# Measuring and Augmenting Large Language Models for Solving Capture-the-Flag Challenges

Zimo Ji
zjiag@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong, China

Daoyuan Wu*†
daoyuanwu@ln.edu.hk
Lingnan University
Hong Kong, China

Wenyuan Jiang
wenyjiang@student.ethz.ch
D-INFK, ETH Zurich
Zurich, Switzerland

Pingchuan Ma
pmaab@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong, China

Zongjie Li
zligo@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong, China

Shuai Wang*
shuaiw@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong, China

## Abstract

Capture-the-Flag (CTF) competitions are crucial for cybersecurity education and training. As large language models (LLMs) evolve, there is increasing interest in their ability to automate CTF challenge solving. For example, DARPA has organized the AIxCC competition since 2023 to advance AI-powered automated offense and defense. However, this demands a combination of multiple abilities, from knowledge to reasoning and further to actions. In this paper, we highlight the importance of technical knowledge in solving CTF problems and deliberately construct a *focused* benchmark, CTF-Know, with 3,992 questions to measure LLMs' performance in this core aspect. Our study offers a focused and innovative measurement of LLMs' capability in understanding CTF knowledge and applying it to solve CTF challenges. Our key findings reveal that while LLMs possess substantial technical knowledge, they falter in accurately applying this knowledge to specific scenarios and adapting their strategies based on feedback from the CTF environment.

Based on insights derived from this measurement study, we propose CTFAgent, a novel LLM-driven framework for advancing CTF problem-solving. CTFAgent introduces two new modules: two-stage Retrieval Augmented Generation (RAG) and interactive Environmental Augmentation, which enhance LLMs' technical knowledge and vulnerability exploitation on CTF, respectively. Our experimental results show that, on two popular CTF datasets, CTFAgent both achieves over 80% performance improvement. Moreover, in the recent picoCTF2024 hosted by CMU, CTFAgent ranked in the top 23.6% of nearly 7,000 participating teams. This reflects the benefit of our measurement study and the potential of our framework in advancing LLMs' capabilities in CTF problem-solving.

## CCS Concepts

**• Social and professional topics → *Computing education*; • Security and privacy; • Computing methodologies → Artificial intelligence**;

*Corresponding authors.
†Work conducted by Daoyuan Wu during his time at HKUST.

**ACM Reference Format:**
Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, and Shuai Wang. 2025. Measuring and Augmenting Large Language Models for Solving Capture-the-Flag Challenges. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744855

## 1 Introduction

Capture-the-Flag (CTF) is universally acknowledged as an essential component of cybersecurity. To simulate real-world vulnerability scenarios and enhance participants' cybersecurity skills and knowledge, CTF competitions have become an indispensable tool for cybersecurity training since their inception at DEFCON in 1993 [39]. In a typical CTF challenge, participants are tasked with identifying and exploiting vulnerabilities in a target system, aiming to discover the hidden "flag" string within the sandbox environment. CTF challenges cover a broad spectrum of domains, such as cryptography, reverse engineering, web exploitation, forensics, and miscellaneous.

To date, CTF competitions have become a popular and "real business" in the cybersecurity community, with numerous competitions held worldwide, such as DEFCON CTF [13], GoogleCTF [16], and picoCTF [26]. Moreover, it is believed that many intelligence agencies use CTF competitions as a recruitment tool to identify top cybersecurity talent [39] and to train their own cybersecurity professionals for various missions. Despite the high benefits and popularity of CTF competitions in cybersecurity, solving CTF challenges requires a combination of technical, problem-solving, and analytical skills, all of which demand human-level intelligence.

As large language models (LLMs) exhibit exceptional capabilities in various security tasks, such as penetration testing [44], vulnerability detection [73], and exploiting zero-day vulnerabilities [47], CTF education and research could also evolve into a new era of *leveraging LLMs to automate CTF challenge solving*. With such intelligent CTF automation, it is anticipated that existing CTF education and training will be significantly enhanced, as cybersecurity learners can use LLM-based "copilots" to quickly identify various attack surfaces across numerous CTF challenges; see more in §6. Moreover, automating offense and defense in CTF competitions, and in cyber-autonomy more generally [38], has long been recognized to stimulate notable research challenges and opportunities, e.g., achieving automated vulnerability discovery and repair in light of software releases now exceeding human review capacity [38], and assessing and enhancing LLMs' vulnerability reasoning [70, 73].

Yet, the full automation of CTF challenge solving by LLMs requires a complex array of composite skills, including scenario comprehension, multi-turn reasoning, and action execution, which are generally hard to assess. To date, few benchmarks have been proposed to evaluate the proficiency of LLMs in CTF competitions [69, 70, 87], in which researchers propose an LLM-based bot in a command-line environment with common security toolkits to solve challenges. However, our tentative exploration finds that existing benchmarks do not delve deeply into the CTF abilities during different phases of challenge solving, including from knowledge to reasoning and further to action.

Given the demand for a more in-depth measurement of LLMs' capability in CTF and the challenges posed by the substantial reasoning and hacking requirements involved, we identify a key enabling factor — *LLM's core technical knowledge* in CTF. We argue that the ability to effectively apply technical knowledge is a critical factor in determining the success of LLMs in solving CTF challenges. Importantly, by focusing on the technical knowledge aspect, it is practically feasible to provide a focused, in-depth, yet not overly complex benchmark. Overall, we construct a new and focused benchmark, CTFKnow, with 3,992 questions to specifically assess LLMs' performance on CTF technical knowledge. We collect 1,084 write-ups from the most well-known CTF competitions over the past five years, including 0CTF [4], UIUCTF [28], GoogleCTF [16], etc. To extract the technical knowledge from these write-ups, we use the recent GPT-4 [35] model, carefully constructing customized prompts to enable it to complete this task more effectively. This approach allows us to identify and extract 1,996 distinct CTF core knowledge points. These knowledge points are then used to construct 1,996 single-choice questions and 1,996 open-ended questions designed to evaluate the technical knowledge of LLMs in scenarios of different difficulty. We employ another LLM to generate these questions, utilizing prompts refined through multiple iterations to ensure the questions' validity and rigor. Moreover, to avoid hallucination and bias in the knowledge extraction and question generation tasks, we use another open-source LLM [12] to check and filter the results of these tasks, which are further manually verified.

With CTFKnow, we measure five mainstream LLM models, including three OpenAI models (GPT-3.5 [17]/4 [18]/4o [19]) and two open-source models, Llama3 [23] and Mixtral [24]. The main findings are twofold: On one hand, LLMs exhibit a strong grasp of technical knowledge in CTF when the potential correct answer is provided in the single-choice questions. This indicates that the vast majority of technical knowledge encountered in most CTF contexts have been adopted by LLMs during their pre-training phase, which is encouraging. On the other hand, with open-ended questions, we find that LLMs exhibit poor capability in matching technical knowledge to specific CTF scenarios. In particular, LLMs struggle more with accurately matching technical knowledge to more challenging CTF scenarios. This underscores how aiding LLMs in effectively mapping their CTF knowledge to specific problems is a crucial area for improvement. Furthermore, the correlation analysis between our results and previous benchmark results [69, 70, 87], along with an in-depth analysis of execution logs from these works, indicates that the absence of tools and unfriendly environments pose significant limitations for LLMs to solve CTF challenges.

To illustrate the benefit of our measurement, we design CTFAgent, a novel LLM-driven framework for advancing CTF problem-solving. CTFAgent introduces two new modules: two-stage Retrieval Augmented Generation (RAG) and interactive Environmental Augmentation (EA). Specifically, we provide CTF knowledge via RAG at different stages of the CTF, including searching for potential vulnerabilities based on relevant CTF vulnerability code snippets during the problem understanding phase and supplying knowledge on how to effectively exploit a specific vulnerability during the problem exploiting phase. In the EA module of CTFAgent, we provide a more interactive CTF environment by integrating interactive command lines and advanced CTF tools, which significantly simplifies the challenge-solving process for LLMs compared to their operation within a native command-line environment.

We conduct an extensive evaluation of CTFAgent on two recent and popular CTF datasets, Intercode-CTF [87] and NYU CTF Dataset [70]. The results show that the CTFAgent framework has enhanced the capability of LLMs in automatically solving CTF problems by 85%, improving from the original 39 out of 100 to 73 out of 100 on Intercode-CTF. Leveraging OpenAI's SOTA model o1 as its backbone, CTFAgent is capable of solving an additional 11 challenges within this dataset. On the more challenging NYU CTF Dataset, CTFAgent still performs commendably by improving the number of solved challenges by over 120%. Furthermore, in the recent picoCTF2024 hosted by CMU, CTFAgent ranked in the top 23.6% of nearly 7,000 human participants, significantly higher than the NYU CTF framework [70], which ranked in the top 47.2%. Meanwhile, as disscussed in §6.3, we thoroughly consider the potential risks of CTFAgent being misused. While taking appropriate actions, we also call on the broader community to remain vigilant about the possible abuse of automated tools.

In sum, we make the following contributions in this paper:

- **CTFKnow: A benchmark for measuring LLMs' CTF knowledge.** We construct CTFKnow with 3,992 questions based on 1,086 CTF write-ups and nearly 2,000 distinct knowledge points extracted from them, enabling targeted assessment of LLMs' technical knowledge across varying difficulty levels.
- **Comprehensive measurement of LLMs on CTF tasks.** Using CTFKnow, we perform a systematic measurement with five mainstream LLMs, revealing both strengths and limitations in their understanding and application of CTF knowledge.
- **CTFAgent: Augmenting LLMs for automated CTF solving.** We propose CTFAgent, which integrates tailored RAG and interactive environment modules, achieving substantial performance gains across two CTF benchmarks and real-world competitions.

**Artifact.** CTFKnow and its evaluation scripts, with the full paper (Appendix included), are accessible at our landing website. CTFKnow is designed to be easily extensible. To minimize potential misuse, the release of CTFAgent is subject to a review process on the website, ensuring that access is restricted to institute-affiliated research personnel only.

## 2 Background

## 2.1 Capture the Flag (CTF)

CTF competitions are a crucial component of cybersecurity education [81]. These gamified competitions expose participants to diverse challenges that encompass a wide range of cybersecurity topics. Each challenge constitutes a carefully designed sandbox environment that mimics real-world security vulnerabilities. In these scenarios, organizers set up services or create situations laden with specific vulnerabilities, containing a hidden text string or "flag."

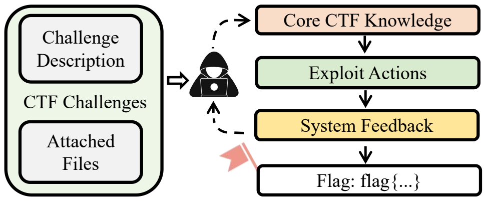


**Figure 1: The typical progress of solving a CTF challenge.**

Figure 1 illustrates the typical progress of solving a CTF challenge, summarized from previous work [42, 69, 87]. Overall, to "capture a flag", the CTF player will need to possess a combination of technical knowledge, reasoning skills, and the ability to execute actions effectively. The process begins with the player identifying the challenge type and understanding the underlying vulnerabilities with his/her core CTF knowledge. Next, the player must apply the appropriate techniques to exploit these vulnerabilities, e.g., crafting a payload to manipulate the target system. Finally, the player executes the payload and monitors the system's response to determine whether the exploit was successful. If successful, the player captures the flag and scores points. Overall, non-trivial CTF challenges require participants to demonstrate both theoretical knowledge and practical skills in cybersecurity. Successfully capturing a flag allows participants to score points in the competition, and the player with the highest score at the end of the CTF competition wins.

