Systematic Capability Benchmarking of Frontier Large Language Models for Offensive Cyber Tasks

Tyler H. Merves[1], Michael H. Conaway[1], Joseph M. Escobar[1], Hakan T. Otal[2], and Unal Tatar[1]
*Corresponding Author: utatar@albany.edu

[1] Department of Cybersecurity
[2] Department of Information Sciences and Technology
College of Emergency Preparedness, Homeland Security, and Cybersecurity
University at Albany, SUNY
Albany, NY, USA

***Abstract*—We present, to our knowledge, the most comprehensive cross-model evaluation of LLM agents on offensive cybersecurity tasks, benchmarking 10 frontier models from 7 providers on all 200 challenges of the NYU CTF Bench. Building on the D-CIPHER multi-agent framework, we extend it with multi-provider backend support, a custom Kali Linux environment with over 100 pre-installed penetration testing tools, and runtime tool-discovery agents. Through a controlled factorial study, we find that the Kali Linux environment yields a $+9.5$ percentage-point improvement over Ubuntu, while auto-prompting and category-specific tips often degrade performance in well-equipped environments. Among models, Claude 4.5 Opus achieves the highest solve rate (59%), followed by Gemini 3 Pro (52%), with Gemini 3 Flash offering the best cost-efficiency at $0.05 per solve. Asymmetric planner/executor model assignments provide no meaningful benefit while coherent same-model configurations consistently outperform mixed-tier pairings. Our results indicate that environment tooling and model selection emerge as the strongest drivers of performance, whereas prompt engineering interventions show diminishing or negative returns in well-equipped environments. Reported performance reflects both model reasoning ability and compatibility with agent tooling and API integration.**

***Index Terms*—large language models, capture the flag, cybersecurity benchmarking, multi-agent systems, penetration testing, LLM evaluation, offensive security**

I. Introduction

Rapid advances in large language models (LLMs) have extended their capabilities into complex technical domains, including cybersecurity [1]. LLM-powered agents can now discover vulnerabilities [2], craft exploits, and conduct multi-step penetration testing with limited human oversight [3]. These capabilities raise dual-use concerns, as the same models that automate defensive workflows may also lower the barrier for offensive cyber operations [4]. Systematic evaluation methodologies are therefore needed to quantify offensive cyber performance under controlled conditions.

Capture The Flag (CTF) competitions serve as a standardized proxy for measuring offensive security capability, requiring cryptanalysis, reverse engineering, binary exploitation, web application attacks, and forensic analysis within sandboxed environments. The NYU CTF Bench [5] formalizes this approach with 200 challenges spanning six categories. Building on this benchmark, increasingly sophisticated agent architectures have been proposed: EnIGMA [6] added interactive tool interfaces, D-CIPHER [7] introduced a multi-agent Planner–Executor architecture, and CRAKEN [8] augmented D-CIPHER with retrieval-augmented generation. Yet comparisons across these studies remain difficult because they differ in model selection, environment, prompt strategy, and evaluation protocol.

To our knowledge, no prior study systematically varies these engineering dimensions on the same benchmark under controlled conditions, making it unclear which factors truly drive performance. This paper addresses that gap through a comprehensive study on the full NYU CTF Bench. We extend D-CIPHER with multi-provider backend support, a custom Kali Linux environment with over 100 pre-installed security tools, and runtime tool-discovery agents. Our contributions are:

1) A systematic comparison of 10 frontier LLMs from 7 providers on 200 CTF challenges, one of the most comprehensive cross-model evaluations on this benchmark to date.
2) A controlled ablation across environment (Ubuntu vs. Kali), prompt strategy (generic vs. tips), and auto-prompting, showing that Kali Linux yields $+9.5$ percentage-point (pp) improvement while auto-prompting generally hurts.
3) A cost-performance analysis revealing that Claude 4.5 Opus achieves the highest solve rate (59%) while Gemini 3 Flash offers the best cost-efficiency at $0.05 per solve.
4) Reproducible experimental infrastructure, including a Kali Linux Docker image with 100+ security tools, runtime tool-discovery agents, a parallel experiment runner, and an automated result parsing pipeline. Source code will be made available upon publication to enable replication and extension of our findings. [1]

