Can Large Language Models reason? Investigating Open-Source Cryptographic Reasoning

Aleksandra Taneva

Supervisor: Assoc. Prof. dr. Zeki Erkin[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS),
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: Aleksandra Taneva
Final project course: CSE3000 Research Project
Thesis committee: Assoc. Prof. dr. Zeki Erkin, dr. Mitchell Olsthoorn

Abstract

Large language models (LLMs) have shown remarkable performance on mathematical competitions (AIME), and recently on the AICrypto benchmark. AICrypto has tested some of the best commercially-available models on capture-the-flag (CTF)-style cryptography challenges across multiple-choice challenges, proof, and open-ended questions. While they do well on the first 2 categories, the LLMs struggle to solve open-ended questions where advanced mathematical reasoning and creativity are required. This research follows an approach similar to the AICrypto benchmark, testing open-source LLMs and their ability to reason. Instead of assigning simple pass/fail scores, the LLM is evaluated qualitatively based on the logic it follows. We aim to demystify the black-box working of commercial LLMs, and potentially lead to developing an open-source framework for solving CTF challenges. The tests are performed on the Qwen3 32B LLM using an agentic ReAct framework. The most common failure modes are discussed, as well their causal factors.

1 Introduction

Modern cryptography is the basis of today's cybersecurity, and studying how both humans and large language models approach cryptographic CTF challenges provides insight into potential weaknesses, failure modes, and misplaced assumptions in automated security analysis. An LLM-based framework that supports structured reasoning and tool-assisted analysis could augment human experts in assessing post-quantum cryptographic schemes and improving the security of deployed protocols. Since large language models have demonstrated strong performance on mathematical benchmarks such as AIME [1], which are often interpreted as requiring multi-step reasoning, it is natural to investigate whether similar reasoning capabilities transfer to solving cryptographic CTF challenges. The tested models on the AICrypto benchmark have shown they're capable of solving multiple choice questions (MCQs) [2], and coming up with proofs, but their performance on open-ended questions can be improved. Demonstrating success on novel cryptographic challenges would suggest that LLMs can generalize beyond memorized solutions and apply problem-solving strategies, though this does not necessarily imply reliable step-by-step reasoning. Furthermore, relying on proprietary, black-box solutions is limiting for research. Developing an open-source solution ensures data privacy, reproducibility, control, and transparency.

However, cryptographic CTFs present challenges compared to other security domains. Unlike web or binary exploitation, which often rely on pattern recognition (e.g., recognizing SQL injection syntax), cryptography requires mathematical reasoning and numerical accuracy. A solution to a CTF challenge requires identifying the underlying encryption protocol, coding a script using tools such as SageMath, and making a correct calculation. This gives an LLM several places to get stuck and fail - identifying the type of problem, pattern matching vs complex reasoning, hallucinating a result, making a coding mistake, and having a calculation mistake are some examples. Even though current LLMs are capable of coding, they can hallucinate mathematical facts and fail to self-correct, as they are confident in their solution, which makes it hard to solve open-ended questions. That makes it challenging for LLMs to reason accurate solutions.

Recent advancements have attempted to automate CTF solving. A promising result is CTFAgent and its improved performance with the plan-and-execute paradigm [3], offering two modes, autonomous and Human-in-the-loop (HITL), with the latter surpassing 94% of the human teams on the challenges from the PicoCTF platform. However, CTFAgent primarily focuses on categories such as Web Exploitation, Reverse Engineering, Forensics, Binary Exploitation, and simple to medium cryptographic puzzles, while not providing in-depth details of the flow of solving questions. It also relies on state-of-the-art (SOTA) models to solve the challenges, and doesn't test the ability of more accessible, privately deployable models. It is also expected that the SOTA models perform better, as they are trained on more data. This creates a clear research gap: there is currently no accessible, open-source framework capable of effectively orchestrating LLMs to solve the complex, open-ended cryptographic challenges found in the AICrypto benchmark.

Consequently, the lack of accessible frameworks for cryptographic reasoning raises a fundamental question about the capabilities of current open-source models. If proprietary models struggle with transparency and reasoning, and fine-tuned open-source agents do not yet exist, we must determine if current state-of-the-art (SOTA) open-source models are sufficient to bridge this gap. This research is guided by the following primary research question:

RQ: How does access to tools and context within a locally-hosted, open-source agentic framework improve performance on the AICrypto benchmark compared to baseline SOTA LLMs?