To support the study and practice of CTF competitions, engaging with various CTF platforms that aggregate, document, and curate challenges from past contests is essential for participants' learning [57]. Representative platforms include picoCTF [25], NYU-CSAW [8], and buuctf [7], and others. Additionally, specialized platforms such as CTFtime [5] compile information on CTF competitions, team data, challenge details, and write-ups. In this paper, all collected write-ups have been sourced from CTFtime. Well-designed CTF challenges encompass a wide range of types that cover most real-world cybersecurity scenarios. The primary categories include:

- **Web:** These focus on web application security, requiring participants to exploit vulnerabilities like SQL injection, Cross-Site Scripting (XSS), and file upload issues to retrieve hidden flags.
- **Reverse (Rev):** Using reverse engineering methods to extract vulnerability information from binary files and write scripts to capture flags, often hidden using complex encryption methods.
- **Pwn:** Concentrating on binary security, these challenges involve vulnerabilities such as stack overflows, heap overflows, requiring extensive knowledge of system and binary code security.
- **Crypto:** These involve attacking cryptographic systems like AES, RSA, and ECDSA, demanding strong mathematical skills, particularly in number theory and abstract algebra.
- **Forensics:** Inspired by real-world computer forensics, these challenges often hide flags within multimedia files and include tasks like traffic packet analysis and steganography.
- **Misc:** These cover a variety of security scenarios including Open Source Intelligence (OSINT), social engineering, etc.

**CTF Value in Cybersecurity**. According to projections by Cybersecurity Ventures, global cybercrime costs are expected to reach $10.5 trillion annually by 2025, reflecting a 15% annual growth rate since 2020 [1]. This increasing threat landscape is prompting organizations to invest more in cybersecurity training and skill development. To date, CTF competitions have been playing a vital role in cybersecurity education and training. These competitions simulate real-world security scenarios, challenging participants across various categories. This encourages continuous learning to the ever-evolving cybersecurity landscape. CTFs have significant industry visibility. Industrial giants like Google and governmental organizations like DARPA actively hosts CTFs to showcase their cybersecurity prowess and attract a talent pool. Leading-edge CTFs often leverage cutting-edge information systems to design challenges that may involve exploiting zero-day vulnerabilities, offering substantial rewards and valuable contributions to the cybersecurity community [3, 9, 27, 29, 72]. These illustrate the importance and value of measuring and augmenting LLMs for CTF challenges.

## 2.2 Large Language Models (LLMs)

Since the release of GPT-3.5 in 2022 [17], Large Language Models (LLMs) have attracted increasing attention. Trained on extensive natural language data, these models are fine-tuned for various downstream tasks [40, 60, 65, 82]. Leading LLMs such as o1 [22], GPT-4 [35], Claude 3.5 Sonnet [11], and Llama-3.1 [23] have excelled in natural language understanding [51] and code generation [92].

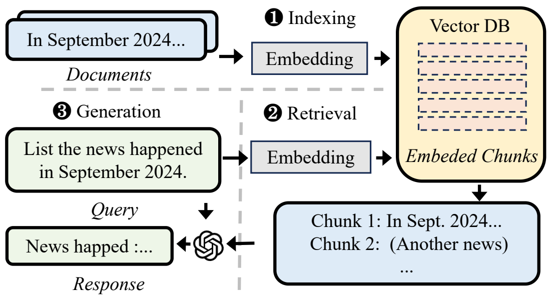


**Figure 2: The standard workflow of RAG.**

In this paper, we enhance LLMs using a customized version of Retrieval-Augmented Generation (RAG) technology, which incorporates knowledge from external databases [48]. As illustrated in Figure 2, RAG allows for pre-retrieval of information, enriching the LLM's context and knowledge base. The standard RAG workflow includes three stages: indexing, retrieval, and generation. During indexing, a pre-collected knowledge dataset is converted into plain text, segmented, and embedded into a vector database. In the retrieval stage, the framework performs cosine similarity searches based on user queries, returning the top-K results for the LLM's reference. Recent enhancements in RAG technology, such as CoN [90] and NoiseRAG [43], have further refined this method.

LLM agents have recently seen widespread application across a variety of commercial tasks [59, 66, 89]. A typical LLM agent comprises three main modules: understand, plan, and act. The understand module is primarily responsible for identifying the task that needs to be completed, based on user input or environmental feedback. The plan module decomposes the understood task into sub-tasks, facilitating their completion by the act module. The act module primarily leverages the tool invocation capability of LLMs, enabling the LLM to utilize externally provided tools to accomplish these sub-tasks and to receive the results following the tool's execution. In this paper, we primarily utilize the OpenAI Assistants API [6] and OpenAI Function Calling [14] to construct our tool.

## 3 Measuring LLM Capability for CTF

### 3.1 LLM4CTF: Understanding and Exploiting

In accordance with our introduction on how human participants solve CTF challenges, as detailed in §2, we synthesize insights from previous work [69, 70] to summarize the steps involved in the LLM4CTF process.

Overall, using LLMs to solve CTF challenges involves two general phases: Understanding and Exploiting.

- *Understanding* entails comprehending the problem's context, identifying the appropriate vulnerability type, and proposing potential exploitation strategies.
- *Exploiting* involves using command-line tools, CTF-specific tools, or scripts to capture the flag, based on a clear understanding.

In the Understanding phase, an LLM must evaluate potential vulnerabilities and attack surfaces from provided code snippets and scenarios, requiring the correlation of vulnerable code characteristics with specific vulnerabilities. This capability is grounded in technical knowledge—a cybersecurity competency that encompasses a comprehensive understanding of vulnerability types and their typical code manifestations. Nevertheless, as defined in previous works [42, 62], CTF technical knowledge should exclude non-technical aspects such as security regulations. We clarify that this definition aligns precisely with our study's focus on the technical understanding of vulnerabilities and their code-level expressions.

Transitioning to the Exploiting phase, the demands on LLMs' capability further increase. First, there is a need for knowledge about using relevant tools or writing scripts, also considered part of technical knowledge. This includes proficiency with command-line interfaces, specialized CTF tools, and Python libraries designed for exploiting vulnerabilities. Second, this phase emphasizes the model's reasoning ability, particularly its capacity to refine strategies based on feedback from tool or script execution, such as debugging Python scripts after runtime errors.

As such, **technical knowledge** is needed in both phases. However, current research primarily evaluates LLMs on their overall performance in solving CTF challenges [69, 70, 87], without recognizing the importance of technical knowledge. To address this, this section presents a benchmark specifically designed to assess the technical knowledge of LLMs, using single-choice and open-ended questions to isolate this evaluation from reasoning ability.

### 3.2 CTFKnow Design

**Motivation.** We recognize the necessity for a benchmark that specifically measures LLM capabilities in CTF. Yet, existing benchmarks, such as Intercode-CTF [87] and NYU CTF Dataset [70], focus on evaluating LLMs' *overall performance* in solving CTF challenges. We see this as a limitation, as it is often challenging to disentangle the technical knowledge from the reasoning ability of LLMs, resulting in a lack of clarity in the evaluation and even bloated performance metrics. We thus champion a targeted benchmark that deliberately measures LLMs' technical knowledge, which is crucial for understanding and exploiting vulnerabilities in CTF challenges. This benchmark should offer a focused and innovative viewpoint on LLM, without incurring biases or bloated performance metrics.

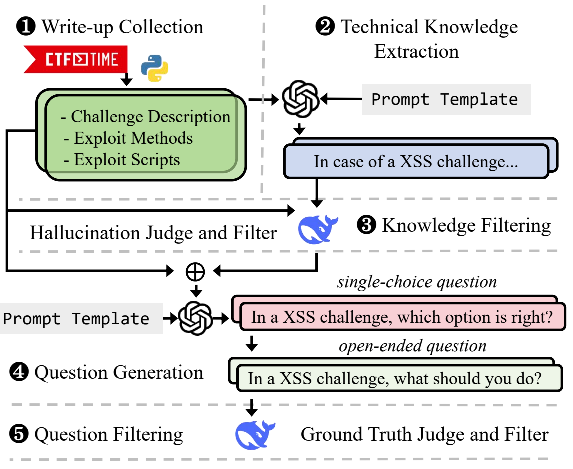


**Figure 3: The workflow of building our benchmark.**

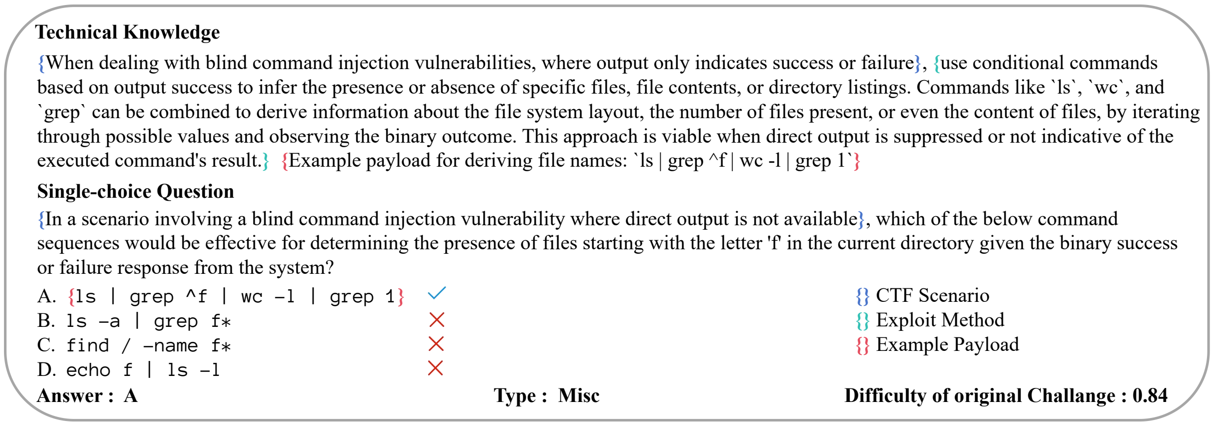
Building on the above motivation, we develop CTFKnow to measure LLMs' technical knowledge in CTF scenarios. As in Figure 3, building CTFKnow was divided into the following five phases.

① **Write-up Collection.** We first select over 700 large-scale international CTF competitions from the past five years, including prestigious events like DEFCON [13], HITCON [20], GoogleCTF [16], and UIUCTF [28], among others. We use the CTFtime platform (introduced in §2.1) as our primary source for the collection of competition information and write-ups.

Specifically, we deployed web scraping scripts to extract all available challenge write-ups from these competitions, converting them from HTML to Markdown format to facilitate easier comprehension by LLMs. This process yielded a collection of over 10,000 write-ups. However, recognizing not all write-ups met our quality standards, we applied the following criteria to refine our selection: (i) A minimum of 30 lines of text, ensuring the write-ups were sufficiently detailed and comprehensive, covering aspects such as Challenge Description and Exploit Method. (ii) Exclusion of images and external resource links. Despite state-of-the-art LLMs' capability to process images, we focused on write-ups relying on textual command-line outputs to avoid challenges and biases associated with image interpretation. (iii) Inclusion of a "Challenge Description" section to provide contextual background, enhancing LLMs' understanding of the scenario and avoiding substantial comprehension biases.

This filtration process yielded 1,084 high-quality write-ups that form the basis for our benchmark's single-choice questions.

② **CTF Knowledge Extraction.** To ensure the scalability of our benchmark and to manage the overall workload efficiently, we use the advanced LLM, GPT-4, for CTF knowledge extraction. Our method employs prompt engineering techniques, utilizing customized prompts that enable the LLM to accurately identify CTF



**Technical Knowledge**

{When dealing with blind command injection vulnerabilities, where output only indicates success or failure}, {use conditional commands based on output success to infer the presence or absence of specific files, file contents, or directory listings. Commands like \`ls\`, \`wc\`, and \`grep\` can be combined to derive information about the file system layout, the number of files present, or even the content of files, by iterating through possible values and observing the binary outcome. This approach is viable when direct output is suppressed or not indicative of the executed command's result.} {Example payload for deriving file names: \`ls | grep ^f | wc -l | grep 1\`}

**Single-choice Question**

{In a scenario involving a blind command injection vulnerability where direct output is not available}, which of the below command sequences would be effective for determining the presence of files starting with the letter 'f' in the current directory given the binary success or failure response from the system?

