Towards Contamination Resistant Benchmarks

Anonymous ACL submission

Abstract

The rapid development of large language models (LLMs) has transformed the landscape of natural language processing. Evaluating LLMs properly is crucial for understanding their potential and addressing concerns such as safety. However, LLM evaluation is confronted by various factors, among which contamination stands out as a key issue that undermines the reliability of evaluations. In this work, we introduce the concept of *contamination resistance* to address this challenge. We propose a benchmark based on Caesar ciphers (e.g., "ab" → "bc" when the shift is 1), which, despite its simplicity, is an excellent example of a contamination resistant benchmark. We test this benchmark on widely used LLMs under various settings, and we find that these models struggle with this benchmark when contamination is controlled. Our findings reveal issues in current LLMs and raise important questions regarding their true capabilities. Our work contributes to the development of contamination resistant benchmarks, enabling more rigorous LLM evaluation and offering insights into the true capabilities and limitations of LLMs.[1]

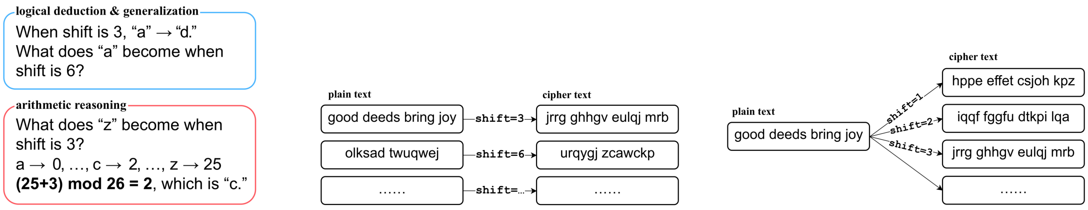
1 Introduction

The advent of large language models (LLMs) has largely changed the field of natural language processing and many facets of daily life (Brown et al., 2020; Chung et al., 2022; Chowdhery et al., 2023; Dubey et al., 2024). Extensive research has asserted that these models possess "human-like abilities" such as reasoning (Kojima et al., 2022; Wei et al., 2022a,b; Bubeck et al., 2023; Hagendorff et al., 2023; Itzhak et al., 2024; Xie et al., 2024). However, LLM evaluation is often confronted by various factors, and contamination, also known as test set contamination or data leakage, stands out as a key issue. Given that LLMs are trained on vast web corpora, there is concern that their apparent reasoning skills are superficial, and that they are merely retrieving memorized information from their training data (Sainz et al., 2023; Dong et al., 2024; Jiang et al., 2024; Li and Flanigan, 2024; Ravaut et al., 2024; White et al., 2024).

A key strategy to address contamination is to create a dynamic benchmark that receives continuous updates, with new test instances added after a model's training cutoff (Jain et al., 2024; Roberts et al., 2024; Shabtay et al., 2024; White et al., 2024; Zhang et al., 2024; Mahdavi et al., 2025). The effectiveness of this method rests on a critical assumption: the models in question have not been trained on these newly added instances. However, given the black box nature of current LLMs, especially commercial ones, it is almost impossible to determine whether developers have continued training their models after the cutoff using the updated instances, and, if so, when this occurred. As a result, the reliability and validity of the newly added data still remain questionable. Furthermore, creating new instances can be laborious, and maintaining a consistent level of difficulty between old and new instances is challenging, which is crucial for preserving the fairness of model performance comparisons across old and new instances.

Based on these considerations, we propose the concept of *contamination resistance*. We argue that a contamination resistant benchmark should meet the following criteria: (a) it should assess certain capabilities of LLMs; (b) it should be dynamic, evolving over time to prevent contamination, and ensure that contamination provides no advantage in model performance; (c) while being dynamic, it should consistently assess the same set of capabilities and maintain the same level of task complexity to ensure fairness in evaluation over time; (d) the effort required to curate new instances should be minimal, making it more update-able than existing static benchmarks.

[1] Our code and data are available at `https://anonymous.4open.science/r/contamination-resistant-E0D4`.



(a) capabilities (b) dynamism (c) update-ability

Figure 1: We propose the concept of *contamination resistance* and establish a contamination resistant benchmark based on Caesar ciphers, which meets the following criteria: (a) this benchmark tests several capabilities, including logical deduction, arithmetic reasoning, and generalization; (b) this benchmark is dynamic, evolving over time so that an LLM can not memorize all possible queries, and the level of task complexity is consistent; (c) it is effortless to curate new test instances, making it more update-able than static benchmarks.

