Do Agent Architectures Matter for Crypto CTFs?

An Empirical Evaluation of LLM Architectures on the AICrypto Benchmark

Querijn Voet[1]

Supervisor(s): Dr. Zeki Erkin[1]

[1]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: Querijn Voet
Final project course: CSE3000 Research Project
Thesis committee: Dr. Zeki Erkin, Dr. Mitchell Olsthoorn

Abstract

Large Language Models (LLMs) demonstrate gold-medal performance in pure mathematics but continue to struggle in professional Capture-The-Flag (CTF) cybersecurity competitions, where the goal is to obtain a flag string as proof. While models can solve textbook equations, the iterative engineering required for cryptographic challenges often exceeds their single-pass capabilities. We explore whether this performance gap is a limitation of the base models or if it can be bridged by using different agent architectures. This paper presents a systematic evaluation of five distinct control flow architectures (i.e., how the agent plans, uses tools, and iterates) applied to the AICrypto CTF benchmark [1]: Chain-Of-Thought (CoT) as the single-pass baseline, ReAct as a reactive method, and three planning/search approaches (ReWOO, ADaPT, and LATS). To isolate the impact of reasoning structures, we restrict the scope to static challenges, which are puzzles that do not require network interaction. Using a fixed base model and tool stack (SageMath execution, command execution and flag submission), we analyze the trade-offs between success rate, time-to-solve, and token consumption to determine if the computational overhead of different architectures contributes to the solving capability. Overall, there is only a 5.4% gain in success rate by using agent architectures compared to the single-pass baseline (35.1% vs. 29.7% success rate), primarily by iteratively debugging solutions rather than discovering new cryptographic ideas. The higher success rates from complex planning/search architectures come at the expense of higher token and time costs, with ADaPT consuming 93% more tokens on average compared to the baseline in successful challenges.

1 Introduction

Large Language Models (LLMs) have achieved major breakthroughs in mathematical problem solving. In mid-2025, frontier models reached the International Mathematical Olympiad (IMO) gold-medal threshold, solving complex problems that require deep logical reasoning [2; 3; 4]. This progress suggests that models possess the core mathematical fundamentals necessary for high-level problem solving. However, this success in pure mathematics does not transfer to the domain of cybersecurity. When entered into professional Capture The Flag (CTF) competitions such as PlaidCTF or DEF CON Qualifiers, where contestants write code to solve security puzzles to obtain a flag string as proof, state-of-the-art models fail to solve any challenges [5]. This difference suggests a gap, models can solve an equation on paper but fail when that equation is part of a live, interactive challenge environment.

Cryptography CTF challenges differ from pure mathematics benchmarks in a couple of ways. First, challenges often rely on less common primitives or custom cipher variants that appear rarely in the training data. Second, the solution process is inherently interactive. A solver cannot find a paper-and-pencil answer. Instead, they must write scripts using specialized libraries like SageMath to recover the flag string. This environment demands an iterative engineering process where the agent must observe errors, debug code, and refine its mathematical strategy based on live feedback. Compared to a single-pass LLM, the lack of a feedback loop means that a single syntax error leads to immediate failure.

Current research in LLM agents proposes various control flow architectures, the control loop that decides when to plan, call tools, and reiterate to handle such multi-step tasks. The Chain-Of-Thought (CoT) baseline generates and executes the solve script in a single-pass [6], while architectures like ReAct interleave reasoning with tool calls (e.g., executing code, commands or submitting a flag) [7]. ReWOO attempts to separate planning from execution to save costs [8]. More advanced methods like LATS treat the problem as a search space, exploring multiple ideas simultaneously [9] and ADaPT plans and decomposes a problem into subproblems when the LLM is not confident it is able to solve the challenge directly [10]. While several of these architectures perform better than a CoT baseline on generic benchmarks like HotPotQA [11] and HumanEval [12], there is a lack of empirical evidence regarding their performance in the domain of cryptographic CTF challenges. Furthermore, it remains unclear if the complexity of these agents is necessary for static cryptographic challenges (challenges that do not require network interactions), or if a simple CoT baseline is sufficient when provided with a fully interactive sandbox.

