Pentester's Copilot: Context-Aware, Agent-Powered, Pentest-Perfected

Yarra Neeli Venkata Rajesh
Independent Researcher
rajesh.yarra1241@gmail.com

Abstract—**In this research, we introduce Pentester's Copilot, a novel framework that leverages Artificial Intelligence. Specifically, it employs Large Language Models (LLMs) to enhance the skills and efficiency of cybersecurity professionals and learners performing offensive security tasks. Pentester's Copilot seamlessly integrates with the penetration-testing lifecycle, offering an intelligent assistant that provides both interactive guidance and autonomous execution of tasks through AI agents. what distinguishes Pentester's Copilot from existing tools is its persistent memory layer, which enables personalized, context aware interactions across sessions. This memory layer is implemented using Mem0 and Supabase, allowing the system to adapt its responses to each user's preferences, goals, and past activities. Moreover, Pentester's Copilot is tailored specifically for offensive security queries, supports multimodal inputs and employs an agentic architecture capable of autonomously planning and executing comprehensive penetration testing workflows. Pentester's Copilot leverages a suite of state-of-the-art LLMs including Meta Llama 3.1-405b, Google Gemini 2.0 Flash, and DeepSeek R1 to generate recommendations, craft exploit payloads, and analyze target environments and many more. The Agent Mode is built on the ReAct (Reasoning and Action) framework, LangChain Framework orchestrates autonomous planning and tool interaction. By combining interactive dialogue with autonomous agents, Pentester's Copilot can carry out end to end penetration testing scenarios, from reconnaissance and vulnerability scanning to exploitation and reporting. The Pentester's Copilot marks a significant shift in the implementation of artificial intelligence in the offensive security domain. The combination of ongoing personalization, multimodal understanding, and self-directed agentic processes presents a solid foundation for future investigation into automated penetration testing enabled through large language models.**

Index Terms—**Penetration Testing, Cybersecurity, Large Language Models (LLMs), AI Agents, ReAct Framework, Memory Layer, Offensive Security, Ethical Hacking, LangChain, Mem0, Supabase.**

I. INTRODUCTION

The domain of cybersecurity is characterized by a relentless arms race between attackers and defenders [1]. As digital infrastructures become more intricate and interconnected, the attack surface expands, and threat actors continuously devise novel methods to exploit vulnerabilities [2], [3]. Penetration testing, the practice of simulating authorized cyberattacks to identify and assess security weaknesses, remains a fundamental component of a proactive defense strategy [4]. However, conventional penetration testing methodologies are often laborious, time-consuming, and heavily reliant on the availability of highly skilled human experts, creating bottlenecks and increasing costs for organizations.

The emergence of strong Large Language Models (LLMs) has created a lot of interest in applying artificial intelligence to automate and enhance a broad spectrum of cybersecurity activities [1], [5], [6]. LLMs have sophisticated Natural Language Processing (NLP), logical reasoning, multi-level planning, and code generation abilities that might make them invaluable tools for penetration testers [7], [8]. They can be used to perform tasks from information gathering and vulnerability scanning to exploit suggestion and report generation [9], [10].

Despite this potential, current solutions present limitations. Standalone security tools (e.g., `Nmap`, `Metasploit`, `Burp Suite`) lack seamless integration and require manual orchestration and interpretation. General-purpose AI chatbots, such as `ChatGPT` or `Claude`, often incorporate safety mechanisms that restrict their ability to provide detailed guidance on offensive techniques, and lack the tool interaction capabilities, and stateful memory necessary for effective penetration testing support. While academic research has demonstrated the feasibility of LLM agents performing autonomous hacking tasks in controlled settings [7], [11]–[18], these systems often function as proofs-of-concept, potentially lacking the robustness, comprehensive tool integration, user-friendliness, or persistent memory needed for practical application. Furthermore, studies indicate a performance gap, with open-source models often lagging behind proprietary counterparts in complex autonomous tasks [15].

To bridge these gaps, we developed Pentester's Copilot, an AI-powered platform specifically designed to serve as an intelligent assistant and automation tool for penetration testers and offensive security practitioners. Our system distinguishes itself by integrating several core functionalities into a unified, user-friendly web application:

1) Interactive Chat Mode: Allows users to converse with various LLMs (Meta `Llama 3.1-405B`, Google `Gemini`

`2.0-flash`, `DeepSeek R1`) tailored for different tasks, from quick queries to complex reasoning.

2) Persistent Memory Layer: Leverages `Mem0` and a `Supabase` vector database to store and retrieve relevant user preferences, past interactions, discovered vulnerabilities, successful commands, and many more enabling personalized and stateful assistance across sessions by leveraging Retrieval Augmented Generation (RAG).
3) Multimodal Input: Supports processing of text, images, and documents (PDFs), enabling analysis of diverse inputs like screenshots or technical documents.
4) Assistant Mode: Envisions real-time, interactive guidance via screen sharing, leveraging Google's `Gemini Live API` to understand context and offer step-by-step help.
5) Autonomous Agent Mode: Implements the LangChain ReAct (Reasoning and Acting) framework [19], allowing the AI to autonomously plan and execute multi-step penetration testing tasks (e.g., `Nmap` scans, vulnerability checks) based on high-level user goals.