We show that a benchmark based on Caesar ciphers is an excellent example of a contamination resistant benchmark (see Figure 1). The Caesar cipher is a simple substitution cipher where each letter in a text is shifted some places down or up the alphabet. Encoding or decoding a Caesar cipher demands multiple capabilities, including logical deduction, arithmetic reasoning, and generalization. This benchmark is dynamic: users can generate an infinite number of unique texts, and it is unlikely that an LLM can memorize all possible queries. Despite its dynamic nature, this benchmark consistently evaluates the same underlying abilities and maintains the same level of task complexity. Moreover, this benchmark is extremely light weight. It requires little effort to generate new test instances and to perform inference.

Despite that solving a Caesar cipher is essentially a simple linear mapping task, and that the state-of-the-art LLMs perform exceptionally well on benchmarks that require much more complex reasoning abilities, they struggle with the Caesar cipher. We summarize our findings as **inconsistent competence of LLMs**, which add to the growing evidence that when contamination is controlled, LLMs are not performing as well as expected, highlighting the issue of contamination. Our findings also align with existing evidence of hallucination and the lack of generalization in LLMs.

Our contributions are summarized as follows:

- We propose the concept of *contamination resistance* and define the criteria a contamination resistant benchmark should meet.
- To establish a contamination resistant benchmark, we propose a benchmark based on Caesar ciphers and show that LLMs struggle with them when contamination is controlled.
- Our work paves the way for the development of contamination resistant benchmarks, enables more rigorous evaluation of LLMs, and sheds light on their true capabilities.

2 Related work

2.1 Model capabilities

The exceptional performance of LLMs on various tasks has led to claims that these models have gained “human-like abilities”, such as reasoning (Kojima et al., 2022; Wei et al., 2022a,b; Bubeck et al., 2023; Saparov and He, 2023; Shi et al., 2023; Webb et al., 2023). However, evidence that contradicts these claims exists, particularly concerning the phenomenon of hallucination (Bang et al., 2023; McKenna et al., 2023; Mündler et al., 2023; Zhang et al., 2023a). Schaeffer et al. (2023) argue that LLMs’ abilities appear due to the choice of metric rather than fundamental changes in model behavior with scale. Another line of research suggests that LLMs rely more on memory than genuine reasoning abilities to solve certain tasks (Reynolds and McDonell, 2021; Merullo et al., 2023; Wang et al., 2023; Zheng et al., 2023; Li et al., 2024; Lu et al., 2024b). Štefánik and Kadlčík (2023) introduce a conceptual learning method aimed at disentangling models’ in-context learning ability from memorization. Their findings indicate that models rely heavily on their pre-trained knowledge than benefiting from the in-context concepts.

Whether LLMs truly possess “human-like abilities” has now gone beyond a technical problem. It affects how users interact with these models and has significant safety implications (Lu et al., 2024a; Bengio et al., 2024). By proposing the concept of contamination resistance, this work sheds light on the true capabilities of LLMs and helps promote better interaction between humans and LLMs.

2.2 Model evaluation

Evaluating LLMs properly is crucial to understand their potential as well as addressing concerns such as safety. The proliferation of benchmarks has led to a multifaceted evaluation process that covers a variety of abilities (Rajpurkar et al., 2018; Wang et al., 2019; Cobbe et al., 2021; Srivastava et al., 2023). However, LLM evaluation is confronted by various factors, among which contamination stands out as one of the key issues. Contamination has become increasingly prominent given that current LLMs are trained on massive web corpora and are scaled up to billions of parameters. Furthermore, investigating potential data leakage is challenging because the training data are often closed source. These factors have severely undermined the reliability of evaluations (Sainz et al., 2023; Ballocu et al., 2024; Dong et al., 2024; Jiang et al., 2024; Ravaut et al., 2024; White et al., 2024). A growing body of research has revealed that LLMs perform better on problems that released before their training cutoff, and that some LLMs are overfitted to the testing sets of popular reasoning benchmarks (Eisenschlos et al., 2023; Li and Flanigan, 2024; Roberts et al., 2024; Zhang et al., 2024).