A. {`ls | grep ^f | wc -l | grep 1`} ✓
B. `ls -a | grep f*` ✗
C. `find / -name f*` ✗
D. `echo f | ls -l` ✗

{} CTF Scenario
{} Exploit Method
{} Example Payload

**Answer : A** **Type : Misc** **Difficulty of original Challange : 0.84**

**Figure 4: A benchmark example includes extracted technical knowledge and a single-choice question. The open-ended question is derived by changing the single-choice question's wording from "which of the below" to "what" and removing all options.**

knowledge. This setup instructs the LLM to extract up to two distinct pieces of CTF knowledge from each write-up since both understanding and exploiting steps have the corresponding knowledge. Each piece of CTF knowledge is categorized into three segments: CTF Scenario, Exploit Method, and Example Payload. This structure ensures that each piece of CTF knowledge maintains as much informational integrity as possible, facilitating further work. We repeat this process for each of the 1,084 selected write-ups, ultimately extracting 2,078 instances of CTF technical knowledge.

③ **Knowledge Filtering.** To mitigate potential hallucinations by LLMs in the CTF Knowledge Extraction task, we employ another LLM to assess and filter the extracted knowledge for hallucinations. To avoid potential bias, we utilize the Deepseek model (version `Deepseek-chat-v2.5`), which differs from GPT-4, for this evaluation and filtering process. The knowledge extracted in step ②, along with its corresponding original write-up, is input into the Deepseek model, which assesses the degree of alignment between the knowledge and the write-up.

For each knowledge point, we retain it only if Deepseek assesses it fully matches the write-up and accurately reflects its content. Following this step, we keep 2,013 high-quality knowledge points that are mostly free from hallucinations to serve as the foundational data for subsequent processing.

④ **Question Generation.** The design process for single-choice questions is facilitated through prompt engineering with GPT-4. To avoid potential context loss that could adversely affect question design, we input both the original write-up and extracted CTF knowledge into the LLM. This approach enables generation of a single-choice question based on each piece of CTF knowledge, ensuring questions remain grounded in the context of the originating write-up. For each question, we ensure the LLM retains the CTF Scenario from the CTF Knowledge as the stem, using either the Exploit Method or Example Payload as the correct option, thereby maintaining the independence and integrity of each question.

Based on these single-choice questions, we slightly adjust the wording of each question, for example, changing "which of the following" to "what" and removing all option information, thus creating our set of open-ended questions. Hence, the number of open-ended questions matches that of the single-choice questions. These open-ended questions do not provide potential answers to the LLM and are primarily used to assess the LLM's ability to match CTF scenarios with CTF technical knowledge, posing a higher level of difficulty. Given that both single-choice and open-ended questions are constructed from technical knowledge points derived from CTF Knowledge Extraction, we treat the assessment results as indicators of the test subjects' understanding of the corresponding technical knowledge, which is commonly adopted in educational and cybersecurity assessment [71, 78] contexts.

⑤ **Question Filtering.** To ensure the accuracy of the ground truth answers for the questions generated in ④, we employed another LLM (Deepseek) for Question Filtering. After this step, we retained a total of 1,996 high-quality questions for subsequent evaluation.

**Manual Verification.** Given the benchmark formed by the above steps, we manually verified the final set of knowledge points and questions. We selected 323 instances[1] for evaluation. Two authors independently cross-checked these knowledge points to assess their relevance to the original write-ups as well as the accuracy of the ground truth answers for the single-choice questions. We find that only two technical knowledge points and questions exhibit slight inaccuracies, indicating that over 99.38% $(1 - 2/323)$ of the knowledge points and single-choice questions are reliable after two rounds of filtering. This further reflects the high quality of CTFKnow.

**An Illustrative Example.** Figure 4 demonstrates a technical knowledge example and a single-choice question crafted using ② and ③. The original challenge write-up employs the payload `ls | grep ^f | wc -l | grep 1` in a blind command injection vulnerability, effectively bypassing restrictions by indicating the presence of files starting with 'f' in the directory. The designed technical knowledge states: "In blind command injection scenarios, where output indicates only success or failure, commands like `ls`, `wc`, and `grep` can be combined... Example payload: `ls | grep ^f | wc -l | grep 1`." The single-choice question utilizes this CTF scenario, presenting the example payload as the correct answer alongside three incorrect options. The open-ended question is crafted by subtly altering the wording to avoid direct repetition of the example.

[1]"323" is determined using the "Statistics of a Random Sample" algorithm at https://www.calculator.net/sample-size-calculator.html. For a population of 1,996 instances, a sample size of 323 ensures a 95% confidence level with a 5% margin of error.

## 3.3 Measurement Settings

Using CTFKnow, we measure LLMs' mastery of CTF technical knowledge. Specifically, we aim to address the following two research questions (RQs):

- **RQ1**: To what extent do LLMs grasp technical knowledge in CTF scenarios?
- **RQ2**: How well can LLMs correctly match and apply this technical knowledge in given CTF scenarios?

**Single-choice Questions.** The evaluation of single-choice questions is straightforward. During the question generation, we generated only one possible ground-truth answer per question, such as answer A in the case shown in Figure 4.

**Open-ended Questions.** The evaluation of open-ended questions is more complex. We use another LLM (GPT-4-Turbo, version `gpt-0125-preview`) as an evaluator to assess the responses of the LLM being tested. The inputs for this evaluator include the open-ended question, the corresponding reference answer, and the response from the LLM under test. A response is considered correct only if it achieves the same effect as the reference answer without modifications needed to solve the problem. This rigorous evaluation standard is adopted to more accurately assess the LLM's ability to precisely match technical knowledge in a given CTF scenario. To avoid possible biases in evaluating GPT-4-Turbo's answer using itself, we add an additional evaluation of the GPT-4-Turbo model using the open-source Qwen model (version `Qwen2.5-72B-Instruct`) as a cross-check.

**Model Selection.** We selected five widely-used LLMs, including three proprietary models: GPT-4 Turbo (version `gpt-4-0125-preview`), GPT-4o (version `gpt-4o-2024-08-06`), and GPT-3.5-Turbo (version `gpt-3.5-turbo-0125`). The two open-source models chosen are Llama 3 (version `llama3-70b`, with an 8,192 context window) and Mixtral-8x7b (with a 32,768 context window). These LLMs collectively represent the best in both proprietary and open-source models, providing comprehensive data support for our exploratory study. Findings are reported in the following sections.

**Human Evaluation.** To assess the quality of questions in CTFKnow and provide a more intuitive comparison of LLM evaluation results, we invited five undergraduate-level CTF players to participate in the *testing*. Each participant was asked to complete 90 single-choice questions and 30 open-ended questions within 120 minutes. These questions were randomly sampled from CTFKnow, ensuring a comprehensive inclusion across all CTF categories. For assessing the correctness of open-ended responses, we employed manual verification instead of LLM-based evaluation. This decision was made because human participants tend to provide brief answers, which could introduce significant bias if an LLM were used as the evaluator, per our observation. It is also worth noting that in this testing, it is the human experts who are involved in answering cybersecurity questions *only*. Thus, no human is under "attack" in any circumstances, and no personal or identifiable information (PII) is collected. Given the nature of this testing, which involves non-interventional expert task responses and no collection of personal data, it qualifies for exemption under Exempt Category 2 [80]. We will also acknowledge the assistance of these invited students in the Acknowledgement section upon publication of this paper.

## 3.4 Knowledge Measurement (RQ1)

Table 1 presents the measurement results of LLMs on single-choice questions. The outcomes show that all five evaluated LLMs perform exceptionally well, with an overall accuracy rate exceeding 70% for each model. Notably, GPT-4o achieved the best results, successfully answering 1,753 out of 1,996 questions, which translates to an impressive accuracy rate of 87.83%. This demonstrates that LLMs have thoroughly mastered a significant amount of technical knowledge during the pre-training phase.

**Finding 1:** LLMs exhibit a strong grasp of technical knowledge on CTF, mastering the vast majority of technical knowledge encountered in most CTF contexts.

Further analysis of the single choice question section in Table 1 across different challenge categories reveals that, generally, LLMs have the most solid understanding of technical knowledge in reverse engineering challenges, while their grasp on Web challenges appears weakest. This aligns with intuition, as reverse engineering challenges often involve understanding and analyzing decompiled code, a skill at which LLMs excel. In contrast, Web challenges involve dynamic operations and usage of penetration testing tools, knowledge that is typically distributed across various multimodal data sources, making it more challenging and costly to learn.

**Finding 2:** LLMs' mastery of technical knowledge varies across different types of CTF challenges, with the strongest performance in Reverse and the weakest in Web.

A longitudinal comparison among different LLMs reveals that, while all demonstrate commendable performance, notable differences exist. Among five LLMs tested, GPT-4-Turbo, GPT-4o, and Llama-3 slightly outperform GPT-3.5-Turbo and Mixtral-8x7b. This is consistent with results from general benchmarks such as MMLU [51], suggesting that for general-purpose models, better overall capabilities correlate with robust mastery of CTF technical knowledge.

## 3.5 Comparative Analysis (RQ2)

Despite LLMs demonstrating a strong grasp of technical knowledge in CTF scenarios, successfully solving CTF challenges requires not only mastering this knowledge but also accurately matching and applying it to the scenarios. This is the focus of RQ2, for which we employed the aforementioned open-ended question evaluation method.

Table 1 summarizes the experimental results, indicating that LLMs perform significantly worse on open-ended questions compared to single-choice questions. Even the best-performing models, GPT-4o and GPT-4 Turbo, barely reach an accuracy of 50%, with the poorest performer, Mixtral-8x7b, achieving only 30% accuracy. The evaluation results of GPT-4 Turbo using GPT-4 Turbo and Qwen as evaluator are quite similar, the result evaluated by Qwen is even slightly higher than that evaluated by GPT-4 Turbo, indicating that the evaluation is free of bias in this task. The accuracy rates for all models dropped by about half when compared to single-choice questions. This suggests that without potential correct answers, LLMs struggle to precisely apply their mastered technical knowledge based on CTF scenarios, a critical capability