To answer this, we investigate three sub-questions:

- What is the zero-shot performance of SOTA open-source models (specifically the Qwen3-based models) on the AICrypto benchmark compared to commercial alternatives?
- How does a specialized ReAct agent architecture [4] with access to domain-specific tools(SageMath) improve the success rate on complex cryptographic puzzles compared to a standard single-turn approach?
- What are the failure modes of the open-source agents, and can they be mitigated?

Our experiment is based on the ReAct agent architecture [4], which interleaves reasoning with tool use.

Section 2 reviews related work on LLM-based CTF solvers, reasoning benchmarks, and agentic architectures. Section 3 describes the experimental methodology, including the agent design, models, tools, and evaluation setup. Section 4 presents a qualitative analysis of the agent's performance

and failure modes on selected cryptographic challenges. Section 5 discusses responsible research considerations. Section 6 goes over the limitations and current state of the 30B class models. Finally, Section 7 concludes the paper and outlines directions for future work.

2 Background and related work

This section goes over the existing solutions to automated LLM-enhanced CTF solvers. We explore existing solutions as well as potential future solutions, with their respective applicability.

2.1 Fine-tuned models

There exists the idea that given enough training data, a fine-tuned model can be created that should be capable of solving challenges similar to the ones in the training dataset. However, cryptographic challenges often involve obscure mathematical constructions or custom-designed problems. Most such challenges are unique, and when solutions exist, they are typically found only in isolated write-ups rather than in large, structured datasets. As a result, training large language models on this domain is difficult, and one cannot reasonably expect most models to have been exposed to these problems during training. Another limitation is the presence of a fixed training cut-off: when models are presented with challenges created after that point, they are unlikely to have encountered them during training. This makes such challenges suitable for evaluating the reasoning capabilities of large language models rather than their ability to recall memorized solutions. Constructing a suitable training dataset would require collecting a large number of cryptographic problems and having expert practitioners provide high-quality solutions for model training. For example, the gamified platform CryptoHack offers challenges similar to those in the AICrypto benchmark, yet many of these lack publicly available solutions [5]. Once solved, such challenges could serve as valuable resources for future model training or for in-context learning [6], where solved examples are provided at inference time.

For model training, a suitable dataset would need to be constructed by scraping challenge write-ups from repositories such as CTFArchive [7] and applying data decontamination procedures as described in the Cyber-Zero agent [8; 9]. The Cyber-Zero agent is reported to achieve SOTA performance on several benchmarks; however, these results are difficult to reproduce, as the model weights are not publicly released. Reproducing the reported performance would therefore require access to a SOTA base model for training, which is impractical to host privately due to substantial computational requirements or high inference costs.

Even if such training were feasible and resulted in improved performance, it would primarily demonstrate a specialized model's ability to solve problems similar to those seen during training, rather than a genuine improvement in general reasoning capability. This underscores that, with sufficient task-specific training, performance gains may reflect memorization or pattern matching rather than improved reasoning.

2.2 Reasoning capabilities

Reasoning capability needs to be tested on complex challenges. Results show that SOTA LLMs are performing well on competitions such as AIME, where the answers are correct numerical values. Interesting results are shown on the 2025 USA Mathematical Olympiad (USAMO), for which participants qualify through prior competitions, such as AIME. There, the best-scoring LLM gets an average score of less than 25%, suggesting reasoning flaws. Petrov et al. report multiple failure modes, such as flawed reasoning, limited creativity, and unjustified assumptions[10]. To further explore this topic, we test open-source models on cryptographic open-ended questions, which need a practical solution consisting of both a reasoning part and a coding part.

2.3 Benchmark

Inspired by the AICrypto benchmark [2], which explores the solve rate of different SOTA LLMs on 3 types of challenges—Multiple choice questions, Open-ended coding challenges and Proof questions—we want to dive deeper into why, despite the huge investments in recent years into developing language models, there still seems to be an inherent inability to solve complex challenges. Rather than measuring aggregate solve rates across the full benchmark, we focus on understanding why models fail, even on comparatively simple instances. We have decided that multiple-choice questions are not interesting, since the model is either trained or not in answering them, which tests memorization and not reasoning. And we have selected Open-ended questions because LLMs are promised to be good at both mathematical tasks and coding challenges, and this is where we can best test reasoning.