[1] Source code available at: https://github.com/TATAR-LAB/ctf-agents

II. Related Work

A. CTF Benchmarks

Several benchmarks evaluate LLM capabilities on offensive security tasks. The NYU CTF Bench [5] provides 200 challenges from the annual CSAW CTF competitions (2016–2024) across six categories with Dockerized environments and ground-truth flags; in its initial evaluation, GPT-4 achieved only single-digit solve rates [9]. Cybench [10] offers 90 challenges from recent competitions and introduces subtask decomposition for partial-credit evaluation. CTFusion [11] demonstrates that data contamination inflates results on static benchmarks: an agent with web search nearly doubled its solve rate by retrieving published writeups, while performance on live competitions dropped by roughly half. Although absolute solve rates should be interpreted cautiously, relative comparisons between configurations are likely to remain valid, as all models are evaluated under identical conditions. BountyBench [12] moves beyond CTFs to evaluate agents on real-world bug bounties, finding that defensive tasks are substantially easier than offensive ones. OCCULT [4] complements these CTF-centric benchmarks by evaluating LLMs against realistic offensive cyber operation tactics used by modern threat actors, finding significant recent advancement in AI-enabled cyber risk. However, differences in training data exposure and tool integration across providers may still influence observed performance.

B. LLM Agents for Offensive Security

PentestGPT [3] was among the first to apply LLMs to multi-step offensive security tasks. Its key finding that context loss over long interaction sequences is the primary bottleneck motivated subsequent multi-agent designs. EnIGMA [6] builds on SWE-agent with Interactive Agent Tools (IATs) for operating interactive programs such as debuggers and server connections, achieving 3× higher solve rates than the NYU CTF Bench baseline. EnIGMA also identified a "solil-oquizing" phenomenon where agents hallucinate observations from memorized training data, and found that most successful solutions occur within the first 20 steps, which informs our timeout design.

D-CIPHER [7] introduces the hierarchical Planner–Executor architecture that our work builds upon. By separating high-level strategy from low-level task execution, D-CIPHER achieved state-of-the-art results across NYU CTF Bench, Cybench, and HackTheBox, reporting a 6% improvement over single-agent baselines and a 3% gain from auto-prompting with Claude 3.5 Sonnet. CRAKEN [8] extends D-CIPHER with retrieval-augmented generation over CTF writeups and attack payloads, achieving 22% on NYU CTF Bench, notably the same as D-CIPHER, indicating that RAG augmentation did not improve performance on this benchmark.

Our work differs in scope: rather than proposing a new architectural component, we conduct the first controlled evaluation that systematically varies environment, prompting, model selection, and architecture assignment across 10 frontier LLMs on the full NYU CTF Bench.

III. Methodology

We extend the D-CIPHER multi-agent framework [7] with new model backends, execution environments, and tooling, then conduct a systematic evaluation across four research questions on the NYU CTF Bench [5].

A. D-CIPHER Framework

D-CIPHER is a multi-agent system built on a hierarchical Planner–Executor architecture designed for solving CTF challenges [7]. Mirroring collaborative CTF teams, a *Planner* agent handles high-level problem decomposition, reasoning about attack strategies, and delegating sub-tasks to one or more *Executor* agents. Each Executor carries out specific low-level operations (running shell commands, invoking security tools, generating and executing code, and interpreting outputs) and reports results back to the Planner for iterative refinement. This separation of concerns breaks complex challenges into manageable steps, increasing solving efficiency over monolithic single-agent approaches.

D-CIPHER also includes an optional *Auto-Prompter* agent that runs before the Planner–Executor loop. The Auto-Prompter serves as a foundational step to increase the likelihood that the system follows a productive solution path. It explores the challenge's Docker environment (examining file structures, identifying available tools, and reading challenge descriptions), then generates a tailored initial prompt intended to steer the system toward productive attack paths. The original D-CIPHER study reported that auto-prompting improved solve rates by approximately 3% with Claude 3.5 Sonnet [7]; however, as we discuss in Section IV, our experiments with Gemini 3 Pro yield different conclusions about its efficacy.

Additionally, D-CIPHER leverages the categorical structure of the NYU CTF Bench by implementing specialized *tips* for each challenge category. These tips provide actionable guidance for approaching challenges using common attack paths of each category, such as suggesting the use of `pwntools` for binary exploitation or recommending frequency analysis for cryptography challenges.

