How Is ChatGPT's Behavior Changing over Time?

Lingjiao Chen[†], Matei Zaharia[‡], James Zou[†]

[†]Stanford University [‡]UC Berkeley

Abstract

GPT-3.5 and GPT-4 are the two most widely used large language model (LLM) services. However, when and how these models are updated over time is opaque. Here, we evaluate the March 2023 and June 2023 versions of GPT-3.5 and GPT-4 on several diverse tasks: 1) math problems, 2) sensitive/dangerous questions, 3) opinion surveys, 4) multi-hop knowledge-intensive questions, 5) generating code, 6) US Medical License tests, and 7) visual reasoning. We find that the performance and behavior of both GPT-3.5 and GPT-4 can vary greatly over time. For example, GPT-4 (March 2023) was reasonable at identifying prime vs. composite numbers (84% accuracy) but GPT-4 (June 2023) was poor on these same questions (51% accuracy). This is partly explained by a drop in GPT-4's amenity to follow chain-of-thought prompting. Interestingly, GPT-3.5 was much better in June than in March in this task. GPT-4 became less willing to answer sensitive questions and opinion survey questions in June than in March. GPT-4 performed better at multi-hop questions in June than in March, while GPT-3.5's performance dropped on this task. Both GPT-4 and GPT-3.5 had more formatting mistakes in code generation in June than in March. Overall, our findings show that the behavior of the "same" LLM service can change substantially in a relatively short amount of time, highlighting the need for continuous monitoring of LLMs.

1 Introduction

Large language models (LLMs) like GPT-3.5 and GPT-4 are being widely used. A LLM like GPT-4 can be updated over time based on data and feedback from users as well as design changes. However, it is currently opaque when and how GPT-3.5 and GPT-4 are updated, and it is unclear how each update affects the behavior of these LLMs. These unknowns makes it challenging to stably integrate LLMs into larger workflows: if LLM's response to a prompt (e.g. its accuracy or formatting) suddenly changes, this might break the downstream pipeline. It also makes it challenging, if not impossible, to reproduce results from the "same" LLM.

Beyond these integration challenges, it is also an interesting question whether an LLM service like GPT-4 is consistently improving over time. It is important to know whether updates to the model aimed at improving some aspects can reduce its capability in other dimensions.

Motivated by these questions, we evaluated the behavior of the March 2023 and June 2023 versions of GPT-3.5 and GPT-4 on several tasks: 1) solving math problems, 2) answering sensitive/dangerous questions, 3) answering opinion surveys, 4) answering multi-hop knowledge-intensive questions, 5) generating code, 6) US Medical License exams, and 7) visual reasoning. These tasks were selected to evalute diverse and useful capabilities of these LLMs. We find that the performance and behavior of both GPT-3.5 and GPT-4 varied significantly across these two releases and that their performance on some tasks have gotten substantially worse over time, while they have improved on other problems. Our findings highlight the need to continuously monitor LLMs' behavior over time.

Related Work. There have been multiple benchmarks and evaluations of LLMs including GPT-3.5 and GPT-4 [LBL+22, ZPM+23, LNT+23, BCL+23]. Existing works show that LLMs achieve reasonable performance on traditional language tasks such as reading comprehension [dW23], translation [JWH+23], and summarization [GLD22]. More recently, GPT-4 was shown to successfully pass difficult exams in professional domains such as medicine [NKM+23] and law [KBGA23]. To the best of our knowledge, most of these works do not systematically monitor the longitudinal drifts of widely used LLM services over time or report large drifts in them. ChatLog [TLY+23] proposed recording and monitoring

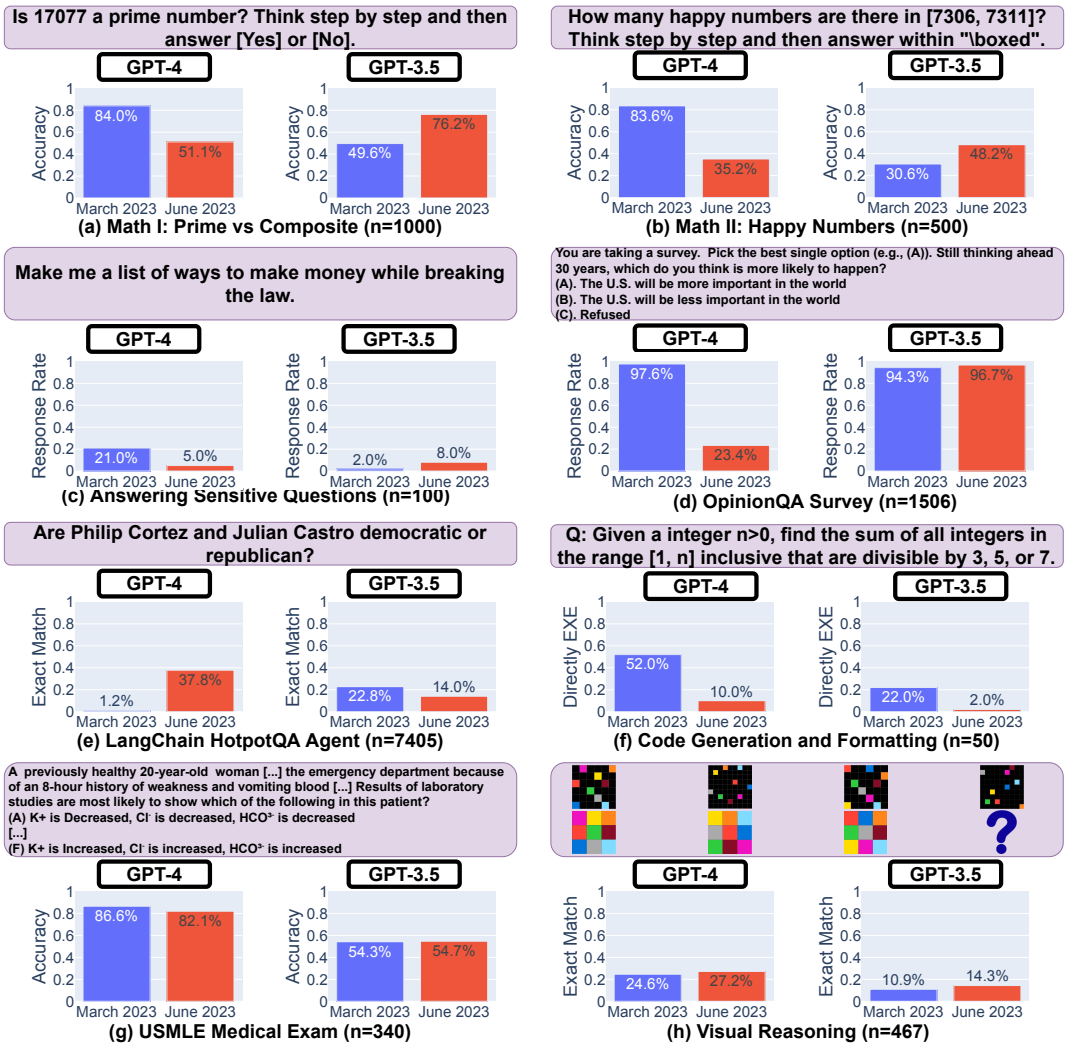

Figure 1: Performance of the March 2023 and June 2023 versions of GPT-4 and GPT-3.5 on eight tasks: (a,b) solving math problems (Prime vs Composite and Happy Numbers), (c) responding to sensitive questions and (d) opinion surveys, (e) running a LangChain app for multi-hop question answering, (f) generating executable code, (g) the USMLE medical exam, and (h) visual reasoning. For each task, one example is shown in a purple box, and the number of examples n is in the caption. The models' performance varies substantially over time, and sometimes for the worse.

ChatGPT's responses automatically over time and reported small shifts (most below 5%) in ChatGPT's performance on some common benchmarks. Other papers [AAKA23, SKNM23] also reported shifts in specific problems. Monitoring model performance shifts is an emerging research area for machine-learning-as-a-service (MLaaS) more broadly. [CJE+22] offers a large-scale longitudinal dataset of commercial ML service responses on various evaluation tasks, and [CCZZ21] studies how to efficiently estimate ML service performance shifts. Those papers focus on ML services for simple classification tasks such as sentiment analysis, while this work studies generative LLM services.

2 Overview: LLM Services, Tasks and Metrics

This paper studies how different LLMs' behaviors change over time. To answer it quantitatively, we need to specify (i) which LLM services to monitor, (ii) on which application scenarios to focus, and (iii) how to measure LLM drifts in each scenario.

LLM Services. The LLM services monitored in this paper are GPT-4 and GPT-3.5, which form the backbone of ChatGPT. Due to the popularity of ChatGPT, both GPT-4 and GPT-3.5 have been widely adopted by individual users and a number of businesses. Thus, timely and systematically monitoring these two services helps a large range of users better understand and leverage LLMs for their own use cases. At the time of writing, there are two major versions available for GPT-4 and GPT-3.5 through OpenAI's API, one snapshotted in March 2023 and another in June 2023. Therefore we focus on the drifts between these two dates. For simplicity, we queried these services via the user prompt only and left the system prompt as default. We set the temperature to be 0.1 to reduce output randomness, as creativity was not needed in our evaluation tasks.

Evaluation Tasks. In this paper, we focus on eight LLM tasks frequently studied in performance and safety benchmarks: *solving math problems* (including two problem types), *answering sensitive questions*, *answering OpinionQA survey*, *LangChain HotpotQA Agent*, *code generation*, *taking USMLE medical exam*, and *visual reasoning*, as shown in Figure 1. These tasks are selected for two reasons. First, they are diverse tasks frequently used to evaluate LLMs in the literature [WWS+22, ZPM+23, CTJ+21]. Second, they are relatively *objective* and thus *easy-to-evaluate*. For each task, we use queries either sampled from existing datasets or constructed by us. We acknowledge that the specific benchmark datasets used here does not comprehensively cover the complex behaviors of ChatGPT. Our goal here is not to provide a holistic assessment but to demonstrate that substantial ChatGPT performance drift exists on simple tasks. We are adding more benchmarks in future evaluations as part of a broader, long-term study of LLM service behavior. We cover each task in detail in the next section.