We propose a comparative study that isolates the agent control-flow architecture variable. We fix the base model and the available tool stack (Linux shell command execution, interactive SageMath code execution and flag submission), including a Dockerized Linux environment to ensure that any difference in performance can be attributed solely to how the agent plans, executes tools, and iterates. We evaluate these architectures on the static subset of the AICrypto benchmark, which provides a categorized dataset of cryptographic challenges containing RSA, elliptic curves, and lattice-based schemes. By removing network-dependent challenges and retrieval tools, we focus on the agent's ability to reason, code, and debug mathematically complex attacks.

In this paper, we implement and evaluate five agent control flow architectures on the static CTF challenges of the AICrypto benchmark. We map architecture strengths to specific cryptographic subdomains, identifying where simple reasoning suffices and where agentic overhead becomes a liability. Finally, we provide a cost-efficiency analysis that compares success rates, measured in Pass@1, which is the percentage of problems an agent solves correctly on its first attempt against token consumption and duration.

2 Background and Related Work

This section explains the technical foundations for autonomous cryptanalysis. We first compare formal reasoning and iterative engineering, followed by an analysis of existing agentic control flows and the specialized requirements of the AICrypto benchmark.

2.1 Formal Reasoning vs. Iterative Engineering

The difference between solving math problems and cryptographic CTF challenges comes down to a shift from closed-form to open-ended reasoning. In pure mathematics, a solution is often a static proof or a numerical value, while in a CTF challenge, it is an engineering task. An agent must translate a mathematical attack into a functioning script that handles integer arithmetic, file reading, and data conversion. Recent studies show that LLMs excel at textbook cryptography where the solution is a known formula [1]. However, they struggle when presented in an interactive environment, for example, a standard LLM without a feedback loop fails immediately upon a syntax error. This paper focuses on Agentic Control Flows, the logic that enables a model to observe errors, debug code, and refine its strategy.

2.2 Agent Architectures

The agent architectures can be categorized into four categories:

- **Linear Reasoning (Chain-of-Thought): [6]** CoT is the baseline approach where the model reasons and finishes with a tool call in a single turn. Because it only runs in a single turn, it lacks the ability to correct errors once execution is finished.
- **Reactive Loops (ReAct): [7]** ReAct (Reason + Act) alternates thought generation with tool execution. The agent observes the output of a previously executed tool (e.g., a SageMath or command execution) and thinks about this output to decide what to do next.
- **Planning-Based (ReWOO & ADaPT):**
  - **Reasoning Without Observation (ReWOO): [8]** attempts to separate planning from execution to save tokens, generating a blueprint of dependent tool calls upfront.
  - **As-Needed Decomposition (ADaPT): [10]** manages complexity by recursively breaking tasks into sub-problems only when a direct attempt fails, potentially offering a balance between efficiency and depth.
- **Language Agent Tree Search (LATS): [9]** Language Agent Tree Search approaches the solution process as a search problem over a decision tree. Using Monte Carlo Tree Search (MCTS) logic, it explores multiple potential exploit strategies, backtracks when a path fails, and utilizes self-reflection to evaluate states. This represents the highest computational cost and complexity.

2.3 AICrypto

The AICrypto benchmark [1] is a suite of cryptographic tasks categorizing problems into subdomains such as RSA, Elliptic Curve Cryptography (ECC), Lattice-based cryptography, and Symmetric Ciphers.
To evaluate reasoning capabilities, this paper focuses exclusively on static challenges. In dynamic CTF challenges, an agent interacts with a remote server, introducing variables such as network errors, timeouts, server crashes and hidden server-side states. Static challenges, however, provide all necessary artifacts (scripts, packet captures, or ciphertexts) as files. The flag string is derived purely through mathematical derivation and local script execution. This restriction ensures that any failure is due to a lack of reasoning or implementation capability, rather than environmental instability or the unavailability or errors of a remote server.