A straight forward approach to address contamination is to ensure that the LLM has never been exposed to the evaluation data during training. Chandran et al. (2024) introduce the notion of private benchmarking, where the evaluation data remain confidential and only the evaluation results are revealed. However, this approach may raise concerns regarding transparency and reproducibility. Another line of work focuses on dynamic benchmarking which features frequently updated questions from recent information sources introduced after a model's training cutoff (Jain et al., 2024; Shabtay et al., 2024; White et al., 2024; Mahdavi et al., 2025). A key assumption of this method is that the model has not been trained on data beyond the cutoff, which may not hold as model developers may continue training their models on newer information sources after the cutoff. To address these concerns, we propose the concept of contamination resistance, and we show how it contributes to more reliable LLM evaluations.

3 A contamination resistant benchmark based on Caesar ciphers

The Caesar cipher is a substitution cipher in which each letter is shifted certain places down or up the alphabet. Solving a Caesar cipher requires several capabilities, including the following:

- **Logical deduction:** the ability to deduce, for example, "a" $\rightarrow$ "g" when `shift=6`, given the knowledge that "a" $\rightarrow$ "d" when `shift=3`.
- **Arithmetic reasoning:** the ability to calculate a letter's position given a shift.
- **Generalization:** the ability to extend the capability of solving, for example, `shift=3` ciphers, to solving ciphers with other shifts.

We refer to the text used for encoding a Caesar cipher as plain text, and the encoded text as cipher text. Based on the observation that LLM performance is influenced by the probability of the task and the probability of the input/output in their pre-training data (McCoy et al., 2024a,b), we curated the benchmark using ciphers with different shifts (i.e., different tasks) and different types of plain texts that consist of natural language English words (**natural**) and random non-sense words (**random**) (i.e., different inputs/outputs). Table 10 in Appendix A shows examples of the data used in the benchmark. In practice, words from any alphabetical language can be used as the plain text for the Caesar cipher. Even a made-up alphabet can be used, provided that it is informed in the prompt. Caesar ciphers are dynamic, as they allow the generation of an infinite number of instances.

The dynamic nature of the data is a crucial aspect of this benchmark that makes it contamination resistant. It is also important to note that regardless of the shifts and types of plain text used to generate the cipher, the capabilities required to solve the Caesar cipher remain the same. Intuitively, it should be easy to generalize from, for example, solving the Caesar cipher with `shift=3` to solving one with `shift=6`. Given the linear mapping nature of the Caesar cipher, solving ciphers with plain texts consisting of random words is not more challenging than solving those with plain texts in natural language English words. Furthermore, this benchmark is lightweight–it is easy to generate a large number of ciphers, and since the ciphers are typically not lengthy, it does not cost much time during inference.

We have 25 plain texts in natural language English words and 25 in random non-sense words. With 4 shift values $[3, 6, 9, 12]$, this results in a total of 100 data for each type of plain text. See Table 10 for examples of the data in our benchmark.

4 Experimental setup

Model We tested widely used LLMs on our benchmark: OpenAI GPT-4o[2], LLaMA3.1-8B/70B (Dubey et al., 2024), Qwen2.5-7B/32B (Yang et al., 2024), and QwQ-32B. These models cover a range of sizes and different architectures. All of the models "know" what the Caesar cipher is (see Table 12). We utilize the OpenAI API, making requests without employing any external tools.

type	prompt
`open`	[Encode/Decode] the following [text to a Caesar cipher/Caesar cipher text]. The shift is `$#$`.
`base`	[Encode/Decode] the following [text to a Caesar cipher/Caesar cipher text]. The shift is `$#$`. Output the cipher text only.
`dict`	[Encode/Decode] the following [text to a Caesar cipher/Caesar cipher text]. The shift is `$#$`. Output a lookup table and the cipher text in a Python dictionary: `{"lookup_table": {}, "cipher_text":..}`. Output the dictionary only.
`code`	[Encode/Decode] the following [text to a Caesar cipher/Caesar cipher text]. The shift is `$#$`. Write a Python function and generate the answer. Output the function and the cipher text only.

Table 1: The prompts used in our experiments. The number of shifts (`$#$`) is given in all the prompts.

