

STRIATUM-CTF: A Protocol-Driven Agentic Framework for General-Purpose CTF Solving

James Hugglestone*, Samuel Jacob Chacko*, Dawson Stoller, Ryan Schmidt, Xiuwen Liu
Department of Computer Science, Florida State University, Tallahassee, USA

Email: jah21e@fsu.edu, sj21j@fsu.edu, dawson.stoller@aol.com, res22f@fsu.edu, xliu@fsu.edu

*These authors contributed equally.

Abstract—Large Language Models (LLMs) have demonstrated potential in code generation, yet they struggle with the multi-step, stateful reasoning required for offensive cybersecurity operations. Existing research often relies on static benchmarks that fail to capture the dynamic nature of real-world vulnerabilities. In this work, we introduce STRIATUM-CTF (A Search-based Test-time Reasoning Inference Agent for Tactical Utility Maximization in Cybersecurity), a modular agentic framework built upon the Model Context Protocol (MCP). By standardizing tool interfaces for system introspection, decompilation, and runtime debugging, STRIATUM-CTF enables the agent to maintain a coherent context window across extended exploit trajectories.

We validate this approach not merely on synthetic datasets, but in a live competitive environment. Our system participated in a university-hosted Capture-the-Flag (CTF) competition in late 2025, where it operated autonomously to identify and exploit vulnerabilities in real-time. STRIATUM-CTF secured First Place, outperforming 21 human teams and demonstrating strong adaptability in a dynamic problem-solving setting. We analyze the agent’s decision-making logs to show how MCP-based tool abstraction significantly reduces hallucination compared to naive prompting strategies. These results suggest that standardized context protocols are a critical path toward robust autonomous cyber-reasoning systems.

Index Terms—Autonomous cyber-reasoning, Model context protocol, LLM agents, Tool-augmented generation, Offensive cybersecurity.

I. INTRODUCTION

Offensive security assessments, including penetration testing, red-teaming, and Capture-the-Flag (CTF) competitions, remain predominantly manual and labor-intensive endeavors. To successfully compromise a target, analysts must execute a complex, non-linear workflow: enumerating vast attack surfaces, interpreting verbose scanner outputs, selecting appropriate exploitation primitives, and iteratively tuning payloads to bypass defenses. While powerful tools exist for specific phases (e.g., Nmap, Ghidra, Hashcat), the orchestration of these tools remains a cognitive bottleneck. Human operators must act as the “glue,” manually parsing data from one tool to feed into another, a process that is prone to errors and cognitive biases such as confirmation bias [1], and is increasingly difficult to scale with the rapid cadence of modern software delivery pipelines. As system complexity grows, the sheer volume of alerts and the requirement for continuous context switching lead to alert fatigue, degrading the operator’s ability to maintain high-fidelity mental models of the target environment [2].

The emergence of Large Language Models (LLMs) [3] offers a potential paradigm shift. Their ability to process unstructured text and generate code makes them well-suited to automate the high-level reasoning required to connect these disparate tools. However, early attempts to build autonomous security agents have faced significant architectural hurdles [4]. Existing methods largely fall into two categories: “copilots” that require constant human prompting, or “naive autonomous agents” that simply pipe model output into a shell [5].

Recent evaluations of these naive approaches reveal critical vulnerabilities beyond simple hallucination. For instance, *Inter-Agent Trust Exploitation* [6] has shown that LLMs often blindly execute malicious payloads if they originate from peer agents, lacking the introspection to verify the origin of a command. Furthermore, while models like GPT-4o [7] excel at high-level planning, they suffer from a *Reasoning-Action Gap* [8], where the model correctly identifies a vulnerability (e.g., SQL injection) but fails to generate the syntactically correct payload to exploit it due to a lack of grounded environment feedback, frequently hallucinating non-existent command-line flags or software packages, a failure mode that introduces downstream supply-chain risks such as *slopsquatting* [9]. This disconnect is deepened in long-horizon tasks, where “context drift” causes the agent to lose track of early reconnaissance data [10], further observed in recent surveys on LLM hallucinations in cybersecurity [11], leading to loops of redundant, failed attempts.

To address these limitations, we introduce **STRIATUM-CTF** (A Search-based Test-time Reasoning Inference Agent for Tactical Utility Maximization in Cybersecurity), a neuro-symbolic agentic framework that decouples neural reasoning from tool execution using the Model Context Protocol (MCP) [12] for solving CTF problems. By formalizing the interface between the LLM and the execution environment, we transform the model from a passive text generator into an active system operator. Internally, the agent is equipped with a curated arsenal of offensive capabilities, including symbolic execution via Angr [13], static analysis with Ghidra [14], and runtime debugging through GDB. Externally, a containerized environment exposes industry-standard reconnaissance and exploitation utilities such as Nmap, Nuclei, FFUF, and TLSX, allowing the agent to interact with the target system safely and reproducibly. Upon receiving a high-level objective, STRIATUM-CTF autonomously formulates a multi-step plan,

invokes tools via structured MCP calls, parses standard error streams to self-correct failed attempts, and iterates until the objective is achieved or the search space is exhausted.