3 Methodology

We design a controlled experimental environment to isolate the impact of agent architectures on cryptographic problem-solving.

3.1 System Architecture

The experimental setup runs in a containerized environment to ensure reproducibility and safety.

The Agent Runner: The agent runner runs as the orchestrator and implements the different architectures. It manages the LLM interaction with the remote API (GPT-5.2) and maintains the conversation state. The agent runner also checks for flag correctness and implements error handling, time limits and logging.

The Sandbox: All code runs within an isolated Docker container limited to a single CPU core and 4GB of RAM. This environment has SageMath, python and pycryptodome installed.

The Challenge Loader: A framework that loads static challenges from the AICrypto benchmark, injecting the challenge files into the sandbox filesystem before the agent starts.

3.2 Model Context Protocol (MCP)

The Agent Runner exposes three different tools to the agent:

- Command Execution: Allows the agent to run shell commands (e.g., ls, cat, python).
- Sagemath execution: Allows the agent to interactively execute code in a SageMath environment through ipython.
- Flag Submission: An interface that validates the obtained flag and stops the agent when the submitted flag is correct.

To run these tools, the challenge sandbox runs a python MCP server through which the Agent Runner communicates and forwards the tool calls.

3.3 Architecture Implementation Details

We implement five architectures representing different reasoning-action workflows.

Chain-of-Thought (CoT): The CoT architecture represents the non-agentic baseline. As shown in Figure 1, after the initial exploration phase, the model generates a linear reasoning path that terminates after a single tool call and therefore lacks a feedback mechanism. If the generated SageMath script fails due to a syntax error or a logical flaw in the attack, the task is marked as a failure. This architecture tests the model's ability to *one-shot* a challenge using only the context gathered during discovery.

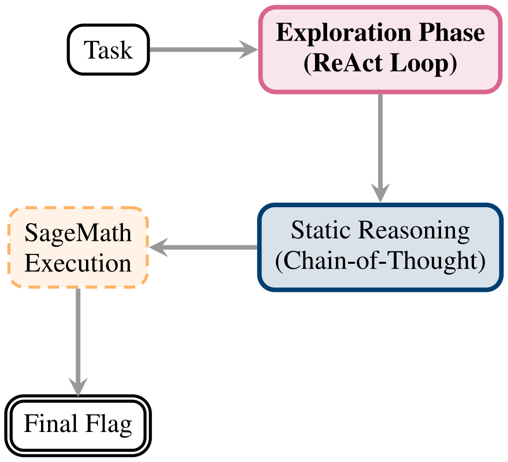
Task
Exploration Phase (ReAct Loop)
Static Reasoning (Chain-of-Thought)
SageMath Execution
Final Flag

Figure 1: The CoT architecture: The Exploration Phase gathers context for a single-pass reasoning and execution step.

ReAct (Reason + Act): The ReAct architecture uses a loop for both discovery and solving. As illustrated in Figure 2, it interleaves thoughts, actions, and observations. This architecture is particularly effective for debugging, as the agent can observe the traceback of a failed script and improve its implementation in the next turn.

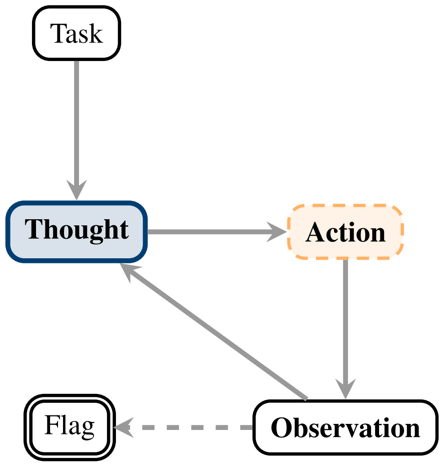

Figure 2: The ReAct architecture: A single loop for exploration and solving.