Prompt Table 1 shows the prompts used in our experiments. The `open` prompt is a straight forward one that most users may use initially. It is open-ended, with no specifications regarding the output format. The `base` prompt serves as a baseline where the model is instructed to generate only the cipher text or plain text. The `dict` prompt instructs the model to produce a lookup table as an intermediate reasoning step before generating the answer. The `code` prompt instructs the model to write a function and then generate the answer. Intuitively, using programming code is an effective approach for solving the Caesar cipher, as it is essentially a linear mapping task. The number of shifts is given in all the prompts. We keep the prompts in their simplest form, specifying only the task (encoding or decoding), the number of shifts, and the desired output format. Given the simplicity of the Caesar cipher, we believe these prompts are adequate for solving it. All the prompts except for `open` specify the output format for easier processing.

Evaluation We evaluate the output using exact match accuracy and character error rate. The character error rate is a more continuous metric, allowing us to capture any partial improvements in the performance of the models. We tested our benchmark under both the zero-shot and few-shot settings (see Table 11 for the few-shot prompts we used). As we do not specify the output format in `open`, we manually extracted the final answer from the generated text to assess performance.

See Appendix B for more details regarding the experimental setup.

5 Results

Table 2 shows the overall results of the experiments. GPT-4o is the only model that shows some success in certain cases, whereas all the other models fail to solve this task almost entirely. The performance of GPT-4o shows a strong negative correlation between the two metrics ($r = -0.6395$). The comparison of character error rates between LLaMA3.1-8B and LLaMA3.1-70B, as well as Qwen2.5-7B and Qwen2.5-32B, indicates that larger models have advantages over their smaller counterparts.

Though most models fail on this benchmark, they still generate reasoning chains that appear correct, which reflects the issue of hallucination.

In the following sections, we analyze the performance of LLMs on our benchmark across different variables and show its resistance to contamination. We summarize our findings as **inconsistent competence in LLMs**. Our discussion focuses on GPT-4o since it is the only model that achieves some success in exact match accuracy.

5.1 The effect of shift

Table 2 shows clear inconsistencies in model performance between `shift=3` and other shifts regardless of the shift, type of plain text, or prompt. We attribute these inconsistencies to contamination, as Caesar ciphers with `shift=3` are more prevalent in the web corpora than those with other shifts (McCoy et al., 2024a) (see also Table 12). LLMs exhibit **inconsistent competence across tasks**, which indicates that they lack generalization in these tasks.

When using the `open` prompt, GPT-4o outputs a reasoning chain that verbalizes all the letter mappings regardless of the shift (see Table 14 for an example). However, this occurs only in the decoding task[3], suggesting that the model was trained

[2]GPT-4o results were obtained on Dec 30, 2024.

[3]In the encoding task, GPT-4o outputs only the final answer without reasoning chains.

type	model	`shift=3`			`shift=6`			`shift=9`		
		`open`	`base`	`dict`	`open`	`base`	`dict`	`open`	`base`	`dict`
	GPT-4o	0.8200	0.6400	0.8000	0.4000	0.1100	0.1000	0.3400	0.0400	0.0200
	LLaMA3.1-8B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
natural	LLaMA3.1-70B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Qwen2.5-7B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Qwen2.5-32B	0.1400	0.0400	0.0200	0.1000	0.0000	0.0000	0.0000	0.0000	0.0000
	GPT-4o	0.1600	0.1000	0.0600	0.2000	0.0000	0.0000	0.1800	0.0200	0.0200
	LLaMA3.1-8B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
random	LLaMA3.1-70B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Qwen2.5-7B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Qwen2.5-32B	0.0400	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
AVERAGE (w/o GPT-4o)		0.0225	0.0050	0.0025	0.0125	0.0000	0.0000	0.0000	0.0000	0.0000