This paper makes the following contributions to the field of autonomous cyber-operations: 1) **A Protocol-Driven Agentic Architecture:** We propose a novel framework that utilizes the Model Context Protocol to standardize LLM interactions with offensive security tools, reducing hallucinations by collapsing the output manifold via strict schema compliance for tool invocation, and enabling robust error recovery through structured feedback loops; 2) **Integration of Specialized Analysis Primitives:** We integrate complex, domain-specific tooling, including symbolic execution (Angr) and dynamic debugging (GDB), within an LLM’s context window, allowing deep binary analysis and exploit generation beyond simple scripting; and 3) **Validation via Live Competition:** We empirically demonstrate the system’s efficacy with a first-place victory in a live, multi-team Capture-the-Flag competition, providing a rigorous case study of autonomous agent performance under time constraints and dynamic uncertainty compared to human baselines. The code and data will be made available upon acceptance.

II. RELATED WORK

The evolution of LLMs from static knowledge bases to active agents relies on scaling test-time compute, allocating computational resources to iterative reasoning during inference rather than relying solely on pre-trained weights [15]. While early approaches like Toolformer [16] and Gorilla [17] attempted to bake tool usage into the model via fine-tuning, this strategy remains rigid and prone to becoming obsolete as APIs change. Modern research increasingly favors dynamic, inference-time planning; however, purely neural reasoning strategies like Chain-of-Thought [18] still suffer from functional hallucination, where the model confidently generates syntactically invalid commands. Our work bridges this gap by extending test-time compute into the symbolic domain. By utilizing the MCP as a hard constraint layer, similar to grammar-constrained decoding [2], [19], we transform the inference process from a probabilistic random walk into a structured, deterministic search, effectively grounding the model’s “thinking” time in verified system feedback.

While techniques such as ReAct (Reasoning and Acting) [20] successfully integrate trace generation with action execution, they fundamentally rely on the Large Language Model’s context window to maintain the entire history of the conversation. In long-horizon scenarios such as CTF competitions, which often require hundreds of interaction turns, this dependency becomes a critical vulnerability. Empirical analyses on long-context attention [10] demonstrate that transformer architectures suffer from “context drift,” where the model’s ability to retrieve and attend to early-state constraints degrades as the context window fills.

In the domain of offensive security, this manifests as error propagation [21]. A single hallucinated file path or misinterpretation of a port scan in step t becomes the ground truth

for step $t + 1$, leading to a cascading deviation from the initial objective. Unlike “stateless” agents that attempt to re-summarize the entire history at every step, STRIATUM-CTF offloads the knowledge-state management to the MCP layer. By treating the environment, rather than the token window, as the primary source of truth, our system mitigates the “Lost in the Middle” phenomenon, allowing the model to focus its attention solely on the immediate tactical decision.

The deployment of LLMs in offensive security has typically followed a “copilot” paradigm, prioritizing safety and interpretability over autonomy. Frameworks such as Pentest-GPT [22] adopt this approach, utilizing a “Pentesting Task Tree” (PTT) architecture to decompose high-level objectives into sub-tasks, while still requiring a human operator to manually execute generated commands and paste outputs back into the model’s context. Similarly, concurrent research such as PentestMCP [23] has introduced toolkits for agent-based penetration testing. However, while these frameworks primarily validate performance on historical examples and known vulnerability classes, STRIATUM-CTF is explicitly designed and tested on novel, dynamic challenges. This distinction is crucial, as it ensures the system’s success is driven by generalized reasoning rather than the retrieval of memorized exploit paths from the training distribution.

While these human-in-the-loop designs, as seen in Pentest-GPT, mitigate the risk of accidental damage, they also create a latency bottleneck that throttles the reasoning loop. In dynamic environments, such as CTF competitions, where state changes occur rapidly, the delay introduced by human intervention prevents the model from maintaining synchronization with the target system. STRIATUM-CTF addresses this by replacing the human verifier with deterministic schema validation at the MCP layer. This closes the feedback loop, enabling the agent to execute “machine-speed” iterations, rapidly testing hypotheses, analyzing error streams, and refining strategies without the cognitive overhead of manual orchestration.

III. METHODOLOGY

A. Formal Problem Formulation

We model the CTF challenge as a Partially Observable Markov Decision Process (POMDP) [24] defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R} \rangle$. The environment state $s_t \in \mathcal{S}$ represents the hidden configuration of the target server (e.g., memory layout, open ports). The agent cannot observe s_t directly but receives a structured observation $o_t \in \mathcal{O}$ via the MCP interface after executing action $a_t \in \mathcal{A}$. We define the history $h_t = (o_0, a_0, \dots, o_{t-1})$ as the accumulated context. The reward function \mathcal{R} is sparse, returning 1 only upon flag capture.

The core contribution of STRIATUM-CTF is the introduction of a symbolic constraint function $C : \mathcal{A} \rightarrow \{0, 1\}$ enforced by the Protocol Layer. In a standard LLM agent, the policy $\pi_\theta(a_t|h_t)$ generates a probability distribution over an unconstrained token space, leading to hallucinations (actions where $C(a_t) = 0$). Our architecture modifies the effective