Pentester's Copilot aims to streamline workflows, accelerate vulnerability discovery, and empower cybersecurity professionals by providing a focused, capable, and context-aware AI assistant for offensive security tasks. This paper details its motivation, objectives, architecture, implementation, and discusses its place within the landscape of AI in cybersecurity.

A. Objectives

The core objectives guiding the development of Pentester's Copilot are:

- Accelerate vulnerability discovery.
- Demonstrate practical AI application in offensive security.
- Explore the boundaries of LLM capabilities in hacking contexts.
- Enhance pentesting efficiency and speed through automation.
- Provide personalized, context-aware assistance via persistent memory.
- Enable intelligent automation of tasks using the Agent Mode.

II. RELATED WORK

The development of Pentester's Copilot builds upon and extends several intersecting lines of research in cybersecurity automation and Large Language Model (LLM) applications. This section reviews the pertinent literature, situating our contribution within the current state-of- the-art.

A. Traditional Penetration Testing Tools and Automation

The foundation of modern penetration testing relies on a vast array of specialized tools. Network scanners like `Nmap`, vulnerability scanners such as `Nessus` and `OpenVAS`, web proxies like `Burp Suite`, and exploitation frameworks including `Metasploit` are indispensable for identifying weaknesses. While powerful, these tools typically excel at specific tasks and often operate in isolation. Effective utilization requires significant manual configuration, execution, interpretation, and correlation of findings across different tools, contributing to the time-consuming nature of comprehensive penetration tests. Early attempts at automation were focused mainly on automating mundane tasks or coding heuristic-based vulnerability scanning software, which, while useful, are inflexible, lack reasoning, and contextual knowledge necessary for the complex, multi-step attacks prevalent in today's environments.

B. AI and Machine Learning in Cybersecurity

Prior to the widespread adoption of LLMs, AI and Machine Learning (ML) techniques were already being explored for various cybersecurity applications [20]. Research focused on areas like intrusion detection, malware classification, and anomaly detection, often using supervised or unsupervised learning models trained on specific network traffic or system log datasets. While valuable for pattern recognition and classification, these earlier AI approaches typically lacked the generative, planning, contextual understanding in and complex reasoning capabilities that the modern LLMs has, limiting their applicability to automating the offensive simulation tasks involved in penetration testing.

C. Large Language Models in Cybersecurity Tasks

The emergence of powerful LLMs [1], [21] has spurred a wave of research into their application across numerous cybersecurity domains beyond penetration testing [3], [5]. Studies have demonstrated LLM proficiency in analyzing threat intelligence reports [10], [22], synthesizing security information, detecting vulnerabilities in source code [23], [24], suggesting code repairs, generating security policies, and even helping in security report generation [6]. These applications highlight the general utility of LLMs in processing and reasoning about security-related textual and code data, providing a strong rationale for their integration into more complex workflows like penetration testing. However, as noted in our Introduction, general-purpose LLMs often face limitations due to safety Guardrails and lack of specialized offensive security knowledge.

D. LLM-Assisted and Autonomous Penetration Testing

Recognizing both the capabilities and limitations of LLMs, recent research has bifurcated into two main streams: LLM-assisted (human-in-the-loop) systems and fully autonomous LLM agents for penetration testing.

In the assisted paradigm, a prominent example is Pentest-GPT [4]. It utilizes an LLM, typically GPT-4, to guide human testers through a structured process involving reasoning, command generation, and parsing of tool outputs, organized within a "Pentesting Task Tree." While useful in helping users with specific issues, PentestGPT relies on the human operator to a great extent for command execution, interpretation of ambiguous results, and the taking of necessary decisions, and hence is a primarily advanced advisory system [4]. Similarly, Goyal et al. [25] introduce the concept of "LLM Augmented Pentesting" with their tool, also named "Pentest Copilot." Their approach focuses on leveraging LLMs (specifically GPT-4-Turbo) to automate discrete sub- tasks within the human tester's workflow, such as utilizing testing tools, interpreting outputs, and suggesting follow-up actions. Employing techniques like Chain-of-Thought [26] to optimize token utilization and enhance decision-making, and Retrieval-Augmented Generation (RAG) [27] to address hallucinations and offer access to current cybersecurity information, their system is intended to combine LLM capabilities with some pentesting requirements in an effort to enhance the productivity of the human tester without attempting full workflow automation.

Concurrently, research into fully autonomous agents has also evidenced growing sophistication. Pathbreaking work by Fang et al. [15]–[17] has proved that agents using frontier models (GPT-4) can independently use web vulnerabilities, such as sophisticated SQL injections and zero-day exploits, and hence underscore the requirement for advanced tool use, document comprehension, and planning capabilities. Happe and Cito [18], [28] focused on autonomous Linux privilege escalation, developing benchmarks and demonstrating both the potential and the common-sense limitations of LLM agents in such tasks. Other frameworks like `AutoAttacker` [29] concentrate on automating post-breach activities, while `VulnBot` [7] utilizes a multi-agent system and a task graph for broader workflow simulation. Platforms like `HackSynth` [14] test autonomous agents within CTF environments. Despite progress, these autonomous systems, particularly those relying on open-source models [15], can still struggle with strong robustness, full real-world tool integration, dynamic environment adjustment, and long-term state management against human experts. Long-term context management and state tracking for long, multi-step interactions are a longstanding challenge for autonomous agents, typically constrained by the native context window of the LLM.

