OEDIPUS: LLM-enchanced Reasoning CAPTCHA Solver

Gelei Deng*
Nanyang Technological University
Singapore
gdeng003@e.ntu.edu.sg

Haoran Ou*
Nanyang Technological University
Singapore
haoran.ou@ntu.edu.sg

Yi Liu†
Nanyang Technological University
Singapore
yi009@e.ntu.edu.sg

Jie Zhang
CFAR and IHPC, A*STAR
Singapore
zhang_jie@cfar.a-star.edu.sg

Tianwei Zhang
Nanyang Technological University
Singapore
tianwei.zhang@ntu.edu.sg

Yang Liu
Nanyang Technological University
Singapore
yangliu@ntu.edu.sg

Abstract

CAPTCHAs have become a ubiquitous tool in safeguarding applications from automated bots. Over time, the arms race between CAPTCHA development and evasion techniques has led to increasingly sophisticated and diverse designs. The latest iteration, reasoning CAPTCHAs, exploits tasks that are intuitively simple for humans but challenging for conventional AI technologies, thereby enhancing security measures.

Driven by the evolving AI capabilities, particularly the advancements in Large Language Models (LLMs), we investigate the potential of multimodal LLMs to solve modern reasoning CAPTCHAs. Our empirical analysis reveals that, despite their reasoning capabilities, LLMs struggle to solve these CAPTCHAs effectively. In response, we introduce OEDIPUS, an innovative end-to-end framework for automated reasoning CAPTCHA solving. Central to this framework is a novel strategy that dissects the complex and human-easy-AI-hard tasks into a sequence of simpler and AI-easy steps. This is achieved through the development of a Domain Specific Language (DSL) for CAPTCHAs that guides LLMs in generating actionable sub-steps for each challenge. The DSL is customized to ensure that each unit operation is a highly solvable subtask by LLMs as revealed in our empirical study. These sub-steps are then tackled sequentially using the Chain-of-Thought methodology.

Our evaluation shows that OEDIPUS effectively resolves the studied CAPTCHAs, achieving an average success rate of 63.5%. Remarkably, it also shows adaptability to the most recent CAPTCHA designs introduced in late 2023, which are not included initial study. This prompts a discussion on future strategies for designing reasoning CAPTCHAs that can effectively counter advanced AI solutions.

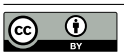
CCS Concepts

• **Security and privacy → Web application security**.

Keywords

Large Language Model, Web Application Security, CAPTCHA

*Both authors contributed equally to this research.
†Corresponding author



CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3744872

ACM Reference Format:
Gelei Deng, Haoran Ou, Yi Liu, Jie Zhang, Tianwei Zhang, and Yang Liu. 2025. OEDIPUS: LLM-enchanced Reasoning CAPTCHA Solver. 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/10.1145/3719027.3744872

1 Introduction

The pervasive threat posed by automated bots has necessitated robust countermeasures to safeguard the integrity and functionality of various applications, leading to the development and wide application of CAPTCHAs [1] (Completely Automated Public Turing test to tell Computers and Humans Apart). Ingeniously tapping into the distinct cognitive capabilities of humans versus the computational limitations of machines, CAPTCHAs present tasks that are trivial for humans but substantially challenging for automated systems. This delineation exemplifies Moravec's paradox [2], which suggests that activities requiring minimal human thought are disproportionately difficult for artificial intelligence to replicate. By designing CAPTCHAs around this concept—creating tasks that are simple for humans but difficult for AI—these tests act as a barrier against automated intrusions.

However, as AI technology evolves, the effectiveness of traditional CAPTCHAs has diminished. Some early designs such as ReCAPTCHA [3] focus on straightforward tasks like text recognition or basic image identification. These tasks leverage the visual and cognitive abilities that are innate to humans but were initially difficult for computer algorithms. With the advance of computer vision [4] and machine learning [5] techniques, these early defenses become increasingly vulnerable to automated solutions [6, 7].

This rapid development necessitates the creation of more complex verification methods, notably reasoning-based CAPTCHAs [8–10]. These methods mark a significant departure from earlier approaches, relying less on object recognition. Instead, they rely on tasks requiring logical reasoning, problem-solving, and interpretation of complex instructions. These activities demand a level of cognitive engagement that current AIs struggle to provide, making reasoning-based challenges a more robust defense mechanism. Despite recent AI advancements, these CAPTCHAs continue to pose a significant obstacle to automated solvers, leading to their adoption by many popular online platforms such as LinkedIn [11], TikTok [12], and Twitter [13].

The landscape of AI technology, particularly with the evolution of Large Language Models (LLMs) [14, 15], has been marked by

significant advancements, notably in reasoning capabilities and multimodal processing. These developments have paved the way for novel approaches to tackling reasoning CAPTCHAs, tasks traditionally considered challenging for AI due to their reliance on human-like cognitive processes. To evaluate the efficacy of these advanced AI models in the context of reasoning CAPTCHA solving, this paper embarks on an empirical investigation, which engages two leading multimodal LLMs, GPT-4V(ision) [16] and Gemini [17], utilizing zero-shot prompting [18] and the Chain-of-Thoughts (CoT) strategy [19] as primary methodologies (Section 3). Our objective is to explore the extent of their capabilities and delineate the boundaries within which these models operate when confronted with various reasoning CAPTCHAs.

The outcomes of our investigation reveal that, despite the unprecedented capabilities of LLMs, they currently fall short in effectively solving reasoning CAPTCHAs. Our analysis yields some key insights: (1) LLMs exhibit a comprehensive understanding of CAPTCHA tasks, including the challenges posed and objectives to be achieved. (2) Unexpectedly, LLMs are capable of deconstructing complex reasoning tasks into simpler, manageable steps and addressing each through the CoT strategy similar to the human being. However, the efficacy of this approach is contingent upon the models' success in accurately completing every step in the sequence; failure of any step inevitably results in the failure of the entire task. (3) The primary reason of models' failure in CAPTCHA solving is their limited capabilities of recognizing objects. While they can recognize and attribute characteristics to singular objects, this ability significantly diminishes when tasked with simultaneously discerning attributes of multiple objects, which is often required in reasoning CAPTCHAs. (4) LLMs face challenges in executing multiple reasoning steps within a single prompt, with a notable increase in errors and hallucinations as the number of required reasoning steps escalates, leading to unsuccessful attempts. (5) Models that undergo supervised fine-tuning (SFT) do not show significant improvement in addressing these challenges. Additionally, there is minimal transferability observed in this task, indicating that fine-tuning a model on one type of CAPTCHA does not necessarily enhance its performance on other types. This lack of transferability complicates the process of tuning a single model to handle multiple CAPTCHA challenges effectively.

This empirical study prompts us to explore the possibility of breaking down an AI-hard reasoning CAPTCHA challenge into a series of AI-easy tasks that can be more readily solved by LLMs. To implement this, we propose a strategy that decomposes a given reasoning CAPTCHA into a sequence of detailed operations, each aligned with the capabilities of LLMs as identified in our empirical findings. To this end, we introduce the CAPTCHA Domain Specific Language (DSL) [20] to regulate this task breakdown process (Section 4). The operations and syntax of this CAPTCHA DSL are meticulously designed to ensure that syntax-correct CAPTCHA DSL scripts contain only operations that are highly solvable by LLMs. We direct LLMs to generate challenge solutions in CAPTCHA DSL scripts that adhere to these syntax principles. A local debugger assists in refining these solutions by identifying and correcting inaccuracies, thereby enabling a systematic approach to generating solutions for reasoning CAPTCHA challenges.

Building upon this, we present an end-to-end framework Oedipus[1] to automate the solving of reasoning CAPTCHAs (Section 5). The workflow initiates with the creation of a DSL script for a given CAPTCHA challenge, followed by its translation into natural language instructions that LLMs can understand and execute. By providing the natural language instructions together with the original challenge, a multimodal LLM is able to solve it step by step. Our framework offers two profound advantages. First, compared to traditional deep learning-based CAPTCHA solving strategies [4, 21], Oedipus does not require any training process or collection of labeled data, which significantly reduces manual efforts. Second, Oedipus is adaptable to new CAPTCHA types as long as the unit operations required for solving them are covered by the DSL.

The efficacy of Oedipus is validated through extensive evaluations and analysis. We deploy Oedipus on 4 types of reasoning CAPTCHAs designed by 2023 and commercially available online. The experimental results are promising, demonstrating that Oedipus achieves a success rate of up to 73.8% in solving these CAPTCHAs on average, with a cost as low as 1.03 USD per 100 CAPTCHA solving. Among these CAPTCHAs, two types have never been solved by any existing solutions discussed in academia. Additionally, Oedipus has shown proficiency in resolving two newly developed CAPTCHAs after 2023 with an average success rate of 44.1%, whose solutions are not used to guide the development of the CAPTCHA DSL generator component. This underscores the versatility of Oedipus: as long as the required operations are within the scope of DSL, Oedipus can consistently perform effectively in CAPTCHA solving without further training process. To benefit the open-source community for future research, we have open-sourced our complete dataset [22] and share the complete source code upon request to avoid potential misuse.

In light of our findings, we propose three strategies for designing CAPTCHAs that could potentially remain unsolvable by LLMs. First, we suggest CAPTCHAs that demand complex reasoning chains beyond LLMs' current reach, pushing the limits of AI problem-solving. Second, we recommend employing adversarial examples [23] to exploit and confuse LLMs' object recognition abilities. Finally, we advocate for CAPTCHAs requiring an understanding of concepts or operations that lie outside the scope of existing LLMs, such as intricate real-world interactions. Acknowledging the rapid advancement of LLMs, these strategies aim to maintain CAPTCHAs as effective security measures by adapting to AI developments.

Ethical Declaration. We emphasize that our research and the development of Oedipus have not been leveraged for any unethical activities or financial gain. We are acutely aware of the ethical implications of our work. In compliance with ethical standards, we refrain from releasing a fully automated CAPTCHA solving tool. Instead, we provide access to a partial solution that generates CAPTCHA resolutions in natural language [22].

