Measuring LLM Tool-Use Efficiency in Cryptographic Capture-the-Flag Competitions

Iordache Mihai Bogdan

Responsible Professor and Supervisor: Dr. Z. Erkin

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 25, 2026

Name of the student: Iordache Mihai Bogdan
Final project course: CSE3000 Research Project
Thesis committee: Dr. Z. Erkin, Mitchell Olsthoorn

Abstract

Large Language Models demonstrate strong capability in mathematics, yet struggle with cryptographic tasks requiring precise algorithmic implementation. We investigate this capability gap through an ablation study that quantifies the marginal utility of giving the models access to multiple tools. Models interact with these tools through Python REPL with unrestricted access to external libraries to autonomously install and configure their own computational environments within a custom ReAct framework. Using 15 tasks from the AICrypto benchmark covering diverse cryptographic categories, we isolate the contributions of general purpose programming versus domain-specific libraries and tools across three configurations and three models, measuring success rate, time to solve, stability, and iteration depth. Our analysis reveals three principal findings: transitioning from pure reasoning to code execution with python and no additional libraries or access to external software yields a +37.78 percentage point performance increase; expanding to unrestricted access produces diminishing returns due to tool management and installation overhead, with effects varying by model type (reasoning / non-reasoning); and tool and Python library utility is task-dependent, with large gains for complex algorithmic and computationally intensive challenges but penalties for pattern-based cryptanalysis. These findings establish the existence of a reasoning capability threshold for LLMs below which additional tools consume planning capacity, demonstrating that the Python Standard Library provides optimal performance for most models and cryptographic categories in our selected dataset. This work provides a quantitative basis for integrating large language models with tool usage in cryptographic CTFs.

1 Introduction

General-purpose Large Language Models (LLMs) show strong capability in mathematics [11], yet this fails to translate to the applied domain of cryptographic Capture-the-Flag (CTF) challenges [11; 6] when models rely on pure inference without access to external computational methods. The core problem is that abstract mathematical reasoning and pure theory do not guarantee good performance in applied cryptanalysis, which requires precise algorithm recall, multi-step implementation, adaptive problem-solving, a working memory, and the effective use of the environment in which the LLM is operating. CTF tasks mimic real-world scenarios, involving unknown, non-deterministic, or misconfigured environments, unfamiliar problems, and a complete lack of documentation.

AICrypto [11] shows that while top models excel at routine proofs and theory memorization, they underperform on tasks demanding dynamic analysis and planning. The LLMs default to familiar attack patterns rather than engaging in deep analytical thinking. This reliance is a pattern-matching behavior, which is to be expected due to the statistical nature of current LLM generation, which, based on training data, predicts the most probable next word [7]. This prediction mechanism fails to replicate the precise reasoning required for cryptanalysis, where a single incorrect character in a hash or key, for example, renders the entire solution invalid. This is the result of the generation process being non-deterministic. Tool usage, being code-based CLI utilities as well as Python libraries, is substantially more deterministic by comparison, thus providing a way to address this limitation.

Even in complex agentic systems that incorporate tool usage, several failures occur. Studies like PentestGPT [3] and EnIGMA [1] show that augmenting agents with tools improves success rates, yet models still struggle with context maintenance and hallucinate tool outputs. These failures highlight that tool availability alone is insufficient without strong pre-prompting for guiding the AI model in a strict manner.

While recent work has shown that tool augmentation can improve performance, we lack quantitative estimates of the contribution of each specific utility. We address the following research questions: How do external tools influence an LLM's performance on solving cryptographic CTF challenges regarding success rate, time to solve, and stability? And how large is the marginal effect of each instrument when added to a configuration? We aim to answer this using an ablation study which allows us to systematically isolate the marginal contributions of different tool access levels to LLM performance on cryptographic challenges.

The remainder of this paper details the experimental setup, presents the statistical analysis of tool efficacy, and discusses the implications for automated cryptanalysis.

The codebase along with the dataset and data processing functionality can be accessed at: bogdansys/Research-Project-TU-Delft-CSE3000 .

2 Background and Related Work

2.1 Large Language Models in Cybersecurity

The field of large language models in cybersecurity evolved from code completion to vulnerability analysis. Sun et al. [10] document this in LLM4Vuln, a framework that compares reasoning performance with and without external capabilities. While non-reasoning models benefit from external help for logic-heavy tasks, their reasoning counterparts perform better without any additional help. Knowledge retrieval in particular degrades performance for the reasoning models by introducing noise that distracts from their intrinsic verification processes. Large language models can be categorized as reasoning models, which employ internal verifications and steps where they generate reasoning steps to evaluate the next approach, or standard models, which generate responses directly.