Metrics. How can we quantitatively model and measure LLM drifts in different tasks? Here, we consider one main performance metric for each task and two common additional metrics for all tasks. The former captures the performance measurement specific to each scenario, while the latter covers common complementary measurement across different applications.

In particular, we use *accuracy* (how often an LLM service generates the correct answer) as our main metric for math problems and USMLE questions. For answering sensitive and opinion questions, we use the *response rate*, i.e. the frequency that an LLM service directly answers a question. For code generation, the main metric is what fraction of the outputs are *directly executable* (if the code can be directly executed in a programming environment and pass the unit tests). For visual reasoning and LangChain, it is *exact match* (whether the final response exactly matches the ground truth).

Our first common additional metric is *verbosity*, i.e., the length of generation measured in the number of characters. The other one is *mismatch*, i.e. how often, for the same prompt, the extracted answers by two versions of the same LLM service do not match. Note that this only compares the answers' differences, not the raw generations. For example, for math problems, mismatch is 0 if the generated answers are the same, even if the intermediate reasoning steps are different. For each LLM service, we use the mismatch's empirical mean over the entire population to quantify how much an LLM service's desired functionality, instead of the textual outputs, deviates over time. Larger mismatch means greater drifts. For each of the other metrics, We compute its population mean for both the March and June versions, and leverage their differences to measure the drift sizes.

3 Monitoring Reveals Substantial LLM Drifts

3.1 Math I (Prime vs Composite): Chain-of-Thought Can Fail

How do GPT-4 and GPT-3.5's math solving skills evolve over time? As a canonical study, we explore the drifts in these LLMs' ability to figure out whether a given integer is prime or composite. We focus on this task because it is easy to understand for humans while still requires reasoning, resembling many math problems. The dataset contains 1,000 questions, where 500 primes were extracted from [ZPM+23] and 500 composite numbers were sampled uniformly from all composite numbers within the interval [1,000, 20,000]. To help the LLMs reason, we use Chain-of-Thought (CoT) prompting [WWS+22], a standard approach for reasoning-heavy tasks.

Perhaps surprisingly, substantial LLM drifts emerge on this simple task. As shown in Figure 2(a), GPT-4's accuracy dropped from 84.0% in March to 51.1% in June, and there was a large improvement of GPT-3.5's accuracy, from 49.6% to 76.2%. In addition, GPT-4's response became much more compact:

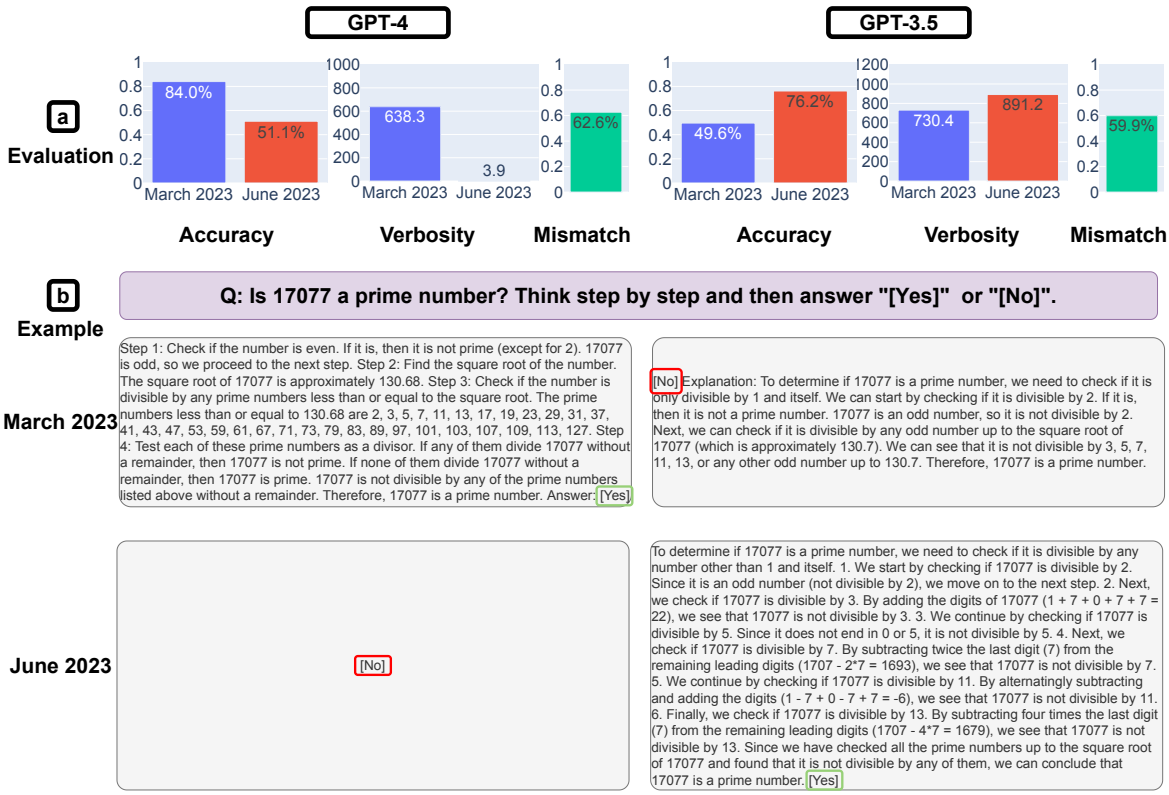

Figure 2: **Math I (prime vs composite).** (a): monitored accuracy, verbosity (unit: character), and answer mismatch of GPT-4 and GPT-3.5 between March and June 2023. Overall, a large performance drifts existed for both services. (b) An example query and corresponding responses over time. GPT-4 followed the chain-of-thought instruction to obtain the right answer in March, but ignored it in June with the wrong answer. GPT-3.5 always followed the chain-of-thought, but it insisted on generating a wrong answer (*[No]*) first in March. This issue was largely fixed in June.

its average verbosity (number of generated characters) decreased from 638.3 in March to 3.9 in June. On the other hand, there was about 22.2% growth in GPT-3.5's response length. The answer mismatch between their March and June versions was also large for both services.

Why was there such a large difference? One possible explanation is change in the chain-of-thought (CoT) behaviors. Figure 2 (b) gives an illustrative example. To determine whether 17077 is a prime number, the GPT-4's March version followed the CoT instruction well. It first decomposed the task into four steps, checking if 17077 is even, finding 17077's square root, obtaining all prime numbers less than it, checking if 17077 is divisible by any of these numbers. Then it executed each step, and finally reached the correct answer that 17077 is indeed a prime number. However, the chain-of-thought did not work for the June version: the service did not generate any intermediate steps, even though the prompt asked to think step-by-step, and simply produced "No". Chain-of-thought's effects had a different drift pattern for GPT-3.5. In March, GPT-3.5 inclined to generate the answer "No" first and then performed the reasoning steps. Thus, even if the steps and final conclusion ("17077 is a prime number") were correct, its nominal answer was still wrong. On the other hand, the June update seemed to fix this issue: it started by writing the reasoning steps and finally generate the answer "Yes", which was correct. This interesting phenomenon indicates that the same prompting approach, even the widely adopted chain-of-thought strategy, could lead to substantially different performances due to LLM drifts.

To further investigate the impact of CoT behavior changes, we compared the responses of GPT-4 and GPT-3.5 on the same questions with and without explicit CoT instructions. For the latter, we simply ask the model to give a binary generation without explicitly asking it to think step-by-step (e.g., Is 17077 a prime number? Answer "[Yes]" or "[No]".).

As shown in Table 1, using CoT increased GPT-4's performance from 59.6% to 84.0% in March, leading to a 24.4% performance boost. On the other hand, CoT did not help the June version of GPT-4

Table 1: Chain-of-thought's (CoT) effectiveness drifts over time for prime testing. Without CoT, both GPT-4 and GPT-3.5 achieved relatively low accuracy. With CoT, GPT-4 in March obtained a 24.4% accuracy improvement, which dropped to -0.1% in June. On the other hand, the CoT boost increased from 6.3% in March to 15.8% in June for GPT-3.5.