Unlike the exhaustive evaluation conducted by AICrypto, our study is intentionally selective. Due to limited computational resources and the time it takes to do repeated multi-step inference, we do not attempt to complete the full benchmark. Instead, we perform an in-depth analysis on a small number of static challenges that are easy to interpret and solve by humans. This choice allows us to isolate reasoning flaws without conflating them with task difficulty or benchmark scale. For each challenge, we log the model's intermediate reasoning paths, tool invocations, retries, and failure patterns. This enables a qualitative analysis of systematic pitfalls.

2.4 ReAct agent architecture and tool calling

Traditional approaches to LLM-based automation are divided into two categories: reasoning methods that generate chain-of-thought (COT) explanations without taking actions, and action-based methods that execute tools without reasoning traces. The ReAct (Reasoning + Acting) architecture, introduced by Yao et al., unifies these two methods by interleaving reasoning traces with action execution in a single generation loop [4].

In the ReAct paradigm, the model switches between three phases:

- <think>block - the model classifies a problem and thinks of a way to solve it.

- "action:" - the model decides which tool should be executed next
- <think>- where the model observes the execution, judges correctness and decides to continue or not with the problem, based on its confidence.

This cycle repeats until the model reaches a solution according to itself or exhausts its allowed attempts.

This architecture mirrors human problem-solving behavior, where experts continuously reflect on intermediate results, adjust their strategies, and seek additional information when stuck. For CTF challenges specifically, a solver must read challenge files, hypothesize the cipher or vulnerability, write and execute code to test the hypothesis, interpret error messages or partial outputs, and iteratively refine the approach.

Unlike black-box action sequences, ReAct agents produce human-readable logs that show how a particular tool was invoked and how the model interpreted its output. This logging allows us to distinguish between errors in reasoning (e.g., misidentifying a cipher type, solution mode), errors in execution (e.g., syntax bugs in generated code) or errors in its self-confidence. However, the ReAct framework introduces new failure modes not present in pure reasoning systems. The model must correctly format tool calls and output answers in a well-structured manner. Failing this leads to parsing problems. Additionally, the iterative nature can lead to strategy loops, where the model repeatedly does the same thing. We observe these failure modes in our experiments.

For our experiments, the following tools were allowed to the agent:

- **execute_python**: Executes Python code in an isolated environment.
- **execute_sage**: Executes SageMath code, providing access to advanced number-theoretic functions such as polynomial factorization, elliptic curve arithmetic, and lattice reduction algorithms commonly used in cryptanalysis.
- **read_file**: Reads the contents of a file from the challenge workspace. This allows the agent to inspect ciphertexts, encryption scripts, or any supplementary data provided with the challenge.
- **write_file**: Writes content to a file in the workspace. Used for saving intermediate results or preparing input files for subsequent computations.
- **submit_flag**: Submits a candidate flag string for verification against the expected solution. The agent receives binary feedback (correct or incorrect) without hints about partial correctness.

3 Methodology

To answer the research questions asked in the previous section, we measure the improvements over a baseline agent in an ablation study. Our methodology progresses from a zero-shot baseline agent to a ReAct agent.

3.1 Experimental Setup and Infrastructure

All experiments are conducted on the DelftBlue High-Performance Computing (DHPC) cluster [11]. The computational environment consists of a GPU node equipped with two NVIDIA A100 GPUs (80GB VRAM each). Main scripts and logic are stored on the login node, which has access to the internet. The models, logs, and large files are stored on the cluster's scratch filesystem. To manage dependencies and ensure reproducibility, we utilize Apptainer as recommended by the HPC documentation. A custom container image (`qwen3-vllm.sif`) was built containing vLLM v0.4.x, PyTorch with CUDA 12.1 support, and cryptographic libraries including PyCryptodome, SymPy, gmpy2, and SageMath.

Model Serving

The Qwen3-32B model is served using the vLLM inference engine [12], which provides high-throughput batched inference via an OpenAI-compatible API. We configure vLLM with the following parameters:

- **Tensor parallelism:** 2 (distributing the model across both A100 GPUs)
- **Maximum context length:** 32,768 tokens (extended via YaRN RoPE scaling [13])
- **GPU memory utilization:** 90%

The server starts in the background and is checked for readiness before the benchmark begins. The job terminates if the server crashes to not waste compute resources.

Agent Orchestration

The benchmark orchestrator (`run_qwen3_orchestrator.py`) iterates over challenge directories, spawns isolated sandbox containers for code execution, and manages the ReAct agent loop. Each challenge is executed under the following constraints:

- **Maximum attempts:** 10 retries per challenge.
- **Turns per attempt:** 20 LLM calls before forced termination.
- **Time limit:** 1600 seconds (26 minutes) per challenge.
- **Execution timeout:** 120 seconds per tool invocation.