(a) exact match accuracy↑

type	model	`shift=3`			`shift=6`			`shift=9`		
		`open`	`base`	`dict`	`open`	`base`	`dict`	`open`	`base`	`dict`
	GPT-4o	0.0430	0.1234	0.0802	0.2172	0.4440	0.4058	0.2357	0.5648	0.5501
	LLaMA3.1-8B	0.8709	0.9076	0.9385	0.8932	0.9288	0.9648	0.9117	0.9326	0.9584
natural	LLaMA3.1-70B	0.7208	0.7756	0.8202	0.8692	0.9056	0.9232	0.9114	0.9172	0.9112
	Qwen2.5-7B	0.7972	0.8338	0.8577	0.8462	0.9052	0.9198	0.9161	0.9196	0.9176
	Qwen2.5-32B	0.5630	0.6714	0.7152	0.5614	0.7873	0.8018	0.7965	0.8516	0.8307
	GPT-4o	0.2414	0.3082	0.2490	0.2561	0.3892	0.3545	0.2747	0.4965	0.4330
	LLaMA3.1-8B	0.8554	0.8868	0.9208	0.8880	0.9104	0.9506	0.9398	0.9068	0.9341
random	LLaMA3.1-70B	0.7931	0.8564	0.8493	0.8262	0.9067	0.8742	0.8563	0.9189	0.8957
	Qwen2.5-7B	0.7765	0.8303	0.8385	0.8437	0.8744	0.8968	0.8969	0.8902	0.9116
	Qwen2.5-32B	0.4830	0.6632	0.6923	0.5995	0.7773	0.7878	0.7799	0.8285	0.8374
AVERAGE (w/o GPT-4o)		0.7325	0.8031	0.8290	0.7909	0.8745	0.8899	0.8761	0.8957	0.8996

(b) character error rate↓

Table 2: Exact match accuracy and character error rate. The rest of the results can be found in Table 13.

primarily on decoding tasks. While verbalization generally leads to better performance (see Table 3), we observe cases where the generated reasoning chain is correct but the final answer is incorrect (see Table 15).

task	type	`shift=3`	`shift=6`	`shift=9`
decoding	**natural**	1.00	0.80	0.68
	random	0.28	0.40	0.32
encoding	**natural**	0.64	0.00	0.00
	random	0.04	0.00	0.04

Table 3: Exact match accuracy; `open`; GPT-4o; the zero-shot setting. Verbalization is only observed in the decoding task.

5.2 The effect of plain text

Table 2 and 3 indicate that using random non-sense words as plain text diminishes GPT-4o's performance on the task, even when `shift=3` where the model performs well with natural language English words as plain text. This could be attributed to contamination, as the model may have memorized the mappings of certain natural language words to their corresponding cipher texts. This is further supported by the observation that GPT-4o performs much better with natural language English words than with random non-sense words as plain text when `shift=3` with the `base` prompt, which instructs the model to generate only the final answer with no reasoning chains. The simplest explanation for this performance gap is that the model has memorized this mappings.

GPT-4o exhibits **inconsistent competence across different types of plain text**, which again indicates a lack of generalization.

5.3 The effect of prompt

Table 2 suggests that GPT-4o performs best with the `open` prompt. As noted earlier, GPT-4o employs a verbalization strategy, explicitly listing all letter mappings in a reasoning chain, which improves its performance. The same strategy is ob-

type	model	legal rate↑	accuracy↑
natural	GPT-4o	0.9900	1.0000
	LLaMA3.1-8B	0.4550	0.4400
	LLaMA3.1-70B	0.4750	0.1100
	Qwen2.5-7B	0.4200	0.1550
	Qwen2.5-32B	0.8750	0.6100
random	GPT-4o	0.9750	1.0000
	LLaMA3.1-8B	0.5900	0.1600
	LLaMA3.1-70B	0.7900	0.4250
	Qwen2.5-7B	0.5650	0.1900
	Qwen2.5-32B	0.9400	0.5950

Table 4: Legal rate and accuracy of lookup tables generated using `dict`; the zero-shot setting.

served in Qwen2.5-32B, which also shows some success when `shift=3` with the open prompt.

Compared to the base prompt, only GPT-4o benefits from the `dict` prompt, showing improvement in both exact match accuracy and character error rate. For the other models, `dict` does not lead to a lower character error rate in general.

We further examine the correctness of the lookup tables models generated using `dict`. Specifically, we look into the legal rate (i.e., whether the lookup table is a legitimate Python dictionary) and accuracy (i.e., whether the mappings of the letters in the lookup table are correct) of the lookup table. For GPT-4o, we specified `json_object` as the output format in the API.

As shown in Table 4, GPT-4o, and Qwen2.5-32B are capable of generating lookup tables that are correct in both format and content, and it is independent of the type of plain text. There is a huge discrepancy between the correctness of the lookup tables and the accuracy of the final outputs shown in Table 2. Though a lookup table intuitively serves as an intermediate reasoning chain equivalent to the one produced by the open prompt, GPT-4o performs worse with `dict` than with open. This suggests that reasoning chains in the form of a lookup table are less effective than those in natural language. It is possible that the generated lookup tables are not the result of reasoning but rather something the model memorizes. Our findings suggest **an inconsistency in LLM competence between generating lookup tables and final answers**.