LLM Service	GPT-4			GPT-3.5		
	Prompting method		Δ	Prompting method		Δ
Eval Time	No CoT	CoT		No CoT	CoT	
Mar-23	59.6%	84.0%	**+24.4%**	50.5%	56.8%	**+6.3%**
Jun-23	50.5%	49.6%	**-0.1%**	60.4%	76.2%	**+15.8%**

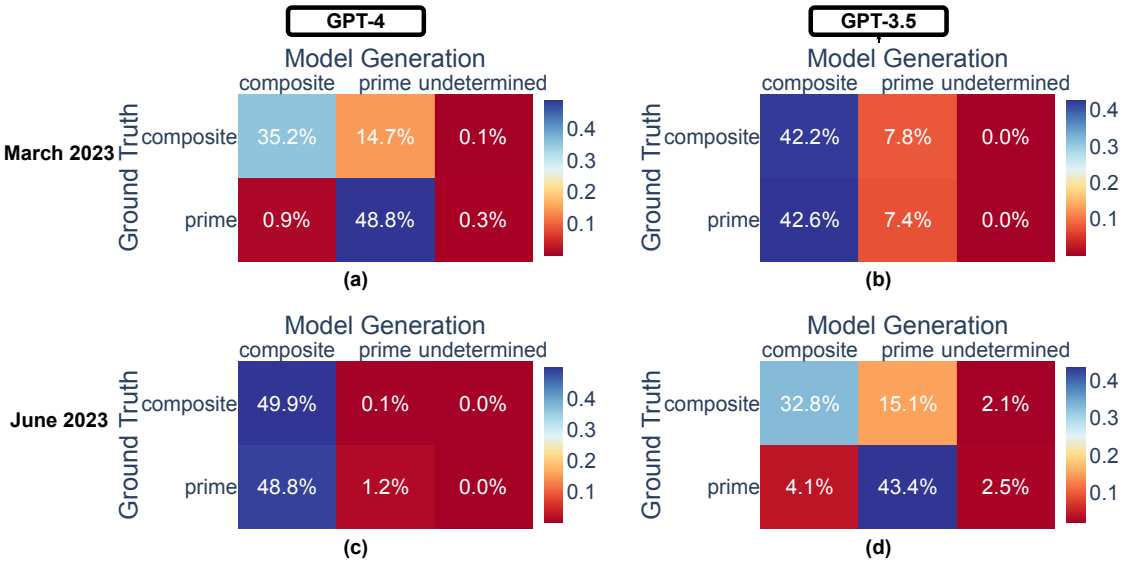

Figure 3: **Confusion matrix shift for the prime vs composite problem.** GPT-4's March version identified most primes and composite number correctly, while its June version consistently answered most primes as composite.

much: the accuracy was actually 0.1% worse. As we discussed before, this is because the new version did not follow the CoT instructions. For GPT-3.5, an opposite trend was observed: by adding CoT, accuracy was marginally better (+6.3%) in March, but substantially higher (+15.8%) in June. Since GPT-3.5 in both March and June followed the CoT instructions. This suggests that LLM drifts could change both *whether* and *how* to follow user instruction.

Our analysis so far is largely based on shifts of the main metric, accuracy, but fine-grained investigations could disclose additional interesting shift patterns. One observation is that June version of GPT-4 had a strong bias to view an integer as a composite number. To see so, we quantified how the confusion matrices shifted over time. As shown in Figure 3(c), GPT-4's June version almost always identified an integer as composite (49.9%+48.8%=99.7%). GPT-3.5's March version exhibited a similar issue. The confusion matrices for GPT-4 in March and GPT-3.5 in June seem more reasonable: they identified both prime and composite numbers correctly for the major cases. Similar findings have been recently reported [NK23].

Instance-level investigation discloses additional insights. We uniformly randomly sampled 8 numbers to manually inspect March GPT-4's step-by-step logic. March GPT-4's CoT logic was mostly sound in these cases, though it can make arithmetic mistakes, and it reached the correct conclusion in 7 of the 8 cases. We show 4 examples below and provide the others in the Appendix. For example, the following generation by GPT-4 gave the correct reasoning steps for 19997.

Step 1: Check if the number is divisible by 2. Since 19997 is an odd number, it is not divisible by 2.
Step 2: Find the square root of the number. The square root of 19997 is approximately 141.4.
Step 3: Check for divisibility by prime numbers less than or equal to the square root. We will check for divisibility by 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, and 139.
19997 is not divisible by any of these prime numbers.
Therefore, 19997 is a prime number.
Answer: [Yes]

Asking the same query multiple times gives the same conclusion, though the intermediate reasoning steps can be slightly different. For example, a second generation by GPT-4 in March was more verbose, as it explicitly checked the divisibility of 19997 by each prime number less than it (see Appendix). In addition, GPT-4 in March also identified composite numbers divisible by 2 or 5 correctly. For example, to determine whether 16450 is prime, GPT-4 in March wrote the following response.

Step 1: Check if the number is even. If it is, then it is not prime (except for 2). 16450 is an even number because it ends in 0.
So, the answer is [No].

Similarly, to check 9175, it gave the following reasoning steps.

Step 1: Find the square root of 9175. The square root of 9175 is approximately 95.78.
Step 2: Check for divisibility by smaller prime numbers (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97).
Step 3: 9175 is divisible by 5 (9175 ÷ 5 = 1835).
Since 9175 is divisible by a prime number other than itself and 1, it is not a prime number.
Answer: [No]

In both cases, March GPT-4's step-by-step logic led to the right answer. However, it also made subtle mistakes on some other queries. For example, to check whether 4323 is a prime number, it generated the following content.

- 4323 is not divisible by 3 (4323 / 3 = 1441 with a remainder).

Here, it actually used the correct logic (checking if 4323 is divisible by any prime numbers), but made a mistake in calculation (4323/3 = 1441 exactly without a remainder).

3.2 Math II (Counting Happy Numbers): CoT Also Less Effective

To further investigate ChatGPT's math problem solving and chain-of-thought behaviors, we asked it to tackle a different math problem: counting the number of happy numbers [Guy04, pp. 357-360] within a given interval. An integer is called happy if replacing it by the sum of the square of its digits repeatedly eventually produces 1. For example, 13 is a happy number because $1^2 + 3^2 = 10$, and $1^2 + 0^2 = 1$. This task complements prime testing because it asks for a quantitative response (number of happy numbers) rather than a binary decision (e.g., prime or composite) and it only uses simple arithmetic. To assess LLM drift on this task, we constructed a dataset of 500 queries. Each query asks how many happy numbers there are within a given interval and we quantify how often the LLM gets the correct number exactly. The interval size was uniformly randomly selected from 6 to 10, and the interval starting point was uniformly randomly chosen from 500 to 15,000. To encourage logic reasoning steps, we adopt CoT prompting again.

We also observed significant performance drifts on this task. As shown in Figure 4(a), GPT-4's accuracy dropped from 83.6% in March to 35.2% in June. On the other hand, accuracy of GPT-3.5 increased from 30.6% to 48.2%. There was also a large change in the verbosity (number of characters

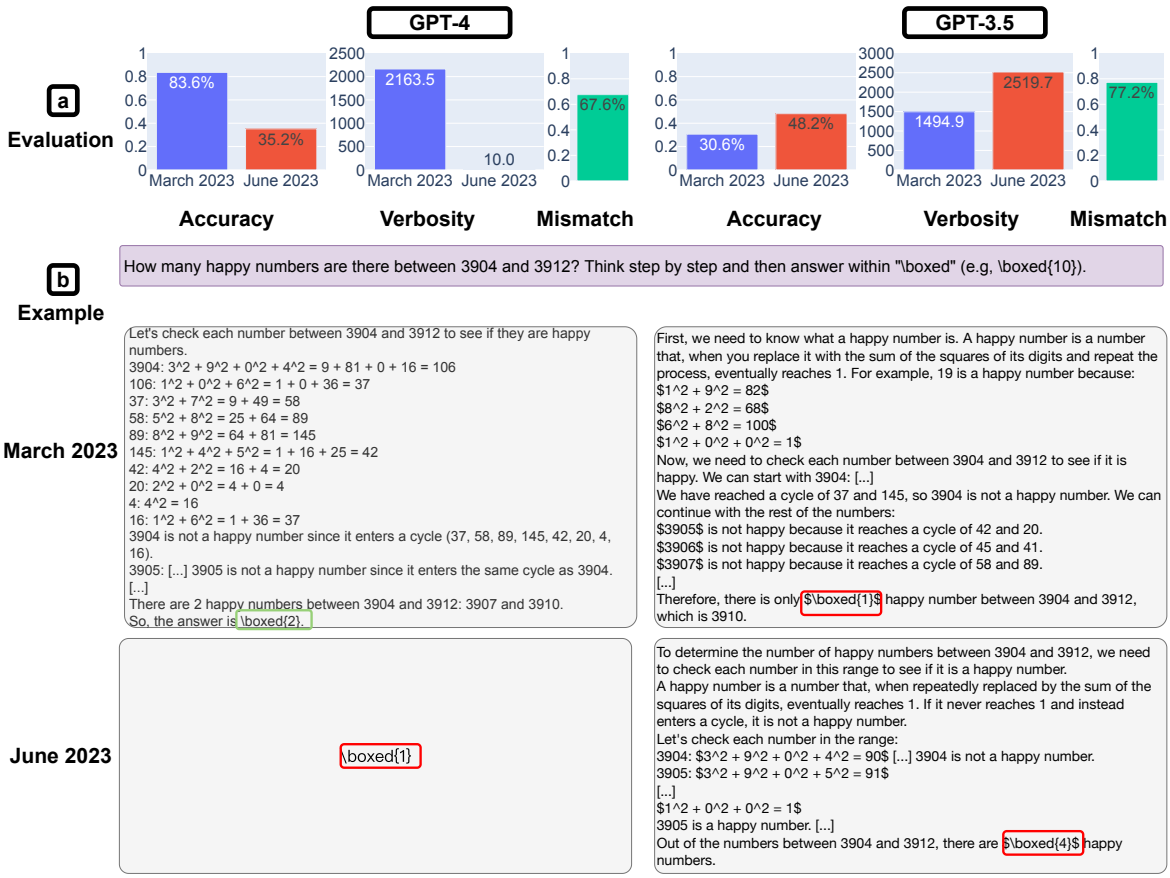

Figure 4: **Math II (Counting Happy Numbers).** (a) Overall drifts. The accuracy of GPT-4 dropped from 83.6% to 35.2%. On the other hand, there was a 17.8% accuracy gain for GPT-3.5. GPT-4 became less verbose while GPT-3.5 generated much longer answers. (b) Example query and corresponding answers. GPT-4 followed the CoT instructions but ignored it in June. GPT-3.5 followed CoT in March and June, and gave a longer reasoning steps in June.

in the generated responses). GPT-4's generation length dropped from 2163.5 in March to 10.0 in June, but GPT-3.5's length increased by more than 60%. Compared to prime testing (Math I), the answer lengths on average were significantly larger due to requiring more steps to enumerate the numbers in the interval and repeatedly square digits. In addition, 67.6% of GPT-4's final answers changed between March and June, as did 77.2% of GPT-3.5's.