E. Positioning Pentester's Copilot

Pentester's Copilot, the framework presented in this paper, aims to synthesize the strengths and address the limitations of these existing approaches. It distinguishes itself from purely human-in-the-loop tools like PentestGPT [4] and the Pentest Copilot by Goyal et al. [25] by incorporating a fully Autonomous Agent Mode leveraging the ReAct framework [19]. Unlike task-specific autonomous agents [15], [18] or frameworks focused on particular phases [29], our system provides a unified platform integrating multiple interaction modes (Chat, Assistant, Agent) to cater flexibly to diverse user needs and task complexities within the end-to-end penetration testing lifecycle.

Crucially, Pentester's Copilot introduces a Persistent Memory Layer (`Mem0`/`Supabase`) explicitly designed for personalization and maintaining stateful context across sessions. This addresses a key limitation identified in both assisted and autonomous approaches, enabling the system to learn user preferences and recall crucial information (e.g., user preferred tools, user goals, discovered vulnerabilities, successful commands) from previous interactions. By combining this persistent memory, multimodal input processing, specialized LLM selection, and the ReAct agent architecture, Pentester's Copilot endeavors to provide a more practical, adaptable, and effective AI-driven solution for augmenting and automating real-world penetration testing.

F. Differentiation Summary

Pentester's Copilot's novelty lies in its integrated approach, combining:

- Multiple Interaction Modes: Chat, Assistant, and autonomous Agent modes within one platform.
- Persistent, Personalized Memory: Using `Mem0`/`Supabase` for statefulness across sessions and modes.
- Specialized LLM Selection: Offering distinct models for different task requirements (speed, power, reasoning).
- Offensive Security Focus: Designed and prompted specifically to assist with hacking tasks, aiming for higher utility than restricted general models.
- Human Augmentation: Primarily designed to assist a human pentester, rather than aiming for full, unsupervised end-to-end automation.

III. SYSTEM ARCHITECTURE AND IMPLEMENTATION

This section details the practical implementation of the Pentester's Copilot framework, translating the architectural design into a functional system. We focus on the core technologies, component interactions, data persistence strategies, and key implementation choices that enable the platform's multi-modal interaction, persistent memory, and autonomous agent capabilities, with specific attention to the operational workflows and prompt engineering techniques employed.

A. Core Technologies and Framework

Pentester's Copilot utilizes a modern, modular web architecture selected for performance, scalability, developer experience, and robust AI integration. The frontend user interface is developed using `Next.js 15` and `React 19` with `TypeScript`, leveraging `shadcn/ui` components and `Tailwind CSS`. State management employs `SWR` for data fetching/caching and custom React hooks. The backend is an asynchronous API service built with `Python 3.8+` and the `FastAPI` framework. Orchestration of LLM interactions and agent workflows relies on the LangChain framework [30]. PostgreSQL (hosted on Vercel Neon), managed via the Drizzle ORM, handles structured data. The Vercel AI SDK [31] aids in managing streaming responses and tool definitions.

B. Component Implementation Overview

a) Frontend: Provides interface for user authentication (`next-auth`), chat interactions, multimodal input support (text, file uploads via Vercel Blob), artifacts rendering/editing, and agent control communicates with the backend via APIs.

b) Backend API (FastAPI): Serves as the central orchestration layer. It handles API requests, manages interactions with multiple LLM APIs (Google Gemini, Meta Llama, DeepSeek via TogetherAI using a customProvider abstraction), invokes the LangChain agent, controls the memory layer (`Mem0` and `Supabase`), processes files (Gemini Multimodal), performs web searches (Gemini Search), and persists data to PostgreSQL Storage in Supabase.

c) Persistent Memory Layer (`Mem0` + `Supabase`): Enables personalization and state-fullness. It extracts relevant facts from conversations using a custom-prompted Gemini model, generates embeddings, stores facts/embeddings in Supabase, and retrieves relevant memories via semantic search (RAG) [27] to inject into LLMs Context Window.

d) Database (PostgreSQL + Drizzle): Stores structured data including user profiles, chat histories, messages (with tool calls/results as JSONB), user feedback (votes), and versioned artifacts (documents).

C. Operational Modes and Workflows

Pentester's Copilot operates in distinct modes, each leveraging the core components differently:

1) Chat Mode Operation: The interactive Chat Mode facilitates conversational interaction. Its workflow involves:
   1) User Input: Query submission via the frontend.
   2) Memory Retrieval (RAG): The backend queries the Supabase vector store to retrieve relevant user/session memories based on semantic similarity to the input query.
   3) Prompt Construction: A comprehensive prompt is dynamically assembled, incorporating: (i) A system prompt defining the "Pentester's Copilot" persona (acting as a form of ethical jailbreak [18], [26], [32]), (ii) Retrieved memories for personalization, (iii) Recent chat history for conversational context, and (iv) The current user query.
   4) LLM Interaction: The constructed prompt is sent to the selected LLM API (e.g., `Llama 3.1`, `Gemini 2.0 flash`, `DeepSeek R1`).
   5) Response Generation & Delivery: The LLM generates a response, which is streamed back to the frontend using the Vercel AI SDK [31].
   6) Memory Storage: The interaction turn (query + response) is processed by Mem0 and Gemini to extract and store new facts/embeddings in Supabase, enabling continuous learning.
2) Assistant Mode Operation: This mode provides real-time, interactive guidance. Leveraging the Google Gemini Live API, it processes a stream of screen-sharing data alongside user text/voice input. The backend analyzes the multimodal context (screen content + user query) and interacts with the Gemini LLM to generate step-by- step guidance, debugging assistance, or planning suggestions relevant to the user's immediate task environment (e.g., a CTF challenge interface, a tool's output). A key enhancement implemented involves adding a specific function call within the Gemini Live interaction loop to explicitly request and display the AI's current plan or next steps directly within the user's console or interface, providing transparency and better user assistance during complex procedures.
3) Autonomous Agent Mode Operation: This mode utilizes the LangChain ReAct (Reasoning and Action) framework [19] for autonomous task execution based on high-level user goals.
   1) Goal Definition: The user provides a high-level objective (e.g., "Perform Nmap scan on 192.168.1.1," "Find SQL injection vulnerabilities on example.com").
   2) Reasoning and Planning: The ReAct agent, typically utilizing LLM `Gemini-2.0-flash`, analyzes the goal and breaks it down into a sequence of steps, selecting appropriate tools for each step.
   3) Tool Selection and Execution: The agent iteratively selects and executes tools from its defined toolkit. Our implementation equips the agent with:
      - `ShellTool`: For executing standard shell commands (e.g., `nmap`, `curl`).
      - `PythonREPLTool`: For executing generated Python code snippets.
      - `pinecone_web_hacking_assistant`: A custom RAG tool querying a specialized Pinecone index containing web hacking documentation for technique retrieval.
      - `web_searching`: A custom tool leveraging the Gemini Grounding for general web information gathering.
      - `together_ai_tool`: A planning assistance tool that leverages Mixture of Agents (MoA) [33] for complex strategy formulation.
      - `WebBrowserTaskTool`: A significant integration using the *browser-use* library, allowing the agent to directly interact with web pages (visit URLs, inspect elements, submit forms). This advances beyond prior work [15] that primarily used non- interactive browser libraries like Playwright, enabling the agent to perform complex web-based attacks like SQL injection or XSS autonomously.
   4) Observation and Iteration: The agent parses the output (observation) from the executed tool, updates its internal state, and uses the ReAct framework's reasoning capabilities to determine the next action, looping until the goal is achieved or a stopping condition is met.

D. Prompt Engineering and Safety

Effective prompt engineering is crucial across all modes. As detailed in the Chat Mode workflow (Sec III-C1), system prompts establish the AI's persona ("Pentester's Copilot"), define operational boundaries, and incorporate dynamically retrieved context (memory, history). This persona-based approach [32] is essential for navigating LLM safety filters [18], [26] in the offensive security domain. User prompts are structured to clearly convey the task, while output formats are constrained where necessary (e.g., requesting single commands in Agent

Mode) to ensure reliable execution. The design aims to balance enabling offensive security tasks for ethical purposes with preventing misuse through the specific framing and constraints embedded within the prompts.

IV. EXPERIMENTAL EVALUATION

To empirically evaluate the capability of Large Language Models (LLMs) in assisting offensive security tasks, we designed and executed two distinct experiments based on the project's evaluation section. The first experiment assessed the LLM's effectiveness in an interactive, guided Chat Mode during a standard Capture The Flag (CTF) exercise. The second experiment evaluated the LLM's autonomous Agent Mode capabilities by tasking it with performing a basic pen test on a controlled web application environment. For both experiments, the `Gemini-2.0-flash` model was utilized.