Code executes inside the Apptainer container with the `--contain` flag to prevent network access and filesystem escape. The agent communicates with the vLLM server over localhost, and all reasoning traces, tool calls, and outputs are logged to JSONL files.

3.2 Models and Baseline:

We use SOTA mixture-of-experts (MoE) reasoning models, capable of tool calling and clear output, such as Qwen 3 32B [14], which fits on the GPUs available. For bigger models, more gpus need to be available.

Baseline agent

The baseline agent receives all the files of the challenge, and has to output a solution in one go, without tool calling. The performance so far on both Qwen 3 32B and DeepSeekR1-distilled-Qwen is 0% solve rate. As this was also expected,

since the agents can't run code and are just supposed to know the answer, no extensive reasoning analysis has been conducted. If the agents had solved any of the questions, it would suggest that there is data contamination.

Tool-Enhanced Agent
The tool-enhanced agent operates in a ReAct loop, alternating between reasoning and action phases. The agent implementation (`qwen_agent_worker.py`) uses Qwen3's native thinking mode, which structures generation into two phases:

1. **Thinking Phase:** The model generates internal reasoning within `<think>...</think>` tags, classifying the problem type, formulating hypotheses, and planning the next action. This phase is allocated up to 4,096 tokens.
2. **Action Phase:** After reasoning, the model outputs a structured action in a predefined format (e.g., `Action: execute_python` followed by a code block). The orchestrator parses this output and invokes the corresponding tool.

The agent has access to five tools: `execute_python` , `execute_sage` (for advanced number-theoretic computations), `read_file`, `write_file`, and `submit_flag`. Tool outputs are truncated to 5,000 characters to prevent context overflow. The flow of the agent's work can be seen in Figure 1. It loops and is guided to make tool calls, the output of the agent is logged, whenever possible, as sometimes a parsing error breaks the flow, leading to agent retrying and repeating the same strategy.

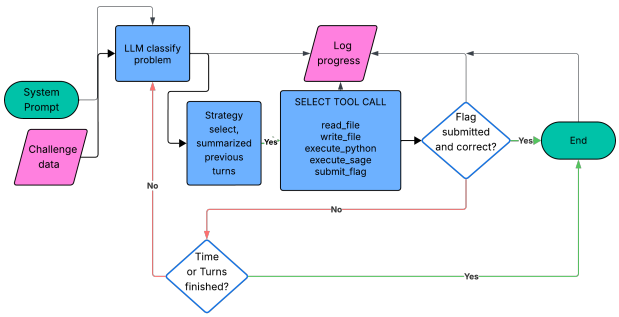

Figure 1: Agent logic flow

Context Management
To handle long conversations within the 32K context window, the agent implements history summarization. When the conversation exceeds 50,000 characters, earlier turns are compressed into a summary of strategies tried and their outcomes, preserving only the most recent exchanges. This allows the agent to maintain awareness of failed approaches without exceeding model limits.

Attempt and Retry Logic
Each challenge allows up to 10 independent attempts. If an attempt exhausts its 20-turn limit without success, the agent restarts the conversation but gets a summary of previously tried strategies. This should encourage trying new strategies and not repeating mistakes.

Logging and running simulations
Simulations are submitted to the queue of DHPC and run once, with detailed logging of the output of the LLM, of what it is currently doing, and which challenge it is on, so that for the next run, the agent can be enhanced and see how the thinking limit and the prompt influence its response.

The agent receives a structured system prompt defining available tools, attack strategies by category (RSA, Classic, Stream, etc.), and behavioral constraints. The complete prompt is provided in the repository.

4 Analysis of Results

In this section, we will discuss the different failure modes of our Qwen3 based ReAct agent. So far, all of the runs and strategies (different prompts, different thinking budgets) have not resulted in correctly solving a challenge from the open-ended questions.