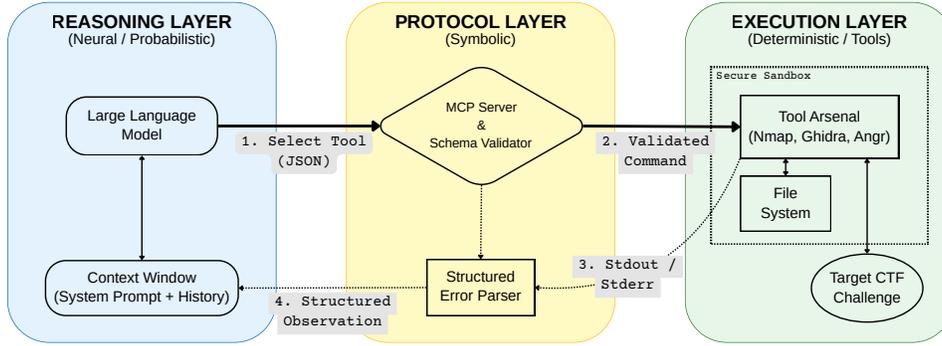


Fig. 1. High-Level Neuro-Symbolic Architecture. The system decouples probabilistic reasoning from deterministic execution. The Reasoning Layer (Left) acts as the strategic planner, emitting JSON payloads that must pass through the Protocol Layer (Center). This symbolic interface enforces strict schema validation, effectively filtering out hallucinated commands, before invoking tools in the Execution Layer (Right). The resulting system feedback (stdout/stderr) is structurally parsed and re-injected into the context window, grounding the agent’s latent state in verifiable reality.

policy π^* to project the output manifold onto the valid schema space $\mathcal{V} = \{a \in \mathcal{A} \mid C(a) = 1\}$:

$$\pi^*(a_t|h_t) = \frac{\pi_\theta(a_t|h_t) \cdot \mathbb{I}[a_t \in \mathcal{V}]}{Z} \quad (1)$$

where $\mathbb{I}[\cdot]$ is the indicator function, and Z is the partition function. By rejecting all $a_t \notin \mathcal{V}$ before execution, we ensure that the transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$ is only defined for valid semantic primitives, preventing undefined behavior in the target environment.

B. High-Level Neuro-Symbolic Architecture

Our framework strictly decouples *neural reasoning* from *system execution*. As illustrated in Figure 1, the system architecture comprises three distinct layers that function as a closed-loop neuro-symbolic controller:

- **The Reasoning Layer (Probabilistic):** We utilize Claude Sonnet 4.5 as the central planner. Unlike standard chatbots, this model is instantiated with a system prompt that enforces an *agentic persona*. It generates high-level strategies but is physically isolated from the operating system, preventing direct execution of hallucinated commands.
- **The Protocol Layer (Symbolic):** To bridge the gap between the probabilistic LLM and the deterministic operating system, we implement the Model Context Protocol. It functions as a schema validator, enforcing strict typing on all function calls (e.g., ensuring a port number is an integer between 1-65535). If the Reasoning Layer emits a syntactically invalid request, the Protocol Layer rejects it before it reaches the sandbox, effectively collapsing the model’s output manifold onto valid execution paths.
- **The Execution Layer (Deterministic):** This layer consists of containerized security tools (Angr, Ghidra, GDB) wrapped in MCP servers. These tools execute in a secure, ephemeral sandbox, returning structured JSON observations rather than raw text streams.

C. The MCP Toolbox

To bridge the abstraction gap between high-level intent and low-level execution, we expose a curated arsenal of semantic primitives, strictly typed JSON-schema functions, rather than raw command-line shells. This design enforces type safety and reduces the token overhead of parsing verbose tool outputs. For **Symbolic and Static Analysis**, we integrate Angr and Ghidra, exposing atomic functions such as `solve_path(addr, cond)` and `get_xrefs(func_name)`. The Protocol Layer enforces strict input constraints (e.g., requiring valid hexadecimal strings for addresses) and returns simplified UNSAT or Stdin strings, abstracting away the complexity of simulation state management. For **Dynamic Analysis**, our GDB wrapper provides an `inspect_heap(count)` function that transforms raw memory dumps into structured JSON objects, allowing the agent to traverse the heap without parsing visual ASCII layouts. Finally, **Reconnaissance and System** operations (via Nmap and Bash) are gated by allow-lists that restrict execution to valid IPv4 ranges and pre-approved binaries, preventing the agent from executing destructive commands outside the engagement scope.

D. Algorithmic Lifecycle

The system follows a recursive four-step control loop designed to maintain state coherence across complex attack chains:

- **Plan (Strategic Decomposition):** The agent analyzes the high-level objective (e.g., “Find the flag”) and decomposes it into a queue of atomic tasks, maintained in the context window and dynamically re-ordered based on observations.
- **Execute (Protocol-Mediated Invocation):** The agent selects the appropriate tool and sends a JSON payload. The Protocol Layer acts as a “circuit breaker”, rejecting hallucinated parameters and preventing state corruption, as seen in Figure 2 (Phase 2).
- **Parse (Structured Ingestion):** The MCP layer executes valid commands and returns a structured JSON object, which the agent parses to extract salient fields (open ports,

gadget offsets, memory leaks) while filtering irrelevant data to conserve context tokens.

- **Iterate (Contextual Refinement):** Based on the newly acquired grounded state, the agent evaluates the success of the prior action, refining subsequent steps (Self-Correction) or terminating the loop to generate a final report if the objective is met.

Fig. 2 shows this lifecycle in a real-world scenario, demonstrating the system’s ability to transition from Reconnaissance (Phase 1) to Autonomous Self-Correction (Phase 2), where a schema violation triggers immediate plan refinement, ultimately leading to Exploitation and Victory (Phase 3).