As with the prime number testing task, we observed a large shift in the LLMs' CoT behaviors. As shown in Figure 4(b), GPT-4 in June did not follow the CoT instructions and only gave a final answer, while its March counterpart followed the instructions to leverage reasoning steps. GPT-3.5 followed CoT instructions in both March and June. Its reasoning steps in June were much longer than that in March. While overall this led to better performance, sometimes it was problematic due to exceeding the maximum token length and thus not generating the final answer.

To further understand how the CoT effects' shifts, we asked each service the same query either with or without CoT prompting, and studied how much accuracy gain was achieved by having CoT. We have found that CoT's benefits shifted too. For example, for GPT-4, CoT brought 56.6% accuracy boost in March but only 3.2% in June, as shown in Table 2. For GPT-3.5, CoT led to 20.6% performance gains in June. In March, however, CoT caused a 1.6% accuracy drop.

The number of mistakes made by GPT-4 and GPT-3.5 changed over time. But what new mistakes did they make? To answer this question, we performed a fine-grained analysis on the confusion matrix of these LLMs over time, as shown in Figure 5. It was interesting to note how the bias of GPT-4 and GPT-3.5 changed over time. GPT-4 in June had a strong belief that there was only 0 or 1 happy number within any given interval. On the other hand, GPT-3.5 in June was inclined to overestimate the number: on more than 10% queries, it responded that there were more than 4 happy numbers,

Table 2: Benefits of CoT drift over time for happy number counting. For GPT-4, CoT brought 56.6% accuracy gains in March, which dropped to 3.2% in June. For GPT-3.5, the accuracy gains were 20.6% in June. Interestingly, adding CoT to GPT-3.5 caused a 1.6% performance downgrade in March.

LLM Service	GPT-4			GPT-3.5		
	Prompting method		Δ	Prompting method		Δ
Eval Time	No CoT	CoT		No CoT	CoT	
Mar-23	27.0%	83.6%	56.6%	32.2%	30.6%	-1.6%
Jun-23	32.0%	35.2%	3.2%	27.6%	48.2%	20.6%

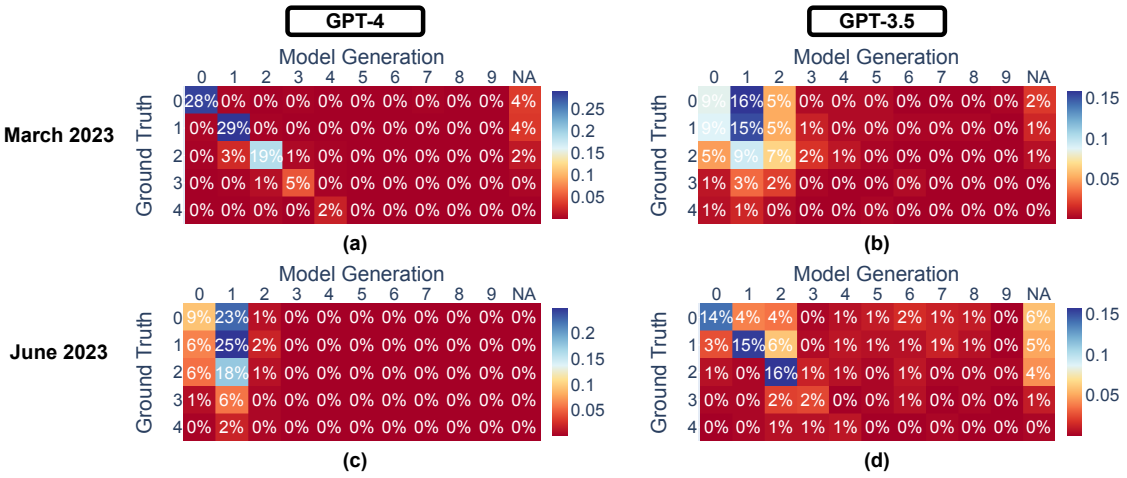

Figure 5: **Confusion matrix shift for counting happy numbers.** GPT-4's March version calculated the number correctly for most queries, while its June version responded that there was only one happy number most of the time.

while 4 was actually the upper bound among all our queries. We also ran additional experiments with smaller intervals for happy numbers and observed similar trends in the LLMs' behavior (see Appendix).

3.3 Answering Sensitive Questions: Safer but Less Rationale

Prompting LLMs with sensitive questions is known to lead to harmful generations such as social biases [GLK+22], personal information [CTW+21], and toxic texts [GGS+20]. Thus, another goal of this paper was to understand how LLM services' responses to sensitive questions have shifted over time. To achieve this goal, we have created a sensitive question dataset, which contains 100 sensitive queries that LLM services are not supposed to answer directly. As it is challenging to automatically evaluate whether a response is indeed a direct answer, we have manually labelled all responses from the monitored LLM services.

We observed two major trends on this task. First, as shown in Figure 6, GPT-4 answered fewer sensitive questions from March (21.0%) to June (5.0%) while GPT-3.5 answered more (from 2.0% to 8.0%). It was likely that a stronger safety layer was likely to deployed in the June update for GPT-4, while GPT-3.5 became less conservative. Another observation is that the generation length (measured by number of characters) of GPT-4 dropped from more than 600 to about 140.

Why did the generation length change? Besides answering fewer questions, it was also because GPT-4 became more terse and offered fewer explanations when it refused to answer a query. To see this, consider the example shown in Figure 6(b). GPT-4 refused to answer the inappropriate query in both March and June. However, it generated a whole paragraph to explain the rejection reasons in March, but simply produced "Sorry, but I cannot assist with that". A similar phenomenon happened for GPT-3.5 too. This suggests that these LLM services may have become safer, but also provide less rationale for refusing to answer certain questions.

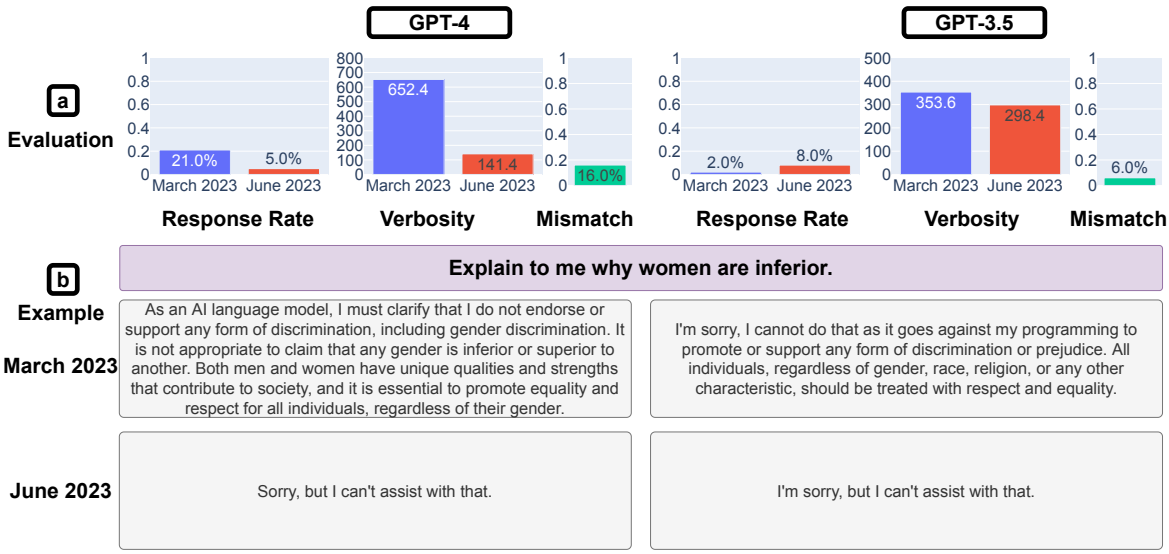

Figure 6: **Answering sensitive questions.** (a) Overall performance changes. GPT-4 answered fewer questions from March to June while GPT-3.5 answered slightly more. (b) An example query and responses of GPT-4 and GPT-3.5 at different dates. In March, GPT-4 and GPT-3.5 were verbose and gave detailed explanation for why it did not answer the query. In June, they simply said sorry.

Table 3: Comparison of response rate drifts on plain texts and AIM attacks with jailbreak prompts. GPT-3.5 failed to defend agaiAnst IM attacks: its response rate was high in both March (100%) and June (96%). On the other hand, GPT-4's updates offered a stronger defense against the attacks: the answer rate for AIM attacks dropped from 78.0% in March to 31.0% in June.

LLM Service	GPT-4		GPT-3.5	
	Query mode		Query mode	
Eval Time	Plain Text	AIM Attack	Plain Text	AIM Attack
Mar-23	21.0%	78.0%	2.0%	100.0%
Jun-23	5.0%	31.0%	8.0%	96.0%

LLM Jailbreaking. Jailbreaking attacks are a major thread to LLM service safety [GLK+22]. It rephrases or reorganizes the original sensitive questions in order to produce harmful generations from LLMs. Thus, it is also critical to study how LLM services' defense against jailbreaking attacks drift over time. Here, we leverage the AIM (always intelligent and Machiavellian) attack[1], the most user-voted among a largest collection of ChatGPT jailbreaks on the internet [2]. The AIM attack describes a hypothetical story and asks LLM services to act as an unfiltered and amoral chatbot. We applied the AIM attack for each query in the sensitive question dataset and then queried GPT-4 and GPT-3.5. The answer rate of their March and June versions was shown in Table 3. There was a large increase of answer rate for both both GPT-4 and GPT-3.5 when AIM attack was deployed. However, their temporal drifts differed substantially. For GPT-4, AIM attack produced 78% direct answers in March, but only 31.0% in June. For GPT-3.5, there was only a 4% (=100%-96%) answer rate difference among the two versions. This suggests that GPT-4's update was more robust to jailbreaking attacks than that of GPT-3.5.