Li et al. [6] show that LLMs struggle with cryptography because they predict text rather than compute exact values. Rather than strictly adhering to decryption rules, models succumb to semantic inference, attempting to guess plaintext based on linguistic patterns instead of executing required

character-level operations. Muzsai et al. [8] demonstrate that making models larger does not solve this problem. Instead, models need access to external tools like Python to perform accurate calculations.

2.2 Agentic Architectures and Tool Augmentation

Benchmarks alone are insufficient without the architectural frameworks that enable models to use external tools during challenge solving. The deployment of Large Language Models in cybersecurity has evolved toward dynamic agent architectures capable of autonomous planning. Static approaches fail because models lack the intrinsic ability to execute code or maintain state over long horizons [4]. Agentic approaches are therefore necessary, where the LLM functions as a decision maker that coordinates actions (commands, tool calls, writing code) based on environmental feedback rather than probabilistic prediction [5]. The ReAct (Reason then act) framework, introduced by Yao et al., addresses this need by combining reasoning traces with task-specific actions through a Thought-Action-Observation loop [13]. This iterative cycle is essential for Capture the Flag challenges, which require trial and error: an agent must hypothesize a vulnerability, execute a reconnaissance command, then analyze the output to validate the hypothesis. Yao et al. demonstrated that this action oriented approach constitutes an improvement for knowledge-intensive tasks, making it well-suited for cryptographic problem-solving.

3 Methodology

In this section, we define the experimental framework used to quantify the marginal utility of external tools in automated cryptanalysis. We detail the benchmark selection, the agentic architecture that manages tool usage, and the statistical metrics used to measure performance.

3.1 System Architecture

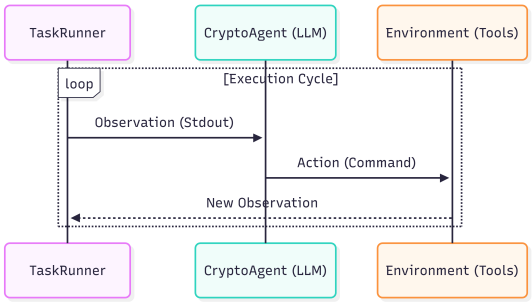

Figure 1: Agentic Architecture: The TaskRunner maintains the execution loop, while the CryptoAgent interacts with the environment.

Throughout this paper, we use the term *tools* to refer broadly to any external computational resource beyond pure language-based reasoning. This encompasses both simple command-line utilities and complex software libraries that the agent can autonomously discover, install, and deploy.

The tool ecosystem is organized into two tiers: a Python 3.10 REPL for general-purpose programming (Tier 1), and any external packages accessible via `pip` installation as well as command line utilities (Tier 2). In the Open Environment configuration, the agent can autonomously install and configure its environment. This design intentionally forces the model to perform tool selection and discovery during reasoning rather than providing a preselected, task specific toolkit. The architecture enforces strict ablation (detailed in Section 3.3) by dynamically constructing system prompts that restrict capabilities of an agent. This is done by hiding information about its own capabilities as well as explicitly instructing the model on allowed actions at each step. For a concrete example, in pure reasoning profile, the LLM is reminded at every step that it is not allowed to use any utility apart from its own thinking. This has been done at every iteration instead of the original pre-prompt since we observed that the models tend to ignore requests after the first 5-10 iterations.

3.2 Model Selection

We selected three Large Language Models. Our selection prioritized diversity in reasoning depth and efficiency to assess how tool augmentation affects models across the performance spectrum.

- *Grok 4.1 (xAI)* [12] We test the *Fast* variant to evaluate a model optimized for high-throughput reasoning.
- *Grok 4.1 Reasoning (xAI)* A variant of Grok optimized for deeper reasoning chains, allowing us to assess whether enhanced reasoning capabilities reduce reliance on external tools.
- *Claude 4.5 Haiku (Anthropic)* [2] selected for its superior speed, representing the class of compact models designed for rapid task execution.

3.3 Experimental Setup

To evaluate applied cryptographic capability, we use the AICrypto benchmark, a selected dataset of CTF challenges. Unlike static dataset evaluations, this benchmark requires the agent to interact with files. To simulate real-world constraints, we enforce a 300-second timeout (matching AICrypto benchmark specifications [11]) and a 25-iteration cap. While empirical findings show successful solutions typically occur within 20–35 iterations [11; 1], this limit captures the majority of efficient solutions, filtering for strategic reasoning over exhaustive search.