We only experimented with the code prompt using GPT-4o. The code prompt does not lead to superior performance compared to the other prompts (see Figure 2). Like other prompts, it achieves better performance when the plain text is natural language English words. To examine the correctness of the code generated by the model, we execute the code and evaluate the accuracy of the output. We find that all the generated code runs successfully and produces correct outputs regardless of the type of plain text and shift. Our findings suggest that there is **an inconsistency in LLM competence between generating code and final answers**. Again, this suggests that the model may have simply memorized the code rather than performing true reasoning.

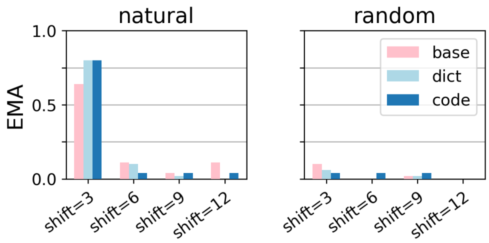


Figure 2: Exact match accuracy (**EMA**); GPT-4o; the zero-shot setting.

5.4 The effect of output position

Table 5 shows the exact match accuracy for the first, second, and third characters in the outputs. For all models, the accuracy is much higher for the initial characters compared to the full-sequence accuracy shown in Table 2. Though plain texts consisting of natural language English words lead to better performance than those with random non-sense words, the impact of plain text type is less prominent than in Table 2. There is a noticeable decline in accuracy from the first character to the third character, suggesting that models **exhibit inconsistent competence in handling characters in different positions**. This also implies the autoregressive nature of LLMs, that models generate higher-probability tokens even in deterministic tasks like solving the Caesar cipher, where the probability should not matter (McCoy et al., 2024a).

Table 6 shows the accuracy of the initial characters generated by GPT-4o using open, the best performing prompt. Notably, when `shift=3` and the plain text is in natural language English words, GPT-4o maintains consistent accuracy across different character positions, which is not observed in any other cases. Additionally, GPT-4o's ability to decode or encode the second and third characters (**`char_id=1`** and **`char_id=2`**) when `shift=3` depends on the type of plain text. This indicates that the model may memorize the plain-text-to-

type	model	id=0	id=1	id=2
natural	GPT-4o	0.8717	0.7467	0.6483
	LLaMA3-8B	0.1100	0.1467	0.0767
	LLaMA3-70B	0.2650	0.2367	0.1400
	Qwen2.5-7B	0.2817	0.2333	0.1100
	Qwen2.5-32B	0.4900	0.3267	0.2100
random	GPT-4o	0.8350	0.6833	0.5167
	LLaMA3-8B	0.0733	0.1267	0.0683
	LLaMA3-70B	0.1800	0.1583	0.0733
	Qwen2.5-7B	0.2350	0.1517	0.1383
	Qwen2.5-32B	0.4250	0.2500	0.1683

Table 5: Exact match accuracy of the first (**id=0**), second (**id=1**), and third output character (**id=2**); the zero-shot setting. LLaMA3 is for LLaMA3.1.

cipher mappings specifically for shift=3 and natural language English plain texts.

shift	char_id=0	char_id=1	char_id=2
3	0.98	1.00	1.00
6	0.92	0.96	0.82
9	0.92	0.88	0.76
12	0.84	0.80	0.78

(a) **natural**

shift	char_id=0	char_id=1	char_id=2
3	0.94	0.82	0.76
6	0.98	0.88	0.76
9	0.98	0.82	0.72
12	0.92	0.84	0.64

(b) **random**

Table 6: Exact match accuracy of the first (**char_id=0**), second (**char_id=1**), and third output character (**char_id=2**); open; GPT-4o; the zero-shot setting.