**Table 1: Overall performance of mainstream LLMs on our CTF technical knowledge measurement. The numbers in parentheses indicate the counts of correctly solved questions. The same convention applies to all subsequent tables. Since the human baseline is tested on a subset of CTFKnow, we omit the parentheses in the "Human Eval." row to avoid misleading.**

| Models / Human | Web (218) | Pwn (459) | Misc (332) | Crypto (638) | Reverse (128) | Forensics (221) | Total (1996) |
|---|---|---|---|---|---|---|---|
| **Single-choice Questions** | | | | | | | |
| GPT-3.5-Turbo | 72.02% (157) | 79.08% (363) | 82.23% (273) | 81.03% (517) | 79.69% (102) | 76.47% (169) | 79.21% (1581) |
| GPT-4-Turbo | 80.73% (176) | 85.40% (392) | 87.95% (292) | 86.21% (550) | **90.62% (116)** | 85.07% (188) | 85.87% (1714) |
| GPT-4o | 83.03% (181) | **88.02% (404)** | **89.46% (297)** | **88.56% (565)** | 89.84% (115) | 86.43% (191) | **87.83% (1753)** |
| Llama-3-70b | **83.03% (181)** | 86.06% (395) | 88.86% (295) | 85.89% (548) | 89.84% (115) | **87.33% (193)** | 86.52% (1727) |
| Mixtral-8x7b | 77.98% (170) | 78.00% (358) | 84.34% (280) | 79.15% (505) | 78.91% (101) | 75.11% (166) | 79.16% (1580) |
| Sampled Human Eval. | 59.15% | 66.67% | 65.62% | 69.88% | 62.67% | 55.26% | 63.33% |
| **Open-ended Questions** | | | | | | | |
| GPT-3.5-Turbo | 33.49% (73) | 32.68% (150) | 43.07% (143) | 34.01% (217) | 34.38% (44) | 35.75% (79) | 35.37% (706) |
| GPT-4-Turbo | 46.79% (102) | **50.33% (231)** | 56.63% (188) | **51.72% (330)** | 49.22% (63) | **55.20% (122)** | **51.90% (1036)** |
| GPT-4o | **47.71% (104)** | 48.58% (223) | **55.72% (185)** | **48.28% (308)** | **54.69% (70)** | **55.20% (122)** | 50.70% (1012) |
| Llama-3-70b | 34.40% (75) | 32.03% (147) | 40.96% (136) | 36.68% (234) | 42.19% (54) | 40.27% (89) | 36.82% (735) |
| Mixtral-8x7b | 33.94% (74) | 26.36% (121) | 34.34% (114) | 28.37% (181) | 34.38% (44) | 32.58% (72) | 30.36% (606) |
| Sampled Human Eval. | 12% | 36% | 52% | 16% | 40% | 40% | 32.66% |
| GPT-4-Turbo (Evaluated by Qwen) | 51.83% (113) | 54.90% (252) | 59.34% (197) | 54.55% (348) | 57.03% (73) | 54.30% (120) | 55.26% (1103) |

in actual problem-solving. Additionally, the data distribution for open-ended questions across different types of CTF challenges remains consistent with that of single-choice questions, with LLMs still showing a preference for Reverse over other challenges.

The human evaluation results demonstrate that both types of questions present significant challenges even for human experts. For the single-choice questions, the average accuracy is generally lower than that of the LLMs. For the open-ended questions, while human participants achieved performance comparable to some LLMs (e.g., GPT-3.5-Turbo, Mixtral-8x7b, and Llama-3-70b), we observe that advanced LLMs can notably outperform human participants. We find these results encouraging, as they illustrate both the difficulty of our CTF questions and the potential of LLMs in this domain.

**Finding 3:** LLMs exhibit poor capability in matching technical knowledge to specific CTF scenarios, highlighting a crucial area for enhancing LLMs' ability to effectively solve CTF problems.

Given that our open-ended questions were derived from write-ups corresponding to original CTF challenges, the difficulty of these challenges might influence LLMs' responses to these questions. To explore this, we analyze the performance of the five LLMs across open-ended questions for CTF challenges of varying difficulties.

Figure 5 shows the results, where the x-axis represents the difficulty of the original CTF challenges, calculated as the score of the current challenge divided by the highest score in the same competition. Values closer to 1 indicate higher difficulty, while values closer to 0 suggest easier challenges. The data and trends indicate that for all LLMs, accuracy decreases with increasing challenge difficulty. Notably, the GPT-4o model shows significant variation, achieving nearly 80% accuracy for questions with difficulty below 0.1. Yet, it achieves about 45% for questions with a difficulty above 0.9. Even the Llama-3 model, which exhibited the least variation, showed a difference of over 10% in accuracy.

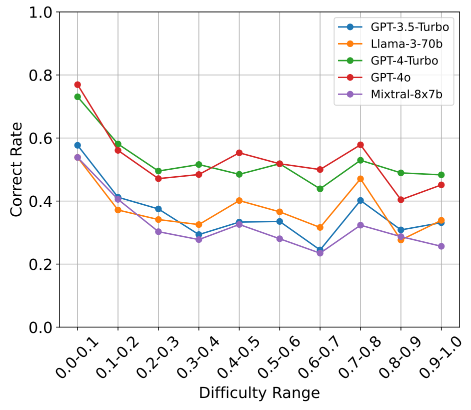


**Figure 5: LLMs' correctness rate by difficulty of original challenges in open-ended questions.**

**Finding 4:** In general, LLMs appear to demonstrate a visible decline in its performance as the difficulty of the original CTF challenges increases, indicating that they struggle in matching technical knowledge to those more challenging CTF scenarios.

However, previous studies [69, 70, 87] indicate that even for simpler CTF problems, the proportion of challenges that LLMs can correctly solve in actual CTF solving processes does not exceed 30%, which is lower than our evaluation results for open-ended questions. Our research and observations from some examples suggest that this discrepancy occurs because, often, even when LLMs correctly match the necessary technical knowledge, they fail to solve problems successfully due to the absence of specific CTF tools, installation failures, or an inability to adjust the solution payload based on feedback from the environment. For example, by analyzing the execution logs of Intercode-CTF [87], we observed that many failures stem from LLM agents lacking necessary Python libraries when writing solution scripts, coupled with their inability to resolve system-related errors during library installation. Typical missing packages include `cryptography` and `gmpy2`. Additionally, even when agents successfully wrote complete scripts with all dependencies installed, they often failed to debug the scripts, ultimately leading to task failure. Failures due to missing tools or guidance were also frequent. Notably, in the `Mini RSA` challenge, while the agent correctly identified the need to factorize the RSA modulus `N`, it failed to use online integer factorization databases, instead attempting alternative factorization algorithms, which resulted in timeout-induced failures. Similar patterns were observed in the execution logs of NYU CTF Bench [70].

**Finding 5:** Our knowledge measurement and analysis of previous works shows that when LLMs attempt to solve CTF challenges and interact with the CTF environment to capture flags, the absence of tools and the presence of unfriendly environments pose significant limitations.

**Broader Implications for the CTF Domain.** Our findings highlight both the encouraging potential and implications of LLMs in the CTF domain. Overall, the strong grasp of technical knowledge suggests LLMs can serve as effective *educational aids*, providing learners with instant access to security breach insight. Specifically, LLMs could function as intelligent tutoring systems and interactive knowledge bases that guide learners through CTF challenges by explaining vulnerabilities and suggesting relevant technical concepts on demand. This is encouraging, as it aligns with the growing trend of using LLMs in educational settings, where they can assist learners in understanding complex topics and provide personalized learning experiences. Moreover, we believe LLMs can also augment the CTF community by serving as valuable tools for *CTF content creation and competition judging*. They could automate the generation of new challenges, develop realistic scenarios, assess competitor performance, and even provide hints during competitions. This reduces the burden on CTF organizers and fostering a more vibrant and engaging competition environment. This could lead to more frequent and diverse CTF events, benefiting both seasoned professionals and aspiring cybersecurity enthusiasts.

## 4 Augmenting LLMs for CTF with CTFAgent

Our above discussion ("Broader Implications for the CTF Domain") sheds light on the promising potential of LLMs in accelerating various aspects of the CTF community (e.g., education, training, and competition). That said, the measurement study findings also uncover *technical challenges* that LLMs face when solving CTF. To effectively address them and unleash the full potential of LLMs in the CTF domain, we propose CTFAgent to augment LLMs for CTF. Below, we first elaborate on how findings from our measurement study inform the design of CTFAgent and then present the design.

### 4.1 Reflection from Measurement Findings

In Findings 1 and 2 of §3.4, we observed that LLMs demonstrate a strong grasp of CTF technical knowledge. Even in the relatively low-performing Web category, the accuracy exceeds 70%, highlighting their significant potential for automating the solution of CTF challenges. Despite these encouraging observations, the following limitations motivate our further exploration and augmentation of LLMs for CTF challenges:

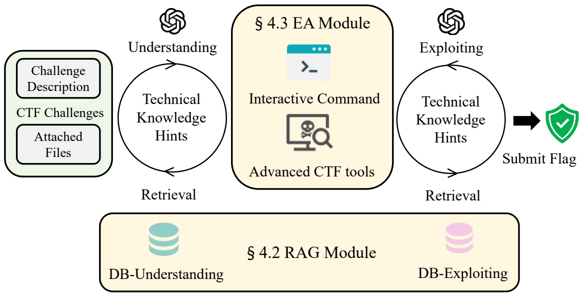


**Figure 6: Overview of CTFAgent.**

In Finding 3, we observed that LLMs struggle to match technical knowledge to specific CTF scenarios, a challenge also common among human CTF participants due to the vast and complex nature of technical details, which can be difficult to recall. As a result, human solvers often rely on web browsers, personal wikis, or blogs to retrieve detailed hints and knowledge for their problem-solving efforts. In Finding 4, we discovered that for complex and difficult CTF scenarios, it is even more difficult for LLMs to precisely provide effective technical knowledge to solve CTF challenges, which increasingly shows that for a given CTF scenario, we need an effective means to provide LLMs with accurate technical knowledge to imitate the behavior of human player to retrieve external resources. As a result, we design a customized version of RAG, called *two-stage RAG*, which allows LLMs to perform searches based on CTF scenarios and more accurately match technical knowledge during both the Understanding and Exploiting phases.

In Finding 5, we discovered that the usefulness of the interaction environment affects the performance of LLMs in CTF scenarios. Human CTF participants often require a variety of specialized tools to complete complex problem-solving processes. Therefore, we aim to simplify operations for LLMs within the CTF environment by providing them with interactive commands and advanced tools through *Interactive Environmental Augmentation.*

**CTFAgent Overview.** Based on the design principles above, we designed CTFAgent as illustrated in Figure 6. CTFAgent primarily comprises two modules: the Retrieval-Augmented Generation (RAG) and Environmental Augmentation (EA). Upon receiving a CTF challenge, including both the challenge description and attached files, CTFAgent initially engages with the EA module. This interaction involves executing commands such as `cat` or decompilation instructions to read the code within the CTF challenge. Once the RAG module detects that the LLM has received code from the EA, it uses DB-Understanding to employ the code as a key. This facilitates the retrieval of the two most closely related pieces of technical knowledge based on vector similarity, which are then returned to the LLM along with the code itself as hints. Following the LLM's successful understanding, identification of vulnerabilities, and formulation of exploitation ideas, it attempts to execute commands within the EA module to exploit these vulnerabilities and hence, discover the flag. Throughout this process, the RAG module employs DB-Exploiting to retrieve technical knowledge closely aligned with the LLM's exploit ideas, aiding in capturing the flag. This continues until the LLM successfully secures the flag or terminates the problem-solving attempt. Details of these two modules are presented below.

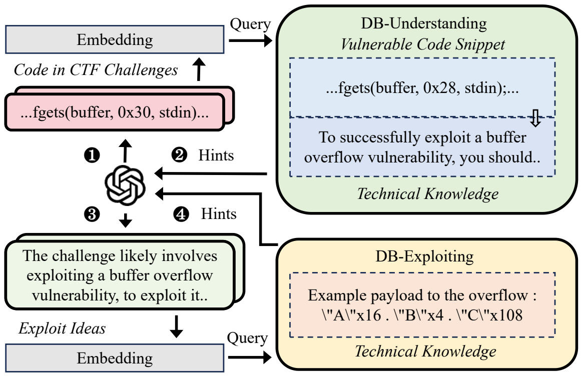

Embedding
Query
DB-Understanding
Vulnerable Code Snippet
Code in CTF Challenges
...fgets(buffer, 0x30, stdin)...
...fgets(buffer, 0x28, stdin);...
To successfully exploit a buffer overflow vulnerability, you should..
❶
❷ Hints
Technical Knowledge
❸
❹ Hints
The challenge likely involves exploiting a buffer overflow vulnerability, to exploit it..
DB-Exploiting
Example payload to the overflow : \"A\"x16 . \"B\"x4 . \"C\"x108
Exploit Ideas
Query
Embedding
Technical Knowledge

**Figure 7: The architecture of two-stage RAG system. Firstly, upon reading the code within a CTF challenge, the DB-Understanding conducts a search based on the code and returns Hints. When the LLM proceeds with subsequent exploitation, each output of Exploit Ideas is searched in the BD-Exploit and Hints are returned accordingly.**

## 4.2 Two-stage RAG for CTF Knowledge

To construct technical knowledge pieces as the database for RAG, the optimal retrieval results should be condensed and focused[2] CTF knowledge trunks. These trunks should include relevant background on CTF scenarios, causes of vulnerabilities, and ideas for exploitation. Furthermore, we need to provide CTF knowledge via RAG at different stages of CTF. During the Understanding phase, the LLM needs to search for potential vulnerabilities based on relevant code snippets. Similarly, during the Exploit phase, it requires knowledge on how to effectively exploit a specific vulnerability. To accommodate these two distinct scenarios, we design a two-stage RAG system, consisting of RAG-Understanding and RAG-Exploiting, whose architecture is shown in Figure 7.