This dataset was selected because it demands multi-step reasoning and implementation accuracy. The AICrypto benchmark minimizes training data contamination through temporal recency (challenges from 2023+), structural obfuscation (standardized prompts without problem description context), and empirical validation that reasoning is more valuable than recall [11]. While complete elimination remains unverifiable for closed-source models, these safeguards reduce memorization risk.

From this larger corpus, we selected a subset of 15 challenges for intensive ablation study. Running the complete 150-task benchmark would require 4,050 execution runs (150×3 configs $\times$ 3 attempts $\times$ 3 models), so we restricted

scope to 405 runs (15 tasks) to enable manual inspection. We focused on challenges where tools were expected to improve performance: tasks requiring modular arithmetic and or algorithmic implementation.

We included challenging Stream/Hash tasks to avoid selection bias toward solvable problems, testing at the boundary where tools matter most. Our findings therefore apply specifically to computationally intensive cryptographic tasks.

Table 1: The subset of 15 challenges selected for the ablation study.

Category	Challenge Name
Classic	Greek Cipher
Classic	Valentines Day
RSA	Blue Hens RSA 1
RSA	Blue Hens RSA 2
RSA	Big E
RSA	16-RSA
RSA	EzRSA 2020
RSA	BabyRSA 2019
RSA	XorSA
Homemade	Alibos
Stream/Hash	LF3R
Stream/Hash	Hyper512
Stream/Hash	My Array Generator
Lattice	Squares vs Cubes
Lattice	Naptime

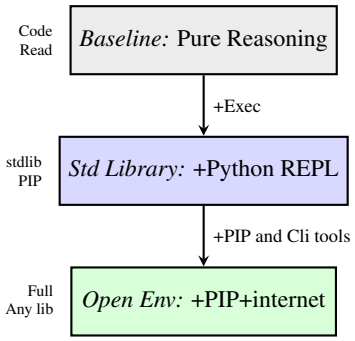

Figure 2: Experimental configurations showing progressive expansion from pure reasoning to full tool access.

Figure 2 shows the three experimental configurations, which differ fundamentally in computational capabilities:

- *Reasoning:* The agent can only read challenge files and reason through solutions linguistically. It cannot execute any code or verify computations, isolating the model's reasoning ability.
- *Standard Library:* The agent gains access to a Python 3.10 REPL restricted to the standard library. It can implement algorithms and verify results, but must build all cryptographic primitives itself using only built-in Python functions.
- *Open Environment:* The agent has autonomy with unrestricted `pip` access and internet connectivity, autonomously discovering and installing any library or tool needed. Agent prompts explicitly prohibited searching for challenge solutions ("looking for the direct challenge solution is CHEATING and strictly forbidden"), restricting internet use to documentation and tool acquisition; log analysis confirmed zero HTTP requests to writeup sites across 135 runs. This mirrors human competitor capabilities.

To enforce these configurations, we use three separate system prompt files. Selected LLMs are biased to utilize any capability defined in their context window, so dynamic system prompt selection ensures that the agent cannot hallucinate tool usage or bypass restrictions. Observed performance differences are thus attributable to the presence or absence of the tooling itself.

We measure performance using five quantitative metrics:

- *Success Rate:* The binary success of capturing the correct flag aggregated over all attempts.
- *Time to Solve:* The wall-clock duration from the initial prompt to successful verification, measuring the efficiency gain provided by specialized tools.
- *Iteration Depth:* The number of reasoning cycles required to reach a solution.
- *Stability:* The standard deviation of success rates, quantifying outcome consistency.
- *Marginal Effect:* The statistical utility of a specific tool layer, defined as the difference in success rates between two configurations: $\Delta(C_i \rightarrow C_j) = S_{C_j} - S_{C_i}$, where S represents the overall success rate across 405 runs (15 tasks $\times$ 3 models $\times$ 3 profiles $\times$ 3 attempts), and where C represents the model's profile.

4 Results

This section presents our evaluation of the three experimental profiles across the chosen CTF challenge dataset. The analysis focuses on capability (success rate, marginal effect), efficiency (time to solve, iteration depth), and stability.

4.1 Capabilities

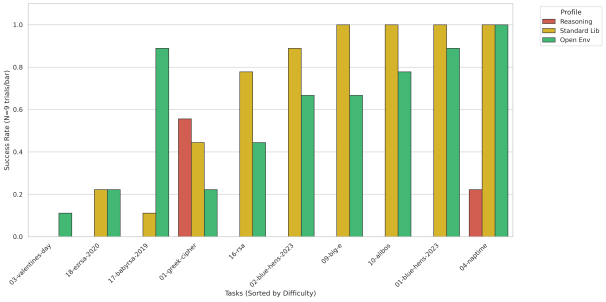

Figure 3: The Capability Staircase: Comparative success rates across profiles.

