to manipulate database.
Brute Force	Attempting many passwords or passphrases with the hope of eventually guessing correctly.
File upload	Malicious files are uploaded to exploit the server or other users who download or interact with the file.
Security Misconfiguration	Improper system configurations that could leave the system vulnerable to attacks.
Broken Access Control	Restrictions on what authenticated users are allowed to do are not properly enforced.
Insecure Design	Flaws in software design that lead to vulnerable applications.
Vulnerable and Outdated Components	Using components with known vulnerabilities or failing to update components.
Identification and Authentication Failures	Flaws in the mechanisms that verify user identity and manage session control.

The difficulty of the target machine in the experiment is distinguished by the minimum number of tasks it needs to complete, where a task is defined as a single operation to detect, analyze, or exploit a specific vulnerability. For example, an easy target might only need to identify an open port and verify its service vulnerabilities, while a difficult target might require a complex multi-step task to gain top access to the system or access sensitive data. For details, see Table 4.3, which lists the classification basis of task complexity.

Table 4.3: Classification of Task Complexity

Difficulty	**Number of Task**
Easy	6 ~ 8
Medium	9 ~ 11
Hard	12+

During the experiment, the success conditions for the targets to be overcome are flexibly defined according to the actual Settings, such as obtaining the highest permission of the system (root permission), successfully accessing the sensitive data of the target, or completing the specified penetration test challenge task. At the same time, the evaluation criteria of different tools vary according to their functional characteristics. For example, for the Nessus evaluation, a successful scan of a target's primary mission vulnerability is considered a breach.

Through the configuration of the above experimental environment, the performance and capability of the proposed automated PT framework can be evaluated on an efficient software and hardware platform for a variety of test target environments.

4.1.2 Overview of Comparison Tools

In order to comprehensively analyze the advantages and potential for improvement of the proposed automated PT framework, three representative tools - NESSUS, Deep Exploit, and PentestGPT - were selected as the baseline for comparative analysis. They have their own characteristics in technical architecture, functional design and application scenarios, which provide a multi-dimensional reference for the comparative experiment of this study.

NESSUS is a widely used commercial vulnerability scanning tool. Its main advantages are high vulnerability coverage and detailed report generation capability. With built-in vulnerability scanning rules, NESSUS is able to quickly locate security vulnerabilities in target systems and generate structured reports that provide users with detailed fix recommendations. However, the design concept is primarily focused on vulnerability scanning rather than complex PT processes. The execution flow depends on static pre-defined rules and cannot be dynamically adjusted based on the intermediate results of task execution. For example, NESSUS cannot use scan results as input for further planning of vulnerability exploitation policies or other tasks, so it lacks flexibility in a multi-step PT scenario.

Deep Exploit is a RL(Reinforcement Learning)-based PT framework whose core capability lies in automating vulnerability exploitation. Through RL algorithms, Deep Exploit can dynamically adjust the attack path according to the characteristics of the target environment and select the optimal policy to execute the exploit. It has a high degree of automation and performs well in fixed or predefined environments. However, it relies on training data and the stability of the environment, and may show inadequate adaptability in the face of non-standardized or dynamically changing environments. In addition, its RL model is weak in generalization to new scenarios, which may lead to utilization failure in complex scenarios.

PentestGPT is a LLM(Large Language Model) interactive PT tool designed to enable an intelligent and user-friendly PT experience through natural language processing technology. PentestGPT is able to parse the user's natural language input, provide PT recommendations, and generate corresponding commands. However, despite its contextual understanding capabilities, PentestGPT has a relatively low degree of automation. Its workflow relies on a lot of human-computer interaction, and users need to provide input or adjust instructions several times during the testing process, which significantly increases the need for human intervention in the task. In addition, PentestGPT has some shortcomings in the accuracy of command generation, and its hallucination problem is more obvious. For example, when dealing with fuzzy input or complex scenarios, PentestGPT has the potential to generate invalid or irrelevant output, affecting the efficiency of the task.

In summary, NESSUS, Deep Exploit, and PentestGPT represent three different technology paths, respectively, traditional vulnerability scanning tools, RL-based automation frameworks, and LLM-driven intelligence tools. Their performance characteristics and technical limitations provide a multi-dimensional analysis perspective for the comparative experiments in this study

4.1.3 Test Method

In order to ensure the fairness of the experiment and the scientificity of the comparison results, this study set a uniform input and test objective for all comparison tools to evaluate their performance in different tasks. Specific test tasks are designed according to the typical flow of PT, including information collection, vulnerability discovery and utilization, and test result generation and reporting. This testing process covers all the key steps from the analysis of target environment information in the early stages of PT to the exploitation and report generation in the later stages to fully evaluate the performance, accuracy and adaptability of the tool.

In the experiment, all tools were run in the same experimental environment, the target system was consistent, and the test scope covered multiple vulnerability types in the experimental targets. The input conditions and execution process of each test task are strictly unified to ensure the comparability of test results. For example, in the information collection phase, all tools are required to scan for the same target IP and port; In the stage of vulnerability discovery and exploitation, the tool should try to exploit the same type of vulnerability; During the results reporting phase, the generated report should contain a detailed analysis of the target environment and recommendations for bug fixes.

During the experiment, the performance of each tool in various test stages was recorded, including key indicators such as completion time, task success rate and output quality. At the same time, the behavior of the tool during execution is observed, and its adaptability to dynamic environment, context memory ability and potential hallucination problem are analyzed. Through rigorous experimental design and data recording, this study can comprehensively compare the advantages and disadvantages of the proposed framework and comparison tool in practical PT tasks, so as to verify its performance and practicability.

4.2 Evaluation metrics

In order to comprehensively evaluate the performance of the AutoPenGPT framework in PT tasks[43], this study designed three core evaluation indicators: degree of automation, contextual memory ability, and hallucination frequency. These metrics cover the ability to automate task execution, the ability to handle context in complex scenarios, and the reliability of the generated output. Through a clear quantitative approach,

these metrics provide a solid theoretical basis for validating framework performance and discovering optimization directions.