**RAG-Understanding** aims to assist LLMs in better identifying potential vulnerabilities in the tested CTF-related code. Vulnerability code snippets serve as the retrieval key in the Understanding scenario. During the offline preparation, we use another LLM to extract or reconstruct relevant vulnerability code snippets from original write-ups and establish their mapping to the extracted technical knowledge trunks. During the online testing, when the LLM solving the problem accesses code files via command-line tools, RAG-Understanding automatically retrieves the vulnerable code snippets based on the content of the code using cosine similarity ranking and returns the corresponding technical knowledge trunks as hints to the LLM. This process effectively informs the LLM about potential vulnerabilities in the current code, aiding in understanding the problem and facilitating the next steps for exploitation.

**RAG-Exploiting** focuses on the Exploit phase, where the LLM has already recognized potential vulnerabilities in the CTF scenario. At this stage, our goal is to assist it in retrieving knowledge on how to exploit these vulnerabilities. Since the CTF knowledge trunks contain both the CTF scenario and exploit methods, retrieval is directly based on the similarity of the trunks themselves. Whenever the LLM outputs a potential exploitation idea ("Exploit Idea" in Figure 7), we use this output to retrieve more specific and effective strategies for exploiting the vulnerability, using relevant CTF tools, writing Python scripts, and more.

[2]Prior research [69, 70] indicates that as the context for LLMs increases, issues such as incoherence and forgetting may arise.

## 4.3 Interactive Environmental Augmentation

As mentioned in the second design principle in §4.1, providing a more potent CTF environment is as important as, if not more important than, technical knowledge. However, previous work on CTF environments [69, 70, 87] provided only static command-line access in sandboxed Linux machines and non-specialized, general-purpose tools. Thus, in the Environmental Augmentation (EA) module of CTFAgent, we aim to provide a more interactive CTF environment by integrating interactive command lines and advanced CTF tools.

**Interactive Commands.** This represents our enhancement to command-line tools. Replicating previous work, we observed numerous difficulties LLMs faced while interacting with the accompanying static command-line tools. For example, without timely prompts, LLMs may fail to solve a problem because they cannot read a CTF attachment due to insufficient user permissions; or they cannot interact in real-time with remote servers using the `netcat` tool, which only accepts results after the pipeline outputs EOF, preventing LLMs from testing payloads and receiving immediate feedback, significantly impacting problem-solving. Based on these observations, we made substantial adjustments to the Command Line environment as follows.

First, we provided targeted hints based on potential issues LLMs might encounter when using different commands, which we refer to as *tool use hints*. We manually collected common issues encountered by LLMs during the interaction, such as difficulties in using decompilation tools, and crafted specific hints. When LLMs invoke related tools, timely prompts are provided to enhance their ability to use various tools and commands.

For example, we observed that when CTFAgent involved in some `Reverse` challenges, after it has read the disassembled code of a given attached binary executable, it may start writing solving scripts immediately, without reading the discompile code, even if that is far easier to understand. In that case, if CTFAgent tries to use the `disassemble` function call, a prompt saying "Don't rush to analyze, try looking at the decompiled results." would be given. All tool use hints are included in a json file, making it easily scalable.

Second, we upgraded the Command Line from "static" to "dynamic," meaning that we enable LLMs to receive real-time feedback from the CTF environment, akin to human interaction. Through our adjustments and alignment between LLMs and CTF environment subprocesses, LLMs can now use `netcat` in a line-by-line input manner. We implemented this functionality by providing three additional function calls to CTFAgent:

- `start_nc_session`. CTFAgent can create a new `netcat` session using this tool with ip and port passed as parameters. This tool will return the initial response of the session to CTFAgent.
- `nc_send_line`. After a session been created, CTFAgent can interact with it via this tool. By adjusting the `stdout` pipe to non-blocking, CTFAgent can receive the result of sending some data immediately, the session would still be retained unless broken.
- `close_nc_session`. If a session is broken or no more needed, CTFAgent can close it using this tool.

With each output from LLM, our augmented Command Line immediately presents the results of this interaction.

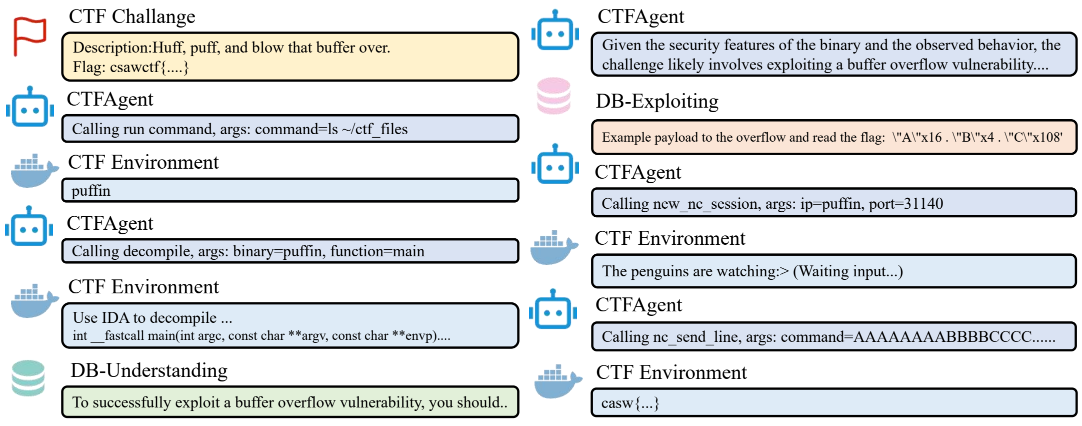

CTF Challange
Description:Huff, puff, and blow that buffer over.
Flag: csawctf{....}
CTFAgent
Calling run command, args: command=ls ~/ctf_files
CTF Environment
puffin
CTFAgent
Calling decompile, args: binary=puffin, function=main
CTF Environment
Use IDA to decompile ...
int __fastcall main(int argc, const char **argv, const char **envp)....
DB-Understanding
To successfully exploit a buffer overflow vulnerability, you should..
CTFAgent
Given the security features of the binary and the observed behavior, the challenge likely involves exploiting a buffer overflow vulnerability....
DB-Exploiting
Example payload to the overflow and read the flag: \"A\"x16 . \"B\"x4 . \"C\"x108'
CTFAgent
Calling new_nc_session, args: ip=puffin, port=31140
CTF Environment
The penguins are watching:> (Waiting input...)
CTFAgent
Calling nc_send_line, args: command=AAAAAAAABBBBCCCC......
CTF Environment
casw{...}

**Figure 8: An example illustrating the working progress of CTFAgent.**

**Advanced CTF Tools.** These are essential for solving CTF challenges, just as they are for human CTF players. For instance, without sophisticated code auditing tools and code editors, it can be nearly impossible to track and examine vulnerable code or locate vulnerabilities within a large project. To address this, we have provided LLMs with a more powerful toolkit for CTF challenges. For example, in previous work [69, 70], the decompiler used was the community edition of Ghidra [15], which struggled with issues like failing to correctly identify arrays and producing poorly readable decompiled code. This impacted the performance of LLMs in CTF scenarios that heavily rely on decompilers, such as Reverse and Pwn. Therefore, we upgraded the decompilation tool available to LLMs to IDA Pro [21], which offers more readable and understandable decompiled code.

### 4.4 An Illustrative Example

To demonstrate the workflow of CTFAgent and how its modules are coordinated, we present an example in the form of a CTF challenge from the NYU CSAW called "puffin," categorized under the Pwn type. The challenge provides a binary executable file, whose simplified code is given below:

```
int main() {
  char buffer[0x20]; int secret_value = 0;
  printf("The penguins are watching: ");
  fgets(buffer, 0x30, stdin);
  if (secret_value) system("cat /flag.txt");
  else printf("penguins\n");
  return 0;
}
```

In this challenge, the binary file (compiled from the above source code) is deployed on a remote server. Players must connect to the remote server using tools like nc from their local machines to interact with the remote service and capture the flag. Players first need to use appropriate tools to decompile the given binary program and, based on the decompiled code, identify that the challenge's vulnerability is a buffer overflow. They must overflow the buffer in such a way as to modify the value of `secret_value` out-of-bounds, ultimately capturing the flag.

Figure 8 illustrates the process of CTFAgent solving this challenge. Upon completing the decompilation of an attachment within EA and accessing the core code, RAG's DB-Understanding performs a vector similarity search based on the vulnerable code. It identifies that the code most closely aligns with technical knowledge regarding buffer overflow and thus returns this piece of knowledge as a hint to the LLM. Upon reviewing this hint, the LLM generates an exploit idea targeted at buffer overflow. When RAG's DB-Exploit receives this idea, it searches its database for the most closely related piece of technical knowledge, which includes an example payload for buffer overflow. This is returned as a hint to the LLM too. Leveraging this example payload, the LLM successfully executes the exploit and captures the flag.

## 5 Evaluation of CTFAgent

Following § 3.3, this section evaluates CTFAgent in response to the following research questions:

- **RQ3 (Performance)**: How does CTFAgent perform in automated CTF challenge solving compared to the methods proposed in previous work?
- **RQ4 (Ablation)**: What roles do the two modules of CTFAgent play in the process of solving CTF challenges?
- **RQ5 (Practicality)**: Is CTFAgent effective in recent real CTF competitions?
- **RQ6 (Failure)**: In cases where CTFAgent fails to solve a challenge, what are the reasons behind these failures, and what insights do they provide for future research?

### 5.1 Evaluation Settings

In the evaluations that follow, we primarily use the advanced GPT-4 model, due to its peak performance during benchmark phases, as our test subject. We use Intercode-CTF [87] and NYU-CTF [70] as our evaluation datasets and baselines, respectively. These datasets are chosen because each comes with its corresponding environment, allowing us to directly use the built-in environments of these datasets as our baselines. To clarify, we do *not* use CTFKnow here, as it is designed for measuring LLMs' knowledge acquisition ability (and it is already maintained in CTFAgent's RAG system), not for evaluating their end-to-end CTF solving ability.

The Intercode-CTF dataset comprises 100 CTF challenges collected from the picoCTF [25] platform. The Intercode-CTF dataset contains 100 challenges from picoCTF [25]. Since picoCTF lacks a `Misc` category but has `General Skills`, we use `Misc` to denote

`General Skills` for consistency. As for the NYU-CTF Dataset, it includes 200 CTF challenges from the CSAW competition. We use these two dataset to test the overall performance of CTFAgent.

Our code is developed based on the environment provided by the NYU-CTF Dataset [70] To ensure the stability and reproducibility of our results, we set the temperature parameter of the GPT-4-Turbo model to 0 and for each CTF challenge in the two datasets, we conduct a single test iteration. However, regarding the maximum number of interaction rounds for CTFAgent to solve each challenge, we use 30 rounds for both datasets.