IV. EXPERIMENTAL SETTING AND RESULTS

A. Dataset, LLM Models, and Implementation Details

We assembled a benchmark of 15 CTF challenges drawn from multiple sources: picoCTF¹ (5 challenges), the official angr.io tutorials² (3 challenges), an archived university CTF (1 challenge), and legacy challenges drawn from internal repositories (6 challenges), for which complete provenance metadata is unavailable. Table I summarizes the distribution across vulnerability categories. Difficulty ratings were assigned by the LLM itself after initial exposure, based on the number of prerequisite techniques required. We acknowledge this introduces potential bias; future work will incorporate independent difficulty ratings. While the dataset size ($N = 15$) is constrained by the manual effort required for ground-truth analysis, the selected challenges represent a high-entropy distribution of difficulty, requiring distinct reasoning chains (e.g., heap traversal vs. side-channel timing) that effectively stress-test the agent’s generalization capabilities.

TABLE I
BENCHMARK DATASET COMPOSITION BY VULNERABILITY CATEGORY.

Category	Challenges
Memory Corruption	4
Reverse Engineering	4
Web Exploitation	4
Cryptography	3
Total	15

1) *Model Configuration:* All experiments used Claude Sonnet 4.5 via Claude Desktop for Linux with extended thinking enabled. Model parameters (temperature, max tokens) were left at default settings. Each challenge was attempted three times per condition, yielding $15 \times 4 \times 3 = 180$ total experimental runs.

2) *Ablation Conditions:* To isolate the contribution of structured documentation, we evaluated four conditions with varying levels of contextual guidance provided via Claude’s project file system:

- **Baseline** (4,147 lines): Complete documentation including attack templates, lessons learned from prior failures,

and the `trriage.py` classification tool. This configuration represents the full STRIATUM-CTF system.

- **Templates** (1,976 lines): The structured solving protocol (`AUTONOMOUS_SOLVER_README.md`) with phase-based attack templates, plus `trriage.py`. No lessons or supplementary guides.
- **Lessons** (1,478 lines): Lessons extracted from debugging sessions (e.g., heap exploitation pitfalls, web framework patterns) plus `trriage.py`, but without the structured templates.
- **Minimal** (55 lines): Only MCP server definitions and tool schemas. No `trriage.py`, no documentation. This serves as the control condition.

The `trriage.py` tool performs automated challenge classification via static and dynamic analysis, predicting vulnerability type before exploitation begins. Its presence in all conditions except Minimal tests whether structured pre-analysis contributes to success.

3) *Implementation Environment:* Experiments were conducted on an Ubuntu 22.04 LTS virtual machine allocated 16 GB RAM, 8 CPU cores, and 100 GB storage, hosted on a Windows 11 system with an Intel i9-14900HX processor. The MCP server ecosystem comprised both community tools and custom implementations:

- **GDB Server:** `mcp-gdb v0.1.1` [25]
- **Ghidra Server:** `GhidraMCP v1.4` [26] with Ghidra 11.4.2
- **Angr Server:** Custom implementation wrapping `angr 9.2.176`
- **SecOps Server:** `secops-mcp` [27]
- **Commands Server:** Custom implementation for shell execution

4) *Evaluation Protocol:* Each run was subject to a 60-minute timeout. To manage context window limitations, execution was manually checkpointed every 10 minutes, the session was terminated, and resumed with a “Continue” prompt. A challenge was marked as *successful* if the correct flag was extracted within the time limit, regardless of the number of intermediate failures or approach pivots. Runs were invalidated and restarted if the agent attempted to access pre-written solutions or otherwise compromised experimental integrity.

We used the community-developed Claude Desktop for Linux³ as the host. We conducted informal preliminary testing across Claude Opus 4.1, Claude Sonnet 4.5, and Claude Opus 4.5, selecting Sonnet 4.5 based on cost-performance tradeoffs.

B. Benchmark Evaluation Results and Analysis

We executed 180 experimental runs (15 challenges \times 4 conditions \times 3 trials) to evaluate the impact of structured documentation on autonomous CTF solving. Figure 3 presents the success rates with 95% Wilson score confidence intervals.

1) *Overall Performance:* Table II summarizes performance across conditions. The Baseline configuration achieved the highest success rate at 86.7%, with progressively lower rates as documentation was reduced. However, statistical analysis