Capability Analysis
This figure shows the success rate for each profile across all tasks, sorted by difficulty based on overall success rate.

- The *Standard Library* and *Open Environment* profiles outperform the *Reasoning* profile on mathematical tasks.
- Peak Capability vs. Stability: While the *Open Environment* profile is less stable on average (underperforming the Standard Library due to environment overhead), it demonstrates higher peak capability. For example, it was the only profile capable of solving the Valentines Day challenge (1/9 runs, 11% success rate), which the other profiles failed.
- Filtering: Tasks yielding 0% success across all profiles were excluded from this graph as they show no statistically valuable information.

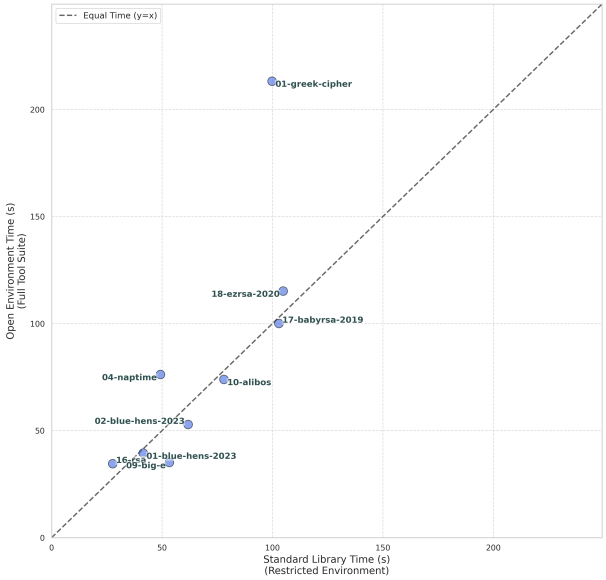

Figure 4: Execution Time Comparison: Standard Library vs Open Environment efficiency.

Efficiency Analysis
This scatter plot compares the execution time of successful runs between the Standard Library and Open Environment profiles.

- Filter: Data included only for tasks solved by both comparison profiles.
- *X-Axis:* Time taken (seconds) by *Exclusive* (Standard Library) profile.
- *Y-Axis:* Time taken (seconds) by *Full* (Open Environment) profile.
- *Reference:* A diagonal $y = x$ line.

Analysis of the 9 intersecting tasks (where both profiles obtained at least one successful result) shows a measurable efficiency penalty for unrestricted tool access. The Standard Library profile achieved an average solution time of 68.8 seconds, whereas the Open Environment averaged 82.7 seconds, 20% slower.

Table 2: Comparative Performance and Stability. Percentage of individual attempts (N=45 per model/profile) that resulted in each outcome. Success requires verified flag capture before timeout. Remaining percentage represents failures that neither crashed nor timed out.

Profile	Model	Success (%)	Crash (%)	Timeout (%)
Reasoning	Claude 4.5 Haiku	6.7	0.0	44.4
	Grok 4.1	4.4	0.0	28.9
	Grok Reasoning	4.4	0.0	48.9
Std Library	Claude 4.5 Haiku	40.0	0.0	60.0
	Grok 4.1	40.0	0.0	55.6
	Grok Reasoning	48.9	0.0	51.1
Open Env	Claude 4.5 Haiku	24.4	4.4	44.4
	Grok 4.1	40.0	0.0	60.0
	Grok Reasoning	53.3	0.0	46.7

Stability Analysis
Table 2 details the stability trade-offs across configurations. While Reasoning and Standard Library profiles show high robustness (0% Crash Rate), the Open Environment introduces instability with a 4.4% crash rate for Claude 4.5 Haiku and varied timeout rates. Here we can observe that unrestricted tool access incurs a reliability penalty.

Iteration Depth
We analyzed the conversation length in assistant turns required to reach a solution. The global median for successful runs is 8.0 steps and mean of 9.0 (successful runs only). Breaking down results by model shows different efficiency patterns: Claude 4.5 Haiku requires 18% more steps in the Open Environment (11.2 vs 9.4) compared to the Standard Library, while Grok Reasoning demonstrates an 11% reduction in steps (7.2 vs 8.2) when granted full tool access.