## 5.2 Performance Evaluation (RQ3)

**Table 2: Performance of CTFAgent, CTFAgent-w/o-RAG, CTFAgent-w/o-EA, and Baseline on Intercode-CTF Dataset.**

| Model | | CTFAgent | Intercode | CTFAgent -w/o-RAG | CTFAgent -w/o-EA |
|---|---|---|---|---|---|
| Pwn | ✓✗ | 100% (4) | 25% (1) | 75% (3) | 0% (0) |
| | ✓ | 50% (2) | 25% (1) | 50% (2) | 0% (0) |
| Reverse | ✓✗ | 78% (21) | 30% (8) | 74% (20) | 74% (20) |
| | ✓ | 70% (19) | 26% (7) | 48% (13) | 59% (16) |
| Misc | ✓✗ | 91% (30) | 70% (23) | 94% (31) | 85% (28) |
| | ✓ | 91% (30) | 61% (20) | 91% (30) | 82% (27) |
| Crypto | ✓✗ | 79% (15) | 53% (10) | 58% (11) | 63% (12) |
| | ✓ | 58% (11) | 26% (5) | 32% (6) | 53% (10) |
| Forensics | ✓✗ | 67% (10) | 33% (5) | 60% (9) | 47% (7) |
| | ✓ | 60% (9) | 33% (5) | 40% (6) | 40% (6) |
| Web | ✓✗ | 100% (2) | 50% (1) | 100% (2) | 100% (2) |
| | ✓ | 100% (2) | 50% (1) | 100% (2) | 100% (2) |
| Total | ✓✗ | **82% (82)** | 48% (48) | 76% (76) | 69% (69) |
| | ✓ | **73% (73)** | 39% (39) | 59% (59) | 61% (61) |

**Table 3: Overall performance of CTFAgent, CTFAgent-w/o-RAG, CTFAgent-w/o-EA, and Baseline on the NYU CTF Dataset. We obtained experimental data of NYU CTF directly from the paper by Shao *et al.* [70].**

| Model | CTFAgent | NYU CTF | CTFAgent -w/o-RAG | CTFAgent -w/o-EA |
|---|---|---|---|---|
| Pwn | 7.89% (3) | 5.08% | 5.26% (2) | 5.26% (2) |
| Reverse | 11.76% (6) | 9.80% | 11.76% (6) | 5.88% (3) |
| Misc | 16.67% (4) | 0% | 12.5% (3) | 8.33% (2) |
| Crypto | 3.77% (2) | 0% | 1.89% (1) | 1.89% (1) |
| Forensics | 20% (3) | 5.26% | 6.67% (1) | 0% (0) |
| Web | 0% (0) | 1.92% | 0% (0) | 0% (0) |
| Total | **9% (18)** | 4% | 7.5% (13) | 4% (8) |

Table 2 presents the test results on the Intercode-CTF dataset. We meticulously documented the performance of CTFAgent and the Intercode framework on each challenge. If the LLM successfully submitted the correct flag, we denoted this outcome with ✓, indicating that the LLM fully solved the challenge. If the LLM failed to solve the challenge correctly but generated a valid approach during the attempt, correctly identifying the vulnerability associated with the CTF challenge, we considered this a partial completion and marked it with a combination of ✓ and ✗. Otherwise, the result was recorded as ✗. Based on these data, we observe that, overall, the CTFAgent framework has enhanced the capability of LLMs in automatically solving CTF problems by 85%, improving from the original 39 out of 100 to 73 out of 100. Moreover, CTFAgent outperforms the Intercode-CTF baseline across every category of CTF challenge. We interpret the results as highly encouraging.

**Table 4: Performance of CTFAgent-o1-preview in Intercode-CTF Dataset. The number of challenges solved is presented as (GPT-4-Turbo solved + o1-preview newly solved).**

| Model | CTFAgent-o1-preview |
|---|---|
| Pwn | 50% (2+0) |
| Reverse | 85% (19+4) |
| Misc | 94% (30+1) |
| Crypto | 68% (11+2) |
| Forensics | 87% (9+4) |
| Web | 100% (2+0) |
| Total | **84% (73+11)** |

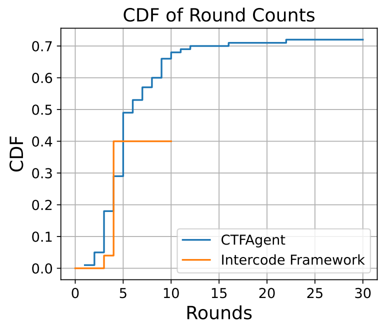


**Figure 9: Cumulative distribution function for the number of rounds taken by CTFAgent (GPT-4-Turbo) and Intercode framework to complete the solved tasks.**

Furthermore, to fully harness the potential of CTFAgent, we conducted an additional evaluation using OpenAI's newly released SOTA o1 model (version `o1-preview`) [22]. Due to the high cost associated with this model, our evaluation focused on challenges from the Intercode-CTF Dataset that remained unsolved by CTFAgent using GPT-4-Turbo. Our preliminary observations suggest that challenges solved by GPT-4-Turbo can be readily addressed by the o1 model. Given that the `o1-preview` model lacks function-calling capabilities, we adapted the ReAct [89] prompt template.

As illustrated in Table 4, CTFAgent-o1-preview successfully solved an additional 11 challenges compared to CTFAgent with GPT-4-Turbo. This finding indicates that CTFAgent's performance can be significantly enhanced by employing more powerful LLMs.

We compiled statistics on the cumulative distribution function for the number of rounds needed to solve challenges in the Intercode CTF Dataset for both the baseline and CTFAgent. As shown in Figure 9, the baseline averaged 3.9 rounds, with no problems solved beyond the fourth round. In contrast, CTFAgent averaged 5.6 rounds, solving some challenges in over 20 rounds. This demonstrates that CTFAgent effectively maintains context throughout the process, significantly enhancing the LLM's ability to tackle more complex CTF problems requiring lengthy rounds interaction.

We present the evaluation results of CTFAgent on NYU CTF Dataset in Table 3. Although NYU CTF Dataset has notably higher difficult challenges, CTFAgent also gains remarkable progress. Totally 18 of 200 challenges solved, CTFAgent makes an improvement of 120% compared with the NYU CTF Baseline (8 challenges solved).

**Table 5: Performance of CTFAgent and NYU Framework in picoCTF2024.**

| Type | CTFAgent | NYU CTF |
|---|---|---|
| Misc(8) | 575'(7) | 225'(4) |
| Pwn(9) | 100'(1) | 0'(0) |
| Web(7) | 200'(3) | 200'(3) |
| Forensics(8) | 400'(4) | 400'(4) |
| Crypto(6) | 300'(2) | 0'(0) |
| Reverse(7) | 300'(2) | 100'(1) |
| Total | **1875'(19)** | 925'(12) |
| Rank | **Top 23.6%** | Top 47.2% |

Which indicates CTFAgent has huge advantages in more challenging CTF challenges as well.

## 5.3 Ablation Study (RQ4)

To investigate the individual contributions of the two modules within CTFAgent, we conduct ablation experiments using the following two variants of CTFAgent:

**CTFAgent-w/o-RAG**: In this variant, the LLM solves problems without any hints, relying solely on its own CTF knowledge and knowledge matching abilities.

**CTFAgent-w/o-EA**: Here, the LLM can only use a native static command-line environment identical to the baseline, equipped with only the most basic CTF tools.

The evaluation results of these three variants on the Intercode CTF Dataset and the NYU Dataset are listed in Table 2 and Table 3. It is evident that the performance of both CTFAgent-w/o-RAG and CTFAgent-w/o-EA is significantly inferior to that of CTFAgent. We observe the following phenomena: In Intercode CTF Dataset, the number of challenges where CTFAgent-w/o-RAG cannot correctly identify vulnerabilities increases from 18 challenges in the case of CTFAgent to 24, and the number of challenges it could solve entirely decreases by 14. Meanwhile, the number of challenges CTFAgent-w/o-EA could solve entirely decreases significantly as well, from 73% to 61% compared to CTFAgent. This indicates that the RAG module aids CTFAgent both in correctly identifying potential vulnerabilities in challenges and in exploiting them, while the EA module primarily plays a role during the Exploiting phase of CTFAgent, which aligns perfectly with the design motivations and principles of CTFAgent. We can draw the same conclusion from the experimental results of the NYU CTF Dataset.

Simultaneously, we observe that the performance of CTFAgent-w/o-RAG and CTFAgent-w/o-EA on some challenges is even worse than the baseline, which indicates that in certain CTF challenges, the RAG system and the interactive environment play critical roles. Without either of them, the overall system performance quickly drops under the CTFAgent framework.

## 5.4 Practicality Study (RQ5)

To assess the practicality of CTFAgent, we opt for an evaluation beyond standardized datasets by selecting the picoCTF2024, held in May 2024[26], to evaluate CTFAgent. picoCTF, hosted by Carnegie Mellon University, is a CTF competition renowned internationally. The picoCTF2024 featured 46 challenges, with 6,957 valid participants. The challenges, varying in difficulty, were assigned points

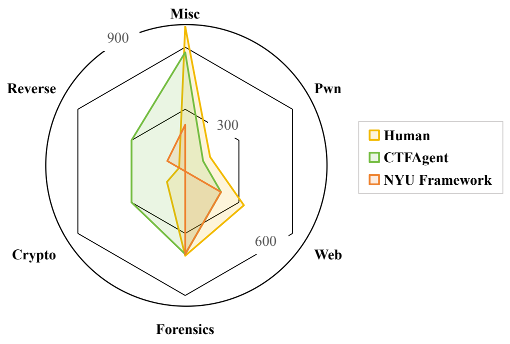


**Figure 10: Average score distribution of all teams with 1,875 points, CTFAgent, and NYU Framework in picoCTF 2024.**

ranging from 25 to 500. We conduct a practicality analysis using CTFAgent and NYU Framework, with the results detailed in Table 5. The competition outcomes reveal that CTFAgent successfully solved 19 CTF challenges, amassing 1,875 points, thereby ranking in the top 23.6% of all participating teams, significantly outperforming NYU Framework. This indicates that CTFAgent can achieve promising results in real CTF competitions.

We conduct an in-depth analysis of the performance disparities between CTFAgent and human CTF participants of comparable skill levels across various challenge categories. This comparison is intended to reveal differences in problem-solving characteristics between CTFAgent and human participants of similar overall proficiency across different types of challenges. We compile the average scores of all human teams who scored the same as CTFAgent, with the results presented in Figure 10. We observe that in the Forensics and Pwn categories, the performance gap between CTFAgent and human contestants is minimal, which is encouraging as CTFAgent manifests a human CTF player-level proficiency. More promisingly, CTFAgent demonstrates superior performance in the Crypto and Reverse categories compared to its human counterparts.

This superiority can be attributed to the challenges in these categories primarily requiring the understanding of source code and the identification of vulnerabilities therein, areas where the capabilities of LLM, augmented by RAG, have a considerable advantage. Conversely, CTFAgent underperforms relative to human participants in the Misc and Web categories. This underperformance is due to these challenges necessitating the extensive use of cybersecurity tools, such as Burp Suite, which involve multi-modal capabilities that CTFAgent's current environment does not yet support. Addressing this requires further research; see §6 for further discussion.