ReWOO (Reasoning Without Observation): ReWOO attempts to mitigate the high token cost of the ReAct loop by generating a static blueprint or dependency graph of tool calls. As shown in Figure 3, the exploration phase informs a Planner which predicts all necessary steps upfront. While this is efficient for predictable tasks, it faces significant challenges in cryptography, where the algebraic result of one tool call often dictates the logic required for the next.

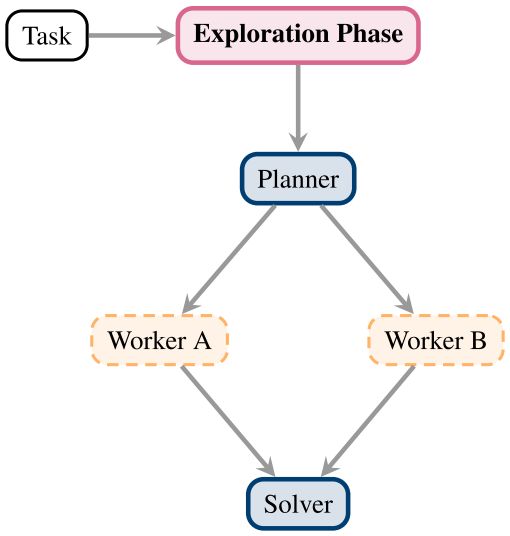

Figure 3: ReWOO architecture: Exploration informs the blueprint generation.

ADaPT (As-Needed Decomposition): The ADaPT architecture uses a recursive controller to break down hard problems. Following exploration, the agent decides if the challenge is complex enough to require decomposition. As seen in Figure 4, if a direct solve attempt fails, ADaPT spawns sub-tasks (e.g., extracting parameters, then implementing the attack). This allows the agent to focus on one implementation detail at a time.

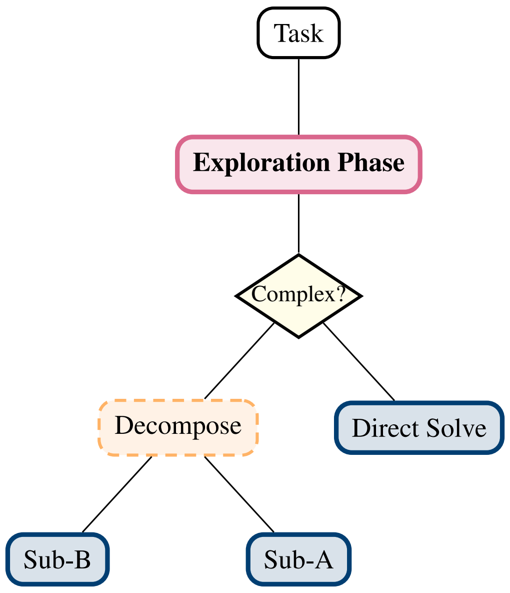

Figure 4: ADaPT architecture: Exploration phase preceding recursive task decomposition.

Language Agent Tree Search (LATS): LATS is the most complex architecture in our study. It treats the exploit process as a search space. As illustrated in Figure 5, after exploration, the agent generates multiple attack solutions. It uses a Monte Carlo Tree Search logic to evaluate which branch is most promising. LATS allows for backpropagation, if an attack fails, it pivots to a different approach.

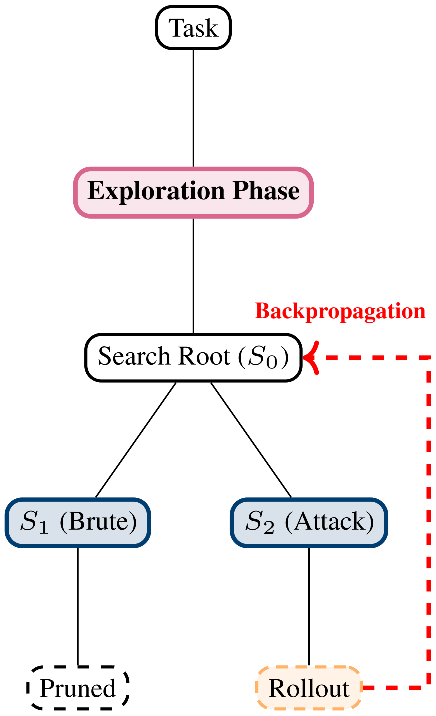

Figure 5: The LATS architecture: An initial discovery phase followed by state S_0, which creates and ranks ideas