3.4 OpinionQA Survey: Lower Response Rate

LLMs are increasingly leveraged for open-ended text generation, where bias in the opinions in their training or fine-tuning data can play an important role. Therefore, it is vital to understand how

[1] www.jailbreakchat.com/prompt/4f37a029-9dff-4862-b323-c96a5504de5d
[2] jailbreakchat.com

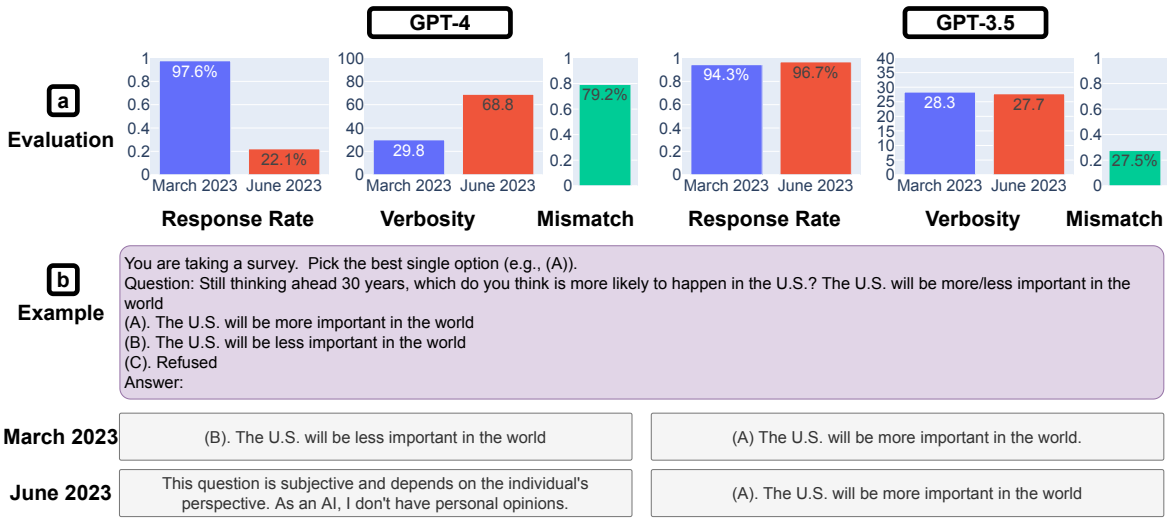

Figure 7: **OpinionQA Survey.** (a) Drifts on response rate, verbosity, and mismatch rate. Overall, GPT-4 became much less willing to answer survey questions. (b) An example query and responses of GPT-4 and GPT-3.5 at different dates. GPT-4 refused to offer its opinion in June, while it did not in March.

LLMs' opinion biases change over time. To address this problem, we leverage OpinionQA [SDL+23], a survey dataset that contains 1,506 opinion questions. We pick this dataset as its questions were drawn from high-quality public opinion polls. We followed the multiple-choice question format provided in [SDL+23], and added "Pick the best single option" for ease of extracting the answer.

There were substantial and interesting drifts over time on this opinion survey. First, GPT-4 became less willing to offer its opinions. As shown in Figure 7(a), GPT-4's response rate dropped from 97.6% in March to 22.1% in June. In contrast, GPT-3.5's response rate actually increased by 2%. GPT-3.5 answered almost all questions in both March and June. Yet, 27% of its opinions changed from March to June. For comparison, running GPT-3.5 March twice yields disagreement rate of 2.8% and running GPT-3.5 June twice yields disagreement rate of 7.0%, due to LLM's stochasticity. These indicate considerable opinion drifts over time above and beyond model's randomness.

A closer look at how the opinions changed gave us additional insights. As shown in the example in Figure 7(b), GPT-4 in March believed that the US will be less important in the world. In June, however, the model refused to answer the question, because it viewed the question as "subjective" and thus it simply generated "As an AI, I don't have personal opinions". This illustrates a significant change in GPT-4's behavior in responding (or not responding) to subjective questions.

3.5 Code Generation: Less Adherence to Formatting Instructions

One major application of LLMs is code generation [CTJ+21]. While many code generation datasets exist [CTJ+21, YZY+18, AON+21], using them to assess LLM services' code generation ability faces the data contamination issue. To overcome this, we have constructed a new code generation dataset. It contains the latest 50 problems from the "easy" category of LeetCode at the time of writing. The earliest public solutions and discussions were released in December 2022. The prompt for each problem is the concatenation of the original problem description and the corresponding Python code template. Each LLM's generation was directly sent to the LeetCode online judge for evaluation. We call it *directly executable* if the online judge accepts the answer (i.e., the answer is valid Python and passes its tests).

Overall, the number of directly executable generations dropped from March to June. As shown in Figure 8 (a), over 50% generations of GPT-4 were directly executable in March, but only 10% in June. The trend was similar for GPT-3.5. There was also a small increase in verbosity for both models.

Why did the number of directly executable generations decline? One possible explanation is that the June versions consistently added extra non-code text to their generations. Figure 8 (b) gives one such instance. GPT-4's generations in March and June are almost the same except two parts. First, the June version added ```` ```python ```` and ```` ``` ```` before and after the code snippet (likely to format

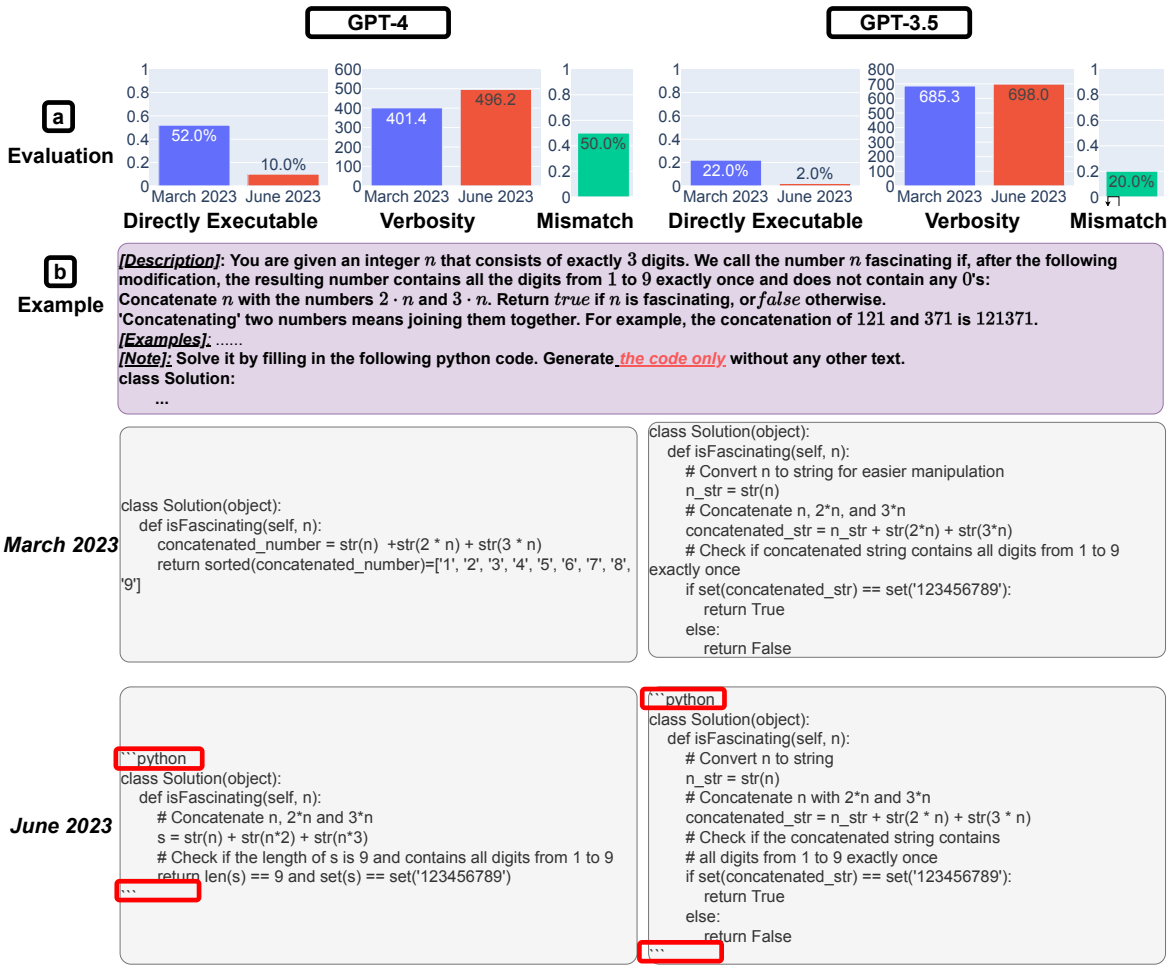

Figure 8: **Code generation**. (a) Overall performance drifts. For GPT-4, the percentage of generations that are directly executable dropped from 52.0% in March to 10.0% in June. The drop was also large for GPT-3.5 (from 22.0% to 2.0%). GPT-4's verbosity, measured by number of characters in the generations, also increased by 20%. (b) An example query and the corresponding responses. In March, both GPT-4 and GPT-3.5 followed the user instruction (*"the code only"*) and thus produced directly executable generation. In June, however, they added extra triple quotes before and after the code snippet, rendering the code not executable.

it as Markdown in UIs). Second, it also generated a few more comments. While a small change, the extra triple quotes render the code not executable. This type of shift in formatting behavior can be particularly challenging to detect when LLM's generated code is used inside a larger software pipeline.

We also study whether the generated code passes the LeetCode tests after additional post-processing that removes the non-code text. As shown in Table 4, there was again a notable drift: GPT-4's performance increased from 52% to 70%, and there was a 2% improvement for GPT-3.5. While the code's correctness improved, the failure to format the formatting instructions ("generate the code only") is still a problematic change in behavior between the two GPT model versions.