4.2.1 Automation Degree

The degree of automation measures the ability of the framework to independently complete all phases of a penetration test task, including critical aspects such as information gathering, vulnerability scanning, vulnerability exploitation, and result report generation. To quantify this ability, the study evaluated three aspects: success rate, time efficiency improvement, and multitasking ability. The success rate is measured by calculating the ratio of the number of targets successfully completed by the framework to the total number of targets, namely:

$$\text{Success rate } (\%) = \frac{N_{success}}{N_{total}} \times 100$$

Where:

- $N_{success}$ is the number of successfully completed targets.
- N_{total} is the total number of targets.

By comparing the difference between the time required to complete the task manually and the time required to complete the task automatically by the framework, the efficiency improvement ratio is calculated.

$$\text{Efficiency boost } (\%) = \frac{T_{manual} - T_{auto}}{T_{manual}} \times 100$$

Where:

- T_{manual} is the time taken to complete the task manually.
- T_{auto} is the time taken to complete the task automatically by the framework.

Multitasking capability Quantifies the scalability of the framework in complex scenarios by testing its performance in concurrent task scenarios, recording the number of multiple tasks being processed at the same time and the average task completion time. These data, obtained through the multi-stage tasks of the experimental design and compared with manual baseline times, can fully reveal the framework's ability to improve efficiency and reduce human intervention.

4.2.2 Context memory

Context memory ability is an important index to evaluate the ability of framework to retain and transfer information in multi-step tasks, especially whether it can correctly use the previous information when dynamically adjusting the task plan. This research quantifies this ability through three aspects: context retention rate, context loss rate and context utilization accuracy rate. Context retention is measured by calculating the ratio of the number of correctly transmitted context messages to the total number of messages.

$$\text{context retention rate } (\%) = \frac{N_{context\,correct}}{N_{context\,total}} \times 100$$

Where:

- $N_{context\,correct}$ is the number of correctly passed context messages.
- $N_{context\,total}$ is the total number of context messages.

Context utilization accuracy is assessed by the proportion of context information that is correctly used in subsequent tasks by the statistical framework.

$$\text{context utilization accuracy } (\%) = \frac{N_{utilized\,correct}}{N_{utilized\,total}} \times 100$$

Where:

- $N_{utilized\,correct}$ is the number of correctly utilized context information.
- $N_{utilized\,total}$ is the total number of utilized context information.

By analyzing task logs and output results, this study can accurately capture the information transfer and utilization of the framework in the complex task chain, so as to reveal its adaptability in multi-stage tasks.

4.2.3 Hallucination frequency

The frequency of hallucination[21] is a key index to measure the possibility of frame generating error or invalid output during task execution, which directly reflects the reliability of frame output. This study evaluated three aspects: hallucination rate, severe hallucination rate and user intervention demand rate. The hallucination rate represents the proportion of the hallucination output generated by the framework to the total output.

$$\text{Hallucination rate } (\%) = \frac{N_{hallucination}}{N_{total\,output}} \times 100$$

Where:

- $N_{hallucination}$ is the number of hallucination outputs generated by the framework.
- $N_{total\,output}$ is the total number of outputs generated by the framework.

The severe hallucination rate further quantifies the proportion of hallucination output that significantly affects normal task completion.

$$\text{severe hallucination rate } (\%) = \frac{N_{severe\,hallucination}}{N_{hallucination}} \times 100$$

Where:

- $N_{severe\,hallucination}$ is the number of severe hallucination outputs that affect task completion.
- $N_{hallucination}$ is the total number of hallucination outputs.

User intervention demand rate measures the frequency of manual correction or intervention due to hallucination problems and is calculated as:

$$\text{User intervention demand rate } (\%) = \frac{N_{intervention}}{N_{total\,tasks}} \times 100$$

Where:

- $N_{intervention}$ is the number of tasks requiring manual intervention.
- $N_{total\,tasks}$ is the total number of tasks performed.

By manually reviewing the output, this study can fully reveal the advantages and disadvantages of the framework in terms of output reliability.

These three core evaluation indexes build a complete performance evaluation system from different dimensions. The degree of automation focuses on the framework's performance in reducing manual dependencies and improving efficiency; Context memory ability was evaluated for information transfer and utilization in multi-step tasks. The frequency of hallucinations further emphasizes the reliability of the frame output. This system provides a clear direction for experimental design and analysis, and the practical application potential and technical boundaries of AutoPenGPT will be revealed through experimental results in the following chapters.

4.3 Results

In this section, we present the findings from our comprehensive evaluation of the AutoPenGPT framework. We start by detailing the framework's performance in automation, measuring its ability to complete tasks with minimal human intervention, improve time efficiency, and manage multitasking. We then examine the contextual memory capabilities of AutoPenGPT, assessing its effectiveness in retaining and utilizing information across complex, multi-step tasks. Additionally, we evaluate the frequency of hallucinations—frequency where the framework generates incorrect or irrelevant outputs—particularly under complex task conditions. The results from these tests are compared against existing tools like Nessus, Deep Exploit, and PentestGPT to highlight AutoPenGPT's strengths and areas for improvement. This analysis provides a detailed picture of AutoPenGPT's performance and outlines potential directions for further enhancing its capabilities in automated penetration testing.