4.2 Marginal Effect Quantification

As defined in Section 3.3, we quantify the marginal utility of each configuration tier using the metric $\Delta(C_i \rightarrow C_j) = S_{C_j} - S_{C_i}$, where S represents the overall success rate across 405 runs (15 tasks $\times$ 3 models $\times$ 3 profiles $\times$ 3 attempts), and where C represents the model's profile.

Overall Marginal Effects. The transitions between profiles yield the following measured effects:

- $\Delta(\text{Reasoning} \rightarrow \text{Standard Library}) = +37.78$ percentage points
- $\Delta(\text{Standard Library} \rightarrow \text{Open Environment}) = -3.70$ percentage points

The first transition quantifies the impact of enabling general-purpose code execution via the Python Standard Library. This large positive delta (+37.78pp) suggests that the primary bottleneck is not conceptual understanding but rather the lack of algorithmic implementation. Models possess the theoretical knowledge to identify attack vectors but fail when restricted to pure reasoning without execution capabilities.

The second transition measures the incremental effect of granting access to specialized cryptographic utilities beyond

the Standard Library. While the aggregate negative delta (-3.70pp) implies a general overhead penalty, this is different between model architectures. Breaking this down by model, lighter models (Claude 4.5 Haiku) perform worse ($\Delta = -15.6$pp), where the complexity of environment management overwhelms the model. Conversely, reasoning-optimized models (Grok 4.1 fast reasoning) successfully use the open environment for a net positive gain ($\Delta = +4.4$pp). Tool management overhead is not universal; rather, there is a reasoning threshold required to effectively manage complex toolchains without suffering from distraction, a nuance consistent with the increased execution times observed in Figure 4.

Category-Level Variation. Marginal effects vary across cryptographic categories. RSA challenges demonstrate a $+57.14$pp gain from Reasoning to Standard Library, indicating that modular arithmetic operations benefit from programmatic implementation. Conversely, Classic ciphers show smaller or negative deltas (-5.56pp), as simpler substitution patterns remain solvable via pure reasoning for basic instances. Homemade ciphers exhibit the maximum delta ($+100.00$pp), confirming that custom cryptosystems require custom code, while Stream/PRNG/Hash challenges yield 0% success across all profiles, indicating capability limitations independent of tooling.

These varied effects show that tool utility is task-dependent: specialized libraries provide marginal value primarily for algorithmically complex tasks. For example, the `BabyRSA-2019` challenge saw success rates jump from 11% (Standard Library) to 89% (Open Environment). Analysis shows that models universally converged on specialized functions like `sympy.nextprime()` to bypass manual prime-finding implementations.

4.3 Global Success Rate Analysis

To distinguish between consistent performance and breakthrough capability, we measured both per-attempt success rate (averaged across all individual runs) and global success rate (proportion of the 15 tasks solved at least once by any of the three models within each profile). Table 3 presents this comparative analysis.

Table 3: Per-Attempt vs Global Success Rates across profiles. Gap quantifies the percentage-point increase from per-attempt to global success, indicating variability in solution paths.

Profile	Per-Attempt	Global	Gap
Reasoning	5.19%	13.33%	8.15pp
Standard Library	42.96%	60.00%	17.04pp
Open Environment	39.26%	66.67%	27.41pp

This table reveals problem-solving reliability. The Open Environment exhibits the largest gap (27.41pp), indicating that while individual attempts are less consistent, the profile solves the most unique tasks overall (66.67% global success vs 60.00% for Standard Library). Unrestricted tool access thus provides breakthrough capability on difficult challenges, even though success on any single attempt is less predictable.

The Reasoning profile shows the smallest gap (8.15pp), reflecting deterministic failure: when the model cannot solve a task through pure reasoning on the first attempt, subsequent attempts rarely succeed without computational tools. The Standard Library achieves a balanced middle ground (17.04pp gap), demonstrating more consistent performance while still benefiting from retry opportunities.

Performance Consistency. The standard deviation metric σ measures outcome variance:

Profile	σ	Interpretation
Reasoning	0.223	Most consistent
Standard Library	0.497	
Open Environment	0.490	

5 Discussion

This section interprets the quantitative findings presented in Section 4, examining the mechanisms underlying the observed performance deltas and their implications for automated cryptanalysis. We organize our analysis around five themes: the basic requirement for execution, the counterintuitive overhead of tool availability, category-specific tool utility, capability threshold hypothesis and limitations.

5.1 Code Execution Requirement: Interpreting the +37.78pp Marginal Effect

The most significant finding of our ablation study is the large performance gain when transitioning from pure reasoning to code execution. The $\Delta(\text{Reasoning} \rightarrow \text{Standard Library}) = +37.78$ percentage points (5.19% to 42.96%) empirically quantifies the gap explained in Introduction.

