

Improving LLM Agents with Reinforcement Learning on Cryptographic CTF Challenges

Lajos Muzsai, David Imolai and András Lukács

Eötvös Loránd University, Institute of Mathematics, AI Research Group

Abstract

We present RANDOM-CRYPTO, a procedurally generated cryptographic Capture The Flag (CTF) dataset designed to unlock the potential of Reinforcement Learning (RL) for LLM-based agents in security-sensitive domains. Cryptographic reasoning offers an ideal RL testbed: it combines precise validation, structured multi-step inference, and reliance on reliable computational tool use. Leveraging these properties, we fine-tune a Python tool-augmented Llama-3.1-8B via Group Relative Policy Optimization (GRPO) in a secure execution environment. The resulting agent achieves a significant improvement in Pass@8 on previously unseen challenges. Moreover, the improvements generalize to two external benchmarks: *picoCTF*, spanning both crypto and non-crypto tasks, and *AICrypto MCQ*, a multiple-choice benchmark of 135 cryptography questions. Ablation studies attribute the gains to enhanced tool usage and procedural reasoning. These findings position RANDOM-CRYPTO as a rich training ground for building intelligent, adaptable LLM agents capable of handling complex cybersecurity tasks.

Keywords

Reinforcement Learning, LLM Agents, Cryptography, Cybersecurity, Penetration Testing

1. Introduction

Large Language Models (LLMs) are rapidly evolving from general-purpose text generators into autonomous agents—capable of complex reasoning and increasingly rivaling human performance across domains [1]. This rapid advancement has caused significant research interest in deploying LLM-based agents in critical cybersecurity contexts, such as autonomous penetration testing and vulnerability detection [2, 3, 4]. Penetration testing, which involves simulating cyberattacks to proactively identify and document exploitable vulnerabilities, is a valuable practice for enhancing system security. Current LLM-based penetration testing agents primarily focus on autonomous exploitation capabilities. Capture The Flag (CTF) challenges have emerged as a natural benchmark for these capabilities: they emulate real-world offensive and defensive scenarios, enforce precise success criteria, and demand structured multi-step reasoning [5, 6, 7, 8]. Despite advancements in scripted and tool-augmented approaches, a significant challenge remains: the absence of scalable, rigorous environments for training these agents, rather than just evaluating them.

Reinforcement Learning (RL) has emerged as a promising approach to enhance the capabilities of LLM agents and has shown strong results in mathematics, coding, and planning. This enhancement is particularly achieved through structured fine-tuning methods that improve multi-step reasoning and decision-making abilities [9]. The recently proposed Group Relative Policy Optimization (GRPO) [10], in particular, supports efficient fine-tuning using ranked outputs, and has shown strong performance on competitive reasoning tasks with a relatively modest compute effort [11]. However, practical RL training requires environments that provide abundant, objectively verifiable tasks, which are often lacking in cybersecurity datasets.

We address this gap with RANDOM-CRYPTO, a procedurally generated dataset for cryptographic Capture The Flag (CTF) challenges that encompasses 50 different algorithmic families

✉ muzsailajos@protonmail.com (L. Muzsai); david@imol.ai (D. Imolai); andras.lukacs@ttk.elte.hu (A. Lukács)

ORCID 0009-0007-3009-6225 (L. Muzsai); 0009-0003-9063-3791 (D. Imolai); 0000-0003-3955-9824 (A. Lukács)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY-SA 4.0).

and over 5,000 unique tasks. Each challenge is randomized in its parameters, while embedding a randomized text flag, and is set within a contextual narrative generated by an LLM. This benchmark enables agent-centric, reward-driven training at scale, while preserving the symbolic accuracy of cryptographic problem-solving. This dataset provides an effectively limitless training corpus, allowing for extensive exploration and thorough evaluation of improvements based on RL. By focusing on cryptographic reasoning—a field that involves verifiable objectives, compositional tool use, and step-by-step logical inference—we create an environment where RL can lead agents not only to achieve correctness, but also to develop competence.

We utilize the RANDOM-CRYPTO dataset to fine-tune a tool-augmented Llama-3.1-8B model [12], using a secure Python execution environment. Our results show that the enhanced agent significantly improves performance on previously unseen crypto tasks, increasing the Pass@8 metric from 0.35 to 0.88. Furthermore, the improvements generalize beyond the RANDOM-CRYPTO distribution to two external benchmarks: *picoCTF*, which spans both cryptographic and non-cryptographic tasks, and *AICrypto MCQ*, a multiple-choice benchmark of 135 cryptography questions covering both theoretical and computational problems. In both cases, the model demonstrates reliable, structured tool use learned through reinforcement.

Our key contributions are as follows:

1. We introduce RANDOM-CRYPTO, a novel procedurally generated cryptographic benchmark for LLM-agent training and evaluation.
2. We demonstrate that GRPO-based RL significantly improves tool-augmented reasoning in LLM agents.
3. We validate cross-domain generalization to *picoCTF* and *AICrypto MCQ*, revealing transferable, RL-acquired strategies for security-relevant problem solving across heterogeneous task formats.