4.3.1 Automation degree evaluation

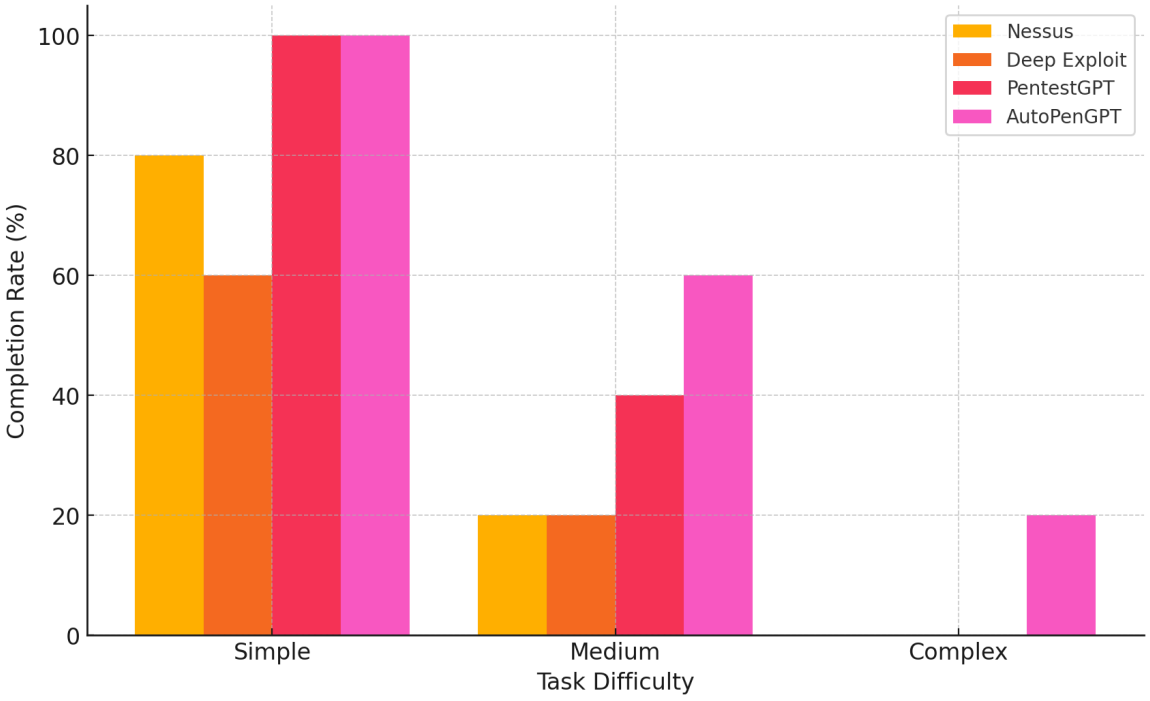

Figure 4.1: Task Completion Rates By Tool And Task Difficulty

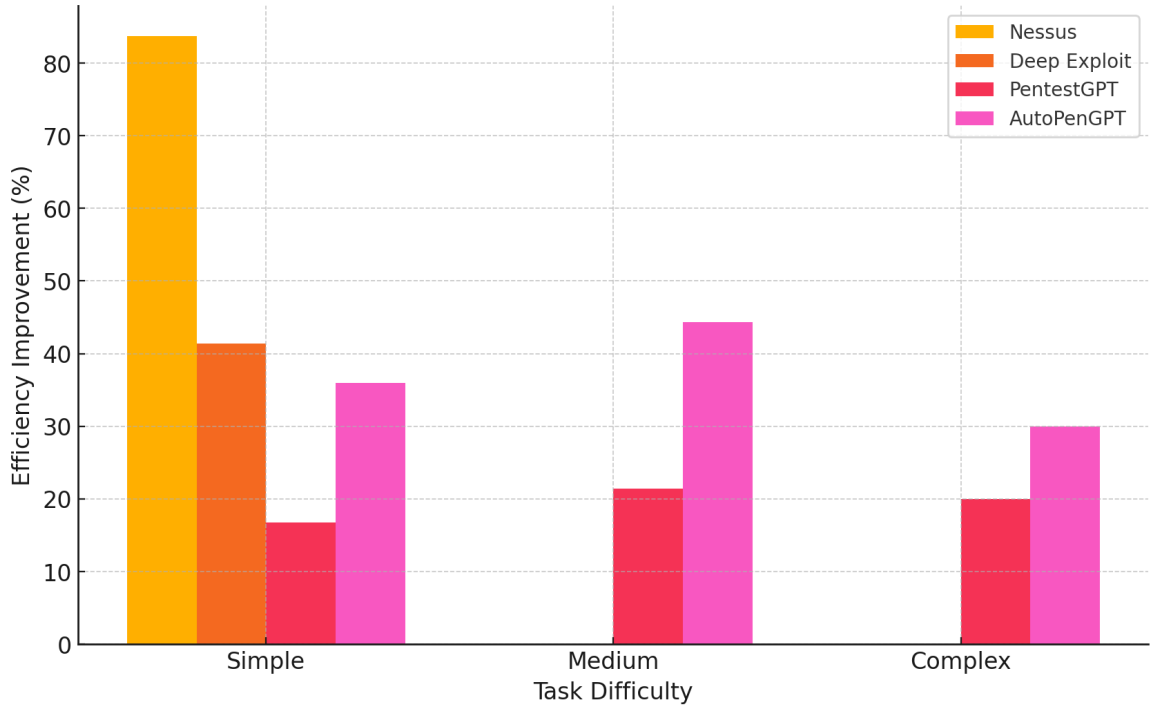

Figure 4.2: Time Efficiency Improvement By Tool And Task Difficulty

The degree of automation is the core measure of how well a tool can accomplish tasks independently during PT. This study comprehensively assesses the automation performance of Nessus, Deep Exploit, PentestGPT, and AutoPenGPT by comparing their task completion ratio, time efficiency improvement, and dynamic task adjustment capabilities at different task difficulties. *The experimental results showed that the task completion rate and efficiency improvement of the tool decreased with the increase of task difficulty, but the performance of AutoPenGPT in complex tasks was significantly better than other tools.*

Nessus had an 80 percent success rate on simple tasks, demonstrated a high level of automation, and its efficient rules engine enabled it to quickly identify and report common vulnerabilities. However, due to the lack of dynamic adjustment capability, Nessus' success rate drops to 20% on medium tasks and completely fails on complex tasks. Deep Exploit has a 60% success rate on simple tasks, but its fixed model limits its ability to adapt to complex scenarios, achieving the same completion rate on medium and complex tasks as Nessus.

Both PentestGPT and AutoPenGPT, based on the LLM, achieved 100% success rates on simple tasks, demonstrating their advantage when dealing with intuitive vulnerabilities. However, as the difficulty of the task increased, PentestGPT's success rate dropped to 40% on medium tasks and to 0% on complex tasks due to its reliance

on user guidance. In contrast, AutoPenGPT, with its dynamic task adjustment capability and autonomous learning mechanism, achieved a success rate of 60 percent in medium tasks and 20 percent in complex tasks, slightly higher than other tools.