A. Chat Mode Evaluation: Penetration Test Assistance on TryHackMe "Blue"

1) Objective: This experiment aimed to quantify the LLM's ability to provide accurate, context-aware, and timely guidance throughout the phases of a penetration test when interacting directly with a human operator. The focus was on evaluating its capacity to suggest commands, interpret tool output, identify vulnerabilities, and assist in exploitation and post-exploitation activities.
2) Experimental Setup:
   - Target System: The `TryHackMe` "Blue" virtual machine (IP: `10.10.129.1`) was selected as the target. This machine is intentionally configured as a vulnerable Windows 7 system, susceptible to the MS17-010 (EternalBlue) SMB vulnerability. While designed for beginners, it provides a standardized environment to test foundational penetration testing guidance.
   - Interaction Model: A human tester engaged with the `Gemini-2.0-flash` model via a text-based chat interface. The tester provided context, executed suggested commands on an attack machine (`Kali Linux` via Attack- Box), and pasted tool output back to the LLM for analysis and next steps. The complete interaction log served as the primary data source.
3) Procedure and Observations: The penetration test simulation followed standard methodology:
   - Reconnaissance: The LLM was prompted for assistance with initial scanning, given the constraint that ICMP was disabled. It correctly recommended an `Nmap` TCP SYN scan (`nmap -sS -p- -v 10.10.129.1`). Upon receiving the `Nmap` output, the LLM accurately identified the number of open ports under 1000 (4: 135, 139, 445, 3389) and correctly inferred the likely vulnerability as MS17-010 based on the presence of open SMB ports (139, 445).
   - Exploitation: The LLM guided the tester through using the Metasploit Framework. It identified the correct exploit module (`exploit/windows/smb/ms17_010_eternalblue`), instructed setting the `RHOSTS` variable, and suggested the `windows/x64/shell/reverse_tcp` payload as per the CTF instructions. Critically, when the initial exploit attempt failed, the LLM diagnosed the issue from the Metasploit output and the provided `ifconfig` data, correctly identifying that the tester had set `LHOST` to the Docker interface IP (`172.17.0.1`) instead of the required VPN interface IP (`10.10.217.182`). It provided the correct `LHOST` setting, leading to successful exploitation and acquisition of a Meterpreter shell.
   - Privilege Escalation & Post-Exploitation: The LLM provided commands for initial system enumeration (`sysinfo`) and identifying the architecture (`x64`). It suggested using `post/multi/recon/local_exploit_suggester` and correctly identified `exploit/windows/local/bypassuac_injection` as a potential escalation path (though the chat log indicates a direct `getsystem` approach or `shell_to_meterpreter` upgrade was subsequently used, possibly following CTF prompts). It successfully guided the conversion of the initial shell to a full Meterpreter session using `post/multi/manage/shell_to_meterpreter`, including setting the required `SESSION` option. It further instructed on achieving `NT AUTHORITY\SYSTEM` privileges (`getsystem`), process listing (`ps`), and process migration (`migrate <PID>`) for persistence and stability. For data exfiltration, it provided the `hashdump` command, identified the non-default user ("Jon"), extracted the NTLM hash (`ffb43f0de35be4d9917ac0cc8ad57f8d`), and suggested hash cracking tools (`john`, `hashcat`) and command syntax. It assisted in troubleshooting john execution by identifying the need to specify the format (`-format=NT`) and remove the username prefix from the hash file. Finally, it guided the retrieval of the three flags from standard locations (`C:\`, `C:\Windows\System32\config`, and the Administrator's Desktop), including troubleshooting path errors for the Desktop using `getenv USERPROFILE`.
4) Performance Summary: The target system was successfully compromised, and all flags were retrieved within approximately 30 minutes of focused interaction. The LLM demonstrated proficiency in recalling specific commands, interpreting tool output, diagnosing common errors (e.g., incorrect LHOST This was the error from the user part but still the LLM was able to get it correct), and following a structured penetration testing workflow based on the CTF's progression. Its interactive nature allowed for real-time troubleshooting and adaptation.

B. Agent Mode Evaluation: Network Scanning on Local DVWA

1) Objective: This experiment assessed the LLM's ability to function as an autonomous agent, utilizing provided tools to execute a penetration testing task without any human intervention. The specific task was to perform network reconnaissance on a locally hosted web application testbed.
2) Experimental Setup:
   - Target System: A locally hosted instance of the Damn Vulnerable Web Application (DVWA) running on `127.0.0.1`. The underlying stack included MariaDB, accessible on port `3306`.
   - Agent Configuration: An LLM agent, presumably utilizing a framework like LangChain ,was instantiated. The agent was equipped with a `shell_tool`, granting it the capability to execute arbitrary shell commands on the host system where the agent script was run.
   - Task Prompt: The agent was given the instruction: "Do a comprehensive `nmap` scan on this target `127.0.0.1`".
3) Procedure and Observations:
   - Task Interpretation and Tool Selection: The agent correctly interpreted the request and identified the `shell_tool` as the appropriate means to execute the Nmap command.
   - Command Formulation and Execution: The agent formulated the Nmap command `nmap -sV -T4 -A 127.0.0.1`. This command requests service version detection (`-sV`), uses an aggressive timing template (`-T4`), and enables OS detection, version detection, script scanning, and traceroute (`-A`). This represents a reasonable interpretation of a "comprehensive" scan. The agent successfully executed this command via the `shell_tool`.
   - Result Processing and Reporting: The agent captured the raw output from the Nmap scan. Significantly, beyond simply returning the raw data, the agent proceeded to analyze the results and autonomously generated a structured markdown report. This report summarized the findings (open port 3306, MariaDB 11.8.1, OS details), performed a preliminary vulnerability assessment (noting potential MariaDB version vulnerabilities, SSL/TLS configuration issues flagged by Nmap, and the use of `mysql_native_password`), and provided actionable security recommendations (vulnerability scanning, patch management, SSL/TLS review, firewalling, etc.).
4) Performance Summary: The agent successfully executed the assigned reconnaissance task. It demonstrated not only the ability to translate a natural language request into a correct command-line execution but also a more advanced capability to parse the tool's output, contextualize the findings within a security assessment framework, and generate a coherent report with relevant recommendations. This indicates a capacity for autonomous analysis extending beyond simple tool invocation.

C. Discussion of Experiments

The experiments validate the core functionalities of Pentester's Copilot across its different modes. The Chat Mode, augmented by persistent memory and capable LLMs like `Gemini 2.0-flash`, proves effective in providing detailed, context-aware guidance through multi-stage penetration tests, significantly aiding users by interpreting tool outputs and suggesting corrective actions. The Agent Mode demonstrates successful autonomous execution of defined tasks, leveraging the ReAct framework for planning and tool use, and showcasing an ability to synthesize results beyond simple command execution. The persistent memory layer appears crucial for maintaining context in the longer Chat Mode interaction, while the agent's ability to select and correctly parameterize tools like `Nmap` validates the core autonomous workflow. The successful integration and use of LLM (`Gemini` for chat/multimodal and agent reasoning) highlights the benefit of a flexible architecture. While these initial experiments focused on well-defined scenarios (a known vulnerable box and basic reconnaissance), They establish a solid foundation and demonstrate the possible capabilities of this AI-based approach to significantly enhance the efficiency and availability of penetration testing. Further assessments with a wider variety of diverse and complex targets, combined with continued development of the agent's toolset and reasoning capabilities (especially concerning error handling and flexibility), are required.