¹<https://picoctf.org>

²<https://github.com/angr/angr-doc>

³<https://github.com/aaddrick/claude-desktop-linux>

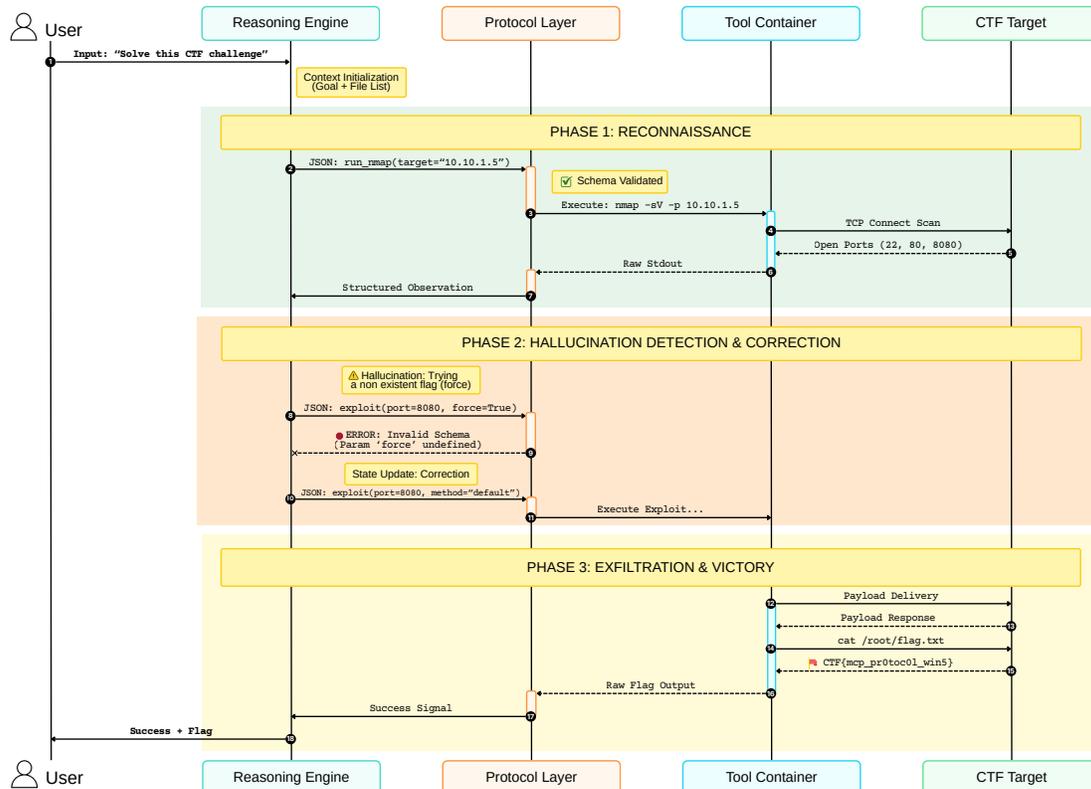


Fig. 2. STRIATUM-CTF Execution Workflow: A sequence trace showing the transition from User Input to Flag Capture. The diagram highlights the system’s error-recovery capability: when the agent attempts an invalid tool configuration (Phase 2), the Protocol Layer enforces schema compliance, triggering an autonomous correction cycle that enables the final successful exploitation (Phase 3).

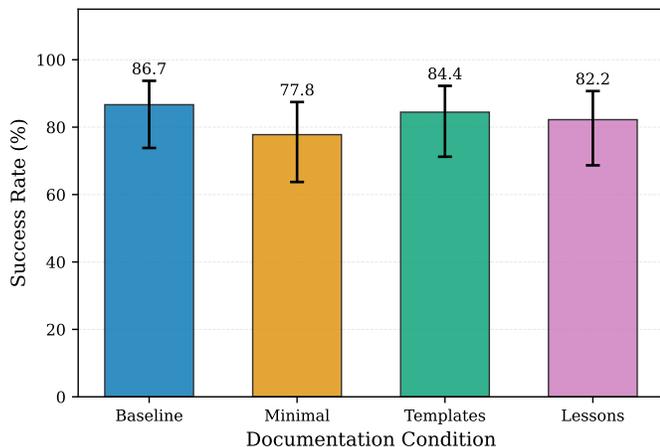


Fig. 3. Success rates of different settings with the 95% Wilson Score confidence interval.

revealed no significant differences between conditions (Chi-square test: $p = 0.71$; Kruskal-Wallis test for durations: $p = 0.77$). Effect sizes were negligible (Cohen’s $d < 0.2$) for all pairwise comparisons.

This result is noteworthy: even the Minimal condition, with only 55 lines of MCP tool definitions and no guidance documentation, achieved a 77.8% success rate. This result

TABLE II
PERFORMANCE SUMMARY ACROSS ABLATION CONDITIONS.

Condition	Success Rate	Mean Duration
Baseline	86.7% (39/45)	17.1 min
Templates	84.4% (38/45)	18.5 min
Lessons	82.2% (37/45)	19.1 min
Minimal	77.8% (35/45)	20.1 min

is consistent with our hypothesis that performance gains are primarily driven by the Protocol Layer’s schematic grounding rather than lengthy prompt engineering, though the small sample size limits the strength of this conclusion. By enforcing type safety at the tool interface, the ‘Minimal’ agent achieves robust performance without requiring the token-heavy context of ‘Baseline’ templates. This suggests that neuro-symbolic grounding via MCP is a more token-efficient scaling law than context-based prompting.

2) *Duration Analysis*: Figure 4 shows the distribution of solve times for successful runs. Mean duration increased as documentation decreased, from 17.1 minutes (Baseline) to 20.1 minutes (Minimal), suggesting that structured guidance may improve efficiency even when overall success rates are comparable. Figure 5 visualizes individual run outcomes, highlighting the variance within conditions.

While differences between conditions did not reach statisti-

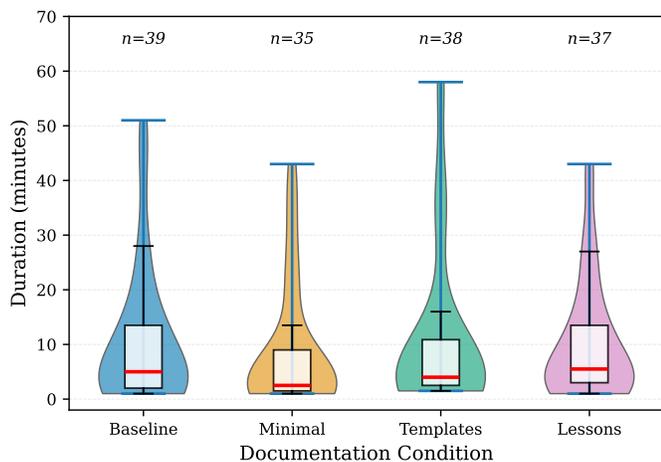


Fig. 4. Distribution of the time taken to solve CTF problems under different settings.