2 Background

2.1 CAPTCHAs and CAPTCHA Solver

CAPTCHAs [24, 25] have evolved from simple text puzzles to increasingly complex challenges designed to distinguish humans

[1] Oedipus was a mythical Greek king who answered the Sphinx's riddles correctly, and defeated this monster.

from bots. This continuous evolution responds to advancements in solver algorithms, transitioning from basic OCR techniques [26] to advanced machine learning models [27–29]. As CAPTCHAs become more intricate, solvers adapt accordingly, creating a persistent arms race in web security. This ongoing cycle highlights the critical need for innovative approaches to security, suggesting a gradual shift towards analyzing user behaviors and incorporating AI-driven methods to maintain the delicate balance between user accessibility and protection against sophisticated automated threats.

2.2 Reasoning CAPTCHAs

The development of reasoning CAPTCHAs [9] marks a significant evolution in the field of web security, shifting the challenge from simple pattern recognition to cognitive tasks that require logical reasoning, contextual understanding, and multi-step deduction. Unlike conventional CAPTCHAs, reasoning CAPTCHAs are specifically designed to thwart automated solvers by exploiting capabilities that are currently unique to human cognition. By exploiting the unique human capability for complex reasoning, reasoning CAPTCHAs provide a stronger defense against bots, marking a critical step forward in protecting online interactions. Because reasoning CAPTCHAs demand complex problem-solving skills, existing methods and solvers cannot be directly compared with strategies targeting these cognitive challenges. This separation has led to reasoning CAPTCHAs being studied as a distinct subcategory in prior research, necessitating specialized datasets and evaluation frameworks. Figures 1 to 3 demonstrates several state-of-the art reasoning CAPTCHAs, which we further illustrate in Section 3.

2.3 Large Language Models

LLMs [15] have emerged as a groundbreaking advancement in AI, demonstrating remarkable abilities in understanding and generating natural language across a broad spectrum of applications. The evolution into multimodal LLMs, capable of processing and integrating various types of data such as text and images, underscores their potential to tackle complex, multi-dimensional problems. Notably, the reasoning capabilities inherent in LLMs position them as promising candidates for security tasks, such as penetration testing [30], protocol fuzzing [31], and blockchain security [32]. This makes LLMs potentially suitable for reasoning CAPTCHAs solving. Although the application of LLMs in this context has yet to be thoroughly explored, their sophisticated advancements in handling complex reasoning tasks suggest a significant potential in overcoming the challenges presented by reasoning CAPTCHAs, highlighting a new frontier in the application of AI in web security.

3 Empirical Study

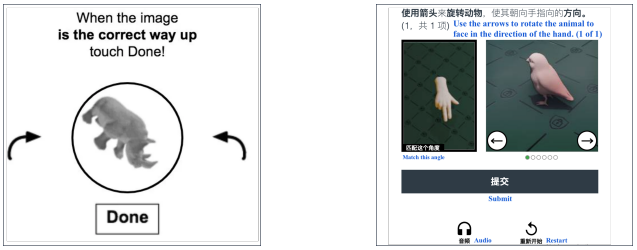
Previous studies [33] have demonstrated the effectiveness of LLMs in solving traditional verification challenges, such as those offered by ReCAPTCHA. However, the potential of LLMs in addressing the more complex reasoning tasks in verification challenges remains unexplored. To bridge this gap, we embark on an empirical study aimed at understanding the capabilities of LLMs in solving reasoning CATPCHAs. This investigation is structured around three pivotal research questions:

- **RQ1 (Categorization)**: What are the different types of reasoning tasks present in reasoning CAPTCHAs?
- **RQ2 (Effectiveness)**: How effective are LLMs in solving reasoning CAPTCHAs, and what factors influence their success rate?
- **RQ3 (Enhancements)**: Can supervised fine-tuning (SFT) improve LLM performance in reasoning CAPTCHA solving?

Below, we detail our strategies to these research questions.

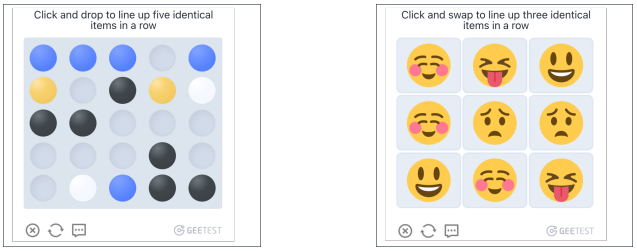
3.1 CAPTCHA Categorization (RQ1)

Dataset Construction. To comprehensively address RQ1, our primary objective is to categorize existing reasoning CAPTCHAs. For this purpose, we embark on a large-scale enumeration of CAPTCHAs available online. Specifically, we review all CAPTCHAs referenced in recent research works [34–37], with a focus on commercially available reasoning CAPTCHAs. It is important to note that our study excludes CAPTCHAs solely discussed in academic research, such as those in [38], due to the absence of readily available APIs and their untested effectiveness in real-world scenarios. Consequently, we have compiled a dataset comprising 5 types of reasoning CAPTCHAs from 3 different vendors, detailed in Table 1.



(a) CAPTCHA 1 by Arkose Labs (b) CAPTCHA 2 by Arkose Labs

Figure 1: Rotation CAPTCHA examples.

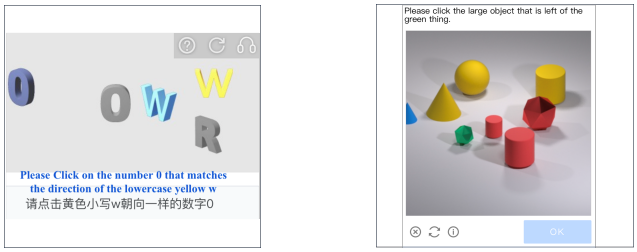


(a) CAPTCHA 1 by GeeTest (b) CAPTCHA 2 by GeeTest

Figure 2: Bingo CAPTCHA examples.

CAPTCHA Categorization. In alignment with preceding works [34, 36], our categorization of CAPTCHAs is fundamentally based on the reasoning tasks embedded in their designs. Ultimately, we identify 3 distinct categories of reasoning CAPTCHAs:

(1) **Rotation CAPTCHAs.** Displayed in Figure 1, rotation CAPTCHAs compel users to adjust an object's orientation to match that of a reference object. This type has evolved from the conventional rotation CAPTCHAs (seen in Figure 1(a)) described in [36], which merely required upward orientation of objects. The newer variants, as depicted in Figure 1(b), introduce increased complexity



(a) CAPTCHA 1 by YiDun (b) CAPTCHA 2 by GeeTest

Figure 3: 3D CAPTCHA examples.

by necessitating users to discern and align with the orientation of a reference item, such as a finger in the provided example.

(2) **Bingo CAPTCHAs.** In Figure 2, bingo CAPTCHAs present the challenge of identifying elements on a board and rearranging them to form a line of identical items. These CAPTCHAs vary greatly in terms of element types and manipulation rules, contingent on the provider. For instance, Figure 2(a) allows arbitrary swapping of any two items, while Figure 2(b) restricts users to only swap adjacent items.

(3) **3D Logical CAPTCHAs.** Illustrated in Figure 3, 3D Logical CAPTCHAs task users with selecting an object from a 3D space, based on intricate logical relationships involving attributes like shape, color, and orientation. For example, Figure 3(a) challenges users to identify the number 0 that shares the same orientation as a yellow letter W; whereas Figure 3(b) requires selecting the larger object positioned to the left of a green object.

The unique set of challenges posed by each CAPTCHA category necessitates specialized solving strategies. By categorizing these CAPTCHAs, our goal is to gain a thorough understanding of the array of reasoning tasks they encompass and critically assess the proficiency of LLMs in tackling these diverse challenges.

Two noteworthy observations emerge from our study. Firstly, there is a significant diversity among vendors in the types of reasoning tasks developed, with little overlap in design. This finding is underscored by our discovery that companies have obtained patents for their unique designs, as verified in [37]. Secondly, tasks can be broadly classified into two categories based on their design complexity and the feasibility of exhaustively cataloging their variations. The first category, "Limited Variability" tasks, such as rotation tasks, presents a relatively small and finite set of challenge variations. Theoretically, it is possible to collect all possible variations of these tasks, and the design of a new task under this type requires the generation of new elements within the tasks (such as a new animal type in rotation tasks). Conversely, the "Dynamic Complexity" tasks, exemplified by Geetest visual reasoning tasks, generates challenges through the combination of multiple elements or objects in various configurations. Due to this dynamic complexity, the number of potential task variations is vast and continually evolving, making it impractical to collect all possible challenges.

3.2 LLMs for Solving CAPTCHAs (RQ2)

We further evaluate if LLMs can be used to solve existing reasoning CAPTCHA problems, and if not, what are the key challenges that hinder their application.

3.2.1 Evaluation Strategy. Our evaluation is designed to probe two critical facets of LLMs performance: (1) their capabilities to accurately comprehend the given CAPTCHA task and (2) their proficiency in methodically executing the necessary steps to resolve the CAPTCHA challenge. To do this, we employ two distinct approaches to test these capabilities: zero-shot prompting [18] and Chain-of-Thoughts (CoT) strategy [19]. These methodologies have been previously applied to diverse reasoning tasks, including symbolic [39] and mathematical reasoning [40], offering a proven framework for assessing LLMs. To rigorously evaluate the effectiveness of LLMs in solving reasoning CAPTCHAs, we select two state-of-the-art models, GPT-4V [16] and Google Gemini [17].