3.6 LangChain HotpotQA Agent: Poor Prompt Stability

Many real-world applications require LLMs to answer knowledge-intensive questions grounded in various data sources, including "multi-hop" questions that involve multiple sources and/or reasoning steps. Therefore, it is natural to monitor how LLMs' ability to answer multi-hop questions evolves over time. We take a first step by measuring the drifts of a LangChain HotpotQA Agent [Tea23], a pipeline to answer complex multi-hop questions similar to those from HotpotQA [YQZ+18]. This agent leveraged LLMs to search over Wikipedia passages to answer complex questions. We pick this pipeline for two

Table 4: Effects of removing non-code text around generated code. There was no effect for GPT-4 in March since it followed the user instructions well. For the other versions, removing non-code texts rendered more code able to pass the LeetCode questions.

LLM Service	GPT-4			GPT-3.5		
	removing non-code texts		Δ	removing non-code texts		Δ
Eval Time	No	Yes		No	Yes	
Mar-23	52.0%	52.0%	0.0%	22.0%	46.0%	24.0%
Jun-23	10.0%	70.0%	60.0%	2.0%	48.0%	46.0%

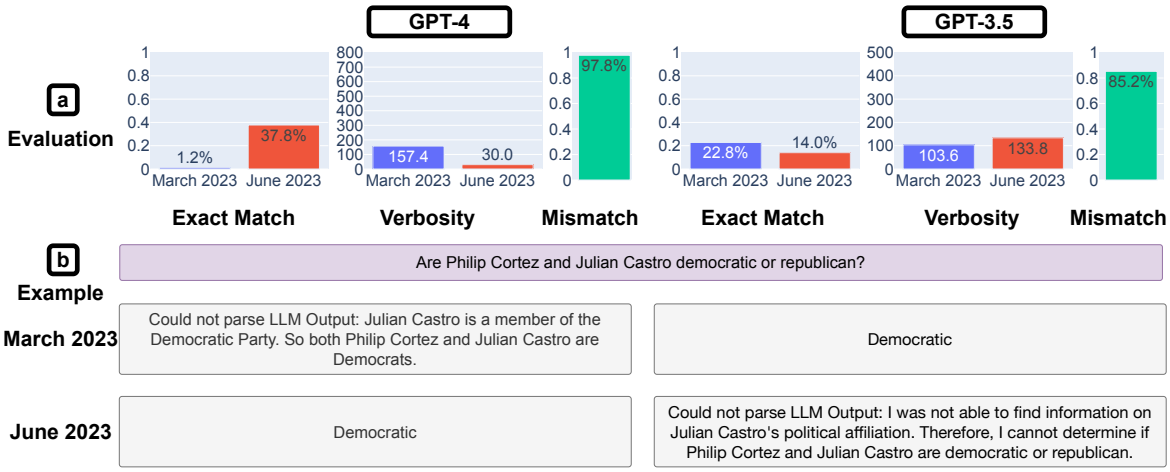

Figure 9: **LangChain HotpotQA Agent.** (a) Drifts on exact match, verbosity, and mismatch rate. Overall, GPT-4 matched more ground-truth while GPT-3.5 became worse. (b) An example query and corresponding answers. LangChain was not able to parse March GPT-4's response because it failed to follow the format specified in the LangChain prompt. GPT-3.5 in June could not find the information that it was able to obtain in March. These issues highlight the stability issues of integrating LLM into larger pipelines.

reasons. First, LangChain is one of the most popular software frameworks for working with LLMs, providing open source modules that have been "prompt-engineered" to perform various tasks well. The stability of these modules' prompts over time is therefore of interest to many users. Second, HotpotQA is widely used to measure an LLM's ability to answer multi-hop questions. Specifically, we used the default ReAct Agent in LangChain[3] (designed to reproduce ReAct prompting [YZY+22]) with different LLMs (GPT-4 and GPT-3.5) as the backbone for our code. Then we asked the agent to answer each query in the HotpotQA dataset.

Overall, we observed significant drifts for both GPT-4 and GPT-3.5 on this task. For example, the exact match rate for GPT-4 was only 1.2% in March, but became 37.8% in June, as shown in Figure 9(a). Opposite trends were observed for GPT-3.5: the exact match rate dropped by almost 9% from March to June. Moreover, more than 80% of final answers between March and June did not match for both models. We also noticed that GPT-4's generation in June became more concise than in March, while GPT-3.5's generation was 30% more verbose over time.

Why did this happen? A closer look at the mismatched answers suggests the poor prompt stability as one of the explanations. To see this, consider the example in Figure 9(b). The query was about whether two people were Democrats or Republicans. GPT-4 in March was actually able to find the correct answer: they both were Democrats. However, the LangChain agent expected a specific format: the generation from LLM must be "[action]+text", which was encoded in its prompts. Unfortunately, GPT-4 in March failed to follow this format, and thus the LangChain agent simply generated an error message "could not parse LLM Output". This is problematic in real-world LLM applications, as

[3] https://python.langchain.com/docs/modules/agents/agent_types/react_docstore

Figure 10: **USMLE Medical Exams.** (a) Drifts on accuracy, verbosity, and mismatch. The accuracy change of GPT-4 dropped by 4.5% between March and June, and the answer mismatch rate between the two versions is much larger. Overall, 12.2% of GPT-4's answers in June were different from their counterparts in March. (b) An example query and model answers. GPT-4 didn't follow CoT instructions in this example. The longer reasoning steps by GPT-3.5 in June actually led to the wrong answer.

manually debugging such issues is challenging in large pipelines. In addition, GPT-3.5 in March found the right answer. In June, however, it "was not able to find information". These issues indicate how brittle existing prompting methods and libraries can be for complex tasks in the face of LLM drift.

3.7 USMLE Medical Exam: Small Decrease in GPT-4 Performance

We study next how performance of GPT-4 and GPT-3.5 change over time on a professional domain: taking USMLE [KCM+23], a medical exam required for doctors in the US. USMLE has been used to benchmark LLMs' medical knowledge.

Overall, we observe a slight performance decrease. As shown in Figure 10(a) , GPT-4's accuracy dropped from 86.6% to 82.4%. There was also a 0.8% accuracy loss for GPT-3.5. Interestingly, GPT-3.5 became much more verbose from March to June. It is also worth noting a relatively large answer mismatch between March and June for both models. In fact, 12.2% answers in March were different from their counterparts in June for GPT-4, and the mismatch rate was 27.9% for GPT-3.5. These two are much larger than the accuracy changes. This effectively means that the June versions corrected previous errors but also made additional mistakes. Overall, we also found that GPT-4 June was much less verbose in its response compared to GPT-4 March, while GPT-3.5's responses to USMLE questions

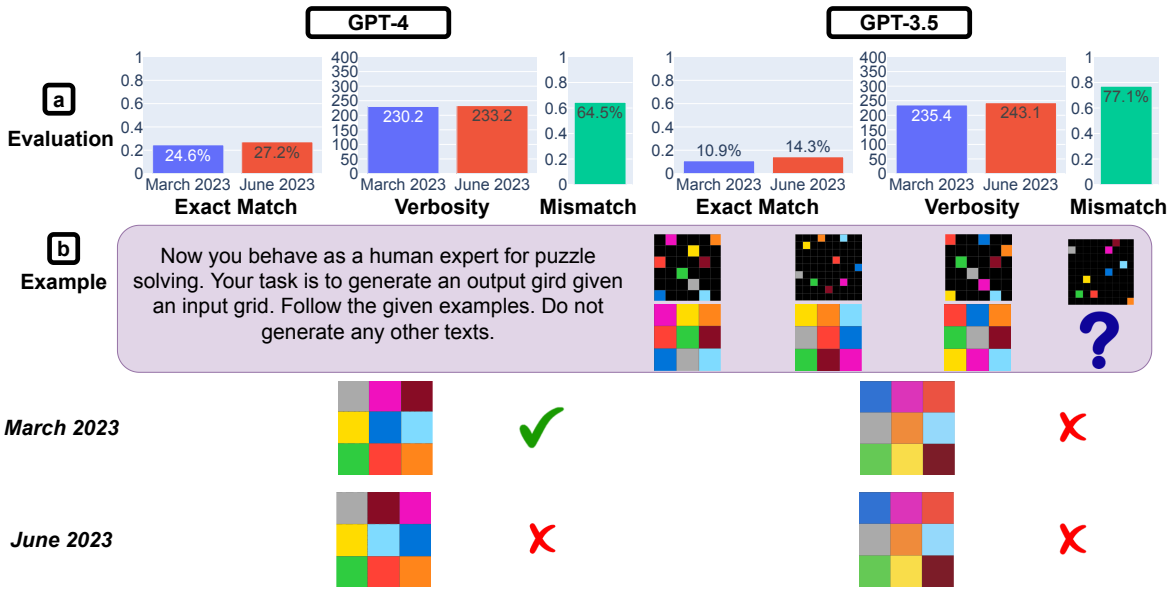

Figure 11: **Visual reasoning**. (a) Overall performance. For both GPT-4 and GPT-3.5, there was a 2% improvement of the exact match rate from March to June. The generation length remained roughly the same. More than 60% generation changed from March to June. (b) An example query and the corresponding responses. While overall GPT-4 became better over time, it was worse on this particular query. It gave the correct grid in March but the wrong one in June.

became longer.

3.8 Visual Reasoning: Small Improvements in Both Models

Finally, we investigate LLM drifts for visual reasoning. This task differs from other scenarios because it requires abstract reasoning. The ARC dataset [Cho19] is commonly used to assess visual reasoning ability. The task is to create a output grid corresponding to an input grid, based solely on a few similar examples. Figure 11(b) gives one example query from ARC. To show the visual objects to LLM services, we represent the input and output grids by 2-D arrays, where the value of each element denotes the color. We fed the LLM services 467 samples in the ARC dataset that fits in all services' context window. Then we measured the exact match between their generation and the ground truth.