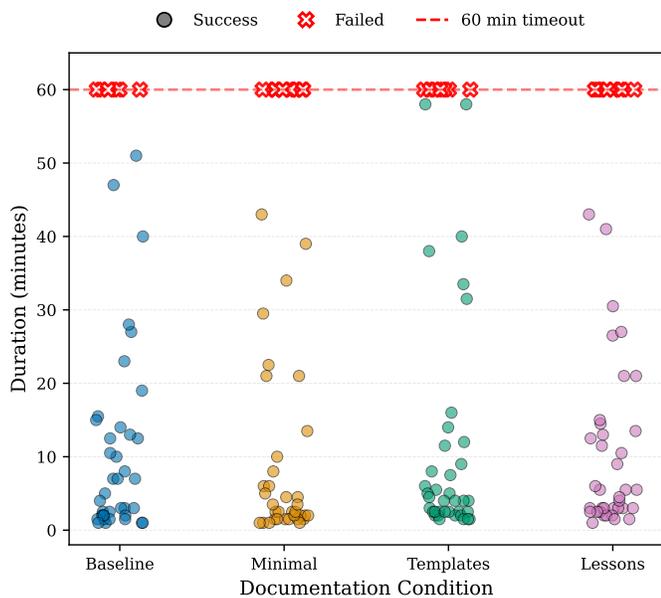


Fig. 5. Time taken of individual runs under different settings.

cal significance, likely due to sample size constraints inherent in the labor-intensive nature of CTF challenge curation, a consistent trend emerged across all metrics. The Baseline condition, representing the full STRIATUM-CTF system, achieved the highest success rate and lowest mean solve time. This suggests that comprehensive documentation provides marginal but consistent benefits, particularly for complex challenges requiring multi-step reasoning.

3) *Case Studies*: To illustrate the system’s behavior, we highlight two representative scenarios:

a) *Timing Side-Channel (sample2)*: This challenge required detecting and exploiting execution time variance to extract a PIN code. The Minimal condition failed all three attempts (0%), while Baseline succeeded once (33%) and Templates twice (67%). Analysis of execution logs revealed that

without explicit guidance on timing attack methodology, the agent attempted symbolic execution, an ineffective approach for side-channel vulnerabilities. This demonstrates that certain vulnerability classes benefit significantly from domain-specific documentation.

b) *Symbolic Execution (angrs_00)*: In contrast, this basic reverse engineering challenge was solved consistently across all conditions in under 2 minutes (100% success rate). The challenge required only invoking the Angr MCP server’s `solve_by_output` function, a capability directly exposed by the Minimal configuration’s tool schema. This suggests that challenges aligning closely with MCP tool primitives require minimal additional guidance.

In addition, we also used another implementation of STRIATUM-CTF to solve the problems worth less than 100 points at <https://ctf.hackucf.org/challenges>. In this version, we used the Claude Code. STRIATUM-CTF was able to establish a connection to hackUCF and pull 30 problems from different categories. With a single request, it was able to solve and collect the flags for 13 problems and provide the remote exploits for 6 problems. An additional request resulted in solving the four scripting problems. Two additional solutions were slightly incorrect. One flag was wrapped in the wrong prefix but was otherwise correct, and the offset of the other exploit was off by 8 bytes. In total, with only two requests and minor corrections to two solutions, STRIATUM-CTF was able to solve 25 problems and score 483 points. If it were a graduate student for the “Offensive Computer Security” course, it would earn an ‘A’ on the offline CTF assignment.

These cases illustrate a key finding: the value of structured documentation is *challenge-dependent*, with the greatest benefits observed for vulnerability classes requiring multi-step reasoning or unconventional approaches.

C. Results on the Live CTF Competition

To further validate the effectiveness of our STRIATUM-CTF, a team (named “Graveyard”) using the system competed in a live and multi-team CTF competition. The competition took place on November 15, 2025, at a university campus⁴ and was open to all undergraduate and graduate students. More than 30 teams registered, of which 22 (including ours) solved at least one challenge. Most participants were students who had taken the “Offensive Computer Security” course. The challenges were mainly in the binary exploitation and web security categories to match the content of the course. The problems were designed to be challenging for the students. For example, we expect an ‘A’ student but with no prior extensive CTF experience to be able to score 60 points. Figure 6 shows the progression (the number of solved problems) plot of the top 10 teams.

For comparison, Fig. 7 shows the score distribution⁵ across all teams that solved at least one problem. STRIATUM-

⁴To comply with the anonymous submission policy, university name and other revealing details are removed in this version; We will provide specific details in the final version.

⁵Generated using MATLAB’s `ksdensity` function.