Our strategy is detailed in Figure 4. Initially, we ❶ prepare each CAPTCHA by isolating its image section with the question texts, employing translation to English when necessary. We then evaluate the LLM's capabilities in CAPTCHA solving with two strategies. ❷ Zero Shot methodology [18]: Here, the LLM is directly prompted with the CAPTCHA image and associated question. To facilitate this, the text portion of the CAPTCHA is integrated into a straightforward prompt ("Please examine the following CAPTCHA challenge and provide the step-by-step solution"), designed to elicit a direct solution from the model. ❸ Chain-of-Thoughts (CoT) methodology [39]: we prompt the LLM to navigate the challenge through a series of iterative and conversational steps. This approach systematically breaks down the CAPTCHA challenge into smaller and more manageable components, each addressed in sequence to construct a comprehensive solution. Detailed guidelines for this process and specific prompt templates are provided in our open-source dataset [22]. ❹ For both zero-shot and CoT methods, we utilize the respective multimodal API endpoints of the models (`gpt-4-vision-preview`[41] and `gemini-pro-vision`[42]), submitting the CAPTCHA image and question as input. ❺ Upon receiving responses from the LLM, we conduct a manual review to determine the correctness of each solution. To enhance experimental efficiency, we employ a manual correction strategy: we continue the testing if a substep in CoT is correctly executed; otherwise, we label the failed trial and provide the correct solution for the LLM to continue on the next substep. This process is repeated until the challenge is fully resolved.

This strategy offers two advantages: firstly, it facilitates a direct comparison between the effectiveness of zero-shot and CoT approaches in CAPTCHA solving. Secondly, the manual correction mechanism allows us to precisely evaluate the LLMs' performance on each sub-step of the solving process, ensuring that the analysis is not hindered by failures in preceding steps.

3.2.2 Experiment Setup. We meticulously select 10 tasks from each identified sub-category of CAPTCHA, resulting in a comprehensive test set of 50 CAPTCHAs. This selection is made to ensure a balanced representation of various CAPTCHA types. To reduce randomness, we repeat each experiment 10 times, resulting in a total number of 1000 trials (i.e., 2 models * 5 types * 10 CAPTCHAs * 10 repetitions). The experiments are conducted following the previously outlined strategies, focusing on several key metrics to assess the performance of the LLMs:

(1) **Task Comprehension Correct Rate:** The primary metric assessed is the ability of LLMs to correctly comprehend the task

Name	Vendor	Category	# of Samples	Description
FunCAPTCHA1	Arkose Labs	Rotation	10	Rotate the object into the indicated direction
Gobang	GeeTest	Bingo	10	Line up five identical items in a row
IconCrush	GeeTest	Bingo	10	Line up three identical items in a row
Space CAPTCHA	GeeTest	Space	10	Based on the space relation click the indicated object
Space Reasoning	YiDun	Space Reasoning	10	Based on the logical relation click the indicated object

Table 1: Summary of collected CAPTCHAs. The names of these CAPTCHAs are given by the vendors.

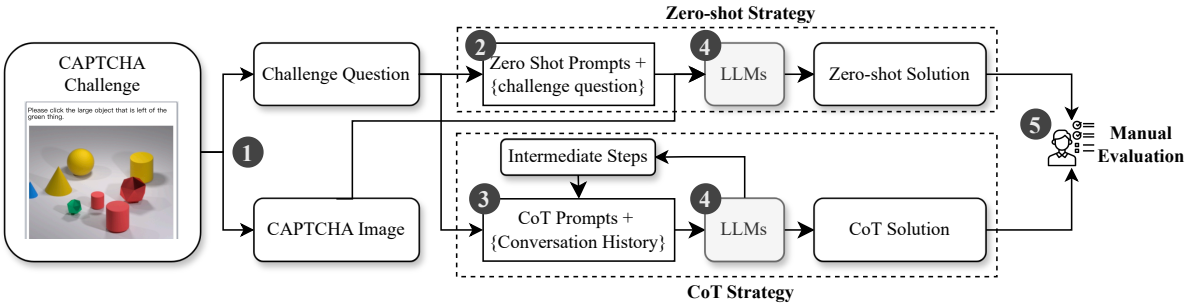


Figure 4: The proposed strategy to test LLMs on CAPTCHA solving.

presented by each CAPTCHA. This is determined by manually evaluating whether the LLMs exhibit an accurate understanding and appropriate intent to solve the given CAPTCHA challenge, based on its image and description.

(2) **Success Rate:** We closely monitor and measure the success rate of LLMs in solving the diverse categories of reasoning CAPTCHAs. We manually review the responses generated by the LLMs, assessing the correctness of their solutions. It is important to note that for the zero-shot approach, a solution is deemed correct only if the LLM response outlines the accurate procedure for resolving the CAPTCHA. In contrast, for the CoT approach, a sub-step is considered successful if the solution proposed by the LLM for that specific sub-step is accurate.

(3) **Failure Reason:** Whenever an LLM fails to solve a CAPTCHA, we meticulously analyze the root causes of the failures. This involves analyzing the detailed LLM responses and delving into its processing and decision-making mechanisms to pinpoint common issues, e.g., task misinterpretation, inadequate reasoning capabilities, or errors in logical deduction.

3.3 Findings

The experimental results of our empirical study are detailed in Table 2. From these results, several intriguing findings emerge. Firstly, it is evident that LLMs are capable of understanding the task of CAPTCHA solving, as they consistently aim to generate plausible outcomes for solving CAPTCHAs. This is demonstrated by the high task understanding rate, with 90.1% for GPT-4V and 86.2% for Gemini. Despite this understanding, both LLMs exhibit notably poor performance in actually solving reasoning CAPTCHAs. Specifically, their success rates in solving the five types of CAPTCHAs are exceedingly low, at only 16.0% for GPT-4V and 12.0% for Gemini. When employing the CoT approach, there is a marginal improvement in performance, reaching 21.0% for GPT-4V and 17.1% for Gemini, but it still remains evident that the models are unable to correctly solve reasoning CAPTCHAs autonomously without human intervention.

Finding 1: Despite the ability to understand the tasks within reasoning CAPTCHAs, the two state-of-the-art LLMs, GPT-4 and Gemini, are unable to effectively solve these tasks without human guidance.

We further examine the CoT-solving process for both models to ascertain the reasons behind their failures. Notably, while the success rates using the CoT approach are low, both models demonstrate a significant capability in resolving a substantial portion of the generated subtasks in the CAPTCHA-solving process. For instance, GPT-4 generates an average of 4.4 subtasks across five different types of tasks and successfully solves 2.4 of them. This represents a substantial 54.5% success rate for individual subtasks, which is significantly higher than the overall task completion rate of 21.0%. This indicates that while LLMs can process and reason through the CAPTCHA-solving steps in a manner akin to humans, they often falter at specific stages. However, in a sequential solving process, the failure of a single subtask inevitably leads to the failure of the overall task.

Finding 2: LLMs could meaningfully decompose a CAPTCHA challenge into subtasks and solve a large portion of them. However, due to the inability to resolve some subtasks, they fail to complete the reasoning process as a whole.

To delve deeper into the nature of both successful and unsuccessful subtasks in the CAPTCHA-solving process, we inspect all subtasks generated during the experiment and categorize them based on their characteristics. We then count the number of unique subtasks generated in this process, and document their rate of being successfully completed by the LLMs in Table 3. It is noticed that LLMs try to complete the challenge with multiple types of

Approach	Zero Shot				CoT							
	Task Understanding		Success Rate		Task Understanding		Average # of subtasks		Average # of successful subtasks		Success Rate	
Model	GPT-4	Gemini	GPT-4	Gemini	GPT-4	Gemini	GPT-4	Gemini	GPT-4	Gemini	GPT-4	Gemini
Angular	37.9%	20.7%	0.0%	0.0%	68.1%	53.4%	4.7	5.1	0.0	0.1	0.0%	0.0%
Gobang	93.1%	81.0%	2.6%	0.0%	96.5%	95.7%	5.6	4.3	2.8	1.3	4.5%	3.3%
IconCrush	94.8%	88.8%	4.2%	0.0%	97.4%	96.6%	4.6	4.7	3.7	1.4	7.9%	4.0%
Space	94.6%	94.8%	44.0%	37.2%	98.3%	95.7%	4.1	3.1	3.3	2.2	55.2%	50.5%
Space Reasoning	59.5%	53.4%	34.3%	25.9%	90.3%	89.7%	3.1	3.2	2.2	1.8	37.3%	27.8%
Average	76.0%	67.8%	16.5%	12.0%	90.1%	86.2%	4.4	4.1	2.4	1.4	21.0%	17.1%

Table 2: Experimental results of applying the testing strategy over the selected CAPTCHAs.

Subtask Category	Unique Subtasks	Success Rate
Understand the task	6	91.8%
Single-criteria object searching	12	80.8%
Multi-criteria object searching	6	25.0%
Orientation identification	2	32.4%
Single-condition Judgement	8	78.2%
Multi-condition Judgement	3	22.4%
Others	4	12.0%

Table 3: Success Rate of Subtasks.

subtasks, such as identifying or localizing an object based on certain attributes (color, shape, etc.) and identifying its orientation. While LLMs exhibit the capability to complete these tasks, their performance is notably better when constrained to a single criterion in different types of tasks. In particular, they can complete single-criteria object searching with a success rate of 80.8%, while it is difficult to handle the case with two or more criteria provided with a success rate of 25.0%. The same conclusion holds for the case when the LLMs are prompted to judge if a declaration holds or not (78.2% vs 22.4%.) An illustrative example is that in the 3D logical CAPTCHA (Figure 3(b)), LLMs can proficiently identify all cylinders. However, when asked to locate a specific red cylinder, they may hallucinate and indicate an incorrect location, or even point to non-cylinder objects.

Finding 3: LLMs demonstrate proficiency in recognizing and understanding natural objects within CAPTCHAs, but their performance significantly diminishes with abstract objects and multi-criteria tasks, revealing a limitation in their cognitive processing capabilities.

Lastly, LLMs exhibit difficulties in processing long instructions encompassing multiple steps, demonstrating an inability to recall outcomes of previous sub-steps, even when the complete conversation history is provided at the LLM endpoints. This challenge could potentially stem from the models' attention-shifting dynamics [43, 44] as revealed in the previous research, which seemingly prioritize the most recent conversational inputs over earlier ones. Additionally, the propensity of LLMs to generate hallucinations increases when they are presented with multiple instructions within a single prompt, complicating their task-solving effectiveness.