We have released the benchmarks and training code used for this paper on GitHub.¹

2. Background

2.1. CTF Challenges

CTF challenges are widely used to train and evaluate cybersecurity skills by simulating real-world attack scenarios in sandbox environments. The goal is typically to discover a hidden text string, called a flag, embedded in vulnerable applications, binaries, or websites. CTF challenges encompass a wide array of cybersecurity domains, including: web exploitation, cryptography, reverse engineering, forensics, and binary exploitation.

Cryptographic challenges, in particular, revolve around exploiting weaknesses or logical flaws in encryption and hashing methods. These challenges commonly involve classical ciphers (e.g., Caesar, Vigenère, substitution), symmetric-key encryption (e.g., AES, DES), and public-key encryption schemes (e.g., RSA, ECC). Participants may also encounter tasks involving hashing algorithms, digital signatures, and custom cryptosystems, which often require creative approaches and deep theoretical understanding to solve.

2.2. LLM Agents in Cybersecurity

Recent advancements in LLMs have sparked significant interest in their application within cybersecurity, particularly in penetration testing and security assessments. Researchers have adapted Capture The Flag (CTF) challenges as standardized benchmarks [5, 6, 8, 7], to evaluate the cybersecurity capabilities of LLM-based agents systematically. Initial evaluations revealed that general-purpose LLMs struggled to effectively handle specialized cybersecurity tasks due to their inherent inability to execute code or interface with operating systems and external tools [13, 5, 6, 14].

¹<https://github.com/aielte-research/HackSynth-GRPO>

In other domains, to bridge this interaction gap, newer architectures introduced tool augmentation, empowering LLM agents to invoke external tools or APIs. Frameworks like ReAct (Reason+Act) demonstrated substantial improvements by integrating logical reasoning with actionable tool use [15]. OpenAI’s ChatGPT further extended this approach through a plugin/function-calling API, enabling deterministic JSON tool calls [16]. Additionally, Anthropic’s Model Context Protocol (MCP) provides a structured interface for models to interact seamlessly with external data sources and tools, facilitating complex cybersecurity tasks that require real-time computational support [17, 18]. These tool-augmented LLM agents have significantly advanced the state-of-the-art. However, they have yet to be fully utilized in autonomous penetration testing.

2.3. Reinforcement Learning to Enhance LLM Reasoning

Reinforcement Learning (RL) has become a central technique for improving the reasoning abilities of LLMs, particularly in tasks requiring multi-step problem solving and generalization. The standard Reinforcement Learning by Human Feedback (RLHF) paradigm fine-tunes models based on human preference signals, typically using Proximal Policy Optimization (PPO) [19], to align model outputs with desired behaviors [20].

Recent work has shown that traditional RLHF, while effective in improving helpfulness and safety, often falls short in domains that require structured, multi-hop reasoning like math, programming, and cybersecurity. To address this, specialized algorithms have emerged, such as Group Relative Policy Optimization (GRPO) [10]. GRPO is a variant of PPO tailored for reasoning-intensive tasks: instead of relying on scalar preference labels, it compares multiple candidate outputs and assigns relative rewards within a group. This relative evaluation strategy encourages diversity and robustness in generated solutions, helping the model refine its internal decision-making process. GRPO has been successfully applied to train DeepSeek-R1 [11], achieving state-of-the-art performance on mathematical reasoning and competitive coding benchmarks. Its low computational overhead and compatibility with low-rank adaptation methods such as QLoRA [21] also make it an attractive choice for resource-constrained environments.

3. Methods

3.1. Random-Crypto Benchmark Dataset

We introduce the RANDOM-CRYPTO benchmark, a programmatically generated dataset created for training LLM-based agents with Reinforcement Learning. The benchmark was designed to allow generating near infinite variations of CTF challenges based on 50 different cryptographic schemes. Solving the challenges requires the ability to exploit vulnerabilities in cryptographic schemes and protocols. Each challenge requires recovering a hidden flag in the form `flag{...}` by leveraging a specific weakness.

The challenges are categorized into eight archetypes, such as *Classical*, *RSA*, *ECC*, etc., each further divided into multiple sub-types detailed in Appendix Table 6. These sub-types are partitioned into three difficulty levels:

- **Easy:** Simple ciphers solvable manually within minutes without complex scripting (e.g., ROT13 cipher).
- **Medium:** Challenges requiring basic scripting and deeper cryptographic understanding (e.g., Morse code decoding with dictionary lookup).
- **Hard:** Complex tasks requiring advanced cryptographic knowledge, scripting capabilities, and potentially external online tools or frequency analysis methods (e.g., substitution cipher with frequency analysis or a vulnerable custom cryptosystem).

The distribution of subtype difficulties is balanced with at least 16 challenges from all three difficulty levels in the 50 subtypes.

Each challenge is generated by first selecting a sub-type and then randomizing parameters specific to the cryptographic scheme, such as cipher keys or substitution dictionaries. Subsequently, a random flag is encrypted using the chosen parameters. An LLM-generated narrative accompanies each ciphertext, providing context and essential information to ensure solvability (see Appendix B, Prompt 1). For instance, a Caesar cipher challenge randomly selects a shift value between 1 and 25, generates a random plaintext flag, encrypts it, and embeds it within an LLM-generated short narrative.