Our results illustrate this limitation most clearly in the RSA category, where models achieved 0% success in reasoning mode but 57.14% with access to Python's standard library. The modular exponentiation, greatest common divisor computation, and prime factorization required for RSA cryptanalysis demand exact arithmetic on large integers. These operations are mathematically well-defined but practically impossible to perform reliably through iterative text generation where a single digit error invalidates the entire solution.

The session context loss described by [3] offers additional explanation: language models cannot maintain persistent computational state across reasoning steps due to limited context windows. The Python REPL aims to provide this missing state, acting as an external memory structure reflected, for example in Lattice problem results ($\Delta = +38.89$pp, $11.11\% \rightarrow 50.00\%$).

The `Greek Cipher` task achieved 100% success in Reasoning mode for Claude 4.5 Haiku, showing pattern-recognition tasks succeed without execution. This exception explains how this gap emerges for computationally intensive cryptanalysis, not for all security tasks.

5.2 Tool Management Overhead: Interpreting the -3.70pp Effect

Model-Specific Divergence

Surprisingly, expanding from the Python Standard Library to the full open environment with unrestricted tool

and package access yielded a negative marginal effect: $\Delta(\text{Standard Library} \rightarrow \text{Open Environment}) = -3.70$ percentage points. This aggregate statistic, however, masks model-specific divergence that is our second major finding. Breaking down this effect by model architecture reveals: Claude 4.5 Haiku performed worse ($\Delta = -15.6$pp), Grok 4.1 reached a performance ceiling ($\Delta = 0.0$pp), while Grok 4.1 Reasoning successfully used specialized tools ($\Delta = +4.4$pp).

Unrestricted tool access adds cognitive load on the model. For models below a certain capability threshold, the complexity of managing an open environment overwhelms the potential benefits of specialized tools. This complexity includes selecting appropriate packages, navigating installation procedures.

Operational Overhead and Failure Mechanisms

We focus on aggregate metrics across all runs, though individual reasoning traces confirm patterns consistent with our quantitative findings: repeated file reads in timeout cases, malformed pip commands in crashes, and convergence on specialized libraries in successful runs.

- The Open Environment averaged 82.7 seconds versus 68.8 seconds for Standard Library (20% time penalty, Figure 4) due to failed installations.
- Claude 4.5 Haiku required 18% more reasoning cycles (11.2 vs 9.4 steps) while Grok 4.1 Reasoning achieved 11% fewer (7.2 vs 8.2), showing tool management competes for planning capacity in sub-threshold models but provides shortcuts for reasoning-optimized ones.
- The Open Environment induced 4.4% crashes for Claude 4.5 Haiku (versus 0% for other profiles), typically resulting from dependency conflicts and malformed installation commands observed during manual review.
- Though timeout remains the dominant failure mechanism (44.4%–60.0%), causes differ. Reasoning profiles get stuck re-reading files while Open Environment timeouts arise from failed installations.

The Valentines Day challenge shows this best: 0% success in Reasoning and Standard Library but 11% in Open Environment, showing genuine benefit despite requiring extensive exploration (25 steps). This represents a complexity threshold where standard Python becomes insufficient, yet the overhead for reaching the correct specialized tool statistically shown to lead to unreliable outcomes.

5.3 Category-Specific Tool Utility and Task Variation

The marginal effects of tool access vary widely across cryptographic categories, revealing that tool utility is task-dependent rather than universal. Table 4 summarizes these divergent effects.

Computational categories like RSA and Homemade show gains from Python execution but negative returns from specialized tools. Models spend unnecessary time searching for non-existent utilities instead of implementing simple logic.

Table 4: Marginal Utility by Category. $\Delta(R \rightarrow S)$: Gain from Standard Library. $\Delta(S \rightarrow F)$: Gain from Open Environment. R represents reasoning, S represents Python environment and F represents Open Environment

Category	$\Delta(R \rightarrow S)$	$\Delta(S \rightarrow F)$
Homemade	+100.0 pp	-22.2 pp
RSA	+57.1 pp	-3.2 pp
Lattice	+38.9 pp	0.0 pp
Classic	-5.6 pp	-5.6 pp
Stream/Hash	0.0 pp	0.0 pp

The exception is `BabyRSA-2019`, where success jumped from 11% to 89% using `sympy.nextprime()`. Pattern-based tasks like Classic Ciphers suffer penalties as tool installation overhead disrupts natural frequency analysis. Lattice challenges show threshold effects: matrix operations are essential, but advanced libraries provide no additional benefit for small-dimension tasks. Stream/Hash challenges remain unsolvable across all profiles, indicating capability limits independent of tooling.