## 5.5 Failure Analysis (RQ6)

**Table 6: Reasons for Failure of CTFAgent in the Intercode-CTF and the NYU CTF Dataset Subset.**

| Type of Failures | Intercode CTF Dataset | NYU CTF Dataset |
|---|---|---|
| Give up | 3.57% (1) | 36.81% (67) |
| Max rounds | 82.13% (23) | 43.41% (79) |
| Context length exceeded | 14.30% (4) | 15.38% (28) |

We analyze the reasons why CTFAgent cannot correctly complete the entire problem-solving process on the two tested baselines, with the results shown in Table 6. It is evident that exceeding the maximum number of attempts is the primary reason for failure across both datasets. Based on a manual analysis of these instances, we believe this may primarily be due to two observations:

- The setting of Interactive Cmd leads CTFAgent to receive many rounds of command-line outputs, and each time, it can only process a limited amount of information, which increases the consumption of attempts.
- After CTFAgent's initial solving approach proves incorrect, even if it identifies the next vulnerability to attempt, it does not continue trying but instead outputs a lot of irrelevant content, such as the significance of CTF competitions, etc.

These observations shed light on future work: we anticipate to further enhance CTFAgent's performance by employing techniques like Tree of Thought (ToT)[88] or Graph of Thought (GoT) [36], which may presumably enhance the LLM's ability to persist in attempting more possible solutions.

## 6 Discussion

### 6.1 CTF Automation for Education & Research

The automation of CTF challenge solving through LLMs carries significant implications for both cybersecurity education and cutting-edge research. From an educational perspective, LLM-powered automation can serve as an intelligent "copilot" for cybersecurity learners [31, 69, 77], helping them rapidly identify attack surfaces across diverse CTF challenges by observing and learning from automated solving processes. This addresses a critical need in security training where the volume and complexity of modern vulnerabilities often outpace traditional teaching methods.

At the research frontier, CTF automation represents a crucial stepping stone toward more advanced AI-powered offensive security capabilities, as evidenced by major initiatives like DARPA's Cyber Grand Challenge [10] and the recent AI Cyber Challenge (AIxCC) [2] with its multi-million dollar prize pool. As noted in [38], the accelerating pace of software development has made automated vulnerability discovery and remediation not just desirable but imperative — with CTF environments serving as ideal testbeds for developing and evaluating these capabilities. The successful automation of CTF solving would directly contribute to addressing fundamental challenges in automated vulnerability mining and exploit generation, subsequently enhancing the security of real-world systems before real attacks occur [32–34].

### 6.2 Comparing With Prior Work

While prior studies such as Intercode-CTF [87] and NYU CTF Bench [70] have made valuable contributions by evaluating LLMs' end-to-end CTF problem-solving capabilities, our work makes three distinct advances. First, we pioneer a knowledge-centric measurement paradigm through CTFKnow, enabling granular analysis of LLMs' technical mastery—an aspect overlooked by previous benchmarks that treated CTF challenges as mostly "black-box" missions. Second, unlike prior tools that focused primarily on command-line automation, CTFAgent's two-stage RAG and Environmental Augmentation modules are explicitly designed to address the knowledge application gaps (see Finding 3 in §3.5) identified by our measurement study. Third, our empirical results demonstrate both quantitative and qualitative improvements: CTFAgent not only achieves 85–120% performance gains over baseline approaches on established benchmarks but also shows superior human-competitive performance in real-world CTF competitions, validating that our knowledge-focused approach translates into practical advantages beyond incremental automation improvements.

### 6.3 Future Work

We discuss several potential directions for future research and extend CTFKnow and CTFAgent.

**RAG Misguidance.** Our dataset CTFKnow incorporates approaching 2,000 entries serving as the database for the RAG system. However, during evaluation, we observed instances where the RAG system, due to inaccurate retrieval, produced CTF Knowledge that is deviated from the scenario. This inadvertently misguides the LLM's solving approach and reduces both efficiency and accuracy. Enhancements to mitigate this limitation could primarily focus on expanding the dataset size and further improving dataset quality.

**Lack of Multi-modal Capabilities.** In real-world competitions, certain categories of CTF challenges, such as OSINT and social engineering, are closely linked with images, necessitating the extraction of substantial information from visual data. As a research prototype, CTFAgent does not exploit the multi-modal capabilities of LLMs, leading to a deficiency in addressing these types of challenges. Additionally, the inability to employ multi-modal functionalities restricts the use of various sophisticated CTF tools, such as Burp Suite [30], which are conveniently accessible through GUIs, thereby limiting the tool's applicability in these scenarios. We anticipate that future research will focus on integrating multi-modal capabilities [63] to address these limitations.

**Further Strengthening CTFAgent's Reasoning Ability.** In this work, although we adopt two-stage RAG to guide CTFAgent through multi-turn interactions in CTF scenarios, we do not enhance its reasoning capabilities at the fundamental model level. How to effectively strengthen the base model's reasoning ability in CTF scenarios involving multi-tool integration and multi-turn interaction remains a meaningful direction for future research. A potentially effective approach is to incorporate o1-like long CoT models [61], enabling the agent to solve CTF challenges through a reasoning process. Reinforcement learning methods like Tool-Integrated RL [58] or LLM Agent RL [41] can also be introduced to further improve the agent's reasoning in complex CTF scenarios.

### 6.4 Ethics Concerns

We posit that, disregarding misuse—namely, when CTFAgent is solely employed for solving CTF challenges—it aligns fully with ethical standards. The rationale is as follows:

- The data used to construct CTFAgent is derived entirely from publicly available vulnerability technical reports, negating concerns related to disclosure or privacy breaches.
- In CTF competitions, the challenges and their environments typically involve specific vulnerabilities and are thus thoroughly isolated from production systems. Therefore, when CTFAgent is used for CTF challenges, it does not engage in unauthorized attacks on external systems.
- Upon obtaining the flag within a CTF environment, CTFAgent ceases its operation, precluding any further malicious activities.

Should CTFAgent be misused inappropriately, it could indeed encounter ethical dilemmas, such as being used to attack unauthorized web applications or crack software. Mitigating such risks requires concerted community efforts. To minimize potential misuse, CTFAgent will be released by review only, while CTFKnow will remain open source. We also advocate for continued research and safeguards on the use of LLMs in offensive security contexts.

## 7 Related Work

Recently, LLMs have become increasingly relevant in cybersecurity domains. CyberSecEval 2 [37], SecBench [54] and CyberMetric [78] have built benchmarks for evaluating the overall cybersecurity knowledge of LLMs. In downstream cybersecurity tasks, FuzzGPT [46], Fuzz4all [84], and CHATAFL [67] introduced LLMs for fuzzing tasks; Happe *et al.* [50] employed LLMs as partners in penetration testing, while PentestGPT [44] and PenHeal [52] developed tools driven by LLMs for penetration testing; PropertyGPT [64] built an LLM agent for formal verification. Since Thapa *et al.* [76] started testing LLMs' capacity in vulnerability detection, many benchmarks and systems including VulBench [49], GPTScan [74], TitanFuzz [45], LLM4Vuln [73], and works by Khare *et al.* [55] and Ullah *et al.* [79] have been devoted to this direction. In contrast, Pearce *et al.* [68] and ACFix [91] focus on using LLMs for vulnerability repair.

Efforts have been made to benchmark LLMs for solving CTF challenges [69, 70, 75, 86, 87]. Besides LLMs, other neural network systems like AutoPwn [85] have also been introduced for CTF solving progress. Furthermore, AutoCTF [53] employed AI systems in CTF challenge design. Significant efforts have also been made to enhance the quality and educational significance of CTF challenges, including Git-based CTF [83] and Pwnable-Sherpa [56].

## 8 Conclusion

This work has conducted a systematic measurement and augmentation on LLM's capability in CTF challenges. We create a novel and targeted benchmark, CTFKnow, to evaluate LLMs' performance in mastering CTF technical knowledge. With findings obtained from the measurement study, we design an enhancement framework, CTFAgent, to improve LLM performance in this domain. Our evaluation results demonstrate the effectiveness of CTFAgent in enhancing LLM in CTF challenges. We believe this work lays a solid foundation for future in-depth evaluations, understanding, and enhancements of LLMs' abilities in CTF.

## 9 Acknowledgement

We thank the anonymous reviewers and the shepherd for their valuable suggestions and feedback.

## References

[1] 2020. Cybercrime To Cost The World $10.5 Trillion Annually By 2025. https://cybersecurityventures.com/cybercrime-will-cost-the-world-16-4-billion-a-day-in-2021/
[2] 2023. AI Cyber Challenge Opens Registration, Adds $4 Million in Prizes, Shows Scoring Algorithm and Challenge Exemplar. https://www.darpa.mil/news/2023/ai-cyber-challenge-opens
[3] 2023. DEF CON® 27 Hacking Conference Contests & Events. https://defcon.org/html/defcon-27/dc-27-ce.html
[4] 2024. 0CTF 2024. https://ctf.0ops.sjtu.cn/. https://ctf.0ops.sjtu.cn/
[5] 2024. All about CTF. https://ctftime.org/. https://ctftime.org/
[6] 2024. Assistants API Overview. https://platform.openai.com/docs/assistants/overview?context=with-streaming. https://platform.openai.com/docs/assistants/overview?context=with-streaming
[7] 2024. BUUCTF. https://buuoj.cn/. https://buuoj.cn/
[8] 2024. Capture the Flag. https://www.csaw.io/ctf. https://www.csaw.io/ctf
[9] 2024. Capture the Flag for Empowered Cybersecurity Training. https://ine.com/blog/capture-the-flag-for-empowered-cybersecurity-training
[10] 2024. CGC: Cyber Grand Challenge. https://www.darpa.mil/research/programs/cyber-grand-challenge
[11] 2024. Claude 3.5 Sonnet. hhttps://www.anthropic.com/news/claude-3-5-sonnet. https://www.anthropic.com/news/claude-3-5-sonnet
[12] 2024. DeepSeek. https://www.deepseek.com/
[13] 2024. DEFCON. https://defcon.org/. https://defcon.org/
[14] 2024. Function calling. https://platform.openai.com/docs/guides/function-calling. https://platform.openai.com/docs/guides/function-calling
[15] 2024. Ghidra. https://ghidra-sre.org/. https://ghidra-sre.org/
[16] 2024. Google CTF. https://capturetheflag.withgoogle.com/. https://capturetheflag.withgoogle.com/
[17] 2024. gpt-3-5-turbo. https://platform.openai.com/docs/models/gpt-3-5-turbo. https://platform.openai.com/docs/models/gpt-3-5-turbo
[18] 2024. gpt-4. https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4. https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4
[19] 2024. gpt-4o. https://platform.openai.com/docs/models/gpt-4o. https://platform.openai.com/docs/models/gpt-4o
[20] 2024. HITCON. https://hitcon.org/2024/CMT/. https://hitcon.org/2024/CMT/
[21] 2024. IDA. https://hex-rays.com/ida-pro/
[22] 2024. Learning to Reason with LLMs | OpenAI. https://openai.com/index/learning-to-reason-with-llms/
[23] 2024. Meet Llama 3.1. https://llama.meta.com/. https://llama.meta.com/
[24] 2024. Mixtral of experts | Mistral AI | Frontier AI in your hands. https://mistral.ai/news/mixtral-of-experts/. https://mistral.ai/news/mixtral-of-experts/
[25] 2024. picoCTF - CMU Cybersecurity Competition. https://picoctf.org/. https://picoctf.org/
[26] 2024. picoCTF2024. https://play.picoctf.org/events/73/scoreboards. https://play.picoctf.org/events/73/scoreboards
[27] 2024. Top 10 Cyber Hacking Competitions - Capture the Flag (CTF). https://www.geeksforgeeks.org/top-cyber-hacking-competitions-capture-the-flag-ctf/
[28] 2024. UIUCTF 2024. https://2024.uiuc.tf/. https://2024.uiuc.tf/
[29] 2024. VicOne & Block Harbor Spearhead Biggest Automotive Capture the Flag Competition for Cybersecurity Enthusiasts Worldwide. https://vicone.com/company/press-releases/vicone-and-block-harbor-spearhead-biggest-automotive-capture-the-flag-competition-for-cybersecurity-enthusiasts-worldwide
[30] 2025. Burp Suite - Application Security Testing Software. https://portswigger.net/burp
[31] 2025. Microsoft Security Copilot Blog. https://techcommunity.microsoft.com/blog/securitycopilotblog/advancing-security-copilot-with-magic-automating-self-correction-in-nl2kql-and-b/4390932
[32] 2025. Proactive Defense: The Role of Offensive Security in Cybersecurity. https://cloudsecurityalliance.org/artifacts/using-ai-for-offensive-security
[33] 2025. Using AI for Offensive Security. https://cloudsecurityalliance.org/artifacts/using-ai-for-offensive-security
[34] 2025. What is Automated Vulnerability Remediation? https://www.sentinelone.com/cybersecurity-101/cybersecurity/what-is-automated-vulnerability-remediation/
[35] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv:2303.08774* (2023).
[36] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proc. AAAI*.
[37] Manish Bhatt, Sahana Chennabasappa, Yue Li, Cyrus Nikolaidis, Daniel Song, Shengye Wan, Faizan Ahmad, Cornelius Aschermann, Yaohui Chen, Dhaval Kapil, et al. 2024. Cyberseceval 2: A wide-ranging cybersecurity evaluation suite for large language models. *arXiv preprint arXiv:2404.13161* (2024).
[38] David Brumley. 2018. The cyber grand challenge and the future of cyber-autonomy. *USENIX Login* 43, 2 (2018), 6–9.
[39] Tanner J Burns, Samuel C Rios, Thomas K Jordan, Qijun Gu, and Trevor Underwood. 2017. Analysis and exercises for engaging beginners in online CTF competitions for security education. In *USENIX Workshop on Advances in Security Education*.
[40] Daihang Chen, Yonghui Liu, Mingyi Zhou, Yanjie Zhao, Haoyu Wang, Shuai Wang, Xiao Chen, Tegawendé F Bissyandé, Jacques Klein, and Li Li. 2024. LLM for Mobile: An Initial Roadmap. *arXiv preprint arXiv:2407.06573* (2024).
[41] Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Fan Yang, Zenan Zhou, Weipeng Chen, Haofen Wang, Jeff Z Pan, et al. 2025. Learning to Reason with Search for LLMs via Reinforcement Learning. *arXiv preprint arXiv:2503.19470* (2025).
[42] Kevin Chung and Julian Cohen. 2014. Learning obstacles in the capture the flag model. In *USENIX 3GSE*.
[43] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The