5.5 The effect of shot

Given that LLMs demonstrate success in in-context learning and can learn simple functions in-context at inference time (Brown et al., 2020; Garg et al., 2022; Akyürek et al., 2023; Zhang et al., 2023b) and that the Caesar cipher is essentially a linear mapping task, state-of-the-art LLMs should be capable of solving the Caesar cipher with few-shot prompts. We include 50 demonstrations in the prompts (see Table 11). These demonstrations have already covered the mappings for all letters in the English alphabet. We tested GPT-4o to encode or decode cipher texts with shift=[6,9,12] using the base prompt, since it does not perform in the zero-shot setting under these conditions (see Table 2). Contrary to intuition, GPT-4o does not benefit from the few-shot demonstrations, as shown in Table 7 and 8. We observe almost no improvement in exact match accuracy and character error rate for the full sequence, and there is only a slight improvement in GPT-4o's performance on the second and third characters.

task	EMA↑		CER↓	
	zero	few	zero	few
decoding	0.1700	0.2400	0.5521	0.5848
encoding	0.1700	0.0800	0.2961	0.4011

Table 7: Exact match accuracy (**EMA**) and character error rate (**CER**); natural language English words; shift=[6,9,12]; base; GPT-4o; the zero-shot setting (**zero**) and few-shot setting (**few**).

task	id=1		id=2	
	zero	few	zero	few
decoding	0.5300	0.5467	0.4800	0.4400
encoding	0.8600	0.9067	0.7400	0.7467

Table 8: Exact match accuracy of the first (**id=1**) and second (**id=2**); natural language English words; shift=[6,9,12]; base; GPT-4o; the zero-shot setting (**zero**) and few-shot setting (**few**).

Given existing evidence that few-shot learning resembles fine-tuning (Dai et al., 2023) and the undesired few-shot performance shown in Table 7 and 8, fine-tuning models on Caesar ciphers may not be a viable solution for improving performance. Moreover, it is unclear whether fine-tuning on a large number random non-sense words would affect model performance on other tasks.

5.6 Advanced reasoning models

Our findings have suggested a clear strategy for solving the Caesar cipher, i.e., the verbalization strategy, where every relevant letter mapping is explicitly verbalized before arriving at the final answer.

Recent advanced reasoning models that have further strengthened the verbalization strategy. These models leverage test-time scaling, which dynamically allocates computational resources during inference to refine intermediate reasoning steps and improve answer accuracy (Akyürek et al., 2024; Snell et al., 2024). They have demonstrated exceptional performance on tasks that were unsolvable for previous models.

We experimented with QwQ-32B with the open prompt. As Table 9 shows, QwQ-32B shows some improvement on the encoding task with natural language English plain texts compared to Qwen2.5-32B, suggesting the effectiveness of test-time scaling. However, the model still fails on random non-sense word plain texts. In the decoding task, QwQ-32B frequently questions its own answer (because the decoded texts are random non-sense words, which it perceives as incorrect) and fails to provide a final answer within the token limit.

type	task	`shift=3`	`shift=6`	`shift=9`
natural	encoding	0.20	0.08	0.08
random	decoding	0.00	0.00	0.00
random	encoding	0.04	0.00	0.00

(a) QwQ-32B

type	task	`shift=3`	`shift=6`	`shift=9`
natural	encoding	0.00	0.00	0.00
random	decoding	0.08	0.00	0.00
random	encoding	0.00	0.00	0.00

(b) Qwen2.5-32B

Table 9: Exact match accuracy; QwQ-32B and Qwen2.5-32B; the zero-shot setting.

As we write this paper, a new DeepSeek-R1 (R1) has been released and captured public attention. However, its server is currently very unstable, preventing us from conducting a comprehensive experiment. At the moment, we only tested the query shown in Table 15, which GPT-4o failed to solve correctly.[4] The plain text in that query consists of random non-sense words, which are shown to be the most challenging for the models we tested.

R1 successfully solves the decoding task. Notably, the inference time is quite long and the intermediate reasoning chain is lengthy (see here for the complete output). Similar to QwQ-32B, R1 frequently questions its own answer because the output "does not make sense in English."

In the encoding task, however, R1 fails to produce the correct answer. Since we currently cannot set parameters such as `seed` or `temperature` to ensure reproducibility, we performed inference 5 times, and R1 failed in all of them. **We encourage you to test the query yourself on the DeepSeek Platform using DeepThink (R1):**

> *Encode the following text to a Caesar cipher. The shift is 3. rakibo zlmqwe.*

[4] R1 results were obtained on Feb 14, 2025.