In terms of time efficiency improvement, Nessus achieved 83.7% efficiency improvement in simple tasks, but its efficiency improvement gradually weakened with the increase of task difficulty. The efficiency of Deep Exploit on simple tasks increased to 41.4%, PentestGPT and AutoPenGPT to 16.8% and 36.0%, respectively. In the medium task, AutoPenGPT performed best with an efficiency increase of 44.3%, PentestGPT with 21.4%, while Nessus and Deep Exploit failed to improve efficiency due to task failures. For complex tasks, AutoPenGPT maintained a 30% efficiency improvement, while PentestGPT achieved only 20%.

In summary, AutoPenGPT has shown advantages in various evaluation indicators of automation degree, especially in dynamic task adjustment and complex task processing, and its capability is superior to existing tools.

4.3.2 Contextual memory ability evaluation

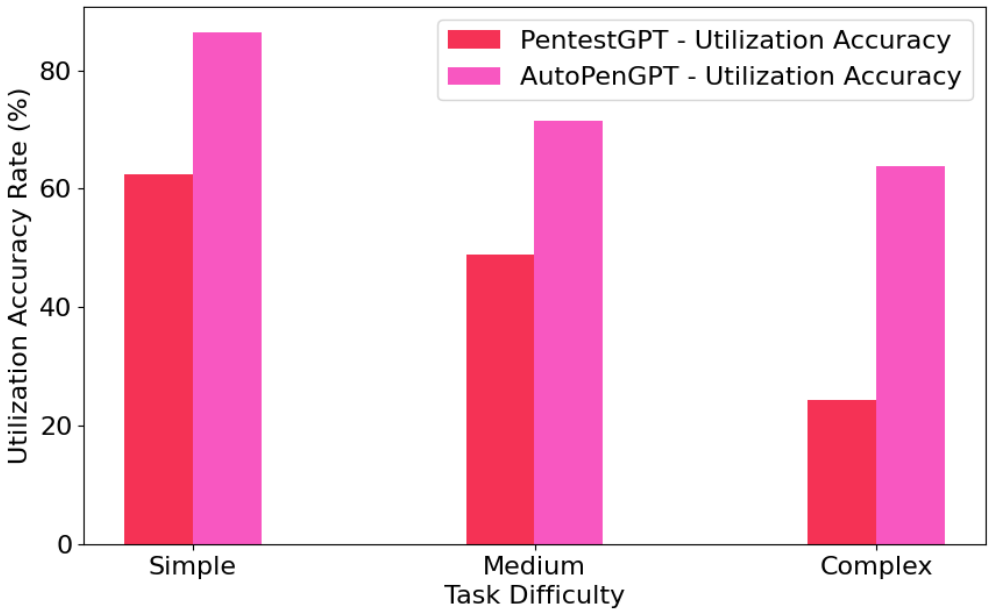

Figure 4.3: Utilization Accuracy Rate by Tool and Task Difficulty

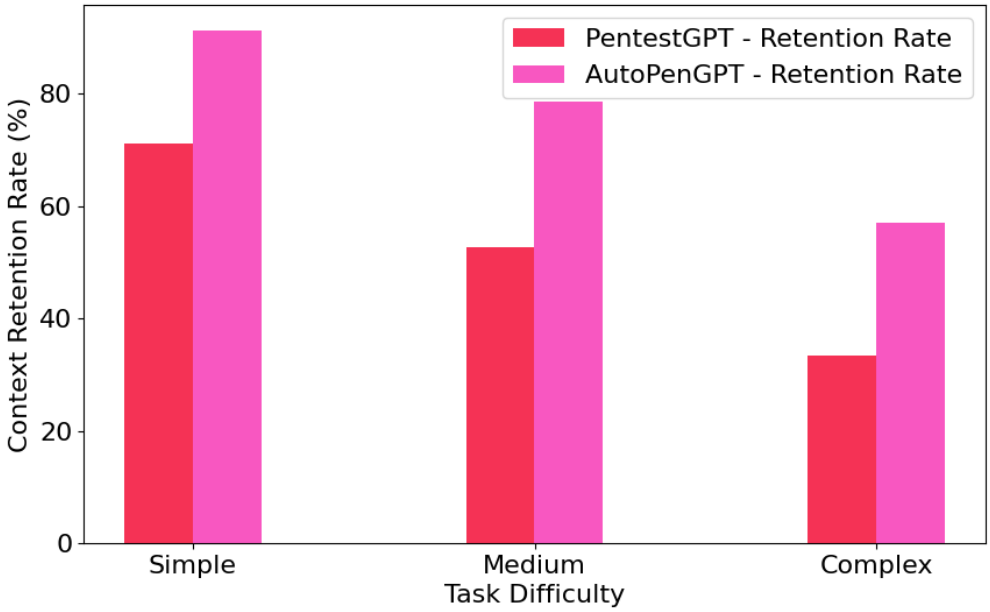

Figure 4.4: Context Retention Rate by Tool and Task Difficulty

The ability to remember context is critical to the success of multi-step tasks, especially in complex PT where information needs to be passed across stages. In this study, context retention rate, loss rate and utilization accuracy rate were used to evaluate the context processing capability of the tool.

In simple tasks, PentestGPT had a context retention rate of 71.2% and utilization accuracy of 62.5%. However, its context retention rate dropped to 52.7 percent on medium tasks and further to 33.4 percent on complex tasks, with an accuracy of 48.9 percent and 24.3 percent, respectively. This decline reflects PentestGPT's significant context passing problems in complex tasks. In contrast, the context retention rate of AutoPenGPT in simple tasks reached 91.3%, and the utilization accuracy rate was 86.4%. In the medium task, the retention rate and accuracy rate were 78.6% and 71.5% respectively. For complex tasks, retention dropped to 57.1 percent and accuracy remained at 63.8 percent. *These results indicate that AutoPenGPT has significantly better contextual memory ability than PentestGPT when processing multi-step tasks.*

The effect of context loss on task completion rate is verified in the experiment. For example, in a moderately difficult task, PentestGPT failed to pass the port information of the scanning phase correctly, resulting in the generation of the wrong attack script in the subsequent exploit phase, and finally the task failed. AutoPenGPT, on the other hand, is able to correctly deliver scan results, successfully exploit vulnerabilities, and

generate effective reports. This case illustrates the importance of context memory to the success of complex tasks.