Finding 4: LLMs struggle with complex and multi-step instructions, leading to challenges in task continuity and an increased likelihood of hallucinations when handling multiple directives simultaneously.

3.4 Fine-tuning LLMs for CAPTCHA Solving

A straightforward strategy to address the limitations of LLMs identified in **RQ2** is to fine-tune the LLMs with solutions specific to reasoning tasks, potentially enhancing their performance in this domain. To evaluate the feasibility of this strategy, we conduct a preliminary study using the Gemini, which supports image-based fine-tuning, whereas OpenAI currently only offers fine-tuning for text-only models (GPT-3.5). Our investigation focuses on three primary questions: Can fine-tuned LLMs more effectively solve reasoning CAPTCHAs? Can fine-tuning on specific categories improve performance on new, similar tasks? Furthermore, is this approach practically viable?

For our study, we collect 20 challenges for each of the 5 types of reasoning CAPTCHA as illustrated in Section 3.1, forming a new experimental dataset with 100 challenges. The manuscript authors manually generate text solutions for all challenges, compiling them into a dataset of 10,400 tokens for supervised fine-tuning (SFT). We then use them to perform the supervised SFT over Gemini[2]. To assess the transferability of fine-tuning, we create six variations of the training dataset: a complete set and five subsets, each missing solutions for one type of challenge. We reevaluate the models using both single-shot prompting and CoT methods on the original test set evaluated in **RQ2**. The experimental results are presented in Table 4.

The results, as demonstrated in Table 4, reveal that the end-to-end CAPTCHA solving rates for the fully fine-tuned model using zero-shot and CoT approaches are 15.1% and 18.9%, respectively. These rates represent increases of only 2.5% and 1.8% compared to the original model, which are not significant. Moreover, when analyzing the detailed success rates of models fine-tuned on partial datasets, we observe significant performance declines when the models are tested on CAPTCHA categories excluded from their training data. Specifically, the solving rates for models lacking training on corresponding challenge types (highlighted with gray cells) average 13.2% and 17.0% for zero-shot and CoT approaches, respectively, showing negligible improvement over the original model's rates of 12.6% and 17.1%. These findings indicate that fine-tuning does not substantially enhance model transferability in CAPTCHA solving tasks, representing a significant drawback for large-scale

[2]We use *gemini-1.0-pro-002*, the only model version that supports SFT to the date of this work.

Challenge	Angular		Gobang		IconCrush		Space		Space Reasoning		Avg.	
Model	Zero Shot	CoT	Zero Shot	CoT	Zero Shot	CoT	Zero Shot	CoT	Zero Shot	CoT	Zero Shot	CoT
Original	0.0%	0.0%	0.0%	3.3%	0.0%	4.0%	37.2%	50.5%	25.9%	27.8%	12.6%	17.1%
Full SFT	1.5%	1.8%	**3.5%**	**4.0%**	**2.5%**	**6.2%**	42.0%	**54.9%**	**29.2%**	**31.1%**	15.1%	**18.9%**
w/o Angular	0.8%	0.2%	1.9%	3.8%	2.4%	4.3%	37.9%	50.9%	26.2%	28.5%	13.9%	17.5%
w/o Gobang	**2.4%**	1.6%	0.3%	2.2%	1.5%	4.9%	41.8%	50.7%	26.4%	28.0%	14.5%	17.5%
w/o IconCrush	2.1%	1.8%	2.4%	3.5%	0.6%	4.1%	**46.9%**	54.3%	29.1%	32.4%	**16.1%**	19.2%
w/o Space	1.4%	0.7%	0.3%	3.2%	2.3%	4.7%	38.2%	50.5%	27.2%	28.0%	13.4%	17.4%
w/o Space Reasoning	0.5%	**2.7%**	2.2%	3.4%	0.9%	4.5%	39.6%	48.7%	25.9%	28.0%	13.8%	14.8%

Table 4: Experimental results of different fine-tuned models over the selected CAPTCHAs. Cells corresponding to models trained without specific data types are shaded gray,

implementation, especially given the continual development of new CAPTCHA types. Considering that fine-tuning requires manual data labeling, incurs additional training costs, and increases token expenses (as fine-tuned models incur higher charges), this approach may not be practically feasible.

Finding 5: SFT does not significantly enhance LLMs' capabilities in solving reasoning CAPTCHAs. While some capabilities are demonstrated, there is a lack of transferability. Given the high costs involved, this strategy is not deemed practical.

3.5 Discussion

The above study demonstrates that while LLMs show promising capabilities in certain aspects of CAPTCHA solving, they struggle with multi-step instructions and task continuity. While recent prompting strategies such as program-of-thought [45] and diagram-of-thoughts [46] offer structured approaches to reasoning, they are fundamentally insufficient for addressing the specific weaknesses we have identified in CAPTCHA tasks. Firstly, reasoning steps generated autonomously by LLMs often fail to align with the nuanced demands of CAPTCHA challenges, particularly in areas where models underperform, such as multi-criteria object recognition and multi-condition decision-making. Secondly, even when explicitly prompted to decompose tasks into more manageable atomic operations, current prompting techniques do not reliably produce correct or consistent task breakdowns, leading to persistent reasoning failures. Given these constraints, we argue that relying solely on prompting is inadequate. Instead, we propose a Domain-Specific Language (DSL) approach, which enforces a structured, controlled decomposition of complex reasoning CAPTCHAs into atomic steps that are specifically aligned with LLM strengths—namely, visual perception and basic logical operations—while systematically avoiding known model weaknesses. In the following sections, we detail the design and implementation of this DSL framework, which we posit as a necessary evolution beyond general-purpose prompting for robust CAPTCHA solving with LLMs.

4 CAPTCHA Domain Specific Language

4.1 Motivation

Our empirical study highlights that LLMs can potentially tackle reasoning CAPTCHAs using the CoT strategy, which decomposes the challenge into smaller subtasks and addresses them in a divide-and-conquer manner. However, the varying levels of LLM proficiency across different subtasks introduce instability into the challenge-solving process. This variability leads us to consider whether it is feasible to formalize the CAPTCHA solving process into a series of steps, each individually tailored to be within the LLM's capabilities.

This consideration has inspired the design of our CAPTCHA Domain Specific Language (DSL) that formally outlines the operations involved in solving CAPTCHAs, essentially *encoding the CAPTCHA solving process with a series of LLM-easy tasks*. Our design rationale is based on the empirical study showing that LLMs can only resolve simple tasks but fail on modern reasoning CAPTCHAs. The proposed DSL only contains atomic, well-defined operations (search, reason, locate, etc.), each aligned with "LLM-easy" tasks as identified in the empirical study. Key challenges—such as LLMs struggling with multi-attribute queries, multi-step reasoning, and complex object recognition—directly inform the DSL constraints, ensuring each operation remains within LLM capabilities. For example, search is restricted to single-attribute queries to prevent compounded failures, while locate explicitly maps objects instead of relying on inference.

The creation of CAPTCHA DSL offers several key advantages. First, the operations delineated by it align with the competencies of LLMs, ensuring that each step of the solution is approachable and solvable, thereby enhancing the overall success rate of solving actual challenges. Second, DSL, being an abstract representation of the natural language process of CAPTCHA solving, can be seamlessly translated back into natural language. This facilitates its integration into LLMs for reasoning and processing. Third, it also brings formalization to the solving procedure. Solutions generated in DSL can be rigorously verified for syntactical correctness, allowing for straightforward identification and rectification of errors. In practice, we implement the DSL through JetBrains MPS with the in-built local syntax verifier (Projection Editing Model) to enforce correctness and provide structured error feedback to Oedipus for automated refinement. This approach enables effective CAPTCHA solving, even for complex challenges, as long as they can be decomposed into solvable subtasks.

In the following, we elaborate on the formal definition of CAPTCHA DSL and showcase its application through illustrative examples. This demonstration will underscore the practical utility and effectiveness of CAPTCHA DSL in streamlining and enhancing the process of solving reasoning CAPTCHAs with LLMs.

4.2 High-level Structure

We first delineate the high-level structure and components of the CAPTCHA DSL. As exemplified in Figure 5, a CAPTCHA DSL script comprises multiple lines of statements, with each line representing a specific operation targeting elements within the CAPTCHA solving context. This structure is reminiscent of the SQL [47] syntax, where scripts are composed of clauses, the fundamental building blocks of statements. Each clause in the CAPTCHA DSL is constructed following specific syntax rules and can include a variable number of components. Below we detail the components of the CAPTCHA DSL.

(1) **`Keywords:`** The operational keywords function as the core unit operations in CAPTCHA DSL. There are four keywords defined: `search`, `reason`, `locate`, and `operate`. They correspond to the atomic operations that our empirical study (Section 3) has shown to be effectively executed by LLMs, i.e., searching objects from the CAPTCHA given requirements, reasoning for a certain task, locating a particular item on the image, or performing a given operation.

(2) **`Objects:`** This component represents the tangible items in the CAPTCHA image, such as animals in the rotation CAPTCHA (Figure 1(b)) and emojis in the Bingo CAPTCHA (Figure 2(b)) Explicit object representation helps mitigate LLMs' inconsistency in object recognition within complex scenes.

(3) **`Attributes:`** These are properties of `objects` encoded in natural language. In CAPTCHA DSL, attributes are limited to a subset that is effective for actual CAPTCHA solving, as informed by our empirical study. This includes characteristics like *size*, *color*, *type*, *orientation*, among others. This constrained attribute set addresses LLMs' tendency to hallucinate or confuse object properties when dealing with multiple attributes simultaneously.

(4) **`Predicates:`** These are conditions that can be evaluated to three-valued logic (3VL) [48] (true/false/unknown) or Boolean values. They are used to constrain the effects of expressions. Predicates function through the standardized set of boolean operators.

(5) **`Expressions:`** We borrow the concept of SQL Expressions [49] into CAPTCHA DSL. Expressions including '=' in CAPTCHA DSL consist of components that operate based on specific logic to yield Boolean or attribute values. These expressions can be used for updating attributes or, in conjunction with predicates, evaluating conditions.