4.1 Reasoning pitfalls

Coming up with a correct strategy to solve a problem is not the only problem. For example, the first challenge in the benchmark is the simple Greek cipher:

Greek Cipher
The challenge provides a ciphertext written entirely in Greek lowercase characters:

$\kappa\sigma\kappa\ \omega\pi\nu\ \alpha\zeta\pi\lambda\ \iota\eta\sigma\iota\ \chi\nu\sigma\nu\theta\ \mu\sigma\gamma\theta\sigma\rho\ \lambda\sigma\theta\ \zeta\pi\iota\ \iota\eta\gamma\ \delta\sigma\rho\theta\ \iota\ \psi\gamma\rho\theta\pi\zeta\ \sigma\zeta\ \eta\sigma\theta\iota\pi\rho\omega\ \theta\nu\theta\psi\gamma\mu\iota\gamma\kappa\ \pi\delta\ \nu\theta\sigma\zeta\epsilon\ \gamma\zeta\mu\rho\omega\psi\iota\sigma\pi\ \zeta?\ \tau\gamma\ \zeta\gamma\sigma\iota\eta\gamma\rho.\ \kappa\sigma\kappa\ \omega\pi\nu\ \alpha\zeta\pi\lambda\ \iota\eta\sigma\iota\ \chi\nu\sigma\nu\theta\ \mu\sigma\gamma\theta\sigma\rho\ \lambda\sigma\theta\ \psi\rho\pi\xi\sigma\xi\omega\ \delta\nu\gamma\zeta\iota\ \sigma\zeta\ \epsilon\rho\gamma\gamma\alpha?\ \tau\gamma\ \zeta\gamma\sigma\iota\eta\gamma\rho.\ \sigma\ \sigma\alpha\gamma\ \eta\pi\lambda\ \epsilon\rho\gamma\ \gamma\alpha\ \mu\eta\sigma\rho\sigma\mu\iota\gamma\rho\ \pi\pi\alpha\ \iota\eta\pi\nu\epsilon\eta,\ \gamma\upsilon\gamma\zeta\ \sigma\delta\ \sigma\ \mu\sigma\zeta'\iota\ \rho\gamma\sigma\kappa\ \iota\eta\gamma\tau\ .\ \sigma\mu\iota\delta\{\sigma\_\epsilon\nu\gamma\theta\theta\_\nu\theta\sigma\zeta\epsilon\_\tau\sigma\zeta\omega\_\epsilon\rho\gamma\gamma\alpha\_\mu\eta\sigma\rho\sigma\mu\iota\gamma\rho\theta\_\kappa\sigma\kappa\zeta'\iota\ \_\theta\iota\pi\psi\_\omega\pi\nu.\_\lambda\gamma\_\psi\sigma\omega\gamma\kappa\_\sigma\_\tau\nu\theta\iota\_\theta\sigma\omega.\mu\pi\zeta\epsilon\rho\sigma\iota\theta!\}$

Expected Solution This is a monoalphabetic substitution cypher where each Greek letter maps to its positional Latin equivalent ($\alpha \rightarrow a$, $\beta \rightarrow b$, etc.): **Flag:** lactf{i_guess_using_many_greek_characters_didn't_stop_you._well_played_i_must_say.congrats!}

Agent Behavior The agent attempted to use the Google Translate API (`googletrans`) rather than recognizing the cipher as a character mapping. This failed because:

1. The text is not Greek *language*, it is English encoded in Greek *letters*
2. Network access was disabled inside the container.
3. It makes an incorrect tool call to Google Translate, trying to translate instead of transliterate.

It did recognise correctly that it is a substitution cipher, and also recognised that frequency analysis can be used, but it didn't apply these strategies correctly. After repeated API failures, the agent exhausted its 40 turns without attempting the correct approach. This shows multiple failure modes: **hallucinated tool call, incorrect strategy application, incorrect conclusion**. The agent struggles until the time runs out, leading to another failure to solve.

In comparison, while prompting without hints to Gemini 3 Pro, the SOTA paid model is capable of solving it in one shot. Despite not having access to the problems in the AICrypto

benchmark in its training data, the model in its core makes tool calls and searches the internet. It finds the solution in a publicly available write-up and circumvents the reasoning process by retrieving the solution. This leads to a problem in benchmarking SOTA models, as they are a complicated black-box solution that utilizes many tools and a vast context. Even without data present during training, they leverage publicly available information to solve problems.

4.2 Small Model Constraints

The small model used in this experiment reveals an additional failure mode. Despite requiring a specific output format—consisting of a `think` block and an `action` block to ensure parsability—the model sometimes breaks this structure, leading to parsing errors. When the output cannot be parsed, the subsequent tool call cannot be extracted, resulting in a generalized failure. Figure 2 shows the distribution of correct tool calls.