3.4 Scope and design constraints

The goal of this research is to evaluate and isolate the architecture as the main variable at the cost of external validity:

Constrained LATS. Our LATS implementation is explicitly budget-limited by only generating three candidate plans after the exploration phase, followed by a ReAct-style implementation. This constraint limits the search budget of LATS and therefore its performance, however, this keeps the cost comparable to the other architectures under the fixed time/turn budgets, which results in a fairer architecture-level comparison.

Static-only challenges and isolation. By restricting the agent to only static challenges and taking away the ability to access the internet, we reduce the dual-use risk, but also take away the ability to retrieve writeups, articles or templates. Therefore, the results describe performance in a contained sandbox, not end-to-end CTF performance.

4 Experimental Setup and Results

We evaluate the success of our agents using the GPT-5.2 model with a temperature of 0 across 74 categorized challenges from the AICrypto benchmark.

4.1 Experimental Environment

Each trial is conducted with a fixed budget of 30 minutes and a maximum of 30 turns for each challenge to enforce a uniform resource budget. We measure three primary metrics: Pass@1 (overall success rate), Time-to-Solve, and Total Token Consumption. Every challenge starts with a mandatory exploration phase where the agent is allowed to read the challenge files, but is explicitly restricted from attempting a final solution.
The code used to evaluate the LLM is available at: https://github.com/QuerijnSharp/crypto-architectures

4.2 Overall Performance

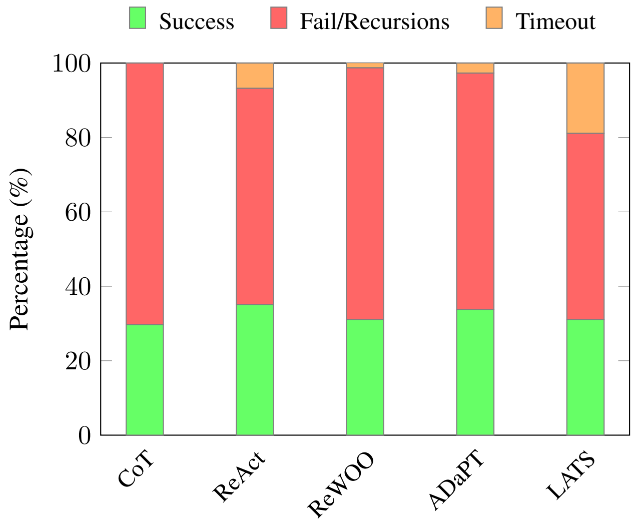

Figure 6: **RQ1: Overall Performance.** Status distribution across all 74 challenges.

As shown in Figure 6, ReAct achieves the highest overall success rate at 35.1%, followed by ADaPT (33.8%) and LATS/ReWOO (31.1%). The baseline CoT achieves just 29.7%. The LLM logs show the reason why the architectures fail, specifically CoT encounters many errors, about 63.6% of runs fail due to engineering failures such as parsing (TYPE_ERROR) using nonexistent methods (NAME_ERROR) or referencing nonexistent attributes (ATTRIBUTE_ERROR & SAGEMATH_METHOD) as shown in table 1. LATS suffers disproportionately from Timeouts (18.9%), despite trailing ReAct by only a few points.

Table 1: Code execution errors by type for CoT.