(6) **`Descriptions:`** A special component in CAPTCHA DSL is the natural language descriptions that can be provided as variables to the `reason` keyword, which can be arbitrarily generated, updated, and handled by the LLM.

4.3 Syntax Constraints

The above CAPTCHA DSL is regulated through a loosely defined syntax, which is primarily designed to address the specific failure modes of LLMs identified in our empirical study. Similar to Python and other programming languages, it is requested that all the variables appearing in the statements are properly defined, and their values are properly updated without use-before-definition. In addition, we define two additional syntax rules to bound the keywords to regulate that each line of the statement in the CAPTCHA DSL script can be translated into a natural language task, and the task is proved to be highly achievable by LLMs as listed in Table 3. (1) The operational keyword `reason` is used by LLMs to perform easy reasoning tasks given an `Description`. To ensure that the task is an 'easy' reasoning task, we only allow the reasoning result to be a 3VL logic, or an `orientation` attribute of an object. This maps to the empirical study, where we realize that LLMs are better at judging if the condition is correct or wrong, but not good at making complex reasoning. (2) The keyword `search` should always and only be bounded by single attributes. This aligns with the fact that LLMs can well tell the objects from the images when only one filtering criterion is given, yet they cannot perform well when more than one is provided.

The above DSL syntax constraints enable a local verification of the unbounded scripts. Given a DSL script, we can parse the prompts through the above DSL syntax constraints and validate if the constraints are met, and if the script fulfills the variable definition logic (such as if the value is not properly defined). This verification process provides crucial feedback for script refinement, compensating for LLMs' inability to self-correct errors in complex reasoning chains.

4.4 A Running Example

We present a detailed running example to illustrate the efficacy and modeling capabilities of our CAPTCHA DSL in the context of a 3D CAPTCHA shown in Figure 3(a). The DSL generated by GPT-4 is presented in the following Figure 5. This 3D CAPTCHA challenge requires the solver to identify and reason about the spatial orientation of alphanumeric characters in a three-dimensional space. Specifically, the task involves finding a number '0' that is oriented in the same direction as the letter 'W.' Our DSL strategy streamlines this complex problem into a series of logical steps, each represented by a single line of instruction within our DSL script, as shown below.

Initially, the script identifies all the objects within the CAPTCHA environment, creating a foundational dataset from which specific objects can be filtered and analyzed. The DSL then proceeds to filter the objects to isolate instances of the number '0' and the letter 'W' each through a separate search operation as mandated by the DSL's syntax constraints. An incorrect attempt is deliberately included to showcase the script's syntax in red color: a line with multiple WHERE clauses is flagged, reflecting an implementation that does not conform to the DSL's syntactic rules. This error can be identified through the local syntax checker.

Following the DSL's structured approach, the script deduces the orientation of the letter 'W' through a reasoning operation. With this information, it then conducts a search for the number '0' that shares this orientation. Finally, the DSL script concludes by locating the correct position of '0' that aligns with 'W,' thus demonstrating the potential for the DSL to effectively model the solution to a CAPTCHA challenge. Through this example, we highlight the precision and clarity of our DSL, alongside its inherent ability to

```
// Identify all objects present in the CAPTCHA
[objs] = SEARCH object IN CAPTCHA;

// Filter the objects to find the number 0 and letter W
[number_zero] = SEARCH obj IN objs WHERE obj.value == "0";
[letter_W] = SEARCH obj IN objs WHERE  obj.value == "W";

// An incorrect attemp is logged below, with more than one
where clause in one instruction.
// [same_direction_objects] = SEARCH obj IN objs WHERE
obj.orientation == letter_W.orientation AND obj.value ==
"0";

// Determine the orientation of the letter W
[W_orientation] = REASON{letter_W.orientation};

// Find the number 0 that has the same orientation as the
letter W
[same_direction_zero] = SEARCH obj IN [number_zero] WHERE
obj.orientation == [W_orientation];

// Return the position of the correctly oriented number 0
[zero_position] = SEARCH same_direction_zero;
```

Figure 5: Example DSL script for 3D CAPTCHA challenge.

self-validate and prevent syntactically incorrect implementations that could otherwise impede the CAPTCHA solving process.

5 Methodology

5.1 Overview

In the pursuit of a solution for solving complex and commercial-level reasoning CAPTCHAs, we introduce Oedipus[3], a comprehensive automated framework that harnesses the capabilities of multi-modal LLMs. Figure 6 provides a schematic overview of Oedipus. The cornerstone of this framework is to leverage the custom-designed CAPTCHA DSL illustrated in Section 4, to systematically articulate the CAPTCHA solving process. Within this DSL, each operation is carefully aligned with the actions that LLMs have high confidence in effectively executing, a strategy grounded in the insights gained from our prior empirical study.

Oedipus operates through the following structured procedures. It contains two main phases: task generation and solution generation. They can be further detailed into the following steps. ❶ *Pre-Solving Preparation*: In the initial phase, a custom LLM undergoes fine-tuning with DSL examples to master its syntax and structure. Faced with a new CAPTCHA challenge, this adept LLM generates a DSL script, meticulously delineating the steps necessary to unravel the CAPTCHA challenge. ❷ *Syntax Verification*: The crafted DSL script is subjected to rigorous verification in a local DSL verifier. This step is crucial for identifying and flagging any syntactical inaccuracies. Detected errors trigger a feedback loop to the CAPTCHA LLM, instigating another round of script generation, informed by the identified syntax errors. ❸ *Solution Translation*: Upon achieving a syntax-error-free DSL script, it proceeds to the instruction translator. This component transforms the DSL script into a format comprehensible in natural language, tailor-made for processing by LLMs. ❹ *CAPTCHA Solving*: The natural language translation of the CAPTCHA solution, coupled with the original CAPTCHA challenge, is then processed by the multimodal CAPTCHA solver LLM. This final phase yields the ultimate solution to the CAPTCHA. Oedipus leverages the analytical prowess of LLMs within the structured confines of the DSL. This harmonization is aimed at dissecting and resolving the reasoning CAPTCHAs methodically with better accuracy. In the following of this section, we detail the design of each module.

[3]Oedipus is the renowned Greek mythological figure known for solving the riddle posed by the Sphinx.

5.2 CAPTCHA DSL Script Generation

Our investigation reveals that LLMs possess the capability to comprehend CAPTCHA challenges and generate meaningful step-by-step solutions using the CoT approach. The creation of the CAPTCHA DSL aims to guide LLMs in generating solutions that adhere to specific rules and structures, thereby enhancing the likelihood of successfully completing each step. To facilitate this, we develop a method for instructing an LLM to automatically generate a CAPTCHA DSL program tailored to a particular type of CAPTCHA challenge.

This process involves three steps. First, we manually generate correct DSL scripts for a given CAPTCHA challenge. This step is straightforward and takes up to 5 minutes for each CAPTCHA challenge given the authors of this work is familiar with the DSL language. Second, we utilize the fine-tuning technique to develop an LLM that specializes in CAPTCHA DSL program generation. The DSL examples generated in the previous step are employed as training data to fine-tune the LLM, enhancing its proficiency and accuracy in generating CAPTCHA DSL programs. Third, we employ few-shot prompting with the LLM, using the sample CAPTCHA DSL programs as illustrative examples. This process is exemplified in the textbox provided below. In practice, we find that this generation task can be effectively completed by fine-tuned code-llama [50] or GPT-3.5 with a minimum number of 20 pieces of examples, or more powerful models such as GPT-4 without any fine-tuning process. We provide more concrete examples on how to instruct the LLM to generate CAPTCHA DSL scripts in our open-source dataset [22].

Once a DSL script is generated by the LLM, it undergoes a thorough examination in a local DSL Verifier. This verifier plays a crucial role in ensuring the syntactical correctness of the script. It checks for compliance with the DSL's syntax rules and verifies that each operation within the script is feasible and logical within the context of the specific CAPTCHA challenge. If any syntax errors or logical inconsistencies are detected, the verifier flags these issues as illustrated in Section 4, which are then used as feedback to refine the LLM's future script generation.

This verification step not only ensures the accuracy and reliability of the generated DSL script but also provides valuable insights for further fine-tuning the LLM to enhance its CAPTCHA solving capabilities. Through this iterative process of generation, verification, and refinement, the LLM becomes increasingly adept at creating effective and accurate CAPTCHA DSL scripts, thereby streamlining the CAPTCHA solving process.

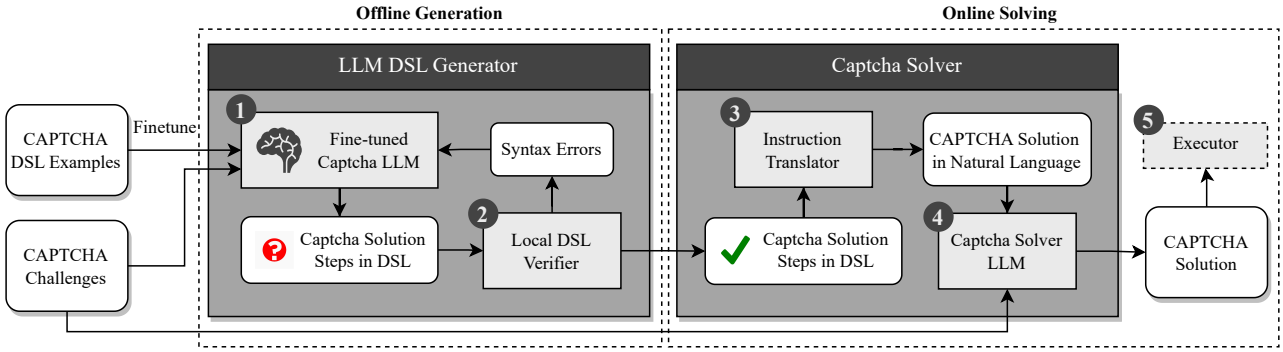


Figure 6: Overview of Oedipus.