V. DISCUSSION

The experimental results presented in Section IV validate the potential of Pentester's Copilot as a versatile AI-driven framework for augmenting penetration testing workflows. Our findings indicate that integrating specialized LLMs, persistent memory, and agentic architectures can significantly enhance both guided assistance and task automation in offensive security contexts. The successful guidance provided by the Chat Mode in completing the TryHackMe "Blue" scenario underscores the effectiveness of LLMs like `Gemini 2.0-flash` when appropriately prompted and augmented with context. The system's ability to accurately interpret `Nmap` outputs, identify the correct MS17-010 vulnerability, guide the Metasploit exploitation process, and crucially, provide effective troubleshooting for configuration errors (like the incorrect `LHOST` setting) highlights the practical utility of AI assistance. The persistent memory layer, operating implicitly via RAG, likely contributed to maintaining context throughout this multi-stage engagement, preventing the need for repetitive information provision by the user, a common limitation in stateless LLM interactions. The Agent Mode's autonomous execution of the DVWA reconnaissance task further demonstrates the viability of the LangChain and ReAct framework for penetration testing automation. The agent's capability to not only select the appropriate tool (`ShellTool`) and formulate the correct command (`nmap -sV -T4 -A ...`) but also to subsequently synthesize the raw output into a structured, insightful Security Assessment Report represents a significant step beyond simple command execution. This indicates potential for agents to handle not just execution but also preliminary analysis and reporting. The integration of the `WebBrowserTaskTool`, leveraging the *browser-use* library, marks a notable advancement over prior agent implementations relying solely on browser automation libraries like Playwright [15], opening possibilities for automating more complex web interactions, including potential SQL injection or XSS exploitation attempts. A key differentiator of Pentester's Copilot is the persistent memory layer implemented with `Mem0` and `Supabase`. This architecture directly addresses the critical limitation of LLM statelessness, enabling personalization based on user preferences (e.g., preferred tools) and maintaining crucial context across complex, potentially multi-session, penetration tests. This stateful capability is largely absent in many existing LLM- assistant or autonomous agent frameworks for pentesting [4], [7], [14], [29]. Despite these promising results, Pentester's Copilot, in its current form, has limitations. The evaluation scope, while representative, covers a limited set of target environments and vulnerabilities. The effectiveness is inherently tied to the capabilities of the underlying LLMs; performance gaps between models exist [15], and handling truly novel or zero- day exploits without prior relevant information remains a significant challenge for current AI [16], [17]. The reliability of integrated tools and the agent's ability to gracefully handle unexpected tool errors or environmental changes require further investigation and hardening. Furthermore, the security implications of granting autonomous agents shell access, even within intended test environments, necessitate robust safeguards not fully implemented in this research prototype. Interaction with complex Graphical User Interfaces (GUIs) also remains an open challenge, primarily addressed conceptually via the Assistant Mode. Compared to prior work, Pentester's Copilot offers a more integrated approach than purely assistive tools like `PentestGPT` [4] by incorporating full autonomy. It provides a broader workflow perspective and incorporates persistent memory, unlike many task-specific autonomous agents [15], [18], [29]. Its multi-modal capabilities and distinct interaction modes offer greater flexibility than frameworks focused solely on a single interaction paradigm.

VI. CONCLUSION

This paper presented Pentester's Copilot, an AI-powered framework designed to enhance the efficiency and effectiveness of penetration testing through intelligent assistance and automation. By integrating multiple interaction modes (Chat, Assistant, Agent), leveraging specialized LLMs, incorporating a novel persistent memory layer via RAG for personalization and statefulness, and utilizing the LangChain and ReAct framework for autonomous task execution, our system addresses key limitations of both traditional manual testing and existing AI- driven approaches. Our experimental evaluations demonstrated the practical utility of the framework. The Chat Mode successfully guided a user through a complete CTF scenario, providing accurate tool usage instructions and effective troubleshooting. The Agent Mode autonomously executed reconnaissance tasks, synthesizing results into comprehensive reports and showcasing the potential for automating complex web interactions via integrated browser tools. These results highlight the significant potential of AI, when architected thoughtfully with features like persistent memory and agentic capabilities, to streamline penetration testing workflows, accelerate vulnerability discovery, and empower cybersecurity professionals. Pentester's Copilot represents a tangible step towards realizing this potential, offering a versatile and context-aware platform for offensive security tasks.