V. DISCUSSION AND FUTURE WORK

While the adoption of the Model Context Protocol (MCP) ensures modularity and interoperability, it introduces inevitable computational overheads compared to highly optimized, bespoke logic such as that employed in Pentest-GPT [22]. A critical bottleneck we are currently addressing is the redundancy in token consumption: the MCP framework typically necessitates re-injecting full tool definitions and schema contexts into the prompt for every interaction. To mitigate this context saturation, we are developing a hierarchical context management strategy. This involves deploying a lightweight, local LLM to act as a semantic filter, dynamically compressing tool descriptions, and maintaining session state locally. This decoupling ensures that the primary reasoning model receives only the most salient information, significantly reducing token costs without sacrificing protocol adherence.

Current architectures predominantly rely on general-purpose reasoning models compatible with the MCP protocol. Unlike static benchmarks such as Cybench [28] or InterCode-CTF [29], which face increasing risks of dataset contamination (where solutions leak into the model’s training set), our evaluation in a live, ephemeral competition ensures that the agent’s performance stems from generalized reasoning rather than memorization. While such models demonstrate adequacy in standard CTF challenges and penetration tasks where training distributions are dense, our extensive empirical analysis reveals their inability to generalize to high-complexity scenarios.

We identify two primary bottlenecks: first, the action space exploration within these models is overly constrained; second, reasoning efficiency degrades significantly during long-horizon tasks, leading to error propagation and planning failures. To overcome these limitations, our future work will focus on two key architectural enhancements. First, to address the constrained action space, we propose the development of a domain-specific primitive layer that maps high-level reasoning to granular, security-oriented operations, effectively expanding the model’s exploratory capabilities. Second, to improve inference efficiency in long-horizon tasks, we intend to implement a hierarchical planning framework. By decomposing complex penetration objectives into manageable sub-goals, this approach will reduce the computational search depth and mitigate the risk of error propagation observed in current general-purpose models [30], [31].

We also seek to enhance the system’s exploratory capabilities through parametrized generalization via concept hierarchies. Abstracting learned sequences into higher-level semantic structures enables more effective transfer of knowledge to novel scenarios. We further propose a formal triphasic lifecycle: a Training Phase for fundamental parameter optimization; an Expansion Phase for accumulating and refining domain knowledge; and a Deployment Phase, where the system is frozen and optimized for time-constrained inference in competition environments. Together, these architectural improvements pave the way for an agentic system capable of rivaling and outperforming human experts in both efficiency

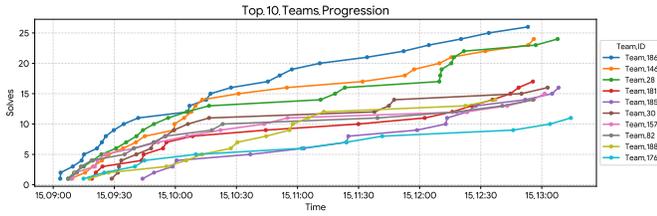


Fig. 6. Cumulative solves progression of the top 10 teams during the four-hour (with a 15-minute extension) competition. STRIATUM-CTF (Team 186) maintained a consistent lead throughout.

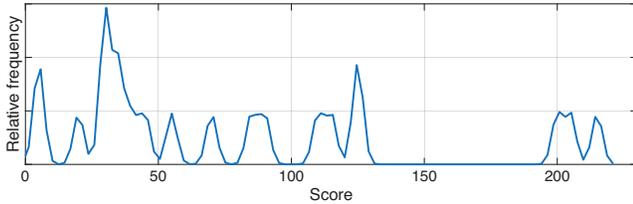


Fig. 7. Relative frequencies of different scores estimated using a kernel density estimator.

CTF achieved 215 points, followed by human teams at 205 (2nd place) and 200 (3rd place), a margin of only 15 points separating first from third. However, the remaining teams scored below 135 points, indicating a substantial gap between the top three and the field.

Based on five years of experience teaching this course, a score of 60 corresponds to an expected ‘A’ for a graduate student. STRIATUM-CTF exceeded this benchmark by a factor of 3.6 \times . Notably, solving all but two challenges and stopping only due to API cost constraints after securing first place, rather than technical limitations. We attribute STRIATUM-CTF’s performance advantage to three factors:

- **Efficiency:** The agent invokes tools and processes output at machine speed, eliminating the latency of manual command entry and output parsing. Tasks that take a human 30–60 seconds in GDB are completed in under one second.
- **Systematic Debugging:** When exploits fail, the agent methodically varies parameters (offsets, payload encodings, timing delays) without the cognitive fatigue that degrades human performance over multi-hour competitions. Failed attempts are logged and avoided, preventing redundant exploration.
- **Resilience Under Time Pressure:** CTF challenges demand precision; a single incorrect byte in a payload causes failure. Under time pressure, humans often guess, while the agent maintains systematic hypothesis testing throughout.

These results show the system now achieves the competency of top students with extensive CTF experience. With further improvements in problem decomposition, tighter integration, and knowledge-guided search, we expect the system to surpass the best students within two years.

and effectiveness, even in highly complex domains.

VI. CONCLUSION

In this paper, we introduced STRIATUM-CTF, a novel framework designed to automate penetration testing via the Model Context Protocol. By leveraging general-purpose reasoning models within a strict symbolic constraint layer, STRIATUM-CTF effectively bridges the gap between static vulnerability scanning and autonomous exploitation. Our empirical results from a live CTF competition demonstrate that this neuro-symbolic approach significantly outperforms human baselines in speed and precision, particularly for tasks aligning with semantic tool primitives. While current architectures still face hurdles regarding long-horizon planning and token efficiency, our findings confirm that protocol-mediated grounding is a viable solution to the model hallucination problem. Ultimately, STRIATUM-CTF establishes a new baseline for agentic security systems, marking a critical step toward the next generation of robust, autonomous cyber-operations.