Extensions to D-CIPHER. The original D-CIPHER framework utilizes an Ubuntu environment with a limited amount of pre-installed cybersecurity tools and supports only OpenAI-formatted API endpoints, limiting both tooling and model selection. We introduce three major extensions (Figure 1):

1) **Multi-provider backend support.** We added Vertex AI, OpenRouter, and Ollama backends, increasing the number of supported model providers. This enables systematic cross-model comparison across proprietary and open-source LLMs within a unified framework.
2) **Kali Linux execution environment.** We built a custom Docker image based on Kali Linux with over 100 pre-installed penetration testing tools (e.g., `nmap`, `sqlmap`, `john`, `hashcat`, `binwalk`, `volatility`) and a structured documentation file (`commands_documentation.csv`) containing usage descriptions for all available commands.

3) **Tool discovery agents.** To ensure agents can leverage the expanded Kali toolset, we created two new tools that are available exclusively in the Kali environment: `ListCommandsTool`, which enumerates all available commands with brief descriptions, and `LookupCommandTool`, which queries compiled documentation for individual commands. Together, these tools reduce tool-discovery friction by allowing agents to first identify relevant commands and then verify their usage before invocation. The performance improvement observed with the Kali environment (Section IV-A) reflects the combined effect of this expanded toolset and the tool discovery capabilities.

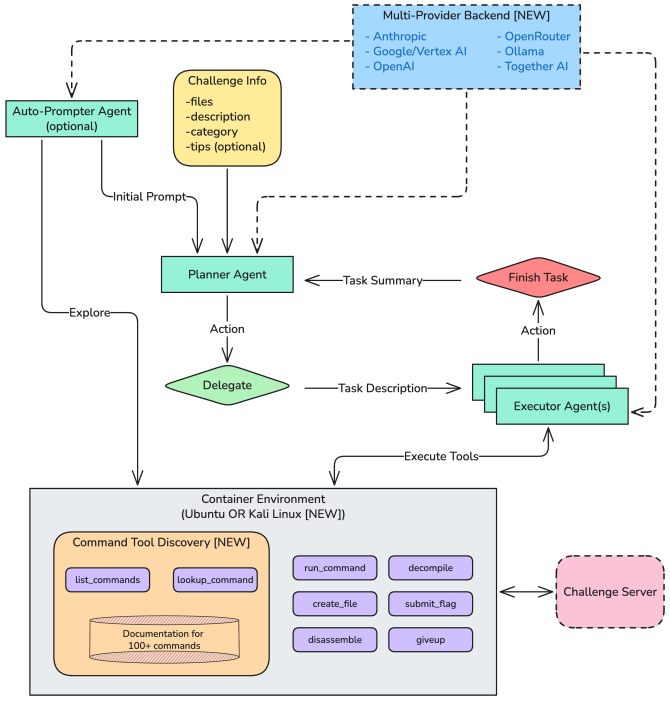

Fig. 1. Overview of the extended D-CIPHER multi-agent architecture.

B. Benchmark Dataset

We evaluate on the NYU CTF Bench [5], a dataset of 200 Capture The Flag challenges sourced from the Cyber Security Awareness Week (CSAW) CTF competitions hosted annually at New York University from 2016 to 2024. The challenges span six categories: cryptography (52), reverse engineering (51), binary exploitation (39), miscellaneous (24), web (19), and forensics (15). Difficulty ranges from beginner to advanced, reflecting a broad spectrum of offensive security tasks encountered by cybersecurity professionals, from web application attacks to low-level binary exploitation. Each challenge includes source files for initialization, Docker configurations for reproducible environmental setup, and a ground-truth flag that enables automated verification of candidate solutions.

C. Experimental Design

We address four research questions through controlled experiments on the full 200-challenge set. Unless otherwise noted, experiments use Gemini 3 Pro (accessed via Vertex AI) as the default model. All agents share the following hyperparameters: temperature T=1.0 (following the D-CIPHER recommendation for optimal performance [7]), 30 planner rounds (100 executor rounds), a nominal $5.00 per-challenge cost limit (never reached in practice), and a 10-minute wall-clock timeout. The timeout reflects findings from D-CIPHER [7] and EnIGMA [6] that most successful solves complete within the first few minutes, with longer runs exhibiting diminishing returns and predominantly ending in failure; a 10-minute cutoff therefore captures the empirically observed productive window while keeping costs tractable across thousands of runs. All experiments report the following metrics: solve rate (number of challenges solved out of 200), total API cost, cost per solved challenge, and per-category solve rates.