Error type	Count
VALUE_ERROR	9
TYPE_ERROR	7
NAME_ERROR	6
ATTRIBUTE_ERROR	5
SAGEMATH_METHOD	3
KEY_ERROR	2
IMPORT_ERROR	1

4.3 Domain-Specific Strengths

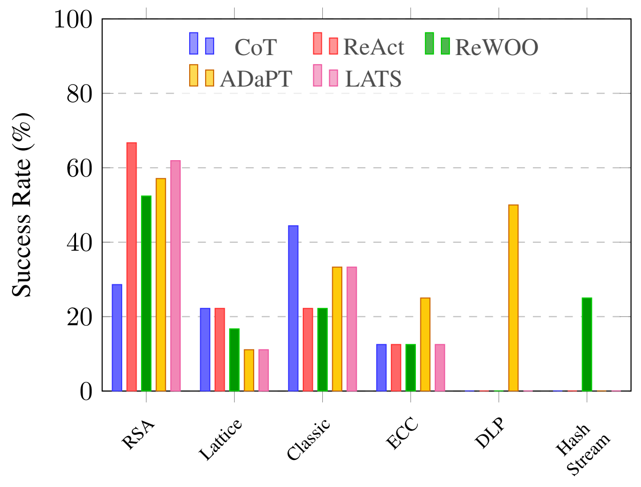

Figure 7: **RQ2: Success Rate by Category.** Success rates by category.

The data in Figure 7 reveals that agent architectures are not universally applicable. For example, ReAct has the best performance in the RSA category (66.7 % success). In contrast, CoT achieves the highest success rate in the Classic category (44.4%) and ties for the best result in Lattice (22.2%). A look at challenges like 'Alilbols' (Lattice) shows that CoT succeeds in 145s, while more complex agents like LATS or ADaPT hit recursion limits. ADaPT and ReWOO are the only two architectures to solve a single challenge within the DLP and Hash / Stream categories respectively. While these single successful results do not provide evidence of a consistent pattern, ADaPT also shows stronger performance in the ECC category compared to the other architectures.

4.4 Efficiency Trade-off

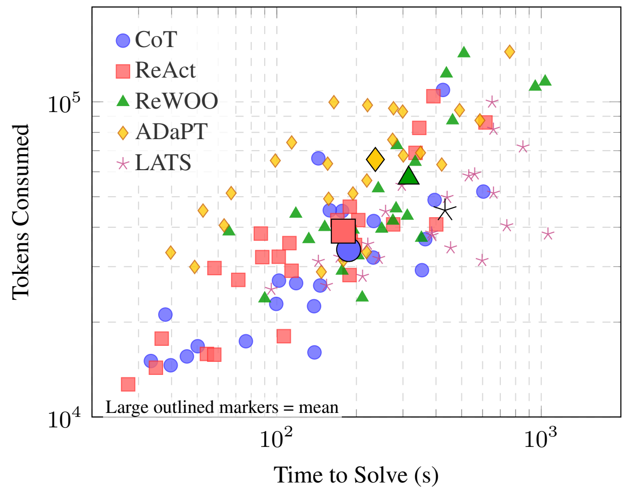

Figure 8: **RQ3: The Cost of Intelligence.** Log-log comparison of successful runs (large outlined markers show per-architecture mean).

Figure 8 shows the log-log relationship between time and tokens for all solved challenges, with CoT being the most efficient architecture (low time, low tokens). ADaPT is the most expensive, often using over 2 times the amount of tokens compared to ReAct for the same challenge.

5 Responsible Research

This research evaluates different LLM agent control-flow architectures on cryptographic CTF challenges. Because we demonstrate the capability of LLMs to reason through and automate the exploitation of cryptographic primitives, it is dual-use. So, we address the ethical implications, safety controls in the setup and societal impacts.

5.1 Dual-Use and Offensive Capabilities

Although our evaluation is conducted on cryptographic CTF challenges, the core skills required for this task (Parsing files, finding bugs and writing and debugging exploit code) are closely related to real-world security tasks. While scientifically valuable to understand the current models and architectural capabilities and limitations, this can also help to build more capable autonomous agents. To reduce this risk, the goal is to measure the effect of control-flow architectures, rather than creating an unconstrained solver.