REFERENCES

- [1] P. Aggarwal, S. Venkatesan, J. Youzwak, R. Chadha, and C. Gonzalez, "Discovering cognitive biases in cyber attackers' network exploitation activities: A case study," in *Human Factors in Cybersecurity*, 15th International Conference on Applied Human Factors and Ergonomics (AHFE 2024)*, 2024.
- [2] S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-constrained decoding for structured nlp tasks without finetuning," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, 2023.
- [3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000–6010, 2017.
- [4] S. B. Hakim, M. Adil, A. Velasquez, S. Xu, and H. H. Song, "Neuro-symbolic ai for cybersecurity: State of the art, challenges, and opportunities," *arXiv preprint arXiv:2509.06921*, 2025.
- [5] A. Happe and J. Cito, "On the surprising efficacy of llms for penetration-testing," *arXiv preprint arXiv:2507.00829*, 2025.
- [6] M. Lupinacci, F. A. Pironti, F. Blefari, F. Romeo, L. Arena, and A. Furfaro, "The dark side of llms: Agent-based attacks for complete computer takeover," *arXiv preprint arXiv:2507.06850*, 2025.
- [7] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.
- [8] K. Stechly, K. Valmееkam, and S. Kambhampati, "On the self-verification limitations of large language models on reasoning and planning tasks," in *The Thirteenth International Conference on Learning Representations*, 2025.
- [9] S. Park, "Slopsquatting: Hallucination in coding agents and vibe coding," *Trend Micro*, 2025.
- [10] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.
- [11] A. K. Sood, S. Zeadally, and E. Hong, "The paradigm of hallucinations in ai-driven cybersecurity systems: Understanding taxonomy, classification outcomes, and mitigations," *Computers and Electrical Engineering*, vol. 124, p. 110307, 2025.
- [12] Anthropic, "Model context protocol." <https://modelcontextprotocol.io>, 2025. Accessed: Jan. 2026.
- [13] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, *et al.*, "Sok:(state of) the art of war: Offensive techniques in binary analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 138–157, IEEE, 2016.
- [14] National Security Agency, "Ghidra: Software reverse engineering framework." <https://github.com/NationalSecurityAgency/ghidra>, 2019. Accessed: 2026-01-29.
- [15] C. V. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning," in *The Thirteenth International Conference on Learning Representations*, 2025.
- [16] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *Advances in Neural Information Processing Systems*, vol. 36, pp. 68539–68551, 2023.
- [17] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive apis," *Advances in Neural Information Processing Systems*, vol. 37, pp. 126544–126565, 2024.
- [18] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 24824–24837, 2022.
- [19] F. Raspanti, T. Ozcebebi, and M. Holenderski, "Grammar-constrained decoding makes large language models better logical parsers," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 485–499, 2025.
- [20] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafraan, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The eleventh international conference on learning representations*, 2022.
- [21] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.
- [22] G. Deng, Y. Liu, V. M. Vilches, P. Liu, Y. Li, Y. Xu, M. Pinzger, S. Rass, T. Zhang, and Y. Liu, "Pentestgpt: Evaluating and harnessing large language models for automated penetration testing," in *USENIX Security Symposium*, 2024.
- [23] Z. Ezetta and W.-c. Feng, "Pentestmcp: A toolkit for agentic penetration testing," *arXiv preprint arXiv:2510.03610*, 2025.
- [24] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration testing== pomdp solving?," in *Workshop on Intelligent Security (Security and Artificial Intelligence)-SecArt-11*, 2011.
- [25] Signal Slot Inc., "MCP GDB Server for Debugging with AI Assistants." <https://github.com/signal-slot/mcp-gdb>, 2025. Accessed: 2026-01-30.
- [26] Maintainers of ghidraMCP, "MCP Server for Ghidra." <https://github.com/LaurieWired/GhidraMCP>, 2025. Accessed: 2026-01-30.
- [27] SecurityforTech, "Security Operations Multi-Tool Platform (MCP)." <https://github.com/securityfortech/secops-mcp>, 2025. Accessed: 2026-01-30.
- [28] A. K. Zhang, N. Perry, R. Dulepet, J. Ji, C. Menders, J. W. Lin, E. Jones, G. Hussein, S. Liu, D. J. Jasper, *et al.*, "Cybench: A framework for evaluating cybersecurity capabilities and risks of language models," in *The Thirteenth International Conference on Learning Representations*, 2025.
- [29] J. Yang, A. Prabhakar, S. Yao, K. Pei, and K. R. Narasimhan, "Language agents as hackers: Evaluating cybersecurity skills with capture the flag," in *Multi-Agent Security Workshop@ NeurIPS'23*, 2023.
- [30] R. F. Del Rosario, K. Krawiecka, and C. S. de Witt, "Architecting resilient llm agents: A guide to secure plan-then-execute implementations," *arXiv preprint arXiv:2509.08646*, 2025.
- [31] R. Fang, R. Bindu, A. Gupta, Q. Zhan, and D. Kang, "Teams of llm agents can exploit zero-day vulnerabilities," *CoRR*, 2024.