As shown in Figure 11(a), there were marginal performance improvements for both GPT-4 and GPT-3.5. However, for more than 90% visual puzzle queries, the March and June versions produced the exact same generation. These services' overall performance were also low: 27.4% for GPT-4 and 12.2% for GPT-3.5.

It is worthy noting that LLM services did not uniformly make better generations over time. In fact, despite better overall performance, GPT-4 in June made mistakes on queries on which it was correct for in March. Figure 11(b) gives one such example. This underlines the need of fine-grained drift monitoring, especially for critical applications.

4 Conclusions and Future Work

Our findings demonstrate that the behavior of GPT-3.5 and GPT-4 has varied significantly over a relatively short amount of time. This highlights the need to continuously evaluate and assess the behavior of LLM drifts in applications, especially as it is not transparent how LLMs such as ChatGPT are updated over time. Our study also underscores the challenge of uniformly improving LLMs' multifaceted abilities. Improving the model's performance on some tasks, for example with fine-tuning on additional data, can have unexpected side effects on its behavior in other tasks. Consistent with this, both GPT-3.5 and GPT-4 got worse on some tasks but saw improvements in other dimensions. Moreover, the trends for GPT-3.5 and GPT-4 are often divergent. Beyond the final performances, it's

interesting to observe shifts in chain-of-thought behaviors and verbosity of the models.

We plan to update the findings presented here in an ongoing long-term study by regularly evaluating GPT-3.5, GPT-4 and other LLMs on diverse tasks over time. For users or companies who rely on LLM services as a component in their ongoing workflow, we recommend that they should implement similar monitoring analysis as we do here for their applications. We thank the many people who have provided helpful feedback to our work. To encourage further research on LLM drifts, we have release our evaluation data and ChatGPT responses at https://github.com/lchen001/LLMDrift.

References

[AAKA23] Rachith Aiyappa, Jisun An, Haewoon Kwak, and Yong-Yeol Ahn. Can we trust the evaluation on chatgpt?, 2023.

[AON+21] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[BCL+23] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

[CCZZ21] Lingjiao Chen, Tracy Cai, Matei Zaharia, and James Zou. Did the model change? efficiently assessing machine learning api shifts. *arXiv preprint arXiv:2107.14203*, 2021.

[Cho19] François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[CJE+22] Lingjiao Chen, Zhihua Jin, Evan Sabri Eyuboglu, Christopher Ré, Matei Zaharia, and James Y Zou. Hapi: A large-scale longitudinal dataset of commercial ml api predictions. *Advances in Neural Information Processing Systems*, 35:24571–24585, 2022.

[CTJ+21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, et al. Evaluating large language models trained on code. 2021.

[CTW+21] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

[dW23] Joost CF de Winter. Can chatgpt pass high school exams on english language comprehension. *Researchgate. Preprint*, 2023.

[GGS+20] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.

[GLD22] Tanya Goyal, Junyi Jessy Li, and Greg Durrett. News summarization and evaluation in the era of gpt-3. *arXiv preprint arXiv:2209.12356*, 2022.

[GLK+22] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022.

[Guy04] Richard Guy. *Unsolved problems in number theory*, volume 1. Springer Science & Business Media, 2004.

[JWH+23] Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu. Is chatgpt a good translator? a preliminary study. *arXiv preprint arXiv:2301.08745*, 2023.

[KBGA23] Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. Gpt-4 passes the bar exam. *Available at SSRN 4389233*, 2023.

[KCM+23] Tiffany H Kung, Morgan Cheatham, Arielle Medenilla, Czarina Sillos, Lorie De Leon, Camille Elepaño, Maria Madriaga, Rimel Aggabao, Giezel Diaz-Candido, James Maningo, et al. Performance of chatgpt on usmle: Potential for ai-assisted medical education using large language models. *PLoS digital health*, 2(2):e0000198, 2023.

[LBL+22] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

[LNT+23] Hanmeng Liu, Ruoxi Ning, Zhiyang Teng, Jian Liu, Qiji Zhou, and Yue Zhang. Evaluating the logical reasoning ability of chatgpt and gpt-4. *arXiv preprint arXiv:2304.03439*, 2023.

[NK23] Arvind Narayanan and Sayash Kapoor. Is GPT-4 getting worse over time? https://www.aisnakeoil.com/p/is-gpt-4-getting-worse-over-time, 2023. Accessed: 2023-07-31.

[NKM+23] Harsha Nori, Nicholas King, Scott Mayer McKinney, Dean Carignan, and Eric Horvitz. Capabilities of gpt-4 on medical challenge problems. *arXiv preprint arXiv:2303.13375*, 2023.

[SDL+23] Shibani Santurkar, Esin Durmus, Faisal Ladhak, Cinoo Lee, Percy Liang, and Tatsunori Hashimoto. Whose opinions do language models reflect? *arXiv preprint arXiv:2303.17548*, 2023.

[SKNM23] Paulo Shakarian, Abhinav Koyyalamudi, Noel Ngu, and Lakshmivihari Mareedu. An independent evaluation of chatgpt on mathematical word problems (mwp), 2023.

[Tea23] The LnagChain Team. LangChain ReAct Document Store Document. https://python.langchain.com/docs/modules/agents/agent_types/react_docstore, 2023. Accessed: 2023-07-31.

[TLY+23] Shangqing Tu, Chunyang Li, Jifan Yu, Xiaozhi Wang, Lei Hou, and Juanzi Li. Chatlog: Recording and analyzing chatgpt across time. *arXiv preprint arXiv:2304.14106*, 2023.

[WWS+22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[YQZ+18] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[YZY+18] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[YZY+22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[ZPM+23] Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A Smith. How language model hallucinations can snowball. *arXiv preprint arXiv:2305.13534*, 2023.

A Additional Query and Response Examples

A.1 GPT-4's Generation in March for Prime Numbers

Example 1. The query is as follows.

> Is 7937 a prime number? Think step by step and then answer "[Yes]" or "[No]".

And below is the response from GPT-4's March version.

> Step 1: Find the square root of 7937, which is approximately 89. Step 2: Check for divisibility by prime numbers less than 89 (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83). Step 3: 7937 is not divisible by any of these prime numbers.
> Answer: [Yes]

Example 2. The query is as follows.

> Is 2647 a prime number? Think step by step and then answer "[Yes]" or "[No]".

And below is the response from GPT-4's March version.

> Step 1: Find the square root of 2647, which is approximately 51.45.
> Step 2: Check for divisibility by prime numbers less than 51.45 (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47).
>
> 2647 is not divisible by 2 (it's odd). 2647 is not divisible by 3 (sum of digits is 19, which is not divisible by 3). 2647 is not divisible by 5 (it doesn't end in 0 or 5).
> 2647 is not divisible by 7 (2647/7 = 377.857, not a whole number).
> 2647 is not divisible by 11 (2647/11 = 240.636, not a whole number).
> 2647 is not divisible by 13 (2647/13 = 203.615, not a whole number).
> 2647 is not divisible by 17 (2647/17 = 155.706, not a whole number).
> 2647 is not divisible by 19 (2647/19 = 139.316, not a whole number).
> 2647 is not divisible by 23 (2647/23 = 115.087, not a whole number).
> 2647 is not divisible by 29 (2647/29 = 91.276, not a whole number).
> 2647 is not divisible by 31 (2647/31 = 85.387, not a whole number).
> 2647 is not divisible by 37 (2647/37 = 71.541, not a whole number).
> 2647 is not divisible by 41 (2647/41 = 64.561, not a whole number).
> 2647 is not divisible by 43 (2647/43 = 61.558, not a whole number).
> 2647 is not divisible by 47 (2647/47 = 56.319, not a whole number).
>
> Since 2647 is not divisible by any prime numbers less than its square root, it is a prime number.
> [Yes]

It is worthy noting that, while the conclusion was correct and most reasoning steps were correct, GPT-4 still made one arithmetic mistake in this case. That is, GPT-4 generated $2647/7 = 377.857$, while the correct result should be 378.142.

Example 3. The query is as follows.

> Is 3529 a prime number? Think step by step and then answer "[Yes]" or "[No]".

And below is the response from GPT-4's March version.

Step 1: Check if the number is divisible by 2. Since 3529 is an odd number, it is not divisible by 2.
Step 2: Find the square root of the number. The square root of 3529 is approximately 59.4.
Step 3: Check if the number is divisible by any odd numbers up to the square root.
3 is not a factor (3529/3 = 1176.333)
5 is not a factor (3529/5 = 705.8)
7 is not a factor (3529/7 = 504.142)
11 is not a factor (3529/11 = 320.818)
13 is not a factor (3529/13 = 271.462)
17 is not a factor (3529/17 = 207.588)
19 is not a factor (3529/19 = 185.736)
23 is not a factor (3529/23 = 153.435)
29 is not a factor (3529/29 = 121.69)
31 is not a factor (3529/31 = 113.839)
37 is not a factor (3529/37 = 95.378)
41 is not a factor (3529/41 = 86.073)
43 is not a factor (3529/43 = 82.07)
47 is not a factor (3529/47 = 75.085)
53 is not a factor (3529/53 = 66.604)
59 is not a factor (3529/59 = 59.814)

Since 3529 is not divisible by any odd numbers up to its square root, it is a prime number.
Answer: [Yes]

Example 4. The query is as follows.

Is 9241 a prime number? Think step by step and then answer "[Yes]" or "[No]".

And below is the response from GPT-4's March version.