VII. FUTURE WORK

Building upon the current Pentester's Copilot framework, several promising avenues for future research and development emerge, aimed at enhancing its capabilities, robustness, and practical applicability:

1) Unified Agent-Chat Interaction: Explore architectures that seamlessly integrate the conversational capabilities of the Chat Mode with the autonomous execution of the Agent Mode. This could allow users to initiate autonomous tasks within a chat, receive progress updates, and intervene or provide guidance conversationally, blending interactive assistance with automation.
2) End-to-End Autonomous Penetration Testing: Significantly expand the Agent Mode's capabilities to cover the full penetration testing lifecycle autonomously. This requires integrating a wider range of tools (beyond `Nmap`, shell, Python, and basic browser interaction), enhancing the agent's planning and reasoning abilities to handle more complex multi-stage attacks (including lateral movement, advanced privilege escalation, and data exfiltration), and improving its adaptability to diverse target environments.
3) Expanded Vulnerability and Technique Knowledge: Continuously enrich the knowledge base accessible via RAG (e.g., the Pinecone index) with more CVE details, exploit code, diverse attack patterns (TTPs), and documentation for a broader array of security tools and platforms.
4) Enhanced Adaptive Reasoning: Improve the agent's ability to handle unexpected errors, adapt to dynamic target environments that change during testing, and reason about novel situations or vulnerabilities not explicitly present in its knowledge base. This could involve integrating more sophisticated planning algorithms or meta-cognitive reflection capabilities within the agent loop.
5) Robust Agent Security and Sandboxing: Implement robust security measures for the Agent Mode's execution capabilities, particularly for tools like `ShellTool`. This includes developing effective sandboxing environments, implementing fine-grained permission controls, and potentially incorporating mandatory user confirmation steps for high-risk actions [34], [35].
6) Advanced Memory and Learning: Investigate more sophisticated long-term memory architectures, potentially integrating knowledge graphs or exploring continual learning techniques to allow the agent to improve its strategies based on experience across multiple penetration tests.

7) Comprehensive Benchmarking: Evaluate Pentester's Copilot against a wider range of benchmarks, including more complex and diverse vulnerable machines (e.g., from HackTheBox, VulnHub) and compare its performance quantitatively against other state-of-the-art AI- driven penetration testing tools [21], [36].
8) Ethical Framework Development: Continue to refine and formalize ethical guidelines and responsible usage protocols for AI-powered offensive security tools like Pentester's Copilot [37], [38].

REFERENCES

[1] J. Zhang, H. Bu, H. Wen, Y. Chen, L. Li, and H. Zhu, "When LLMs meet cybersecurity: A systematic literature review," arXiv preprint arXiv:2405.04648, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:269604648.

[2] F. Nourmohammadzadeh Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, and C. Meinel, "Large language models in cybersecurity: State-of-the-art," Social Science Research Network, 2024, available: https://api.semanticscholar.org/CorpusID:267406465.

[3] I. Hasanov, S. Virtanen, A. Hakkala, and J. Isoaho, "Application of large language models in cybersecurity: A systematic literature review," *Computers & Security*, vol. 145, p. 103974, 2024, available: https://api.semanticscholar.org/CorpusID:274286927.

[4] 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 *USENIX Security Symposium*, 2024, available: https://www.usenix.org/conference/usenixsecurity24/presentation/deng.

[5] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From LLMs to LLM-based agents for software engineering: A survey of current, challenges and future," arXiv preprint arXiv:2407.08841, 2024, available: https://api.semanticscholar.org/CorpusId:271709396.

[6] H. Xu, S. Wang, N. Li, K. Wang, Y. Zhao, K. Chen, T. Yu, Y. Liu, and H. Wang, "LLMs for cyber security: A systematic literature review," arXiv preprint arXiv:2405.05535, 2024.

[7] H. Kong, D. Hu, J. Ge, L. Li, T. Li, and B. Wu, "VulnBot: Autonomous penetration testing for a multi-agent collaborative framework," arXiv preprint arXiv:2501.13411, 2025, available: https://api.semanticscholar.org/CorpusID:275820465.

[8] D. Pratama, N. Suryanto, A. A. Adiputra, T.-T.-H. Le, A. Y. Kadiptya, M. Iqbal, and H. Kim, "CIPHER: Cybersecurity intelligent penetration-testing helper for ethical researcher," in *Italian National Conference on Sensors*, 2024.

[9] S. Lermen, A. Kao, A. Vishwanath, B. Schneier, and F. Heiding, "Evaluating large language models' capability to launch fully automated spear phishing campaigns: Validated on human subjects," arXiv preprint arXiv:2412.01976, 2024, available: https://api.semanticscholar.org/CorpusID:274437751.

[10] P. Liu, X. Dai, Z. Yeh, and P. Tseng, "Using LLMs to automate threat intelligence analysis workflows in security operation centers," arXiv preprint arXiv:2407.09222, 2024, available: https://api.semanticscholar.org/CorpusID:271270843.

[11] S. Nakatani, "RapidPen: Fully automated ip-to-shell penetration testing with LLM-based agents," arXiv preprint arXiv:2502.16730, 2025, available: https://api.semanticscholar.org/CorpusID:276575021.

[12] B. Wu, G. Chen, K. Chen, X. Shang, J. Han, Y. He, W. Zhang, and N. H. Yu, "AutoPT: How far are we from the end2end automated web penetration testing?" arXiv preprint arXiv:2410.16290, 2024, available: https://api.semanticscholar.org/CorpusID:273812360.