5.3 Instruction Translation

Following the generation of the CAPTCHA DSL script, the subsequent phase entails translating these DSL instructions into natural language. This translation is pivotal, as it converts the structured DSL commands into a format that is understandable and actionable by the CAPTCHA Solving module, which then proceeds to address the specific CAPTCHA challenge. Given the critical role of this translation in bridging the gap between the DSL's structured syntax and the LLM's interpretative flexibility, it is paramount to ensure the accuracy and precision in this translation.

To achieve the highest level of translation fidelity, we employ an LLM, utilizing two strategies for optimal results. First, we leverage prompt engineering and meticulously craft prompts that include the definitions of our DSL syntax as described in Section 4, thereby orienting the LLM to accurately grasp and interpret the DSL instructions. This foundational understanding is crucial for the LLM to correctly translate the DSL script into natural language instructions. Second, similar to the multi-shot prompting strategy adopted in the DSL generation phase to furnish the LLM with examples demonstrating the translation of instructions into DSL scripts, we implement a reverse process for this phase. By providing examples that illustrate how DSL scripts can be effectively mapped back to natural language instructions, we facilitate a deeper comprehension by the LLM of the intended semantic and functional translation. Through practical application, we observe that combining prompt engineering with this reversed example provision markedly enhances the accuracy of the instruction generation process, ensuring that the CAPTCHA Solving module receives clear, precise, and actionable directives to solve the challenges at hand.

5.4 CAPTCHA Solving

With the natural language instructions derived from the DSL script, we proceed to the CAPTCHA solving phase. In this step, we use these instructions as CoT prompts to guide the CAPTCHA LLM through the actual CAPTCHA challenge. This process is straightforward: the instructions, along with the original challenge image, are fed into the CAPTCHA LLM. The LLM then follows these step-by-step instructions to systematically tackle and solve the CAPTCHA.

As an additional optional step, the solutions generated by the LLM can be integrated into an automated executor. This executor is programmed to interact with the CAPTCHA on the target website, inputting the solution directly and completing the challenge. Oedipus is designed as an end-to-end framework that automates CAPTCHA solving, taking a CAPTCHA image as input and generating a structured solution. While the current setup requires a screenshot, full automation can be achieved by integrating an agent framework to detect and submit CAPTCHAs dynamically. In practice, we implement this automation through a three-step process: (1) PyAutoGUI scans the CAPTCHA and identifies pixel locations of elements, (2) we follow the concrete steps generated by Oedipus and match the operations to the identified locations of the elements, and (3) the Python script executes these commands to complete the CAPTCHA. To ensure precise interactions, we define standardized mouse operations (e.g., click, drag) and uniquely identifiable CAPTCHA element descriptions that enable the LLM to generate accurate automation commands. This modular design enables seamless integration into a fully autonomous solving pipeline, where Oedipus can solve each step autonomously while maintaining the flexibility to adapt to different CAPTCHA interfaces and requirements.

5.5 Discussion

Our framework is intentionally not calibrated for achieving an exceptionally high success rate in CAPTCHA solving. This design philosophy aligns with the CAPTCHA evaluation standard proposed by Microsoft [51], which suggests comparing the economic cost of automated CAPTCHA solving against the cost of human labor for the same task. While human solvers are expected to achieve a solving rate of above 90% [51, 52], an automated approach like ours could be deemed equally effective if it maintains a 50% success rate, provided its operational cost is less than one-fourth of the labor cost of hiring a human solver ($1 - 0.5^4 = 93.75\%$). Given this cost-benefit analysis, Oedipus is designed to optimize performance within these economic constraints, eschewing complex and cost-incurring enhancements in favor of a more streamlined approach. Consequently, Oedipus does not incorporate an active feedback mechanism to verify the correctness of its substep solutions during the reasoning process. This decision stems from the practical challenge of confirming whether the LLM has accurately identified objects without reliable external references for validation. While iterative identification and majority voting could theoretically improve accuracy, the additional computational and financial costs of

increased API queries make this option untenable. Therefore, our strategy intentionally forgoes such enhancements to maintain the economic viability and operational simplicity of Oedipus.

6 Evaluation

We evaluate the performance of Oedipus on real-world reasoning CAPTCHA challenges. In particular, we are interested in four research questions.

- **RQ1 (Effectiveness)** How effective of Oedipus in addressing real-world reasoning CAPTCHAs automatically?
- **RQ2 (Ablation)** How effective is each strategy contributing to the success of Oedipus?
- **RQ3 (Transferability)** Can Oedipus resolve new CAPTCHA tasks that are not included in our empirical study, i.e., its transferability to new CAPTCHAs?

6.1 Experimental Setup

Evaluation Baselines. We implement Oedipus with 1,554 lines of Python3 code. In our assessment of Oedipus under varied conditions, we incorporate six multimodal LLMs that are currently accessible: OpenAI GPT-4V and GPT-4o, Google Gemini and Gemini-2.0-Flash, Claude-3.7, and miniGPT-4 [53]. The inclusion of GPT-4V and Gemini aligns with our preceding empirical study to ensure consistency in model evaluation; GPT-4o, Gemini-2.0-Flash, and Claude-3.7 are the state-of-the-art multi-modal LLM, with higher response speed and lower cost compared to GPT-4V; miniGPT-4 is chosen for its prominence as the most widely recognized open-source multimodal LLM to date. For all three models, we adjust the LLM response temperature to 0, aiming to minimize the output variability and ensure deterministic responses. To benchmark Oedipus's performance, we draw a comparison with the only discussed solution to reasoning CAPTCHAs in academia, VTT [34]. Given the absence of open-source access to VTT and the discontinuation of the CAPTCHA challenges it was tested on, we endeavor to reconstruct their approach as accurately as possible to facilitate a fair comparison.

Evaluation Datasets. In the absence of existing open-source datasets for reasoning CAPTCHAs, we embark on a systematic collection process. We target three prominent security companies known for providing reasoning CAPTCHAs: Arkose Labs [54], Geetest [55], and NetEase Yidun [56]. By collecting all types of reasoning CAPTCHAs from them designed by the end of 2022 and available online as paid commercialized API services, we finally formulate a dataset, which includes four types of CAPTCHAs covering the three categories summarized in Section 3: *Arkose-Angular*, *Geetest-Gobang*, *Geetest-Space*, and *Yidun-Space-Reasoning*. The complete dataset is also open-sourced at [22]. We meticulously collect examples through their paid CAPTCHA API services, ultimately curating a dataset comprising 100 samples for each type of challenges. Note that *Arkose-FunCAPTCHA* samples exhibit repetition, differing only in their starting positions, due to the limited number of combinations inherent in Rotation-type CAPTCHAs. Following the collection phase, we manually solve all the CAPTCHA challenges to ensure each task is paired with a correct standard answer. Since VTT requires additional data for training, we collect 100 additional samples with manual labels to reconstruct their models.

Experiment Settings For each CAPTCHA challenge, we run three versions of Oedipus and VTT and examine the solutions. To reduce randomness, we repeat each trial for 10 times. Thus, we conduct a total number of 28,000, i.e., 4 projects * 100 samples * 7 settings * 10 repetitions, of experiments, with a total LLM API cost of 1565.26 USD[4].

6.2 (RQ1) CAPTCHA Solving Performance

Our initial exploration into the effectiveness of different models in the reasoning CAPTCHA solving process reveals insightful outcomes, as summarized in Table 5. Notably, Oedipus, when augmented with Claude-3.7, outperforms its counterparts across all four types of reasoning CAPTCHAs, recording an average success rate of 73.8%. This performance is better than that of Oedipus configured with GPT-4o, GPT-4v, and Gemini-flash-2.0. In comparison, VTT demonstrates proficiency in the space reasoning challenges, securing success rates of 55.2% and 44.0%, respectively, which places it slightly behind the LLM-powered Oedipus versions.

There are two notable observations. First, all strategies perform poorly on Angular CAPTCHAs. A closer inspection of the LLM outputs suggests a difficulty in formally recognizing the orientation of objects, leading to selections that approximate but do not exactly match the correct orientation. Despite this challenge, Oedipus showcases commendable performance across the board, indicating that the integration of LLMs with superior reasoning abilities substantially enhances CAPTCHA solving success rates. Second, newer models with stronger reasoning capabilities in general tasks show remarkable improvement over older models. For instance, Oedipus powered by Claude-3.7 achieves an average improvement of 10.3% over GPT-4V, with exceptional performance on Gobang CAPTCHAs (93.4%). These results underscore the pivotal role of advanced reasoning capabilities in improving automated CAPTCHA solving outcomes. The substantial performance gains with newer models highlight that as LLM technology continues to advance, Oedipus will likely achieve even greater effectiveness without requiring significant architectural modifications.

Model	Angular	Gobang	Space	Space Reasoning	Average
GPT-4V	37.9%	79.7%	71.3%	65.0%	63.5%
GPT-4o	37.2%	78.8%	70.9%	63.7%	62.7%
Gemini	31.0%	56.3%	59.5%	58.0%	51.2%
miniGPT-4	7.5%	16.9%	18.0%	15.1%	14.4%
VTT	✗	✗	55.7%	44.5%	25.1%
Gemini-2.0	35.1%	79.1%	82.7%	62.9%	65.0%
Claude-3.7	42.8	93.4%	80.7%	78.2%	73.8%

Table 5: The success rate of CAPTCHA solving.

We proceed to evaluate the practicality of Oedipus by comparing the average cost required to successfully solve 100 reasoning CAPTCHAs, accounting for both successful and unsuccessful attempts. This analysis is juxtaposed against the quotations from two anonymous online CAPTCHA solving services, with specifics withheld for security and ethical considerations. The cost comparison results, detailed in Table 6, reveal that although Oedipus supported by GPT-4 exhibits a higher success rate, its cost per

[4]For OpenAI and Google APIs; mini-GPT4 is implemented locally on a PC with Nvidia RTX 4090.