5.2 Controlled Environment

To reduce the dual-risk, we made several design choices:

- **Sandbox** All the agent interactions were executed in an isolated Docker container with no access to the host machine. This ensures that any unsafe commands or code only impact the container environment. Additionally, it makes it easier to reproduce the results without requiring running untrusted code on the host machines.
- **Static scope** Because we evaluate explicitly on the static subset of the AICrypto benchmark, we reduce the risk of the agent targeting remote servers.
- **Budget constraints** We enforce strict per-run budgets through a wall-clock time and a maximum number of turns. This reduces the depth of iterative trial and error and, as a result, the agent's ability to refine partial ideas into a working attack. They also limit the successful throughput, as we limit the time/turns for each attempt, more attempts fail. To overcome this, more runs are needed, which in turn requires more time. Finally, these constraints discourage compute-intensive strategies, since the agent cannot run long loops or computationally expensive algorithms within a single run.

We note that budgets are a practical mitigation for the released evaluation setup rather than a hard barrier against a motivated actor who modifies the agent runner.

5.3 Resource Efficiency

The inference of LLMs, specifically when used in recursive agents, carries a significant environmental cost. As highlighted in Section 4.4, complex architectures use more tokens compared to the CoT baseline. For example, on successful runs ADaPT uses on average 93% more tokens compared to CoT and 132% more across all runs, so this high-compute overhead of complex agents yields diminishing returns compared to simpler baselines.

6 Discussion

Our results suggest that agent architectures help on cryptographic CTF challenges, but the success rate improvement compared to the baseline CoT is only a 5.4% gain. The gain of these architectures comes from the engineering process of writing the code rather than a source of new cryptographic ideas.

6.1 The limited benefit of Agent Architectures

Prior work on agentic control flows [10; 7; 9; 8] report higher accuracies across different categories of expertise, such as HotPotQA [11], which requires the agent to retrieve two or more Wikipedia excerpts. Higher accuracies have also been observed across programming benchmarks such as HumanEval [12] compared to a CoT baseline. Our results confirm this, but the improvement is limited to 5.4 percentage points in Pass@1. A likely explanation for this is that the sandbox already provides what many agent frameworks try to emulate, an interactive environment where hypotheses can be tested and computations can be executed. In this case, the value of an architecture is less about unlocking new cryptographic insights, but rather about engineering the solution into working code.

This is consistent with the data from Table 1 as discussed in Section 4.2, as 63.6% errors are related to engineering failures. Architectures that can iteratively react and figure out a solution to these errors are therefore more likely to solve the challenge when the cryptographic idea is known, but the implementation requires precise knowledge of the language it is working in.

6.2 Timeouts and Turn-Limit Failures

Figure 6 shows that different architectures fail in different ways. LATS shows a 18.9% timeout across all challenges, whereas ReAct and ADaPT time out 6.8% and 2.7% respectively. A reason for this is that LATS, after its exploration phase, generates and grades three different plans before committing to a solution path. Within the strict 30-minute cap, also having to write and debug its solution, this corresponds to taking on average roughly 2.31 times longer than the CoT baseline to solve a challenge.

The other reason for failure is the 30-turn limit, while reactive agents are able to react and try to solve encountered errors, their troubleshooting process can be inefficient. An agent might decide to do more exploration to understand why their input parsing would fail, despite having the exploration phase data still in its context. Additionally, an agent might also decide to do many sanity checks to explore the SageMath environment or available functions.

6.3 No Single Best Architecture

The category results of Figure 7 show that there is no single best architecture in terms of success rate. However, in a category like RSA, where challenges often involve standard attacks and correcting implementation errors, all architectures outperform CoT, with ReAct having the highest success rate of 66.7%. In the cases where the solution is clear, the additional steps in an architecture, as the case for LATS, ReWOO and ADaPT can become a liability and increase the probability of exhausting the turn budget and time cap. In contrast, there are categories and individual instances where complex architectures outperform ReAct, such as ECC, DLP and Hash / Stream. This suggests that a single best architecture under these constraints is unlikely.