1) RQ1 & RQ2 (Environment and Prompt Engineering): RQ1 and RQ2 both address the effectiveness of the D-CIPHER framework with varying configurations.

RQ1: Does enhanced security tooling improve LLM performance on CTF challenges? We hypothesize that the richer tooling and documentation-retrieval capabilities of the Kali environment (Section III-A) increase solve rates by reducing tool-discovery friction and enabling more sophisticated attack strategies.

RQ2: How do prompting strategies affect performance across different environments? We examine whether category-specific tips and auto-prompting generalize to the Kali Linux environment, given that D-CIPHER's original 3% gain from tips was observed with Claude 3.5 Sonnet on Ubuntu [7].

To address both questions jointly, we conduct a controlled ablation across three binary factors:

- **Environment**: Ubuntu 22.04 (base Docker image with standard tools such as `gdb`, `radare2`, `pwntools`, and `angr`) versus Kali Linux (custom Docker image with 100+ penetration testing tools and runtime-queryable documentation).
- **Prompt strategy**: Generic prompts (minimal task instructions) versus category-specific tips (actionable guidance tailored to each challenge category).
- **Auto-prompting**: Off (planner receives only the static prompt) versus On (auto-prompter explores challenge files and generates a tailored initial prompt before the solve attempt begins).

The combination of these three binary factors yields eight configurations, each systematically run on all 200 challenges with Gemini 3 Pro, for a total of 1,600 challenge runs.

2) RQ3 (Model Selection): Which LLMs perform best on offensive cyber tasks? We hypothesize that larger frontier models tend to outperform smaller and open-source models due to superior tool-use, multi-turn reasoning, and instruction-following capabilities.

Using the best configuration identified in RQ1/RQ2 (Kali + generic prompts, no auto-prompting), we compared ten models

from seven providers, spanning several specialization types:

- **Frontier:** Claude 4.5 Opus (Anthropic), Gemini 3 Pro (Google), GPT-5.2 (OpenAI)
- **Provider-flagship:** GLM-5 (Z-AI)
- **Small/efficient:** Gemini 3 Flash (Google)
- **Code-specialized:** GPT-5.2-Codex (OpenAI)
- **Reasoning:** DeepSeek-R1 (DeepSeek)
- **Open-source:** DeepSeek-V3 (DeepSeek), Llama 3.3 70B (Meta), Qwen 3.5 397B-A17B (Alibaba)

To ensure a fair comparison, all models share the identical hyperparameters, toolset, and timeout described above.

3) RQ4 (Multi-Agent Architecture): Does asymmetric model assignment across planner and executor roles improve cost-performance over symmetric configurations? We hypothesize that using a larger, more capable model for the Planner and a smaller, faster model for the Executor would achieve better cost-performance balance than a uniform model assignment. This assumes that planning demands high-level strategic reasoning, whereas execution is more mechanical, involving tool calls and command translation.

We assessed two symmetric configurations (Gemini 3 Pro for both roles, and Gemini 3 Flash for both roles) and two asymmetric configurations (Pro Planner + Flash Executor, and Flash Planner + Pro Executor), all under the Kali + generic prompt setup.

IV. Experimental Results

We present results for each of the four research questions on the full 200-challenge NYU CTF Bench.

A. Impact of Environment and Prompt Strategy

Figure 2 summarizes the results across all eight configurations (Environment $\times$ Prompts $\times$ AutoPrompt), all run with Gemini 3 Pro.

Kali Linux consistently outperforms Ubuntu under matched conditions. Under identical prompt settings (Generic, no AutoPrompt), Kali achieves 52.0% (104/200) versus 42.5% (85/200) for Ubuntu, a gain of $+9.5$ pp. The advantage is especially pronounced in forensics (46.7% vs. 20.0%) and reverse engineering (58.8% vs. 49.0%), categories that depend heavily on specialized tool availability.

The interaction between environment and prompt strategy is noteworthy. Under Kali, generic prompts outperform category-specific tips (52.0% vs. 40.0%), whereas under Ubuntu the pattern reverses (42.5% vs. 44.5%). One possible explanation is that the richly tooled Kali environment already encodes implicit guidance; layering explicit tips over-constrains the agent's exploration. On Ubuntu, the sparser environment means tips compensate by steering the agent toward appropriate techniques.