5.4 The Reasoning Capability Threshold Hypothesis

Different models respond differently to increased tool complexity, suggesting the existence of a reasoning capability threshold: a minimum level of planning and problem-solving ability required to effectively use complex toolchains. Below this threshold, additional tools impose net negative effects by overwhelming the model's limited reasoning capacity. Above it, tools improve performance by handling low-level implementation details.

- *Claude 4.5 Haiku (Negative Utility)*: Tool complexity consumes reasoning capacity for package selection and installation management. The model generates more iterations (+18%), takes longer, and achieves worse outcomes (-15.6pp) when granted unrestricted access. This negative aggregate effect reflects overhead costs dominating across most tasks, though specific computationally intensive challenges (e.g., Valentines Day) succeeded only with specialized tools. After approximately 10 reasoning cycles, qualitative inspection suggests a success rate drop-off, correlating with repetitive failure loops where models repeat behaviors without strategy adaptation. Models lack the ability to recognize when progress has stalled and change strategy.
- *Grok 4.1 (Neutral Utility)*: The model reaches a performance ceiling independent of tool availability (0.0pp marginal effect). Both Standard Library and Open Environment yield approximately 40% success rates, suggesting task difficulty and algorithmic knowledge limit performance rather than tool access.
- *Grok 4.1 Reasoning (Positive Utility)*: Specialized tools act as shortcuts, enabling the model to bypass manual implementations. The 11% reduction in iteration depth (7.2 vs 8.2 steps) demonstrates efficiency gains. The model uses `sympy.nextprime()` and `numpy` instead of

manual implementations. The +4.4pp marginal effect confirms net positive utility, with successful solutions occurring within the first 8 iterations.

This threshold effect has practical implications for agent deployment: smaller, faster models may achieve better efficiency when restricted to limited, minimal toolsets that reduce decision complexity, while larger reasoning-optimized models justify the computational overhead only when granted unrestricted tool access to fully exploit their management capabilities.

The broader implication is that tool strategies must be matched to model capabilities: there is no universal *best* configuration, only optimal matches between model capability and tool complexity.

5.5 Limitations

Our findings are subject to several methodological constraints. The 15-challenge subset comprises approximately 10% of the AICrypto benchmark, limiting statistical power for detecting small effects and generalizability to excluded cryptographic categories. External LLM APIs introduce nondeterministic variance despite fixed model versions, creating a noise floor in stability metrics that future replications may not reproduce exactly. The 300-second uniform time limit scales disproportionately across task complexity, and the explicit tool-prompting directive inflates usage rates relative to baselines. We thus limit any claims about performance under less constrained time budgets.

6 Ethical Considerations

This research has dual-use implications. Autonomous agents capable of automated cryptanalysis and tool utilization may be repurposed for offensive cyber attacks. We addressed this issue through several measures: all studies took place in isolated sandbox environments; instead of employing real-world exploits, we utilized synthetic CTF challenges from CTF competitions; and we did not instruct the model to use dangerous cybersecurity tools such as Metasploit [9], even if capable of using them. Such tools are deemed outside of our area of research. We implemented safety guardrails to prevent harmful commands that could affect the underlying host system. While such research might lower barriers to sophisticated attacks, the scientific knowledge obtained helps improve defense systems in an era of increasingly automated cyber threats.

7 Conclusion

We have investigated how external computational tools and Python libraries influence Large Language Model performance on cryptographic Capture-the-Flag challenges and quantified the marginal utility of each layer of tool sophistication.

7.1 Key Findings

Our experimental analysis reveals three main findings. First, transitioning from pure reasoning to code execution yielded a +37.78 percentage point performance increase. Second, expanding to unrestricted tool and package access produced a negative aggregate effect ($\Delta = -3.70$pp), masking strong model-dependent differences. Compact models suffered performance degradation ($\Delta = -15.6$pp) while reasoning-optimized models successfully used specialized tools ($\Delta = +4.4$pp), showing a reasoning capability threshold below which additional tools degrade performance. Third, marginal effects vary widely across cryptographic categories (RSA: $+57.14$pp, Classic: -5.56pp), revealing that tool strategies must be task-aware rather than universal.