To verify the solvability of each challenge, the authors conducted manual validation; at least one instance per sub-type was solved. Parameter spaces were also reviewed to avoid trivial or unsolvable configurations. The dataset comprises 50 manually validated challenges used exclusively for testing, along with 5,000 automatically generated challenges designated for agent training.

3.2. Reinforcement Learning

We fine-tuned the Llama-3.1-8B-Instruct [12] model for 250 training steps using the Group Relative Policy Optimization (GRPO) algorithm. The training specifically leveraged a constrained, multi-step code execution interface with a Python execution server via Anthropic’s Model Context Protocol (MCP), allowing the model up to four sequential tool invocations per challenge. During training, we restricted the training problems to only come from the easy subset of all the challenges. The decision to restrict training to easy challenges was motivated by initial experiments showing that more difficult challenges negatively impacted the training process. Models tended to converge prematurely to local optima by prioritizing superficial reward types, i.e., outputs matching formatting or syntax rewards. The rarity of the accuracy reward in the case of these hard challenges caused the models not to pursue solving them.

During each training step, we generated eight candidate trajectories for four distinct challenges, resulting in a batch size of 32. Each candidate trajectory consisted of iterative reasoning followed by structured JSON-formatted tool calls. The model interaction cycle included:

1. Generating initial reasoning text,
2. Invoking the Python MCP server with generated code,
3. Receiving execution output and continuing reasoning,
4. Repeating the process until the flag was recovered or the maximum number of interactions (four loops) was reached.

The composite reward function described in Table 1 guided the model’s learning process, designed to encourage correct and efficient problem-solving behavior. To address the challenge of the model prematurely optimizing towards superficial rewards (such as formatting or tool-calling correctness without genuine problem-solving), we implemented a deduction strategy. Specifically, we penalized scenarios where the model produced a fictitious or made-up output immediately following a tool call without actually sending it to the MCP server. Furthermore, we penalized the model when it produced a hallucinated flag as the answer, even before any reasoning or tool calls happened.

The total token generation was limited to 8192 per interaction sequence, and training was augmented using QLoRA [21] adapters on one A100 80GB GPU.

We explicitly instructed the model to strictly adhere to the interaction protocol with structured prompts (see Appendix B, Prompt 2). The available Python execution server exposed three functionalities:

- `execute_python`: Executes provided Python code and returns output.
- `list_variables`: Lists variables currently stored in memory.
- `install_package`: Installs Python packages into the runtime environment.

Table 1

Reward structure used during GRPO training.

Reward Type	Description	Value
Accuracy Reward	The predicted flag fully matches	1.0
Answer Format Reward	The final answer matches <code>\boxed{flag{...}}</code>	0.1
Tool Calling Reward	The model produces a valid JSON tool call	0.2
Execution Reward	The MCP server executes the code without error	0.5

This structuring mitigated unintended hallucinations by clearly delineating reasoning, tool calls, and output interpretation phases.

3.3. Evaluation Setup

We evaluate both baseline and GRPO-trained agents on three benchmarks: (i) the RANDOM-CRYPTO test split, consisting of 50 manually validated cryptographic challenges, (ii) the public *picoCTF* benchmark [6] containing 120 heterogeneous CTF problems across six categories, and (iii) the *AICrypto MCQ* benchmark [22], a 135-question multiple-choice dataset covering a broad spectrum of cryptography topics, including both theoretical and computational problems. All evaluations use the same tool-augmented prompting format described in Appendix B, with eight independent generations per task, and are scored using Pass@8 and Maj@8 metrics.

4. Results

This section presents the evaluation of five leading LLMs: Llama-4-Scout-17B-16E [23], Llama-3.1-70B [24], Llama-3.1-8B [12], GPT-4.1 [25], and o3 [26]—on the randomized cryptographic benchmark. Additionally, we analyze the impact of RL-based fine-tuning on model performance.

4.1. Benchmarking

To assess the cryptanalytic abilities of contemporary LLMs, we conducted experiments using the 50 verified challenges from the RANDOM-CRYPTO dataset. Each LLM was evaluated under four distinct experimental conditions: (i) *No hint, no tool use* (baseline performance), (ii) *Hint only* (supplementary hints provided), (iii) *Tool use only* (ability to execute Python code), (iv) *Hint and tool use*. In each scenario, models received the question field and any supplementary hints or Python tool access. Models were required to generate a single, structured response formatted as `\boxed{flag{...}}` in order for a challenge to be considered solved. Due to computational constraints, the maximum token generation was limited to 4096 for the non-tool use case and 8192 for the tool use case.

We report two standard LLM evaluation metrics: Pass@8 is the proportion of tasks for which at least one out of eight independent model generations successfully solves the task, and Maj@8 is the proportion of tasks for which at least five out of eight independent model generations successfully solve the task. Table 2 summarizes the experimental outcomes, measured by Pass@8 and Maj@8 metrics. GPT-4.1 and o3 models consistently outperform Llama-family models, highlighting the advanced reasoning capabilities of these architectures.