Auto-prompting, in which the agent explores challenge files and generates a tailored prompt before solving, generally reduces performance across most configurations. Under Kali + Generic, AutoPrompt reduces the solve rate from 52.0% to 39.5% (-12.5 pp); under Ubuntu + Tips, the drop is even sharper: 44.5% to 16.0% (-28.5 pp). One exception stands out: under Kali + Tips, AutoPrompt raises the rate from 40.0% to 48.5% ($+8.5$ pp), suggesting it can partially recover a suboptimal base strategy. We hypothesize that the additional exploration phase typically consumes budget without producing actionable insight, and may introduce misleading preliminary analysis.

The best overall configuration (Kali + Generic without AutoPrompt) costs $44.23 total. The most expensive configuration (Kali + Tips + AutoPrompt, $66.12) achieves only the second-best solve rate (48.5%), reinforcing that simpler configurations can be more effective. Since this ablation uses only Gemini 3 Pro, the magnitude of these effects may vary with other models; however, the directional findings (environment over prompting) are consistent with the broader patterns observed in RQ3.

Solve Rates by Experiment Configuration and Challenge Category

OS	Tips	AP	Crypto	Forensics	Misc	Pwn	Reverse	Web	Overall
Ubuntu	✗	✗	44.2%	20.0%	62.5%	30.8%	49.0%	36.8%	42.5% 85/200
Ubuntu	✗	✓	38.5%	6.7%	37.5%	30.8%	21.6%	42.1%	30.5% 61/200
Ubuntu	✓	✗	50.0%	40.0%	62.5%	33.3%	43.1%	36.8%	44.5% 89/200
Ubuntu	✓	✓	17.3%	6.7%	33.3%	17.9%	5.9%	21.1%	16.0% 32/200
Kali	✗	✗	53.8%	46.7%	70.8%	33.3%	58.8%	47.4%	52.0% 104/200
Kali	✗	✓	44.2%	33.3%	54.2%	33.3%	33.3%	42.1%	39.5% 79/200
Kali	✓	✗	36.5%	40.0%	41.7%	28.2%	51.0%	42.1%	40.0% 80/200
Kali	✓	✓	59.6%	33.3%	62.5%	30.8%	51.0%	42.1%	48.5% 97/200

Fig. 2. Solve rates by configuration and challenge category (RQ1+RQ2).

B. Model Selection

We benchmark ten LLMs from seven providers under the best RQ1/RQ2 configuration (Kali + Generic, no AutoPrompt). Table I reports solve rates, total costs, and cost-per-solve; Figure 4 visualizes the cost–performance tradeoff.

TABLE I
Model benchmark on NYU CTF Bench
(200 Challenges on Kali with Generic prompts)

Model	Provider	Solv.	Rate	Cost	$/Solv.
Claude 4.5 Opus	Anthropic	118	59.0%	249.92	2.12
Gemini 3 Pro	Google	104	52.0%	44.23	0.43
Gemini 3 Flash	Google	54	27.0%	2.69	0.05
GLM-5	Z-AI	39	19.5%	22.09	0.57
GPT-5.2-Codex	OpenAI	36	18.0%	9.23	0.26
GPT-5.2	OpenAI	27	13.5%	7.47	0.28
DeepSeek-V3	DeepSeek	13	6.5%	0.19	0.01
DeepSeek-R1	DeepSeek	8	4.0%	0.13	0.02
Qwen 3.5 397B-A17B	Alibaba	7	3.5%	1.39	0.20
Llama 3.3 70B	Meta	5	2.5%	0.03	0.01

For context, prior state-of-the-art results on NYU CTF Bench were 22% by D-CIPHER [7] and 22% by CRAKEN [8], both using Claude 3.5 Sonnet, while the original benchmark evaluation with GPT-4 achieved only single-digit solve rates [5]. Our best configurations exceed these

baselines; however, direct comparison is limited by generational model differences. Notably, even our Ubuntu baseline with Gemini 3 Pro (42.5%) exceeds prior results, suggesting that the improvement is driven primarily by advances in model capability rather than our framework extensions alone.