power of noise: Redefining retrieval for rag systems. In *Proc. ACM SIGIR*.

[44] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2024. PentestGPT: Evaluating and harnessing large language models for automated penetration testing. In *Proc. USENIX Security*.

[45] Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*. 423–435.

[46] Yinlin Deng, Chunqiu Steven Xia, Chenyuan Yang, Shizhuo Dylan Zhang, Shujing Yang, and Lingming Zhang. 2024. Large language models are edge-case generators: Crafting unusual programs for fuzzing deep learning libraries. In *Proc. IEEE/ACM ICSE*.

[47] Richard Fang, Rohan Bindu, Akul Gupta, Qiusi Zhan, and Daniel Kang. 2024. Teams of LLM Agents can Exploit Zero-Day Vulnerabilities. *arXiv preprint arXiv:2406.01637* (2024).

[48] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[49] Zeyu Gao, Hao Wang, Yuchen Zhou, Wenyu Zhu, and Chao Zhang. 2023. How far have we gone in vulnerability detection using large language models. *arXiv preprint arXiv:2311.12420* (2023).

[50] Andreas Happe and Jürgen Cito. 2023. Getting pwn'd by ai: Penetration testing with large language models. In *Proc. ACM FSE*.

[51] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300* (2020).

[52] Junjie Huang and Quanyan Zhu. 2024. PenHeal: A Two-Stage LLM Framework for Automated Pentesting and Optimal Remediation. *arXiv:2407.17788* (2024).

[53] Patrick Hulin, Andy Davis, Rahul Sridhar, Andrew Fasano, Cody Gallagher, Aaron Sedlacek, Tim Leek, and Brendan Dolan-Gavitt. 2017. {AutoCTF}: Creating diverse pwnables via automated bug injection. In *WOOT*.

[54] Pengfei Jing, Mengyun Tang, Xiaorong Shi, Xing Zheng, Sen Nie, Shi Wu, Yong Yang, and Xiapu Luo. 2024. SecBench: A Comprehensive Multi-Dimensional Benchmarking Dataset for LLMs in Cybersecurity. *arXiv:2412.20787* (2024).

[55] Avishree Khare, Saikat Dutta, Ziyang Li, Alaia Solko-Breslin, Rajeev Alur, and Mayur Naik. 2023. Understanding the effectiveness of large language models in detecting security vulnerabilities. *arXiv preprint arXiv:2311.16169* (2023).

[56] Sung-Kyung Kim, Eun-Tae Jang, Hanjin Park, and Ki-Woong Park. 2023. Pwnable-Sherpa: An interactive coaching system with a case study of pwnable challenges. *Computers & Security* 125 (2023), 103009.

[57] Stela Kucek and Maria Leitner. 2020. An empirical survey of functions and configurations of open-source capture the flag (ctf) environments. *Journal of Network and Computer Applications* 151 (2020), 102470.

[58] Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383* (2025).

[59] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459* (2024).

[60] Zongjie Li, Wenying Qiu, Pingchuan Ma, Yichen Li, You Li, Sijia He, Baozheng Jiang, Shuai Wang, and Weixi Gu. 2024. On the Accuracy and Robustness of Large Language Models in Chinese Industrial Scenarios. In *Proc. ACM/IEEE IPSN*.

[61] Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, et al. 2025. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419* (2025).

[62] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, et al. 2025. Advances and Challenges in Foundation Agents: From Brain-Inspired Intelligence to Evolutionary, Collaborative, and Safe Systems. *arXiv preprint arXiv:2504.01990* (2025).

[63] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. *Advances in neural information processing systems* 36 (2024).

[64] Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, and Yang Liu. 2025. PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation. In *Proc. ISOC NDSS*.

[65] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. InsightPilot: An LLM-empowered automated data exploration system. In *EMNLP: System Demonstrations*.

[66] Wei Ma, Daoyuan Wu, Yuqiang Sun, Tianwen Wang, Shangqing Liu, Jian Zhang, Yue Xue, and Yang Liu. 2025. Combining Fine-Tuning and LLM-based Agents for Intuitive Smart Contract Auditing with Justifications. In *Proc. IEEE/ACM ICSE*.

[67] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large language model guided protocol fuzzing. In *Proc. ISOC NDSS*.

[68] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2023. Examining zero-shot vulnerability repair with large language models. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2339–2356.

[69] Minghao Shao, Boyuan Chen, Sofija Jancheska, Brendan Dolan-Gavitt, Siddharth Garg, Ramesh Karri, and Muhammad Shafique. 2024. An empirical evaluation of llms for solving offensive security challenges. *arXiv:2402.11814* (2024).

[70] Minghao Shao, Sofija Jancheska, Meet Udeshi, Brendan Dolan-Gavitt, Haoran Xi, Kimberly Milner, Boyuan Chen, Max Yin, Siddharth Garg, Prashanth Krishnamurthy, et al. 2024. NYU CTF Dataset: A Scalable Open-Source Benchmark Dataset for Evaluating LLMs in Offensive Security. *arXiv:2406.05590* (2024).

[71] Teotino Gomes Soares, Azhari Azhari, Nur Rokhman, and E Wonarko. 2021. Education question answering systems: a survey. In *Proceedings of The International MultiConference of Engineers and Computer Scientists*.

[72] Nicholas Springer and Wu-chang Feng. 2021. Thunder CTF: Learning Cloud Security on a Dime. *arXiv preprint arXiv:2107.12566* (2021).

[73] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Wei Ma, Lyuye Zhang, Miaolei Shi, and Yang Liu. 2024. LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs' Vulnerability Reasoning. *arXiv:2401.16185* (2024).

[74] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. 2024. Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis. In *Proc. IEEE/ACM ICSE*.

[75] Wesley Tann, Yuancheng Liu, Jun Heng Sim, Choon Meng Seah, and Ee-Chien Chang. 2023. Using large language models for cybersecurity capture-the-flag challenges and certification questions. *arXiv preprint arXiv:2308.10443* (2023).

[76] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. 2022. Transformer-based language models for software vulnerability detection. In *Proc. ACM ACSAC*.

[77] Alba Thaqi, Arbena Musa, and Blerim Rexha. 2024. Leveraging AI for CTF Challenge Optimization. In *Proc. IEEE CIEES*.

[78] Norbert Tihanyi, Mohamed Amine Ferrag, Ridhi Jain, and Merouane Debbah. 2024. Cybermetric: A benchmark dataset for evaluating large language models knowledge in cybersecurity. *arXiv preprint arXiv:2402.07688* (2024).

[79] Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. 2023. Can large language models identify and reason about security vulnerabilities? not yet. *arXiv preprint arXiv:2312.12575* (2023).

[80] U.S. Department of Health and Human Services. 2018. Federal Policy for the Protection of Human Subjects ('Common Rule'). https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-A/part-46#p-46.104(d)(2) Title 45 Code of Federal Regulations Part 46.104(d)(2).

[81] Jan Vykopal, Valdemar Švábenský, and Ee-Chien Chang. 2020. Benefits and pitfalls of using capture the flag games in university courses. In *Proceedings of the 51st ACM Technical symposium on computer science education*. 752–758.

[82] Liwen Wang, Yuanyuan Yuan, Ao Sun, Zongjie Li, Pingchuan Ma, Daoyuan Wu, and Shuai Wang. 2024. Benchmarking Multi-Modal LLMs for Testing Visual Deep Learning Systems Through the Lens of Image Mutation. *arXiv preprint arXiv:2404.13945* (2024).

[83] SeongIl Wi, Jaeseung Choi, and Sang Kil Cha. 2018. Git-based {CTF}: A Simple and Effective Approach to Organizing {In-Course}{Attack-and-Defense} Security Competition. In *2018 USENIX Workshop on Advances in Security Education*.

[84] Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*.

[85] Dandan Xu, Kai Chen, Miaoqian Lin, Chaoyang Lin, and Xiaofeng Wang. 2023. Autopwn: Artifact-assisted heap exploit generation for ctf pwn competitions. *IEEE Transactions on Information Forensics and Security* (2023).

[86] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2024. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems* 36 (2024).

[87] John Yang, Akshara Prabhakar, Shunyu Yao, Kexin Pei, and Karthik R Narasimhan. 2023. Language agents as hackers: Evaluating cybersecurity skills with capture the flag. In *Multi-Agent Security Workshop@ NeurIPS'23*.

[88] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS* (2024).

[89] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[90] Wenhao Yu, Hongming Zhang, Xiaoman Pan, Kaixin Ma, Hongwei Wang, and Dong Yu. 2023. Chain-of-note: Enhancing robustness in retrieval-augmented language models. *arXiv preprint arXiv:2311.09210* (2023).

[91] Lyuye Zhang, Kaixuan Li, Kairan Sun, Daoyuan Wu, Ye Liu, Haoye Tian, and Yang Liu. 2024. Acfix: Guiding llms with mined common rbac practices for context-aware repair of access control vulnerabilities in smart contracts. *arXiv preprint arXiv:2403.06838* (2024).

[92] Li Zhong, Zilong Wang, and Jingbo Shang. 2024. Ldb: A large language model debugger via verifying runtime execution step-by-step. *arXiv preprint arXiv:2402.16906* (2024).