Interestingly, while supplementary hints typically improve performance for scenarios without tool access, this trend reverses when tools are available, especially for smaller models like Llama-3.1-8B. In these cases, hints often prompted models to attempt brute-force approaches. For example, challenges requiring enumeration or brute-forcing strategies (such as ROT13 decoding or poorly salted hashes) led smaller models to generate inefficient Python scripts. Such attempts often resulted in the models generating Python scripts that tried to enumerate exhaustive lists

of possible solutions or unnecessary computations, causing memory exhaustion and crashes in the execution environment. Conversely, larger and more capable models, such as GPT-4.1 and Scout, demonstrated improved performance when hints were available alongside tools, as they effectively incorporated hints into structured, computationally efficient reasoning processes.

Differences in prompt engineering also affected model performance. Specifically, the prompts utilized in this experiment were optimized primarily for Llama-3.1-8B, inadvertently disadvantaging larger Llama-family models.

The o3 model demonstrates exceptional baseline reasoning but experiences notable performance degradation due to the enforced 4096-token limit, cutting short extensive reasoning chains, especially in the presence of hints. OpenAI’s o3 frequently encountered content moderation issues, resulting in lower performance metrics due to incomplete task executions. The o3 model also consistently got confused regarding permissions for code execution, limiting its effective use of available Python tooling.

4.2. Reinforcement Learning

To investigate the impact of Reinforcement Learning (RL) on cryptographic task performance, we conducted three distinct training experiments using the Llama-3.1-8B model: (i) training without hints for 251 steps on easy challenges, (ii) training with hints for 251 steps, (iii) curriculum-based training, initially providing hints for 130 steps, followed by 121 steps without hints.

Initially, the model often generated valid Python code through tool calls but would simultaneously provide incorrect or imagined answers instead of executing the code. As a result, even when the generated scripts could produce the correct outputs, the model’s premature hallucinations led to the termination of the interaction.

Figure 1 illustrates the progression of average rewards throughout training, showing an increase from approximately 0.6 to 0.9 across all scenarios. The hinted training regime consistently achieved higher rewards due to the hints guiding the model to the appropriate solution. The exact breakdown of the different rewards gained during training is shown in Figure 2. Metrics tracking successful Python code execution (defined as code running without errors) improved notably from roughly 39% to approximately 56%, with most of this improvement happening in the first 30 steps of training. The adherence to the correct answer format saw an even more substantial improvement, rising from about 40% to nearly 80%. This improvement happened in the first 50 steps. However, properly formatted tool calls remained steady at approximately 95%. The reason behind this is that the prompt was constructed to direct the model to produce proper tool calls. The rate of successful challenge solutions went from 10% to 30%, and unlike the other rewards, this was gradual during the full training, with variance across training steps.

Post-training results on the easy subset of the RANDOM-CRYPTO benchmark (Table 3) reveal significant performance improvements. The curriculum and hinted models both gained a significant 0.80 improvement to their Pass@8 scores, which is better than GPT-4.1 and only

Table 2

Benchmark results (Pass@8/Maj@8) on the RANDOM-CRYPTO dataset with and without hint or tool use.

Model	Without Tool Use				With Tool Use			
	Without Hint		With Hint		Without Hint		With Hint	
	Pass@8	Maj@8	Pass@8	Maj@8	Pass@8	Maj@8	Pass@8	Maj@8
Llama-4-Scout-17B-16E	0.08	0.02	0.14	0.02	0.12	0.08	0.12	0.08
Llama-3.1-70B	0.04	0.00	0.04	0.00	0.18	0.02	0.12	0.00
Llama-3.1-8B	0.02	0.00	0.04	0.00	0.10	0.02	0.14	0.04
GPT-4.1	0.26	0.14	0.26	0.20	0.34	0.16	0.44	0.18
o3	0.32	0.20	0.32	0.14	0.92	0.38	0.92	0.38