AutoPenGPT combines RAG(Retrieval-Augmented Generation) and LLM Agent technologies and uses the output of previous steps for subsequent tasks through dynamic retrieval and generation mechanisms, effectively reducing the problem of context loss. Even in complex tasks, its context retention rate and utilization accuracy remain ahead, which makes AutoPenGPT more adaptable in tasks requiring multi-step information transfer.

4.3.3 Hallucination frequency Evaluation

Hallucination issues directly affect tool reliability and user experience, especially in complex tasks. The experiment comprehensively analyzed the output quality of the tool under different task difficulty by statistical hallucination rate, severe hallucination rate and user intervention demand rate.

In simple tasks, PentestGPT had a hallucination rate of 8.9 percent, with a severe hallucination rate of 2.5 percent. AutoPenGPT performed better, with a hallucination rate of 7.3% and a severe hallucination rate of only 1.8%. In moderate tasks, PentestGPT's hallucination rate rose to 11.4 percent, severe hallucinations to 4.8 percent, and user intervention needs to rise to 30 percent. In comparison, AutoPenGPT had a hallucination rate of 9.2 percent, a severe hallucination rate of 3.2 percent, and a need for intervention rate of 18 percent. In complex tasks, the hallucination rate of PentestGPT reached 15.3%, the severe hallucination rate was 7.1%, and the intervention demand rate rose to 42%. For AutoPenGPT, the hallucination rate was 12.7%, the severe hallucination rate was 5.5%, and the need for intervention rate was 26%.

These data indicate that AutoPenGPT outperforms PentestGPT in reducing hallucinatory output and the need for user intervention. This advantage is mainly due to AutoPenGPT's counterfactual analysis strategy, where when a potential hallucination output is detected, its model generates an optimized output through a loop, thus significantly reducing the probability of severe hallucinations.

Table 4.4: Hallucination Rate by Tool and Task Difficulty

Tool	Simple	Medium	Complex
PentestGPT	8.9%	11.4%	15.3%
AutoPenGPT	7.3%	9.2%	12.7%

Table 4.5: Severe Hallucination Rate by Tool and Task Difficulty

Tool	Simple	Medium	Complex
PentestGPT	2.5%	4.8%	7.1%
AutoPenGPT	1.8%	3.2%	5.5%

Table 4.6: User Intervention Demand Rate by Tool and Task Difficulty

Tool	Simple	Medium	Complex
PentestGPT	0%	30%	42%
AutoPenGPT	0%	18%	26%

Severe hallucinations not only directly lead to task failure, but also significantly increase the user's need for intervention. During a complex task, PentestGPT generated an unexecutable command, causing the test process to break and requiring manual correction by the user. In the same task, AutoPenGPT can identify the error of the command through the verification mechanism and automatically generate a corrected version, avoiding manual intervention. This capability significantly improves task completion efficiency and user experience.

4.4 Case Studies

This section presents two case studies to evaluate the performance of the proposed AutoPenGPT framework under both successful and failure scenarios. The first case study highlights a successful multi-step penetration test where AutoPenGPT efficiently carried out vulnerability exploitation and privilege escalation. The second case study examines a failure scenario involving complex vulnerabilities and ambiguous inputs, which serves to demonstrate the limitations of the framework in handling uncertainty and complexity. Through these case studies, we provide valuable insights into the strengths and weaknesses of AutoPenGPT, offering a basis for further refinement and development of the framework.

4.4.1 Successful Case Analysis

In an ideal task scenario designed to verify the performance of the AutoPenGPT framework, we explored a multi-step vulnerability exploitation process starting with SQL injection leading to privilege escalation. This chain began with AutoPenGPT

performing an automated scan to identify SQL injection vulnerabilities on a web application. Once identified, the system automatically generated SQL injection payloads to test and exploit the vulnerability without user intervention. Following successful exploitation, AutoPenGPT then executed a series of commands aimed at escalating privileges within the compromised system.

AutoPenGPT's seamless transition from scanning to exploitation and then to privilege escalation demonstrates its superior task automation and dynamic adjustment capabilities. Unlike traditional tools like Nessus, which can only scan and report vulnerabilities, AutoPenGPT not only identified the vulnerability but also took actionable steps to exploit it, thereby demonstrating a comprehensive approach to penetration testing. The entire process was executed with a high degree of accuracy and efficiency, taking a total of 319.8 seconds to complete, which significantly outperformed other tools such as Nessus and PentestGPT in similar scenarios.

4.4.2 Failure Case Analysis

To assess the limitations of AutoPenGPT and other tools under more complex and fuzzy input conditions, a challenging scenario was created involving intricate logic vulnerabilities with multi-step dependencies. The task was designed to mimic a real-world situation where the inputs were ambiguous, and the vulnerabilities were not well-defined, testing the framework's ability to handle complex logic under uncertain conditions.

In this complex scenario, AutoPenGPT started by scanning for potential vulnerabilities but generated several irrelevant results due to the fuzzy nature of the inputs. During the vulnerability validation phase, the exploitation scripts generated by AutoPenGPT were misaligned with the actual vulnerabilities present, leading to failed attempts and incorrect task execution. Despite several dynamic adjustments attempted by the system, the context analysis mechanism struggled to correctly interpret the feedback from previous steps, ultimately affecting the task's overall success.

The analysis of this failure highlighted several issues, primarily hallucination outputs where the system generated incorrect or irrelevant information based on misunderstood or misinterpreted data. This led to a cascade of errors in subsequent tasks, demonstrating a significant challenge in handling complex scenarios with ambiguous information.