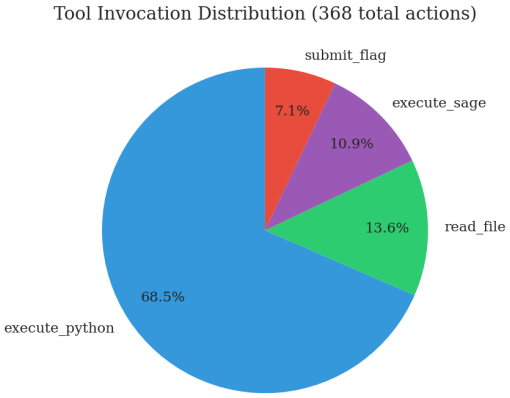

Figure 2: Tool invocation distribution, across the provided tools.

Whereas Figure 3 identifies the primary causes of failure. This is difficult to visualize, since the agent has multiple failure modes per run, as each turn can have a different failure.

We define success in our measurements as the model following the required format and executing the tool call, even if the challenge itself is not solved. Figure 4 summarizes all turns and attempts, and quantifies the different failure types. Most runs exhaust the allocated turns and time while repeatedly trying different approaches. We impose a time limit because the model often repeats the same strategy; for example, one run spent three and a half hours attempting to solve a long Caesar cipher.

To encourage convergence, when time is nearly exhausted or there are no remaining turns, the agent receives a reminder to attempt the best possible approach and submit an answer. This behavior sometimes results in hallucinated flags. Examples include:

- `nwhuy{ilnwpwu_oweq_wowgwj_gwpv_ivgw_wrwn.}`
- `THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG.`
- `CTF{example_flag}`
- `CTF{frequency_analysis_flag}`
- `CTF{f4k3_fl4g_f0r_3x4mpl3}`

Interestingly, some outputs were similar to known correct flags, such as `picoctf{caesarcipherisfun}` and `n00bz{vigenerecipherisfun}`. This may indicate that the questions are outdated and that the model has previously encountered similar answers, leading to near-correct outputs.

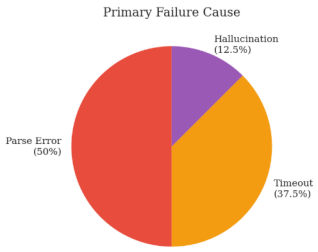

Figure 3: Main failure modes observed in the logs

Challenge	Tries	Turns	Flag?	Correct	Format	Errs	SuccRate	Time	Techniques
01-greek-cipher	3	40	NO	NO	WRONG	29	15%	1506	Translation API
02-selamat-pagi	3	40	YES	NO	WRONG	16	57%	1841	Caesar cipher
03-valentines-day	3	40	NO	NO	WRONG	2	50%	1099	Unknown approach
04-too-loud-to-yap	2	20	YES	NO	WRONG	12	40%	1595	ROT shift, Frequency analysis
05-three-line-crypto	4	60	YES	NO	WRONG	31	43%	1310	Stream cipher, XOR cipher
06-funny-cipher	3	40	YES	NO	WRONG	25	34%	1682	Substitution cipher, Frequency anal
07-crypto-long-caesar	2	20	YES	NO	WRONG	15	25%	1097	Caesar cipher
08-Vinegar	3	40	YES	NO	WRONG	13	41%	1318	Caesar cipher
09-Vinegar2	2	20	YES	NO	WRONG	19	5%	1311	Unknown approach
06-my-array-generator	2	20	NO	NO	WRONG	-	100%	1313	Unknown approach
11-bivalves	3	40	NO	NO	WRONG	12	43%	1202	Unknown approach
12-Hyper512	2	20	NO	NO	WRONG	17	0%	919	Stream cipher, XOR cipher
13-lf3r	2	20	NO	NO	WRONG	17	0%	1274	Stream cipher
01-blue-hens-2023	2	20	YES	NO	WRONG	18	0%	1406	GCD/Euclidean, Factorization
02-blue-hens-2023	2	20	NO	NO	WRONG	4	20%	992	Math library, CRT attack
06-ravin-cryptosystem	3	40	YES	NO	WRONG	40	0%	1158	Factorization, RSA attack

Total: 16 | Solved: 0 (0%) | Correct Format: 0 | Errors: 270 | Avg Time: 1314s | Avg Turns: 31.2

Figure 4: Summary of challenge runs, techniques used, and success rate of tool calls.