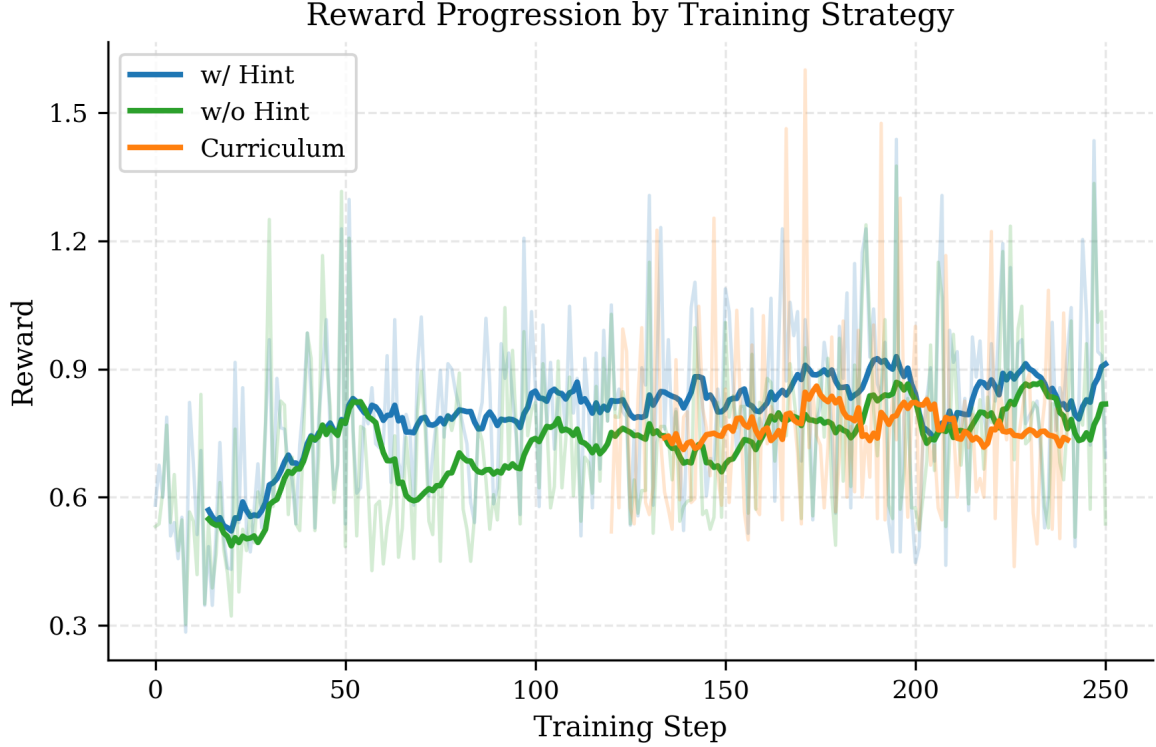


Figure 1: Reward gained during training. The bright lines mark the average, while the shaded lines mark the actual data points. One data point shows the average of all rewards given out in a training step to all benchmarks.

slightly behind the o3 model. The training setup that included no hints achieved a slightly lower Pass@8 score. On the contrary, it produced a significantly higher Maj@8 score, indicating that this training setup produces a more consistent model. Training with hints could guide the model toward accurate solutions more frequently, even when the model couldn’t guess it by chance. However, this could also make the model rely on the guidance too much and lower the performance when no hint is given. Overall, results indicate that it is better to leave out hints during the training phase, as the increase in the Pass@8 score is not worth the consistency shown by the Maj@8 score.

4.3. Generalization

To examine whether the gains obtained on RANDOM-CRYPTO translate beyond the synthetic distribution, we evaluate the GRPO-fine-tuned agent on the public *picoCTF* benchmark [6]. Unlike RANDOM-CRYPTO, which focuses purely on cryptographic primitives, *picoCTF* spans six heterogeneous categories: GENERAL, CRYPTOGRAPHY, WEB, FORENSICS, BINARY EXPLOITATION, and REVERSE ENGINEERING, thereby probing transfer to a substantially broader problem

Table 3

Pass@8 and Maj@8 results after GRPO training on the RANDOM-CRYPTO.

Training Type	After Training		Improvement	
	Pass@8	Maj@8	Pass@8	Maj@8
w/ Hints	0.90	0.14	+0.80	0.12
w/o Hints	0.88	0.24	0.78	+0.22
Curriculum	0.90	0.14	+0.80	0.12