100 CAPTCHA solutions (13.1 USD) significantly surpasses that of Oedipus powered by Gemini (3.1 USD). This discrepancy is primarily due to the GPT-4 API's pricing, which is approximately ten times higher than that of the Gemini Pro API [42]. The rates offered by the two CAPTCHA solving service providers are marginally lower than those associated with Oedipus, yet they reside within the same cost magnitude. Importantly, experiments with newer models demonstrate significant improvements in cost. Oedipus with Gemini-2.0-Flash reaches 64.7% success rate with merely 23.6% of the cost compared to GPT-4o. Notably, it is even lower than commercial CAPTCHA-solving service providers, making it not only more effective but also more economical. These findings highlight the rapid advancement in LLM capabilities and cost reduction. As LLM technology continues to evolve with improved reasoning capabilities and decreased operational costs, we anticipate that Oedipus will become increasingly cost-competitive while maintaining or improving its performance advantage.

Cost(USD)	Angular	Gobang	Space	Space Reasoning
Oedipus-GPT4V	21.6	12.8	8.7	9.3
Oedipus-GPT4o	11.3	7.0	4.2	4.6
Oedipus-Gemini	5.3	3.2	1.8	2.1
Oedipus-Gemini-2.0	0.4	3.6	0.06	0.07
Oedipus-Claude-3.7	5.2	2.7	1.0	0.7
Provider-A	5.0	2.0	0.5	0.5
Provider-B	3.0	2.0	0.3	0.5

Table 6: Average cost of different tools for successfully solving 100 CAPTCHAs.

6.3 (RQ2) Ablation Study

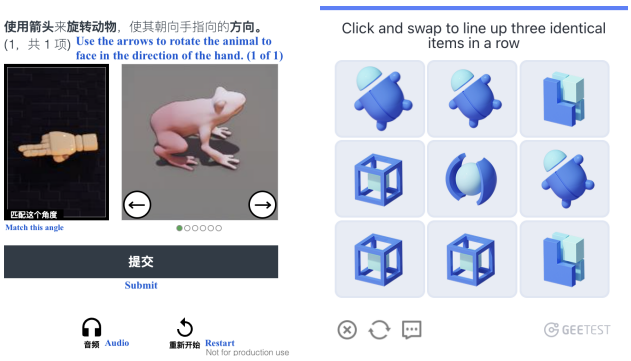
We further investigate if the components in Oedipus (i.e., DSL generation, multi-shot prompting CAPTCHA solving) can successfully improve the performance of the CAPTCHA solving process.

Our initial investigation focuses on the impact of local DSL generation feedback on task completion efficacy. For this purpose, we establish a comparison group in which CAPTCHA DSL scripts generated during the first attempt are directly utilized for CAPTCHA solving. This approach includes a manual review to ascertain the accuracy of the DSL script generation for each task. Results of this examination are detailed in Table 7. It is observed that the initial success rate for generating DSL scripts across the four tasks averages at 59.8%, whereas incorporating feedback significantly enhances this rate to 93.7%. Thus, integrating feedback into the CAPTCHA solving process markedly improves the overall success rate, underscoring the value of iterative refinement in generating effective DSL scripts for CAPTCHA resolution.

Success Rate	Angular	Gobang	Space 1	Space 2	Average
First Trial	61.8%	63.4%	56.8%	57.2%	59.8%
Solving	16.2%	55.3%	47.9%	38.6%	39.5%
Feedback	91.2%	94.4%	93.2%	95.8%	93.7%
Solving	37.4%	80.2%	70.8%	65.4%	63.5%

Table 7: The success rate of DSL generation.

We then examine whether task breakdown contributes to the performance of the solving process. As shown in Table 8, In this table, gray rows highlight outcomes where LLMs attempt CAPTCHA solutions without segmenting the tasks into smaller, more manageable steps. MiniGPT-4 remains incapable of solving CAPTCHA problems due to its inherent limitations. In contrast, task breakdown is evidently of great assistance for GPT-4V, GPT-4o, and Gemini. This underscores the effectiveness and significance of Oedipus. In *Angular* tasks, task breakdown makes it possible for LLMs to carry on such challenges, which they cannot solve directly, albeit with a modest success rate. In *Gobang* and *IconCrush* tasks, there is a sudden increase in success rates, elevating them from initially low levels to relatively high levels. For Space and Space Reasoning tasks, which LLMs can solve with certain levels of success rates, improvements are also observed.



(a) Arkose-FunCAPTCHA (b) Geetest-IconCrush

Figure 7: Two new CAPTCHAs developed in 2023.

	GPT-4V	GPT-4o	Gemini	miniGPT-4
Angular	0.0%	0.2%	0.0%	0.0%
	37.4%	36.8%	31.4%	0.4%
Gobang	2.6%	3.2%	0.0%	0.0%
	80.2%	78.2%	55.7%	1.6%
Space	54.0%	52.1%	49.2%	0.2%
	70.9%	71.4.1%	60.1%	1.8%
Space Reasoning	34.3%	31.9%	25.9%	0.2%
	65.4%	65.2%	57.7%	1.4%

Table 8: The impact of task breakdown. The gray row denotes no task breakdown.

6.4 (RQ3) Transferability

To evaluate Oedipus's ability to adapt to emerging CAPTCHA challenges, we delve into its performance against newly designed reasoning CAPTCHAs. Following the methodology outlined in Section 5, which emphasizes the use of sample DSL scripts and CAPTCHA solutions for few-shot prompting, we assess Oedipus's efficacy in navigating uncharted challenges. Specifically, we focus on two novel CAPTCHA types introduced in 2023: *Arkose-AngularV2* and *Geetest-IconCrush*, with visual samples depicted in Figure 7. Consistent with our established protocol, we prepare a dataset comprising 50 instances of each CAPTCHA variant, subsequently deploying Oedipus powered by four distinct LLMs to tackle these challenges.

	Angular New	IconCrush	Average
GPT-4V	35.2%	67.4%	51.3%
GPT-4o	36.8%	70.3%	53.6%
Gemini	31.4%	42.2%	36.8%
miniGPT-4	4.2%	6.0%	5.1%
Gemini-2.0	34.0%	81.0%	57.5%
Claude-3.7	40.5%	87.0%	63.8%
VTT	✗	✗	✗

Table 9: Transferability study on new CAPTCHAs.

Model	Angular	Gobang	Space	Space Reasoning	Average
GPT-4V	238.1	124.7	39.2	39.3	146.5
GPT-4o	318.1	83.5	23.2	24.8	92.4
Gemini	337.2	102.2	37.1	37.7	128.6
Claude-3.7	–	71.9	26.3	22.9	40.4
Gemini-2.0-flash	191.5	33.3	17.3	12.1	63.6
Provider-A	28.0	20.0	8.0	8.0	16.0
Provider-B	15.0	10.0	3.0	3.0	7.8

Table 10: Time consumption of CAPTCHA solving process.

Table 9 reports the success rates of the experiment. Oedipus powered by GPT-4V and GPT-4o attain average accuracies of 51.3% and 53.6%, respectively. The Claude-3.7 (63.8%) markedly outperforms other solutions, demonstrating appreciable transfer gains with strong model reasoning capabilities. In contrast, miniGPT-4 (5.1%) and the vision-only VTT (—) fail to generalize effectively. These results confirm that Oedipus's pipeline can be extended to CAPTCHAs outside its training distribution, but that its transferability hinges on the reasoning prowess of the underlying LLM. In particular, the step-change from Gemini to Gemini-2.0 and Claude-3.7 highlights how even modest model upgrades can substantially boost adaptability to novel CAPTCHA designs.

Despite the challenges encountered with miniGPT-4, the overall findings validate Oedipus's transferability capability, illustrating its potential to remain an effective tool with stronger models being developed. This adaptability is contingent upon continuous enhancements in LLM reasoning capabilities, ensuring that Oedipus can keep pace with the dynamic nature of the CAPTCHA challenge.

7 Discussion

7.1 Limitation

One limitation of our strategy is the time required to solve CAPTCHAs. Table 10 presents the average solving time of Oedipus across four benchmark reasoning CAPTCHA challenges. Notably, Oedipus-miniGPT4's solving time is excluded due to its dependency on GPU capabilities. On average, Oedipus takes about 80 seconds per task, longer than the sub-20-second duration typical of anonymous CAPTCHA service providers, according to on our quotations.

This efficiency gap underscores the need for faster inference. However, we have seen dramatic speedups within a 6-month period of LLM development: GPT-4V required on average 146.5 seconds, GPT-4o brought this down to 92.4, and the latest Gemini-2.0-flash further reduced it to 63.6. Within half a year of iterative model development, we observed a nearly 30-second decrease in average solve time. These results give us confidence that ongoing advances—both algorithmic and in hardware acceleration—will continue to drive down latency. We anticipate that, with continued optimization, Oedipus will soon match or exceed the performance of commercial CAPTCHA solvers, delivering both state-of-the-art accuracy and real-time responsiveness.

7.2 Model Refusal Issues

During our experiments, we encountered occasional cases where models failed to solve CAPTCHAs due to alignment constraints. Our analysis shows varying refusal rates across different models and CAPTCHA types. Gemini-Pro, Gemini-Flash-2.0, and MiniGPT-4 did not show any refusals across all evaluated samples. GPT-4v and GPT-4o occasionally refused Angular CAPTCHAs (less than 10% of samples), without refusals for other categories. Claude-3.7 showed higher refusal rates, particularly for Angular CAPTCHAs, and occasionally rejected other CAPTCHA types. These refusals typically occur when the model recognizes CAPTCHA-solving as potentially violating safety regulations. Interestingly, this issue can often be circumvented by reattempting the query. We also note that LLM service providers have gradually loosened restrictions on CAPTCHA solving. While OpenAI previously prohibited using LLMs for CAPTCHA solving, their current usage policies [57] have removed this restriction. Similar trends are observed with Gemini [58] and Claude [59], which have no explicit prohibitions on CAPTCHA-solving tasks. This evolving landscape of model policies suggests that refusal issues may become less significant as LLMs continue to mature and their governance frameworks adapt to legitimate CAPTCHA-solving use cases.