Claude 4.5 Opus leads at 59.0% (118/200), followed by Gemini 3 Pro at 52.0% (104/200). These two frontier models form a clear top tier, each solving nearly twice as many challenges as the third-ranked Gemini 3 Flash (27.0%). OpenAI's models cluster in the middle tier: GPT-5.2-Codex (18.0%) outperforms base GPT-5.2 (13.5%), suggesting code specialization provides a meaningful advantage. Open-source and smaller models achieve lower solve rates: GLM-5 (19.5%) attains moderate accuracy but at higher cost ($0.57/solve), while DeepSeek-V3 (6.5%), DeepSeek-R1 (4.0%), Qwen 3.5 (3.5%), and Llama 3.3 70B (2.5%) all fall below 7%. These lower results may reflect a combination of weaker tool-use capabilities, less robust multi-turn agentic reasoning, and differences in how well each model integrates with the agent framework's tool-calling interface.

The DeepSeek pair provides an instructive comparison: the reasoning-specialized R1 (4.0%) does not outperform the standard V3 (6.5%). While both models achieve low solve rates overall, this pattern is consistent with the observation that agentic settings, where success depends on iterative tool calls and environment interaction, may reward robust tool-use over pure chain-of-thought reasoning. However, both models' low scores make it difficult to draw strong conclusions from this comparison alone.

Per-category analysis (Figure 3) reveals that the *misc* category is the easiest across nearly all models, while *pwn* is consistently the hardest. Miscellaneous challenges often involve pattern recognition, encoding, or scripting tasks that align well with LLM strengths, whereas binary exploitation requires precise memory layout reasoning and multi-step exploit construction that current models struggle to sustain. Claude 4.5 Opus outperforms Gemini 3 Pro in every category, with the largest gains in pwn (41.0% vs. 33.3%) and web (52.6% vs. 47.4%).

Gemini 3 Flash occupies a favorable position on the cost–performance frontier (Figure 4): at $0.05/solve, it is the most cost-efficient model tested, solving over half of what Pro achieves at 6% of the cost. Claude 4.5 Opus achieves the highest solve rate but at the highest total cost ($249.92, $2.12/solve), roughly 5× more expensive per solve than Gemini 3 Pro ($44.23, $0.43/solve). For cost-sensitive deployments, Gemini 3 Flash offers the best tradeoff, while Claude 4.5 Opus is the best option when accuracy is the primary objective regardless of cost.

C. Multi-Agent Architecture

Table II presents results for four planner/executor configurations using Gemini 3 Pro and Flash, all under Kali + Generic.

The homogeneous Pro configuration achieves 52.0%, nearly doubling the next-best result. The asymmetric Pro Planner + Flash Executor yields only +1.5 pp over Flash-only (28.5% vs. 27.0%) while increasing cost by 45%.

Notably, the reverse configuration (Flash Planner + Pro Executor) performs worse than Flash-only (23.5% vs. 27.0%), despite deploying the stronger model as executor. This challenges the intuition that executor quality is the bottleneck; instead, it suggests that **coherent planning and execution within the same model family and capability tier** matters more than the raw capability of either role alone. We hypothesize

RQ3: Solve Rates by Model × Category

Model	Crypto	Forensics	Misc	Pwn	Reverse	Web	Overall
Claude 4.5 Opus	61.5%	53.3%	79.2%	41.0%	64.7%	52.6%	59.0% 118/200
Gemini 3 Pro	53.8%	46.7%	70.8%	33.3%	58.8%	47.4%	52.0% 104/200
Gemini 3 Flash	36.5%	13.3%	29.2%	28.2%	17.6%	31.6%	27.0% 54/200
GLM-5	13.5%	20.0%	29.2%	15.4%	23.5%	21.1%	19.5% 39/200
GPT-5.2-Codex	23.1%	6.7%	25.0%	20.5%	9.8%	21.1%	18.0% 36/200
GPT-5.2	19.2%	0.0%	25.0%	5.1%	11.8%	15.8%	13.5% 27/200
DeepSeek-V3	1.9%	6.7%	12.5%	7.7%	9.8%	0.0%	6.5% 13/200
DeepSeek-R1	0.0%	0.0%	8.3%	2.6%	5.9%	10.5%	4.0% 8/200
Qwen 3.5 397B-A17B	1.9%	6.7%	8.3%	0.0%	5.9%	0.0%	3.5% 7/200
Llama 3.3 70B	0.0%	0.0%	12.5%	0.0%	3.9%	0.0%	2.5% 5/200

Fig. 3. Per-category solve rates across all ten models. Darker cells indicate higher solve rates.