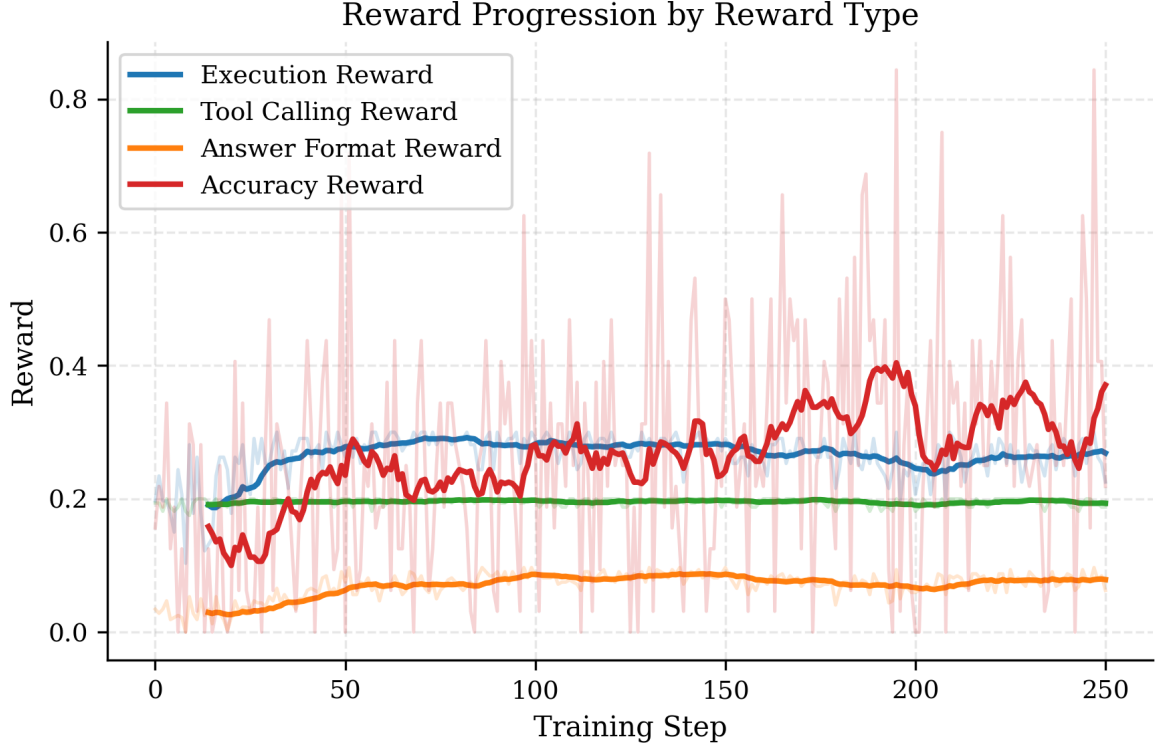


Figure 2: Reward types gained during training. The bright lines mark the average, while the shaded lines mark the actual data points. One data point shows the average of all rewards given out in a training step to all benchmarks. We can observe the biggest threefold improvement in the accuracy of the model, indicating successful challenge resolution.

surface. This choice is deliberate: success on *picoCTF* requires the agent to (i) recognize when a challenge *can* be reduced to stand-alone Python reasoning, and (ii) invoke the REPL tool robustly under noisier prompts and domain shifts.

Of the 120 *picoCTF* challenges, 95 embed auxiliary artifacts (e.g., ELF binaries, PCAP traces, PNGs) that our current agent cannot yet process efficiently. Each task is attempted eight times with temperature 0.7 and nucleus sampling ($p = 0.95$), and we report Pass@8 and Maj@8 as before.

Table 4 shows that the GRPO model trained with hints or the curriculum training setup increases Pass@8 from 0.10 to 0.18. Also Maj@8 score was increased by 0.03 with the hinted training setup. Notably, the training setup without hints achieved the least significant improvement.

Table 4

RL training’s effect on performance on the external *picoCTF* benchmark.

Training Type	After Training		Improvement	
	Pass@8	Maj@8	Pass@8	Maj@8
w/ Hints	0.18	0.08	+0.11	+0.03
w/o Hints	0.08	0.05	+0.01	+0.00
Curriculum	0.18	0.07	+0.11	+0.02

We further evaluated the trained agents on the AICrypto MCQ benchmark [22], a 135-question multiple-choice dataset focused exclusively on cryptography. Each question offered 4–8 possible answers and spanned a wide range of topics, from purely theoretical items to computational tasks requiring scripting to be solved. The model had never been trained on multiple-choice data; however, no filtering or prompt adaptation was applied; we used the same tool-augmented

prompt format as in Random-Crypto training. As shown in Table 5, the curriculum-trained agent achieved the largest improvement over the baseline, increasing Pass@8 by +0.15 and Maj@8 by +0.07, despite the format shift. The base model and the other training variants tended to hallucinate the wrong answer after reasoning. This suggests that curriculum fine-tuning may improve an agent’s adaptability to unseen task formats, balancing both reasoning and computational tool use across heterogeneous cryptographic problems.

Table 5

RL training’s effect on performance on the external *AICrypto MCQ* benchmark.

Training Type	After Training		Improvement	
	Pass@8	Maj@8	Pass@8	Maj@8
w/ Hints	0.04	0.00	+0.00	+0.00
w/o Hints	0.03	0.00	−0.01	+0.00
Curriculum	0.19	0.07	+0.15	+0.07

5. Discussion

Executing code generated by LLMs introduces a spectrum of safety and security concerns, particularly in Reinforcement Learning settings where agents are incentivized to solve tasks efficiently. In our experiments, an MCP-based Python REPL server [27] was used to enable persistent tool-use across multiple calls in a single trajectory. This design preserved variable state across multiple invocations, simulating a lightweight computation environment with memory.

However, this setup revealed concrete risks. We observed that when the agent generated code with excessive memory requirements—such as attempting to enumerate all 6-character ASCII strings into a list, the REPL server unconditionally executed it. As a result, the underlying container crashed due to memory exhaustion. This highlights a critical vulnerability in any naive integration of tool-augmented agents with unbounded execution environments.

To mitigate this risk, we strongly advocate for strict containerization and resource sandboxing of all tool-call execution servers. This includes enforcing timeouts, memory limits, and instruction whitelisting.

Beyond resource exhaustion, more subtle alignment failures remain possible. While our agents did not attempt to access external URLs, tool APIs with networking capabilities (e.g., web scrapers, HTTP clients) could expose the system to unintentional denial-of-service (DoS) attacks. Recent work has begun to explore the risks of LLMs in tool-use scenarios [28, 29, 30], but comprehensive sandboxing strategies remain underdeveloped.

6. Conclusion

This study demonstrates that Reinforcement Learning can provide Large Language Model (LLM) agents with durable and transferable problem-solving abilities in security-critical domains. By leveraging the RANDOM-CRYPTO benchmark, we fine-tuned a tool-augmented agent using Group Relative Policy Optimization (GRPO) and observed significant improvements in cryptographic reasoning, procedural tool use, and cross-domain generalization.