7.3 New CAPTCHA Design as a Defense

The rise of LLMs has weakened traditional CAPTCHAs, including those based on commonsense reasoning [34], which were once seen as robust defenses [60]. In response to these developments, our proposal for enhancing CAPTCHA security involves the conceptualization of new reasoning CAPTCHA challenges from three innovative perspectives, aimed at exploiting the current limitations of LLMs.

Complex Reasoning Chains: The first approach involves crafting CAPTCHAs that necessitate extended reasoning chains far beyond the current processing capabilities of LLMs. For example, a CAPTCHA could present a narrative puzzle that requires understanding a multi-step logical sequence or piecing together information from various parts of a text to arrive at a conclusion. We have primarily tested that by updating the Bingo challenge in Section 2 to move two pieces, which is unsolvable for current LLMs. This type of CAPTCHA can push the boundaries of LLMs' reasoning depth, requiring not just understanding individual components but synthesizing complex relationships over several logical steps.

Deceptive Object Recognition: The second strategy exploits the object recognition capabilities of LLMs by introducing adversarial examples [23] into CAPTCHA designs. For instance, visually distorted objects that appear normal to human observers but are deliberately crafted to mislead LLMs' pattern recognition can be used. This leverages the susceptibility of neural networks to misinterpretation when faced with carefully manipulated input data.

Unit Operations Beyond LLM Capabilities: The third approach seeks to develop CAPTCHAs comprising unit operations that are inherently beyond the capabilities of existing LLMs. For instance,

they can be designed to request intuitive understanding of physical interactions in the real world, such as predicting the outcome of a physical event depicted in a video clip, challenging LLMs' ability to infer physical laws and dynamics from visual data alone.

Incorporating these strategies into the design of CAPTCHAs aims to elevate their security efficacy against automated solvers. However, as LLMs continue to advance, even these challenges may eventually be overcome, reflecting the ongoing cycle of innovation and adaptation in CAPTCHA design and AI development. This eventuality underscores the cat-and-mouse nature of CAPTCHA development and AI evolution—a continuous cycle of action and reaction, where each advancement in CAPTCHA design prompts a corresponding leap in AI problem-solving abilities.

8 Conclusion

In this work, we study the hard AI problems underlying current reasoning CAPTCHAs, and explore an automated methodology to solve these challenges through LLMs. With a custom designed CAPTCHA DSL, we design an end-to-end framework Oedipus that automatically solves reasoning CAPTCHAs. Our solution achieves an average success rate of 63.5%, with a cost comparable to commercialized CAPTCHA solving services. To propose a defense, we further propose three strategies of designing more secure reasoning CAPTCHAs in the future.

Acknowledgments

This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority.

References

[1] CAPTCHA, https://en.wikipedia.org/wiki/CAPTCHA.
[2] Moravec's_paradox, https://en.wikipedia.org/wiki/Moravec%27s_paradox.
[3] ReCAPTCHA, https://www.google.com/recaptcha/about/.
[4] G. Ye, Z. Tang, D. Fang, Z. Zhu, Y. Feng, P. Xu, X. Chen, and Z. Wang, "Yet another text captcha solver: A generative adversarial network based approach," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 332–348.
[5] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell, "The end is nigh: Generic solving of text-based {CAPTCHAs}," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.
[6] R. Jin, L. Huang, J. Duan, W. Zhao, Y. Liao, and P. Zhou, "How secure is your website? a comprehensive investigation on captcha providers and solving services," 2023.
[7] M. I. Hossen, Y. Tu, M. F. Rabby, M. N. Islam, H. Cao, and X. Hei, "An object detection based solver for google's image recaptcha v2," 2021.
[8] H. Wang, F. Zheng, Z. Chen, Y. Lu, J. Gao, and R. Wei, "A captcha design based on visual reasoning," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 1967–1971.
[9] Y. Gao, H. Gao, S. Luo, Y. Zi, S. Zhang, W. Mao, P. Wang, Y. Shen, and J. Yan, "Research on the security of visual reasoning {CAPTCHA}," in *30th USENIX security symposium (USENIX security 21)*, 2021, pp. 3291–3308.
[10] P. Wang, H. Gao, C. Xiao, X. Guo, Y. Gao, and Y. Zi, "Extended research on the security of visual reasoning captcha," *IEEE Transactions on Dependable and Secure Computing*, 2023.
[11] LinkedIn, https://www.linkedin.com/.
[12] TikTok, https://www.tiktok.com/.
[13] Twitter, https://www.twitter.com/.
[14] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.
[15] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, 2023.
[16] GPT-4V, https://openai.com/research/gpt-4v-system-card.
[17] Gemini, https://deepmind.google/technologies/gemini/#introduction.
[18] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.
[19] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.
[20] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM computing surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.
[21] Z. Noury and M. Rezaei, "Deep-captcha: a deep learning based captcha solver for vulnerability assessment," *arXiv preprint arXiv:2006.08296*, 2020.
[22] G. Deng, H. Ou, Y. Liu, J. Zhang, T. Zhang, and Y. Liu, "Oedipus captcha benchmark," May 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15339891
[23] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.
[24] R. Gossweiler, M. Kamvar, and S. Baluja, "What's up captcha? a captcha based on image orientation," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 841–850.
[25] V. P. Singh and P. Pal, "Survey of different types of captcha," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 2, pp. 2242–2245, 2014.
[26] G. Ye, Z. Tang, D. Fang, Z. Zhu, Y. Feng, P. Xu, X. Chen, and Z. Wang, "Yet another text captcha solver: A generative adversarial network based approach," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 332–348.
[27] Z. Noury and M. Rezaei, "Deep-captcha: a deep learning based captcha solver for vulnerability assessment," *arXiv preprint arXiv:2006.08296*, 2020.
[28] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, "Re:{CAPTCHAs—Understanding}{CAPTCHA-Solving} services in an economic context," in *19th USENIX Security Symposium (USENIX Security 10)*, 2010.
[29] M. Korakakis, E. Magkos, and P. Mylonas, "Automated captcha solving: An empirical comparison of selected techniques," in *2014 9th International Workshop on Semantic and Social Media Adaptation and Personalization*. IEEE, 2014, pp. 44–47.
[30] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "Pentestgpt: An llm-empowered automatic penetration testing tool," 2023.
[31] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model guided protocol fuzzing," in *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, 2024.
[32] Z. He, Z. Li, S. Yang, A. Qiao, X. Zhang, X. Luo, and T. Chen, "Large language models for blockchain security: A systematic literature review," 2024.
[33] H. Wang, X. Luo, W. Wang, and X. Yan, "Bot or human? detecting chatgpt imposters with a single question," *arXiv preprint arXiv:2305.06424*, 2023.
[34] Y. Gao, H. Gao, S. luo, Y. Zi, S. Zhang, W. Mao, P. Wang, Y. Shen, and J. Yan, "Research on the security of visual reasoning CAPTCHA," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3291–3308. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/gao
[35] N. T. Dinh and V. T. Hoang, "Recent advances of captcha security analysis: a short literature review," *Procedia Computer Science*, vol. 218, pp. 2550–2562, 2023.
[36] A. Searles, Y. Nakatsuka, E. Ozturk, A. Paverd, G. Tsudik, and A. Enkoji, "An empirical study & evaluation of modern CAPTCHAs," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 3081–3097. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/searles
[37] M. Kumar, M. Jindal, and M. Kumar, "A systematic survey on captcha recognition: types, creation and breaking techniques," *Archives of Computational Methods in Engineering*, vol. 29, no. 2, pp. 1107–1136, 2022.
[38] A. Searles, Y. Nakatsuka, E. Ozturk, A. Paverd, G. Tsudik, and A. Enkoji, "An empirical study & evaluation of modern CAPTCHAs," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 3081–3097. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/searles
[39] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023.

[40] W. Chen, M. Yin, M. Ku, P. Lu, Y. Wan, X. Ma, J. Xu, X. Wang, and T. Xia, "Theoremqa: A theorem-driven question answering dataset," 2023.
[41] gpt-4-vision preview, https://platform.openai.com/docs/guides/vision.
[42] gemini-pro vision, https://labelbox.com/product/model/foundry-models/google-gemini-pro-vision/.
[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.
[44] L. Yang, H. Chen, Z. Li, X. Ding, and X. Wu, "Chatgpt is not enough: Enhancing large language models with knowledge graphs for fact-aware language modeling," 2023.
[45] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," 2023. [Online]. Available: https://arxiv.org/abs/2305.10601
[46] Y. Zhang, Y. Yuan, and A. C.-C. Yao, "On the diagram of thought," 2025. [Online]. Available: https://arxiv.org/abs/2409.10038
[47] SQL, https://en.wikipedia.org/wiki/SQL.
[48] Three-valued_logic, https://en.wikipedia.org/wiki/Three-valued_logic.
[49] C. J. Date, *A Guide to the SQL Standard*. Addison-Wesley Longman Publishing Co., Inc., 1989.
[50] F. tuned code llama, https://github.com/ragntune/code-llama-finetune/blob/main/fine-tune-code-llama.ipynb.
[51] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "Designing human friendly human interaction proofs (hips)," 04 2005, pp. 711–720.
[52] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*. Springer, 2003, pp. 294–311.
[53] miniGPT 4, https://minigpt-4.github.io/.
[54] Arkose_Labs, https://www.arkoselabs.com/.
[55] Geetest, https://www.geetest.com/en/.
[56] NetEase_Yidun, https://dun.163.com/locale/en.
[57] [Online]. Available: https://openai.com/policies/usage-policies
[58] "Gemini app safety and policy guidelinese," https://gemini.google/policy-guidelines/, [Accessed 27-04-2025].
[59] "Updating our Usage Policy — anthropic.com," https://www.anthropic.com/news/updating-our-usage-policy, [Accessed 27-04-2025].
[60] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," in *RSS 2023 Workshop on Learning for Task and Motion Planning*, 2023. [Online]. Available: https://openreview.net/forum?id=tED747HURfX