[13] I. Alshehri, A. Alshehri, A. Almalki, M. Bamardouf, and A. Akbar, "BreachSeek: A multi-agent automated penetration tester," arXiv preprint arXiv:2408.09527, 2024, available: https://api.semanticscholar.org/CorpusID:272463712.

[14] L. Muzsai, D. Imolai, and A. Lukács, "HackSynth: LLM agent and evaluation framework for autonomous penetration testing," arXiv preprint arXiv:2412.01778, 2024, available: https://api.semanticscholar.org/CorpusID:274437431.

[15] R. Fang, A. Gupta, R. Bindu, Q. Zhan, and D. Kang, "LLM agents can autonomously hack websites," arXiv preprint arXiv:2402.06664, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:267627588.

[16] R. Fang, R. Bindu, A. Gupta, and D. Kang, "LLM agents can autonomously exploit one-day vulnerabilities," arXiv preprint arXiv:2404.08144, 2024, available: https://api.semanticscholar.org/CorpusID:269137506.

[17] R. Fang, R. Bindu, A. Gupta, Q. Zhan, and D. Kang, "Teams of LLM agents can exploit zero-day vulnerabilities," arXiv preprint arXiv:2406.01637, 2024, available: https://api.semanticscholar.org/CorpusID:270226495.

[18] A. Happe and J. Cito, "Can LLMs hack enterprise networks? Autonomous assumed breach penetration-testing active directory networks," arXiv preprint arXiv:2502.04227, 2025, available: https://api.semanticscholar.org/CorpusID:276161099.

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[20] A. Handa, A. Sharma, and S. K. Shukla, "Machine learning in cybersecurity: A review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 4, p. e1306, 2019.

[21] I. Isozaki, M. Shrestha, R. Console, and E. Kim, "Towards automated penetration testing: Introducing LLM benchmark, analysis, and improvements," arXiv preprint arXiv:2410.17141, 2024, available: https://api.semanticscholar.org/CorpusID:273507775.

[22] M. A. Ferrag, A. Mechri, A. A. Battah, B. Cherif, F. Alwahedi, and N. Tihanyi, "Generative AI and large language models for cyber security: All insights you need," arXiv preprint arXiv:2405.06742, 2024.

[23] K. Shashwat, F. Hahn, X. Ou, D. Goldgof, L. Hall, J. Ligatti, S. R. Rajgopalan, and A. Z. Tabari, "A preliminary study on using large language models in software pentesting," arXiv preprint arXiv:2402.00933, 2024, available: https://api.semanticscholar.org/CorpusID:267335021.

[24] J. Wang, A. Arya, D. Liu, D. Berzin, Y. Zhang, A. Roychoudhury, M. Mirchev, and O. Chang, "Fixing security vulnerabilities with AI in OSS-Fuzz," arXiv preprint arXiv:2410.18933, 2024.

[25] D. Goyal, S. Subramanian, and A. Peela, "Hacking, the lazy way: LLM augmented pentesting," arXiv preprint arXiv:2409.09493, 2024, available: https://api.semanticscholar.org/CorpusID:272689776.

[26] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 24 824–24 837.

[27] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[28] A. Happe and J. Cito, "Getting pwn'd by AI: Penetration testing with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2023, pp. 2082–2086.

[29] J. Xu, J. W. Stokes, G. McDonald, X. Bai, D. Marshall, S. Wang, A. Swaminathan, and Z. Li, "AutoAttacker: A large language model guided system to implement automatic cyber-attacks," arXiv preprint arXiv:2403.01038, 2024, available: https://api.semanticscholar.org/CorpusID:268230530.

[30] LangChain, "Langchain documentation," https://docs.langchain.com/, accessed: 2025-04-24.

[31] Vercel, "Vercel ai chat sdk," https://github.com/vercel/ai-chatbot, 2024, accessed: 2025-04-24.

[32] Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, K. Wang, and Y. Liu, "Jailbreaking ChatGPT via prompt engineering: An empirical study," arXiv preprint arXiv:2305.13860, 2023.

[33] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou, "Mixture-of-agents enhances large language model capabilities," arXiv preprint arXiv:2406.04692, 2024.

[34] Z. Wang, Y. Yao, Y. Zhang, H. Wang, C. Zhan, J. Huang, H. Zhang, and K. Mei, "Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents," arXiv preprint arXiv:2409.17092, 2024.

[35] M. Balunović, F. S. Tramèr, M. Fischer, L. Beurer-Kellner, E. Debenedetti, and J. Zhang, "AgentDojo: A dynamic environment to evaluate attacks and defenses for LLM agents," arXiv preprint arXiv:2406.13031, 2024.

[36] L. Gioacchini, M. Mellia, I. Drago, A. Delsanto, G. Siracusano, and R. Bifulco, "AutoPenBench: Benchmarking generative agents for penetration testing," arXiv preprint arXiv:2410.03225, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:273162799.

[37] D. Ayzenshteyn, R. Weiss, and Y. Mirsky, "The best defense is a good offense: Countering LLM-powered cyberattacks," arXiv preprint arXiv:2410.16985, 2024.

[38] W. Zhou, D. Ye, T. Zhu, B. Liu, X. Guo, M. Ding, S. Wang, and P. S. Yu, "Unique security and privacy threats of large language model: A comprehensive survey," arXiv preprint arXiv:2406.04847, 2024.