This case study underlines the robust capabilities of AutoPenGPT in handling well-defined, structured tasks while also revealing critical areas for improvement in managing tasks with high complexity and ambiguity. Future developments for the

framework will focus on enhancing the accuracy of context interpretation, improving the detection and correction of hallucinations, and refining the dynamic adjustment mechanisms to better manage the uncertainties inherent in complex penetration testing environments.

4.5 Summary

In this chapter, we conduct a comprehensive evaluation of the AutoPenGPT framework, evaluating its performance in automated penetration testing compared to established tools such as Nessus, Deep Exploit, and PentestGPT. Initially, we detailed the experimental setup, including hardware and software environments, to ensure consistency and repeatability between tests. We then introduce the evaluation criteria - degree of automation, contextual memory ability, and hallucination frequency - that are key to assessing the effectiveness of penetration test automation. Methods for calculating these metrics are discussed, with a focus on how they provide quantitative insights into AutoPenGPT performance. A subsequent section describes a series of experiments designed to challenge these tools with tasks of varying complexity, providing a comprehensive dataset for analysis.

The results section compares the performance of AutoPenGPT with the selected tool in detail. The results show that AutoPenGPT performs well in complex scenarios, effectively reduces human intervention, and maintains context in multi-step tasks, significantly outperforming other methods in these areas. Although AutoPenGPT still needs to be improved in terms of context loss and hallucination frequency in highly complex tasks, the research and application of AutoPenGPT shows that it has the potential to significantly improve the efficiency of penetration test automation. By further optimizing its context memory mechanism and reducing the frequency of hallucinations, AutoPenGPT is expected to become an important technical tool in the field of cybersecurity, providing security experts with more efficient and accurate test results. The evaluation results in this chapter not only verify the practicability and effectiveness of AutoPenGPT in PT, but also provide an important basis for subsequent optimization.

Chapter 5

Conclusion

In this thesis, we develop and comprehensively evaluate the AutoPenGPT framework, an advanced automated penetration testing system that integrates sophisticated techniques such as RAG(Retrieval-Augmented Generation), LLM(Large Language Model) agents, and MoE to address current challenges in cybersecurity testing. AutoPenGPT aims to enhance traditional PT(Penetration Test) methods by providing comprehensive task coverage, from information gathering to vulnerability exploitation and report generation, coupled with dynamic adjustment mechanisms that allow flexibility to adapt to different mission environments. This approach ensures a high degree of automation and minimizes the need for human intervention.

Our evaluation results show that AutoPenGPT not only outperforms traditional tools such as Nessus and Deep Exploit in complex and adaptive scenarios, but also offers considerable improvements over similar tools such as PentestGPT in reducing user involvement and enhancing the autonomous flow of the testing process. While AutoPenGPT is excellent at handling complex multi-step tasks by integrating LLM and retrieval mechanisms, it still faces challenges in terms of efficiency, illusion control, and high computational costs for simple tasks. These findings highlight the potential of AutoPenGPT to transform penetration testing practices by providing a smarter, more adaptable, and thorough testing process, which is critical to managing modern cybersecurity threats.

Overall, the integration of advanced machine learning technologies within the AutoPenGPT framework provides a forward-thinking approach to complex PT environments, ensuring efficiency and adaptability to meet evolving cybersecurity challenges.

5.1 Discussion

The comprehensive assessment of AutoPenGPT underscores its potential to significantly advance the field of automated penetration testing, though it also identifies essential areas for enhancement. Primarily, the system's performance in managing complex and ambiguous testing scenarios needs improvement. Despite achieving commendable context retention and accuracy rates for moderately complex tasks, these metrics fall notably when confronting high-complexity tasks due to the system's limited ability in managing extended contextual dependencies[29][32]. This limitation can influence the system's applicability in intricate security environments, where precise and context-aware task generation is crucial.

Moreover, while AutoPenGPT shows a reduction in hallucinations compared to its counterparts, the persistence of these errors—12.7% for hallucinations and 5.5% for severe hallucinations in complex tasks—highlights the need for more robust mechanisms to mitigate misinformation that can disrupt task flows and increase manual oversight[39][33]. Enhancing these aspects is critical for maintaining task integrity and operational efficiency, particularly in scenarios that demand high accuracy.

Organizations operating in dynamically changing security landscapes stand to benefit significantly from the deployment of AutoPenGPT. Its ability to adaptively generate and adjust testing tasks based on real-time data and external knowledge makes it particularly valuable for sectors where security is paramount. These include finance, healthcare, and government agencies, which frequently encounter sophisticated cyber threats. The continuous development of the system's knowledge base and its refinement of error correction mechanisms will further enhance its utility, making it a robust tool in the arsenal against cyber vulnerabilities. Additionally, while the system's resource intensity and reliance on cutting-edge hardware could limit its application in resource-constrained settings, its benefits for well-resourced environments are substantial, offering a sophisticated and scalable solution to modern cybersecurity challenges.

5.2 Future work

Although AutoPenGPT shows strong potential in the field of PT, there are still many aspects that deserve further exploration. Future research can be extended and optimized from the following directions to improve the performance and applicability of the framework.

First of all, further enhancing context memory capability is an important direction of optimization framework. When dealing with complex tasks, improving context retention and utilization accuracy will significantly improve task completion rates and efficiency. In the future, more advanced context management mechanisms, such as model design based on memory networks or enhanced Transformer architecture, can be introduced to enable more efficient processing of long-term dependent information. In addition, dynamic weight allocation techniques can help models identify and utilize critical context information in tasks more accurately.

Secondly, for the hallucination problem, future research needs to develop more advanced hallucination detection and control mechanisms. For example, a multimodal feedback validation mechanism can be combined to identify and correct potential error outputs in a timely manner by comparing the generated output with real-time feedback from the target environment.

Finally, expanding the scope of application of AutoPenGPT is also an important direction for future work. The current framework is optimized for typical vulnerability testing tasks, but its performance has not been fully validated when dealing with broader cybersecurity scenarios such as threat intelligence analysis, multi-target collaborative testing, etc. By combining a richer external knowledge base with domain-specific fine-tuning[9] techniques, AutoPenGPT is expected to demonstrate its capabilities in a more diverse range of security tasks.