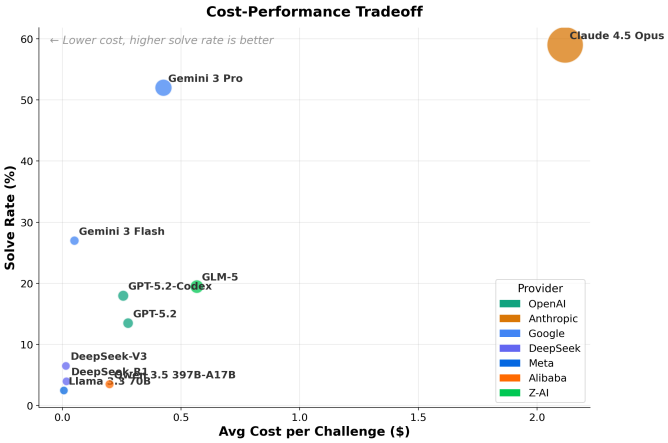

Fig. 4. Cost–performance tradeoff: average cost per challenge vs. solve rate. Bubble size is proportional to total cost.

TABLE II
PLANNER/EXECUTOR ARCHITECTURE COMPARISON (200 CHALLENGES, KALI + GENERIC).

Planner	Executor	Solved	Rate	Cost ($)
Pro	Pro	104	52.0%	$44.23
Pro	Flash	57	28.5%	$3.90
Flash	Flash	54	27.0%	$2.69
Flash	Pro	47	23.5%	$2.92

that mismatched models introduce coordination inefficiencies between planning and execution: the planner's instructions may not align with how the executor model interprets and decomposes tasks, leading to wasted rounds and miscoordination. A weaker planner may issue underspecified or naive instructions that a stronger executor cannot compensate for, whereas a same-model pair shares implicit assumptions about instruction granularity and tool-use patterns. The performance gap is largest in categories requiring sustained multi-step reasoning: crypto (53.8% for Pro-only vs. 21.1% for Pro+Flash) and misc (70.8% vs. 29.2%). The practical implication is that using the same model for both roles appears more effective than mixing model tiers. We note that these findings are based on two models from the same provider (Google); whether the coherence effect generalizes across providers remains an open question.

V. Conclusion

This paper presents a systematic evaluation of frontier large language model (LLM) agents on offensive cybersecurity tasks using the NYU CTF Bench. By varying environment, prompting strategy, model selection, and multi-agent architecture, we identify the primary drivers of performance in agent-based cyber operations.

Our results show that environment and tooling are important: a Kali Linux setup with pre-installed security tools improves performance over a standard Ubuntu environment. In contrast, prompt engineering techniques such as category-specific tips and auto-prompting do not consistently improve performance and can degrade results in well-equipped environments. Model selection remains a dominant factor, with frontier models outperforming smaller and open-source alternatives. We also find that homogeneous planner–executor configurations outperform mixed-model setups, highlighting the importance of coordination within a single model.

These findings suggest that environment configuration and model choice have greater impact than additional prompt complexity, while also emphasizing that performance depends not only on model capability but also on tool integration and API interaction.

This study has several limitations. Results are specific to the D-CIPHER framework, and other architectures may exhibit different behavior. Because LLMs are inherently nondeterministic and all experiments use a sampling temperature of $T{=}1.0$, solve rates may vary across runs; reported figures represent single-trial measurements and should be interpreted as indicative rather than exact. The NYU CTF Bench reflects academic CTF tasks, and model rankings correspond to early-2026 versions. Additionally, as shown in prior work, static CTF benchmarks are susceptible to data contamination. While absolute solve rates may be inflated, relative comparisons remain meaningful due to shared exposure. Validation on live or dynamic benchmarks would strengthen external validity.

Future work will extend this evaluation to additional benchmarks such as CyBench, explore retrieval-augmented generation and adaptive task routing, and investigate persistent challenges in categories like binary exploitation and forensics. Developing standardized and reproducible benchmarking protocols is also an important direction.

As LLM capabilities in offensive cybersecurity continue to advance, rigorous and transparent evaluation will be essential for both capability assessment and responsible use.