6.4 The cost of success

Figure 8 shows that the token cost and time spent differ in each architecture. A likely reason for ADaPT higher token usage is that its controller repeatedly evaluates whether a problem requires decomposition, which can cause overhead for simpler tasks where decomposition is unnecessary, exhausting its time cap, nevertheless it shows success in DLP and ECC categories.

7 Conclusions and Future Work

This paper asked whether agent control-flow architectures improve LLM performance and efficiency on cryptographic CTFs when the base model and tool stack are constant. We evaluated 74 static AICrypto challenges with a 30-minute and 30-turn budget.

7.1 Conclusions

Our results show that the architecture matters, but the gains are limited: ReAct achieved the best overall Pass@1 (35.1%), followed closely by ADaPT (33.8%) and LATS/ReWOO (31.1%), while the non-agentic CoT baseline reached 29.7%.

A key finding is that many failures are not due to missing the cryptographic idea required to solve the challenge, but rather engineering failures caused by SageMath errors or input parsing. Architectures with an observe-debug loop can correct these errors by iteratively responding to the generated tracebacks. However, more complex architectures do not show higher success rates across all categories. Category-level results reveal that there is no single best architecture. ReAct beats all other architectures in RSA (66.7% success), while CoT leads in Classic (44.4%) and matches ReAct performance in Lattice (22.2%). ADaPT and ReWOO are the only architectures that solve any challenge in DLP (50%) and Hash/Stream (25%) categories, respectively. This suggests that architecture choice depends on challenge structure, when the solution path is standard, a direct reactive approach helps. The overhead of the architecture (e.g., decomposition, planner or search) becomes a liability under strict budgets.

Finally, architecture gains come with extra time and token costs. The baseline CoT being the most efficient in token usage among successful runs, while ADaPT and LATS have a high token/time overhead. Overall, the results show that the architecture primarily improves the implementation process rather than coming up with new cryptographic ideas.

7.2 Future Work

This evaluation limits the benchmark to static-only challenges, restrict internet/retrieval, fixed a single base model configuration, and a budget-limited LATS variant. All these limitations reduce external validity for control. Therefore, several extensions would strengthen external validity:

Dynamic challenges and retrieval. By introducing dynamic challenges and access to previous related challenges, a CTF environment is more closely emulated.

Pass@k. LLMs are non-deterministic, small differences in any phase of the solution can change the outcome or solution paths, especially for search/decomposition agents. This makes Pass@1 for individual challenge results difficult to compare and repeated runs (Pass@k) would provide a better confidence interval. Running each challenge multiple times and reporting Pass@k would provide more reliable comparisons than single-shot Pass@1.

Adaptive controller. The results suggest that success depends on a challenge difficulty such as easy vs. harder challenges. An adaptive controller that selects or switches between ReAct, decomposition, or limited search based on early signals could improve overall success rates and efficiency.

Documentation. Since many failures occur due to parsing and SageMath errors, future work should allow the agent to access SageMath documentation.

References

[1] 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.

[2] Google DeepMind. Advanced version of gemini with deep think officially achieves gold medal standard at the international mathematical olympiad, 2025. Accessed: 2025-11-16.

[3] Davide Castelvecchi. Deepmind and openai models solve maths problems at level of top students. *Nature*, 2025. News article.

[4] Reuters. Google clinches milestone gold at global math competition, while openai also claims win. *Reuters Technology News*, 2025. Published July 22, 2025.

[5] Anthropic. Claude is competitive with humans in (some) cyber competitions, 2025. Accessed: 2025-11-16.

[6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

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

[8] Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models, 2023.

[9] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2024.

[10] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models, 2024.

[11] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018.

[12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021.

A Use of LLMs for writing this paper

We used LLMs as a tool to improve grammar, readability and LaTeXcode. All suggested edits were reviewed for mistakes and adjusted to fit the rest of the text. The used prompts are in line with following:

- ”Improve the grammar of the following sentence: SENTENCE_HERE”
- ”Suggest other academic words for: WORD_HERE”
- ”The legend for this figure FIGURE_LATEX_HERE is too large. Make it smaller.