To better integrate AutoPenGPT into existing cybersecurity pipelines, such as Security Operations Center (SOC) workflows, or commercial tools such as Metasploit, future research should consider developing compatibility with interfaces to these systems and tools. Furthermore, it is imperative to address ethical considerations rigorously, acknowledging the risks associated with the misuse of AI in cybersecurity and proposing robust safeguards to mitigate such risks.

Bibliography

[1] Muhammad Arslan, Hussam Ghanem, Saba Munawar, and Christophe Cruz. A survey on rag with llms. *Procedia Computer Science*, 246:3781–3790, 2024.

[2] Ömer Aslan, Semih Serkant Aktuğ, Merve Ozkan-Okay, Abdullah Asim Yilmaz, and Erdal Akin. A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions. *Electronics*, 12(6):1333, 2023.

[3] Deng Cai, Yan Wang, Lemao Liu, and Shuming Shi. Recent advances in retrieval-augmented text generation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pages 3417–3419, 2022.

[4] Neeloy Chakraborty, Melkior Ornik, and Katherine Driggs-Campbell. Hallucination detection in foundation models for decision-making: A flexible definition and review of the state of the art. *ACM Computing Surveys*, 2024.

[5] Jinyin Chen, Shulong Hu, Haibin Zheng, Changyou Xing, and Guomin Zhang. Gail-pt: An intelligent penetration testing framework with generative adversarial imitation learning. *Computers & Security*, 126:103055, 2023.

[6] Junyang Chen, Rui Mi, Huan Wang, Huisi Wu, Jiqian Mo, Jingcai Guo, Zhihui Lai, Liangjie Zhang, and Victor CM Leung. A review of few-shot and zero-shot learning for node classification in social networks. *IEEE Transactions on Computational Social Systems*, 2024.

[7] Gabriel de Jesus Coelho da Silva and Carlos Becker Westphall. A survey of large language models in cybersecurity, 2024.

[8] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. Pentestgpt: An llm-empowered automatic penetration testing tool, 2024.

[9] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[10] John Fields, Kevin Chovanec, and Praveen Madiraju. A survey of text classification with transformers: How wide? how large? how long? how accurate? how expensive? how safe? *IEEE Access*, 2024.

[11] Mohamed C. Ghanem, Thomas M. Chen, and Erivelton G. Nepomuceno. Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks. *Journal of Intelligent Information Systems*, 60(2):281–303, April 2023.

[12] Mohamed Chahine Ghanem. *Towards an efficient automation of network penetration testing using model-based reinforcement learning.* PhD thesis, City, University of London, 2022.

[13] Ross Gruetzemacher and David Paradice. Deep transfer learning & beyond: Transformer language models in information systems research. *ACM Computing Surveys (CSUR)*, 54(10s):1–35, 2022.

[14] Kailash A Hambarde and Hugo Proenca. Information retrieval: recent advances and beyond. *IEEE Access*, 2023.

[15] Ismayil Hasanov, Seppo Virtanen, Antti Hakkala, and Jouni Isoaho. Application of large language models in cybersecurity: A systematic literature review. *IEEE Access*, 2024.

[16] William Grant Hatcher and Wei Yu. A survey of deep learning: Platforms, applications and emerging research trends. *IEEE access*, 6:24411–24432, 2018.

[17] Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 2025.

[18] Eric Hilario, Sami Azam, Jawahar Sundaram, Khwaja Imran Mohammed, and Bharanidharan Shanmugam. Generative ai for pentesting: the good, the bad, the ugly. *International Journal of Information Security*, 23(3):2075–2097, 2024.

[19] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 2024.

[20] Mamoona Humayun, Mahmood Niazi, NZ Jhanjhi, Mohammad Alshayeb, and Sajjad Mahmood. Cyber security threats and vulnerabilities: a systematic mapping study. *Arabian Journal for Science and Engineering*, 45:3171–3189, 2020.

[21] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[22] Rohit Kumar Kaliyar. A multi-layer bidirectional transformer encoder for pre-trained word embedding: A survey of bert. In *2020 10th International conference on cloud computing, data science & engineering (confluence)*, pages 336–340. IEEE, 2020.

[23] Wafaa Kasri, Yassine Himeur, Hamzah Ali Alkhazaleh, Saed Tarapiah, Shadi Atalla, Wathiq Mansoor, and Hussain Al-Ahmad. From vulnerability to defense: The role of large language models in enhancing cybersecurity. *Computation*, 13(2):30, 2025.

[24] Kentaro Katahira, Bai Yu, and Takashi Nakao. Pseudo-learning effects in reinforcement learning model-based analysis: A problem of misspecification of initial preference. 2017.

[25] Johnson Kinyua and Lawrence J. Awuah. Ai/ml in security orchestration, automation and response: Future research directions. *Intelligent Automation and Soft Computing*, 28:527–545, 2021.

[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[27] Qianyu Li, Miao Hu, Hao Hao, Min Zhang, and Yang Li. Innes: An intelligent network penetration testing model based on deep reinforcement learning. *Applied Intelligence*, 53(22):27110–27127, 2023.

[28] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[29] Sifan Long, Jingjing Tan, Bomin Mao, Fengxiao Tang, Yangfan Li, Ming Zhao, and Nei Kato. A survey on intelligent network operations and performance optimization based on large language models. *IEEE Communications Surveys & Tutorials*, 2025.

[30] Sikender Mohsienuddin Mohammad and Lakshmisri Surya. Security automation in information technology. *SSRN Electronic Journal*, 6:901–905, 06 2018.

[31] Rana F Najeeb, Ban N Dhannoon, and Farah Qais Alkhalidi. Topic modeling based bert & sbert transformer pretrained language modeling: A survey (2019-2023). 3169(1), 2025.

[32] Rajvardhan Patil, Sorio Boit, Venkat Gudivada, and Jagadeesh Nandigam. A survey of text representation and embedding techniques in nlp. *IEEE Access*, 11:36120–36146, 2023.