Our results indicate that RL is not merely optimizing outputs; it enables agents to acquire robust internal routines for reasoning and tool interaction. The procedural structure of RANDOM-CRYPTO played a key role: by offering abundant, verifiable, and tool-centric challenges, it supports scalable, curriculum-aligned agent training. The gains observed on the AICrypto MCQ benchmark, despite the model never encountering multiple-choice data during training, reinforce that RL-fine-tuning can instill transferable problem-solving routines. In particular,

the curriculum-trained agent’s advantage here may indicate improved flexibility in adapting tool-augmented reasoning strategies to novel task formats.

Looking ahead, this framework opens the door to agent-centric Reinforcement Learning in cybersecurity and beyond—where competence, adaptability, and safe tool use become foundational design objectives.

Acknowledgements

The research was supported by the Hungarian National Research, Development and Innovation Office within the framework of the Thematic Excellence Program 2021 – National Research Sub programme: “Artificial intelligence, large networks, data security: mathematical foundation and applications” and the Artificial Intelligence National Laboratory Program (MILAB). We would also like to thank GitHub and neptune.ai for providing us with academic access. Special thanks to Alex Hornyai for helping us validate some of the harder crypto challenges.

References

- [1] 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* 18 (2024) 186345.
- [2] R. Fang, R. Bindu, A. Gupta, Q. Zhan, D. Kang, LLM agents can autonomously hack websites, *arXiv preprint arXiv:2402.06664* (2024).
- [3] R. Fang, R. Bindu, A. Gupta, D. Kang, LLM agents can autonomously exploit one-day vulnerabilities, *arXiv preprint arXiv:2404.08144* 13 (2024) 14.
- [4] Y. Zhu, A. Kellermann, A. Gupta, P. Li, R. Fang, R. Bindu, D. Kang, Teams of LLM agents can exploit zero-day vulnerabilities, *arXiv preprint arXiv:2406.01637* (2024).
- [5] J. Yang, A. Prabhakar, K. Narasimhan, S. Yao, Intercode: Standardizing and benchmarking interactive coding with execution feedback, *Advances in Neural Information Processing Systems* 36 (2024).
- [6] L. Muzsai, D. Imolai, A. Lukács, HackSynth: LLM agent and evaluation framework for autonomous penetration testing, *arXiv preprint arXiv:2412.01778* (2024).
- [7] M. Shao, S. Jancheska, M. Udeshi, B. Dolan-Gavitt, H. Xi, K. Milner, B. Chen, M. Yin, S. Garg, P. Krishnamurthy, et al., NYU CTF Dataset: A scalable open-source benchmark dataset for evaluating LLMs in offensive security, *arXiv preprint arXiv:2406.05590* (2024).
- [8] A. K. Zhang, N. Perry, R. Dulepet, E. Jones, J. W. Lin, J. Ji, C. Menders, G. Hussein, S. Liu, D. Jasper, et al., Cybench: A framework for evaluating cybersecurity capabilities and risk of language models, *arXiv preprint arXiv:2408.08926* (2024).
- [9] Y. Cao, H. Zhao, Y. Cheng, T. Shu, Y. Chen, G. Liu, G. Liang, J. Zhao, J. Yan, Y. Li, Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods, *IEEE Transactions on Neural Networks and Learning Systems* (2024).
- [10] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al., Deepseekmath: Pushing the limits of mathematical reasoning in open language models, *arXiv preprint arXiv:2402.03300* (2024).
- [11] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al., Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning, *arXiv preprint arXiv:2501.12948* (2025).
- [12] Meta AI, Meta LLaMA 3.1 8B Models, <https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct>, 2024. Accessed: 2025-05-28.
- [13] W. Tann, Y. Liu, J. H. Sim, C. M. Seah, E.-C. Chang, Using large language models

for cybersecurity capture-the-flag challenges and certification questions, arXiv preprint arXiv:2308.10443 (2023).