Acknowledgment

This work was supported by the SUNY AI Platform, powered by Google Cloud, and the Griffiss Institute (Award Number SA1001202200481). The contents of this paper are the sole responsibility of the authors and do not necessarily represent the official views of the sponsors.

References

[1] M. A. Ferrag, F. Alwahedi, A. Battah, B. Cherif, A. Mechri, N. Tihanyi, T. Bisztray, and M. Debbah, "Generative AI in cybersecurity: A comprehensive review of LLM applications and vulnerabilities," *Internet of Things and Cyber-Physical Systems*, 2025.

[2] R. Fang, R. Bindu, A. Gupta, Q. Zhan, and D. Kang, "LLM agents can autonomously hack websites," *arXiv preprint arXiv:2402.06664*, 2024.

[3] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzón, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *USENIX Security Symposium*, 2024.

[4] M. Kouremetis, M. Dotter, A. Byrne, D. Martin, E. Michalak, G. Russo, M. Threet, and G. Zarrella, "OCCULT: Evaluating large language models for offensive cyber operation capabilities," 2025, arXiv:2502.15797. [Online]. Available: http://arxiv.org/abs/2502.15797

[5] M. Shao, S. Jancheska, M. Udeshi, B. Dolan-Gavitt, H. Xi, K. Milner, B. Chen, M. Yin, S. Garg, P. Krishnamurthy, F. Khorrami, R. Karri, and M. Shafique, "NYU CTF Bench: A scalable open-source benchmark dataset for evaluating LLMs in offensive security," 2025, arXiv:2406.05590. [Online]. Available: http://arxiv.org/abs/2406.05590

[6] T. Abramovich, M. Udeshi, M. Shao, K. Lieret, H. Xi, K. Milner, S. Jancheska, J. Yang, C. E. Jimenez, F. Khorrami, P. Krishnamurthy, B. Dolan-Gavitt, M. Shafique, K. Narasimhan, R. Karri, and O. Press, "EnIGMA: Interactive tools substantially assist LM agents in finding security vulnerabilities," 2025, arXiv:2409.16165. [Online]. Available: http://arxiv.org/abs/2409.16165

[7] M. Udeshi, M. Shao, H. Xi, N. Rani, K. Milner, V. S. C. Putrevu, B. Dolan-Gavitt, S. K. Shukla, P. Krishnamurthy, F. Khorrami, R. Karri, and M. Shafique, "D-CIPHER: Dynamic collaborative intelligent multi-agent system with planner and heterogeneous executors for offensive security," 2025, arXiv:2502.10931. [Online]. Available: http://arxiv.org/abs/2502.10931

[8] M. Shao, H. Xi, N. Rani, M. Udeshi, V. S. C. Putrevu, K. Milner, B. Dolan-Gavitt, S. K. Shukla, P. Krishnamurthy, F. Khorrami, R. Karri, and M. Shafique, "CRAKEN: Cybersecurity LLM agent with knowledge-based execution," 2025, arXiv:2505.17107. [Online]. Available: http://arxiv.org/abs/2505.17107

[9] M. Shao, B. Chen, S. Jancheska, B. Dolan-Gavitt, S. Garg, R. Karri, and M. Shafique, "An empirical evaluation of LLMs for solving offensive security challenges," 2024, arXiv:2402.11814. [Online]. Available: http://arxiv.org/abs/2402.11814

[10] A. K. Zhang, N. Perry, R. Dulepet, J. Ji, C. Menders, J. W. Lin, E. Jones, G. Hussein, S. Liu, D. Jasper *et al.*, "Cybench: A framework for evaluating cybersecurity capabilities and risks of language models," 2025, arXiv:2408.08926. [Online]. Available: http://arxiv.org/abs/2408.08926

[11] Anonymous, "CTFusion: A CTF-based benchmark for LLM agent evaluation," *Under review at ICLR*, 2026.

[12] A. K. Zhang, J. Ji, C. Menders, R. Dulepet, T. Qin, R. Y. Wang, J. Wu, K. Liao, J. Li, J. Hu *et al.*, "BountyBench: Dollar impact of AI agent attackers and defenders on real-world cybersecurity systems," 2025, arXiv:2505.15216. [Online]. Available: http://arxiv.org/abs/2505.15216