4.3 72B model

While most of the experiments were done with a 32B model, as that is what comfortably fits on the provided hardware, an experiment was run on a Qwen2.5-72B-Instruct model to evaluate potential performance gains from increased model capacity. However, the model's input is limited to 16384 context tokens. Even if initial requests are successful, the model must recall all information about files, previous attempts, and failed code to use the ReAct framework, leading to poor requests. ReAct relies on the model's ability to reason over previously observed states, including file contents, earlier hypotheses, failed code attempts, and the outcomes of executed actions. Since the model has no persistent memory across calls, all relevant information must be included in the prompt at each iteration. As the CTF challenges progress, the context reaches the limit leading to bad requests. The API requires a constraint that the total number of prompt tokens plus the maximum number of generated tokens must not exceed the model's context window. Once this limit was exceeded, requests were rejected with a `400 Bad Request` error, preventing further progress. Consequently, later iterations became infeasible, as reducing the prompt size required discarding information that was necessary for correct reasoning and preventing reasoning loops. This illustrates a limitation

of applying iterative, tool-augmented reasoning frameworks such as ReAct to long-running tasks under fixed context budgets. For complex reasoning tasks, which require several iterations, architectural changes such as context compression or external memory can be more important than increasing the size of the model. In contrast, production systems such as ChatGPT and Gemini mitigate context limitations through architectural mechanisms external to the language model itself. These systems maintain a persistent conversation state outside the model and selectively inject only the most relevant information into each prompt. Older interactions are automatically summarized or abstracted, while large auxiliary artifacts such as files or documents are accessed via retrieval-based mechanisms rather than being included verbatim.

As a result, long-horizon tasks are supported not by unbounded context windows, but by a combination of external memory, automated context compression, and dynamic relevance filtering. This allows such systems to sustain extended interactions while remaining within fixed model context limits. In comparison, the ReAct-based setup used in this work relies on replaying prior states through the prompt, making it susceptible to context exhaustion in iterative problem-solving scenarios.

5 Responsible Research

While we work towards having an agent capable of autonomously solving mathematically grounded complex CTF challenges, this research implies that a malicious actor can use the techniques here, substituting the model with a paid one/deploying a bigger private model, to create an agent capable of finding weaknesses in different systems.

However, we argue that creating such agents is inevitable, and open-sourcing the work can also lead to more information on the possible security vulnerabilities, leading to better defense mechanisms. Since the code is publicly available, the baseline models can be upgraded with more powerful ones, and this framework can be used for testing cryptographic implementations.

All experiments conducted in this research were executed privately on the DelftBlue high-performance computing cluster. No external services or internet access were used during model execution; all computation remained contained within the university infrastructure. Special care was taken to respect shared compute resources: jobs were submitted with conservative runtime limits (typically 6–8 hours), and failure condition handling ensures crashing jobs would terminate early rather than waste energy or GPU time. This approach reflects a social responsibility toward fair resource usage in shared academic environments.

From a security perspective, all cryptographic challenges were executed inside isolated containers with restricted filesystem access. The agent had no global file access, no network connectivity, and no ability to execute privileged or system-altering commands. This sandboxed setup prevents unintended side effects and ensures that potentially unsafe code generated by the model cannot affect the host system or other users.

Finally, this work prioritizes reproducibility and transparency. The complete framework, including agent logic, prompts, orchestration code, and experiment configuration, is released. This allows other researchers to reproduce the results, verify findings, and extend the framework with alternative models or strategies. By enabling continuation and scrutiny of the work, we aim to contribute to responsible and sustainable research practices in LLM-based security research.

6 Discussion

The results of this study highlight a gap between the advertised reasoning capabilities of LLMs on benchmarks such as AIME and LiveBench. Such benchmarks suggest that models like Qwen3 32B are capable of advanced logic, while our results show that this doesn't translate to the multi-step, precise solving of CTF open-ended questions.

6.1 Knowledge vs Application

A recurring pattern was the agent's ability to recognize the type of problem and name the solution, but its inability to apply the complete strategy that expert humans could. The LLM lacks the procedural reasoning required to map the concept to fully functioning code. Furthermore, although summarizing previous turns and iterations into a single prompt should theoretically lead to improvements, the LLM demonstrates a limited multi-turn attention span. While the ReAct framework provides transparency and mirrors human problem-solving, it introduces instability. The model struggles to adhere to the required "thinking vs. acting" format, leading to parsing errors and disrupted action flows. It exhibits high confidence in its own flawed reasoning. Such general-purpose Mixture-of-Experts models may struggle to act reliably as agents, as they likely have not been fine-tuned for tool-use orchestration.