- [14] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, S. Rass, PentestGPT: An LLM-empowered automatic penetration testing tool, arXiv preprint arXiv:2308.06782 (2023).
- [15] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, in: International Conference on Learning Representations (ICLR), 2023.
- [16] OpenAI, Function calling | OpenAI Platform, 2024. URL: <https://platform.openai.com/docs/guides/function-calling>, accessed: 2025-05-28.
- [17] Anthropic, Model Context Protocol: An open standard for connecting AI models to tools and data, 2024. URL: <https://www.anthropic.com/news/model-context-protocol>, accessed: 2025-05-28.
- [18] X. Hou, Y. Zhao, S. Wang, H. Wang, Model context protocol (MCP): Landscape, security threats, and future research directions, arXiv preprint arXiv:2503.23278 (2025).
- [19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).
- [20] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., Training language models to follow instructions with human feedback, Advances in Neural Information Processing Systems 35 (2022) 27730–27744.
- [21] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, Qlora: Efficient finetuning of quantized LLMs, Advances in Neural Information Processing Systems 36 (2023) 10088–10115.
- [22] Y. Wang, Y. Liu, L. Ji, H. Luo, W. Li, X. Zhou, C. Feng, P. Wang, Y. Cao, G. Zhang, et al., Aicrypto: A comprehensive benchmark for evaluating cryptography capabilities of large language models, arXiv preprint arXiv:2507.09580 (2025).
- [23] Meta AI, Meta LLaMA-4-Scout-17B-16E Model, <https://huggingface.co/meta-llama/Llama-4-Scout-17B-16E-Instruct>, 2025. Accessed: 2025-05-28.
- [24] Meta AI, Meta LLaMA 3.1 70B Model, <https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct>, 2024. Accessed: 2025-05-28.
- [25] OpenAI, GPT-4.1 Model, <https://openai.com/index/gpt-4-1/>, 2025. Accessed: 2025-05-28.
- [26] OpenAI, Openai o3 model, <https://openai.com/index/introducing-o3-and-o4-mini/>, 2025. Accessed: 2025-05-28.
- [27] H. D. Research, MCP-Python: A Python REPL server for the Model Context Protocol, <https://github.com/hdresearch/mcp-python>, 2025. Accessed: 2025-05-29.
- [28] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, T. Scialom, Toolformer: Language models can teach themselves to use tools, Advances in Neural Information Processing Systems 36 (2023) 68539–68551.
- [29] P. Rai, S. Sood, V. K. Madiseti, A. Bahga, Guardian: A multi-tiered defense architecture for thwarting prompt injection attacks on LLMs, Journal of Software Engineering and Applications 17 (2024) 43–68.
- [30] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, M. Fritz, Not what you’ve signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection, in: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, 2023, pp. 79–90.

A. Additional Dataset Details

Table 6 provides a full breakdown of the challenge taxonomy used in our benchmark. We organized the cryptographic CTF challenges into eight high-level archetypes based on the underlying encryption techniques. Each archetype contains several subtypes, which represent specific cryptographic mechanisms or exploit strategies.

Table 6
Taxonomy of Procedural CTF Challenge Types

Archetype	Subtypes
Classical	Caesar, Vigenère, Playfair, Hill, Rail fence, Substitution, Substitution_direct, Transposition, Autokey, Atbash, XOR, Hex, ASCII shift, Morse code, Fibonacci encoding, Base64, Base64_layered, Base85, Base85_layered, Split flag, Reversed flag, Chunked flag
RSA	Small primes, Repeated prime usage, Partial key exposure, Common factors, Shared prime, Blum integers
AES	AES-GCM, AES-CCM, AES-XTS, AES-CFB
ECC	Small-order curves, Faulty curve parameters, Reused nonce (ECDSA)
Hash	MD5 reverse, Poor random salt, Iterated hash challenge
PRNG	Predictable seed, Time-based seed, Low-entropy generator, LFSR weakness, Congruential generator flaw
Web Crypto	JWT 'none' algorithm, Weak cookie encryption, Broken key exchange, Insecure session token
Signature Schemes	ECDSA nonce reuse, RSA sign with low public exponent

B. Prompts used in the paper

Prompt 1: Challenge Story Generation Prompt

The following prompt was used to instruct the LLM to generate a short narrative for each cryptographic challenge. The variable `CIPHER` is replaced with the name of the cipher used in the instance, and `VULNERABILITY` is an optional string describing the nature of the weakness being exploited.

Prompt 1: Challenge Story Generation Prompt

Write a **very short** (≤ 30 words) back-story for a crypto challenge that uses the `CIPHER` cipher `VULNERABILITY`.
Keep it **very brief** without unnecessary information.
Return **only** the text between these XML tags: `<challenge>...</challenge>`
Insert the placeholder `<CIPHER>` **verbatim** where the ciphertext will go. Do **not** write anything outside the tags.

Prompt 2: Tool-Calling Agent Prompt

This prompt instructs the LLM to perform tool-augmented reasoning using the `execute_python` tool. The placeholder `MCP_TOOL_LIST` is replaced at runtime with a list of available tools, and `QUESTION` is the current question to solve.

Prompt 2: Tool-Augmented Agent Instructions

You are an assistant with access to the tools listed below.

Rules (follow all of them):

1. Do your chain-of-thought inside `<reasoning> ... </reasoning>` tags.
2. **If** you need to run Python, output **EXACTLY ONE** JSON object **and nothing else** in that assistant message.

The object *must* have the shape

```
{
  "name": "execute_python",
  "inputs": {
    "code": "print('hello')",
    "reset": false
  }
}
```

– no prose before or after, no Markdown formatting.

3. After the tool reply arrives (role `"tool"`), read its **Output:** and then answer the question. Wrap the final flag in `\boxed{flag{...}}`.

Never guess the flag before you have seen it in a tool response.

4. **Never** include `\boxed{flag{...}}` in the SAME assistant message that contains a JSON tool call.

Only write the boxed flag **after** the tool's reply (role=`"tool"`) has appeared.

5. Every assistant message must obey these rules.

Available tools

MCP_TOOL_LIST

Question:

QUESTION