[33] Andreas Pester, Ahmed Tammaa, Christian Gütl, Alexander Steinmaurer, and Samir Abou El-Seoud. Conversational agents, virtual worlds, and beyond: A review of large language models enabling immersive learning. pages 1–6, 2024.

[34] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, 1998.

[35] Sandeep Pochu, Srikanth Reddy Kathram, and Senior DevOps Engineer. Automated vulnerability assessment leveraging ai for enhanced security. *Journal of Multidisciplinary Research*, 8(01):14–25, 2022.

[36] Konstantin Pozdniakov, Eduardo Alonso, Vladimir Stankovic, Kimberly Tam, and Kevin Jones. Smart security audit: Reinforcement learning with a deep neural network approximator. In *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pages 1–8. IEEE, 2020.

[37] Chunyuan Qin, Chuan Deng, Jiashun Huang, Kunxian Shu, and Mingze Bai. An efficient faiss-based search method for mass spectral library searching. pages 513–518, 2020.

[38] J Ranjith, Santhi Baskaran, and B Adithya. Mitigating catastrophic forgetting in deep learning models for sentiment analysis. 1:1–7, 2024.

[39] G Pradeep Reddy, YV Pavan Kumar, and K Purna Prakash. Hallucinations in large language models (llms). pages 1–6, 2024.

[40] S Revathi, J Raja, M Mohanraj, K Malathi, Balasubbareddy Mallala, and RG Vidhya. Challenges in cyber physical social systems and internet of things. In *2024 5th International Conference on Smart Electronics and Communication (ICOSEC)*, pages 409–413. IEEE, 2024.

[41] Imtithal A Saeed, Ali Selamat, Mohd Foad Rohani, Ondrej Krejcar, and Junaid Ahsenali Chaudhry. A systematic state-of-the-art analysis of multi-agent intrusion detection. *IEEE Access*, 8:180184–180209, 2020.

[42] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Pomdps make better hackers: Accounting for uncertainty in penetration testing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 1816–1824, 2012.

[43] Mandar Prashant Shah. *Comparative analysis of the automated penetration testing tools*. PhD thesis, Dublin, National College of Ireland, 2020.

[44] Yannic Smeets. Improving the adoption of dynamic web security vulnerability scanners. *Radboud University, NL*, 2015.

[45] Moritz Staudinger, Wojciech Kusa, Florina Piroi, Aldo Lipani, and Allan Hanbury. A reproducibility and generalizability study of large language models for query generation. pages 186–196, 2024.

[46] Olena Sviatun, Olga Goncharuk, Chernysh Roman, Olena Kuzmenko, and Ihor Kozych. Combating cybercrime: Economic and legal aspects. *WSEAS TRANSACTIONS ON ENVIRONMENT AND DEVELOPMENT*, 17:542–553, 05 2021.

[47] Isao Takaesu. Deepexploit: Fully automatic penetration test tool using machine learning. https://github.com/13o-bbr-bbq/machine_learning_security, 2018.

[48] Wesley Tann, Yuancheng Liu, Jun Heng Sim, Choon Meng Seah, and Ee-Chien Chang. Using large language models for cybersecurity capture-the-flag challenges and certification questions, 2023.

[49] Andrea Tundis, Wojciech Mazurczyk, and Max Mühlhäuser. A review of network vulnerabilities scanning tools: Types, capabilities and functioning. In *Proceedings of the 13th international conference on availability, reliability and security*, pages 1–10, 2018.

[50] Junli Wang, Chenyang Zhang, Dongyu Zhang, Haibo Tong, Chungang Yan, and Changjun Jiang. A recent survey on controllable text generation: a causal perspective. *Fundamental Research*, 2024.

[51] Lindsay Wells and Tomasz Bednarz. Explainable ai and reinforcement learning—a systematic review of current approaches and trends. *Frontiers in artificial intelligence*, 4:550030, 2021.

[52] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[53] Lei Wu, Xiaofeng Zhong, Jingju Liu, and Xiang Wang. Ptgroup: An automated penetration testing framework using llms and multiple prompt chains. In *Advanced Intelligent Computing Technology and Applications: 20th International Conference, ICIC 2024, Tianjin, China, August 5–8, 2024, Proceedings, Part IX*, page 220–232, Berlin, Heidelberg, 2024. Springer-Verlag.

[54] Jiacen Xu, Jack W. Stokes, Geoff McDonald, Xuesong Bai, David Marshall, Siyue Wang, Adith Swaminathan, and Zhou Li. Autoattacker: A large language model guided system to implement automatic cyber-attacks, 2024.

[55] Zezhou Yang, Sirong Chen, Cuiyun Gao, Zhenhao Li, Xing Hu, Kui Liu, and Xin Xia. An empirical study of retrieval-augmented code generation: Challenges and opportunities. *ACM Transactions on Software Engineering and Methodology*, 2025.

[56] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

[57] Ziruo Yi, Ting Xiao, and Mark V Albert. A survey on multimodal large language models in radiology for report generation and visual question answering. *Information*, 16(2):136, 2025.

[58] Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8):1177–1193, 2012.

[59] Fabio Massimo Zennaro and László Erdődi. Modelling penetration testing with reinforcement learning using capture-the-flag challenges: Trade-offs between model-free learning and a priori knowledge. *IET Information Security*, 17(3):441–457, 2023.

[60] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey, 2024.

[61] Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Xiaowei Mei, Yiqi Wang, Zhen Wen, Fei Wang, Xiangyu Zhao, Jiliang Tang, et al. Recommender systems in the era of large language models (llms). *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[62] Junhao Zheng, Shengjie Qiu, Chengming Shi, and Qianli Ma. Towards lifelong learning of large language models: A survey. *ACM Computing Surveys*, 2024.

[63] Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. Large language model for vulnerability detection and repair: Literature review and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 2024.

[64] Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. Large language model for vulnerability detection and repair: Literature review and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 2024.

[65] Lin Zhu, Suryadipta Majumdar, and Chinwe Ekenna. An invisible warfare with the internet of battlefield things: A literature review. *Human Behavior and Emerging Technologies*, 2020.