6.2 Contamination and True Reasoning

Our comparison to SOTA model on a simple exercise reveals an issue in current AI evaluation: data contamination. Despite having benchmarks with questions and problems dated after an LLM's training, most paid models make use of tool calls and access publicly available data on the internet to cheat their way into a solution, which obfuscates their reasoning ability. While enforcing an offline environment for our self-deployed model, we isolate the model's reasoning abilities. The resulting 0% success rate serves as a sobering baseline, suggesting that "solving" a CTF is often more about retrieval than genuine cryptographic cryptanalysis.

6.3 Hardware limitations

The decision to use a 32B model due to hardware constraints (2xA100 GPUs) limits our full understanding of the reasoning of all models, as literature claims that reasoning scales with model size. Larger open-weights models (e.g., 70B+ parameters) may possess the necessary depth to self-correct during the thinking phase, which the 32B models consistently failed to do.

7 Conclusions and Future Work

This research investigated the feasibility of using open-source Large Language Models to autonomously solve cryptographic CTF challenges within an offline, tool-augmented framework. We conclude that current open-weight models in the 30B parameter class are insufficient for autonomous cryptographic reasoning.

While the agentic framework successfully provided transparency into the model's decision-making process, it did not improve the solve rate. Our analysis identified three primary failure modes:

- Hallucinated Logic, where models invented mathematical relationships (e.g., invalid Caesar shifts of wrong length)
- Tool Misuse, such as attempting to use online APIs in an offline environment
- Strategic Loops, where the agent failed to pivot away from unsuccessful strategies.

These findings demystify the "black box" of LLM performance, demonstrating that high scores on general math benchmarks do not guarantee competence in specialized, multi-step exploits.

7.1 Future Work

To bridge the gap identified in this study, future research should focus on these areas:

- **Scaling Model Size:** Replicating this benchmark with larger open-source models, such as Qwen-72B or DeepSeek 236B, would determine if the reasoning failures observed are intrinsic to LLMs or simply a limitation of the 32B scale.
- **In-Context Learning (RAG):** Enhancing the agent with a Retrieval-Augmented Generation (RAG) system containing a playbook of solved CTF write-ups. This would allow the model to perform pattern matching against known solutions rather than reasoning from first principles.
- **Domain-Specific Fine-Tuning:** As discussed in Section 2, future work should involve training a model specifically on "trajectories" of successful CTF solutions. This would teach the model the process of using tools (e.g., SageMath) correctly, rather than just the theory of cryptography.

By moving from generalist models to specialized, context-aware agents, we believe open-source frameworks can eventually approach the performance of human experts in automated vulnerability assessment.

References

[1] Artificial Analysis. Aime 2025 evaluation, 2025. [Online; accessed January 25, 2026].

[2] Yu Wang, Yijian Liu, Liheng Ji, Han Luo, Wenjie Li, Xiaofei Zhou, Chiyun Feng, Puji Wang, Yuhan Cao, Geyuan Zhang, Xiaojian Li, Rongwu Xu, Yilei Chen, and Tianxing He. Aicrypto: A comprehensive benchmark for evaluating cryptography capabilities of large language models, 2025.

[3] Yuwen Zou, Jia Liu, and Wenjun Fan. Ctfagent: An llm-powered agent for ctf challenge solving. *Journal of Information Security and Applications*, 96:104305, 2026.

[4] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

[5] CryptoHack. Cryptohack: Learn and practice cryptography online, 2025. [Online; accessed 06-Jan-2026].

[6] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work?, 2022.

[7] Sajjad J̈J̈ Arshad (sajjadium). Ctf archives: Collection of ctf challenges. GitHub repository, 2025. Accessed: 2025-01-06.

[8] Terry Yue Zhuo, Dingmin Wang, Hantian Ding, Varun Kumar, and Zijian Wang. Cyber-zero: Training cybersecurity agents without runtime. *arXiv preprint arXiv:2508.00910*, 2025.

[9] Terry Yue Zhuo, Dingmin Wang, Hantian Ding, Varun Kumar, and Zijian Wang. Training language model agents to find vulnerabilities with ctf-dojo. *arXiv preprint arXiv:2508.18370*, 2025.

[10] Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunović, Nikola Jovanović, and Martin Vechev. Proof or bluff? evaluating llms on 2025 usa math olympiad, 2025.

[11] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 2). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

[13] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models, 2023.

[14] Qwen Team. Qwen3 technical report, 2025.