7.2 Design Implications

For practitioners building autonomous agents for technical tasks, our results suggest two design choices. First, prioritize Standard Library profiles over unrestricted tool access. The Python REPL without package installation delivers the best stability-to-capability ratio across most models and cryptographic categories. Second, in cases where specialized tools are absolutely necessary, refrain from using lightweight (less capable) LLMs and utilize a preconfigured environment to minimize tokens and time wasted on setup. Models below the reasoning capability threshold waste computational budget on environment management rather than problem-solving. Unrestricted tool access should be reserved for reasoning-optimized architectures on computationally intensive tasks where specialized libraries provide measurable advantage.

7.3 Future Work

Three important research directions remain open. First, expanding to the full 150-challenge AICrypto corpus would validate marginal effect stability across underrepresented cryptographic categories. Second, runtime profile escalation systems could dynamically upgrade access based on deterministic metrics when task analysis reveals computational requirements, optimizing the efficiency-capability tradeoff. Third, monitoring mechanisms could detect when models are stuck in repetitive loops. Finally, extending this methodology to real-world deployment for live testing and comparing with human performance.

The evidence shows tool augmentation is essential for LLM performance in computational cryptography, yet the relationship is non-linear and model-dependent. The Python Standard Library provides the highest marginal utility per unit of complexity, representing the optimal configuration for most models and tasks. Specialized utilities deliver value only when paired with sufficiently capable reasoning models and computationally intensive challenges. Future work should focus on smarter, context-aware tool orchestration rather than broader tool access. This requires understanding when, why, and for which models computational tools provide genuine advantages.

References

[1] 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. R. Narasimhan, R. Karri, and O. Press. Enigma: Interactive tools substantially assist lm agents in finding security vulnerabilities, 2025. arXiv. https://arxiv.org/abs/2409.16165.

[2] Anthropic. Claude 4.5 haiku. https://www.anthropic.com/news/claude-haiku-4-5, 2024. Accessed: 2025-05-15.

[3] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass. Pentestgpt: An llm-empowered automatic penetration testing tool, 2023. arXiv. https://arxiv.org/abs/2308.06782.

[4] Z. Huang and J. Zhuge. Multi-agent framework utilizing large language models for solving capture-the-flag challenges in cybersecurity competitions. *Applied Sciences*, 2025. https://www.mdpi.com/2076-3417/15/13/7159.

[5] S. P. Laney. Llm-directed agent models in cyberspace. Master's thesis, Massachusetts Institute of Technology, 2024. https://dspace.mit.edu/handle/1721.1/156291.

[6] Yujia Li, Qipeng Pei, Min Sun, Hong Lin, Chang Ming, Xiang Gao, Ji Wu, Cong He, and Li Wu. Cipherbank: Exploring the boundary of llm reasoning capabilities through cryptography challenges. *arXiv preprint arXiv:2504.19093*, 2025. https://arxiv.org/abs/2504.19093.

[7] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. arXiv. https://arxiv.org/abs/2410.05229.

[8] Lazlo Muzsai, Daniel Imolai, and András Lukács. Improving llm agents with reinforcement learning on cryptographic ctf challenges. *arXiv preprint arXiv:2506.02048*, 2025. https://arxiv.org/abs/2506.02048.

[9] Rapid7. Metasploit: Penetration testing framework. https://www.metasploit.com/, 2026. Accessed: 2026-01-24.

[10] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Wei Ma, Lyuye Zhang, Yang Liu, and Yingjiu Li. Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning. *arXiv preprint arXiv:2401.16185*, 2024. https://arxiv.org/abs/2401.16185.

[11] Y. Wang, Y. Liu, L. Ji, H. Luo, W. Li, X. Zhou, C. Feng, P. Wang, Y. Cao, G. Zhang, X. Li, R. Xu, Y. Chen, and T. He. Aicrypto: A comprehensive benchmark for evaluating cryptography capabilities of large language models, 2025. arXiv. https://arxiv.org/abs/2507.09580.

[12] xAI. Grok 4.1: Fast reasoning. https://x.ai/news/grok-4-1-fast, 2025. Accessed: 2026-01-15.

[13] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. https://arxiv.org/abs/2210.03629.

A LLM Prompts Used for Writing Assistance

To ensure academic integrity and clarify the role of Large Language Models in the writing process of this paper, we list below the specific prompts used.

A.1 Category 1: English Language Refinement

- "Please suggest potential improvements for the grammatical flow of this paragraph, without altering the underlying meaning: '[Insert text]'"
- "Could you suggest a list of academic synonyms for the word 'show' that would be appropriate in the context of data analysis?"

A.2 Category 2: Technical and LaTeX Assistance

- "Please suggest the correct LaTeX table syntax to represent the following data."
- "I am encountering a compilation error in this section. Could you suggest which part of the syntax might be causing the issue?"