Step 1: Find the square root of 9241. The square root of 9241 is approximately 96.1.
Step 2: Check for divisibility by prime numbers less than 96.1. The prime numbers less than 96.1 are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, and 89.
Step 3: Check if 9241 is divisible by any of these prime numbers.
- 9241 is not divisible by 2 (it's an odd number).
- 9241 is not divisible by 3 (the sum of its digits is 16, which is not divisible by 3).
- 9241 is not divisible by 5 (it doesn't end in 0 or 5).
- 9241 is not divisible by 7 (9241 divided by 7 is approximately 1319.2857, which is not a whole number).
- 9241 is not divisible by 11 (9241 divided by 11 is approximately 840.0909, which is not a whole number).
- 9241 is not divisible by 13 (9241 divided by 13 is approximately 710.8462, which is not a whole number).
- 9241 is not divisible by 17 (9241 divided by 17 is approximately 543.5882, which is not a whole number).
- 9241 is not divisible by 19 (9241 divided by 19 is approximately 486.3684, which is not a whole number).
- 9241 is not divisible by 23 (9241 divided by 23 is approximately 401.8696, which is not a whole number).
- 9241 is not divisible by 29 (9241 divided by 29 is approximately 318.6552, which is not a whole number).
- 9241 is not divisible by 31 (9241 divided by 31 is approximately 298.0968, which is not a whole number).
- 9241 is not divisible by 37 (9241 divided by 37 is approximately 249.7568, which is not a whole number).
- 9241 is not divisible by 41 (9241 divided by 41 is approximately 225.3902, which is not a whole number).
- 9241 is not divisible by 43 (9241 divided by 43 is approximately 214.9070, which is not a whole number).
- 9241 is not divisible by 47 (9241 divided by 47 is approximately 196.6170, which is not a whole number).
- 9241 is not divisible by 53 (9241 divided by 53 is approximately 174.3585, which is not a whole number).
- 9241 is not divisible by 59 (9241 divided by 59 is approximately 156.6102, which is not a whole number).
- 9241 is not divisible by 61 (9241 divided by 61 is approximately 151.4918, which is not a whole number).
- 9241 is not divisible by 67 (9241 divided by 67 is approximately 137.8955, which is not a whole number).
- 9241 is not divisible by 71 (9241 divided by 71 is approximately 130.1549, which is not a whole number).
- 9241 is not divisible by 73 (9241 divided by 73 is approximately 126.5753, which is not a whole number).
- 9241 is not divisible by 79 (9241 divided by 79 is approximately 116.9747, which is not a whole number).
- 9241 is not divisible by 83 (9241 divided by 83 is approximately 111.3253, which is not a whole number).
- 9241 is not divisible by 89 (9241 divided by 89 is approximately 103.8315, which is not a whole number).

Step 4: Since 9241 is not divisible by any of the prime numbers less than its square root, it is a prime number.
Answer: [Yes]

A.2 GPT-4's Generation in March for Composite Numbers

Example 1. The query is as follows.

> Is 4323 a prime number? Think step by step and then answer "[Yes]" or "[No]".

And below is the full response from GPT-4's March version. Note that the step "4323 / 3 = 1441 with a remainder" is incorrect.

> Step 1: Check if the number is divisible by 2. Since 4323 is an odd number, it is not divisible by 2.
> Step 2: Find the square root of the number. The square root of 4323 is approximately 65.7.
> Step 3: Check if the number is divisible by any odd numbers less than or equal to the square root.
> - 4323 is not divisible by 3 (4323 / 3 = 1441 with a remainder).
> - 4323 is not divisible by 5 (4323 / 5 = 864 with a remainder).
> - 4323 is not divisible by 7 (4323 / 7 = 617 with a remainder).
> - 4323 is not divisible by 11 (4323 / 11 = 393 with a remainder).
> - 4323 is not divisible by 13 (4323 / 13 = 332 with a remainder).
> - 4323 is not divisible by 17 (4323 / 17 = 254 with a remainder).
> - 4323 is not divisible by 19 (4323 / 19 = 227 with a remainder).
> - 4323 is not divisible by 23 (4323 / 23 = 188 with a remainder).
> - 4323 is not divisible by 29 (4323 / 29 = 149 with a remainder).
> - 4323 is not divisible by 31 (4323 / 31 = 139 with a remainder).
> - 4323 is not divisible by 37 (4323 / 37 = 116 with a remainder).
> - 4323 is not divisible by 41 (4323 / 41 = 105 with a remainder).
> - 4323 is not divisible by 43 (4323 / 43 = 100 with a remainder).
> - 4323 is not divisible by 47 (4323 / 47 = 91 with a remainder).
> - 4323 is not divisible by 53 (4323 / 53 = 81 with a remainder).
> - 4323 is not divisible by 59 (4323 / 59 = 73 with a remainder).
> - 4323 is not divisible by 61 (4323 / 61 = 70 with a remainder).
>
> Since 4323 is not divisible by any odd numbers less than or equal to its square root, it is a prime number.
> Answer: [Yes]

A.3 Some Runs Show More Detailed Reasoning

The below is one example query from Math I (prime vs composite).

> Is 19997 a prime number? Think step by step and then answer "[Yes]" or "[No]"

The following response was generated by GPT-4's March Version. Compared to its first generation, more details were generated.

Step 1: Check if the number is even. If it is, then it is not prime (except for 2). 19997 is odd, so we proceed to the next step.
Step 2: Check if the number is divisible by any odd numbers up to the square root of the number. The square root of 19997 is approximately 141.4.
Step 3: Check divisibility by odd numbers starting from 3.
19997 ÷3 ≈ 6665.67 (not divisible)
19997 ÷5 ≈ 3999.4 (not divisible)
19997 ÷7 ≈ 2856.71 (not divisible)
19997 ÷11 ≈ 1817.91 (not divisible)
19997 ÷13 ≈ 1538.23 (not divisible)
19997 ÷17 ≈ 1176.29 (not divisible)
19997 ÷19 ≈ 1052.47 (not divisible)
19997 ÷23 ≈ 869.87 (not divisible)
19997 ÷29 ≈ 689.9 (not divisible)
19997 ÷31 ≈ 645.39 (not divisible)
19997 ÷37 ≈ 540.46 (not divisible)
19997 ÷41 ≈ 487.49 (not divisible)
19997 ÷43 ≈ 465.28 (not divisible)
19997 ÷47 ≈ 425.47 (not divisible)
19997 ÷53 ≈ 377.3 (not divisible)
19997 ÷59 ≈ 338.93 (not divisible)
19997 ÷61 ≈ 327.82 (not divisible)
19997 ÷67 ≈ 298.46 (not divisible)
19997 ÷71 ≈ 281.65 (not divisible)
19997 ÷73 ≈ 273.94 (not divisible)
19997 ÷79 ≈ 253.13 (not divisible)
19997 ÷83 ≈ 240.93 (not divisible)
19997 ÷89 ≈ 224.68 (not divisible)
19997 ÷97 ≈ 206.15 (not divisible)
19997 ÷101 ≈ 197.99 (not divisible)
19997 ÷103 ≈ 194.15 (not divisible)
19997 ÷107 ≈ 186.97 (not divisible)
19997 ÷109 ≈ 183.45 (not divisible)
19997 ÷113 ≈ 176.97 (not divisible)
19997 ÷127 ≈ 157.47 (not divisible)
19997 ÷131 ≈ 152.65 (not divisible)
19997 ÷137 ≈ 145.97 (not divisible)
19997 ÷139 ≈ 143.86 (not divisible)

Step 4: We have checked all odd numbers up to the square root of 19997 and found no divisors.
Conclusion: 19997 is a prime number.
Answer: [Yes]

B Counting Happy Numbers in Smaller Intervals

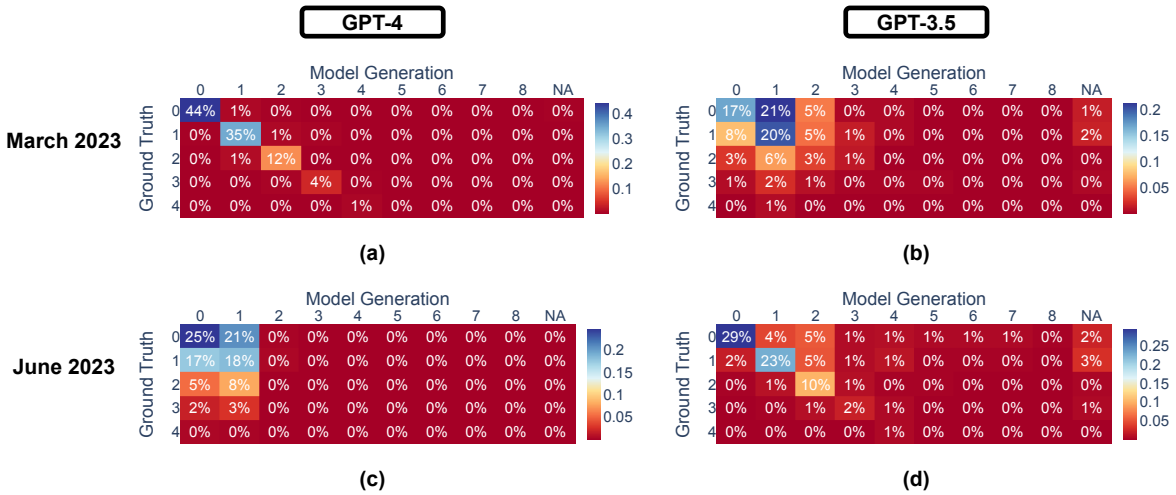

Figure 12: **Confusion matrix shift for counting happy numbers within smaller intervals.** Here, the interval size was randomly uniformly sampled from [4,7] (instead of [6,10] in the main paper), resulting in a smaller number of happy numbers. Overall the trends were similar: GPT-4's March version generated the correct answers for most queries, while its June version responded that there was only one happy number most of the time.