The correct answer is *udnler coptzh*, and the 5 outputs we obtained are: *UDNLER COPTHZ*, *udnler copszh*, *UDNELR COPTZH*, *udnler copthz*, *udnelr coptzh* (we highlighted the incorrect parts in the output, and see here for the complete outputs). Similar to the GPT-4o example in Table 15, R1 is able to generate correct intermediate reasoning chains but fail to produce the correct final answer. It seems that R1, despite being a state-of-the-art advanced reasoning model, still shows inconsistencies in solving the Caesar cipher.

5.7 Integrity check

We did not specify the direction of the shift in the prompts (see Table 1). We intend to use the default rules: for encoding, the shift is rightward (e.g., "a" → "b"), and for decoding, it is leftward (e.g., "b" → "a"). However, models may not adhere to this setting. To address this, we conduct an addition set of evaluations where we relax our evaluation criterion by allowing two ground truths for each query. An output is considered correct if it matches either ground truth. The results show that this relaxed evaluation criterion yields the same exact match accuracy as those in Table 2. This suggests that the models follow the default rules even when the prompt does not explicitly specify them.

6 Conclusion

In this work, we propose the concept of *contamination resistance*. To establish a contamination resistant benchmark, we create a benchmark using Caesar ciphers. Though it is a simple task, we demonstrate how it resists contamination by revealing inconsistencies in model competence. We show that when contamination is controlled, model performance on certain tasks drops greatly or even be completely diminished. Apart from contamination, our findings align with other issues of LLMs, including hallucination and lack of generalization.

We would like to emphasize that this paper does not aim to find a solution for the Caesar cipher – it is a simple task that can be solved in a few lines of code. Instead, using this simple task, we highlight the necessity of a contamination resistant benchmark and demonstrate how it contributes to more reliable evaluations of LLMs and raises important questions regarding their true capabilities.

Limitations

Due to its closed source nature, the reproducibility of the results related to OpenAI GPT-4o is a concern. Reproducibility is also an issue for DeepSeek R1, as we used the DeepSeek Platform and we currently cannot set parameters such as `seed` and `temperature`, and its server is highly unstable at the moment.

We did not test OpenAI o1 due to its high cost. A single inference costs around $1.5, which exceeds our budget. Additionally, we cannot specify parameters to ensure reproducibility, which further limits its usability.

Ethics Statement

There are no ethical concerns associated with this work.

Acknowledgements

References

Ekin Akyürek, Mehul Damani, Linlu Qiu, Han Guo, Yoon Kim, and Jacob Andreas. 2024. The surprising effectiveness of test-time training for abstract reasoning. *CoRR*, abs/2411.07279.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2023. What learning algorithm is in-context learning? investigations with linear models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondrej Dusek. 2024. Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source llms. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - Volume 1: Long Papers, St. Julian's, Malta, March 17-22, 2024*, pages 67–93. Association for Computational Linguistics.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP 2023 -Volume 1: Long Papers, Nusa Dua, Bali, November 1 - 4, 2023*, pages 675–718. Association for Computational Linguistics.

Yoshua Bengio, Geoffrey Hinton, Andrew Yao, Dawn Song, Pieter Abbeel, Trevor Darrell, Yuval Noah Harari, Ya-Qin Zhang, Lan Xue, Shai Shalev-Shwartz, et al. 2024. Managing extreme ai risks amid rapid progress. *Science*, 384(6698):842–845.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott M. Lundberg, Harsha Nori, Hamid Palangi, Marco Túlio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with GPT-4. *CoRR*, abs/2303.12712.

Nishanth Chandran, Sunayana Sitaram, Divya Gupta, Rahul Sharma, Kashish Mittal, and Manohar Swaminathan. 2024. Private benchmarking to prevent contamination and improve comparative evaluation of llms. *CoRR*, abs/2403.00393.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24:240:1–240:113.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, et al. 2022. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023. Why can GPT learn in-context? language models secretly perform gradient descent as meta-optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4005–4019, Toronto, Canada. Association for Computational Linguistics.

Yihong Dong, Xue Jiang, Huanyu Liu, Zhi Jin, Bin Gu, Mengfei Yang, and Ge Li. 2024. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 12039–12050, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, et al. 2024. The llama 3 herd of models.

Julian Martin Eisenschlos, Jeremy R. Cole, Fangyu Liu, and William W. Cohen. 2023. Winodict: Probing language models for in-context word acquisition. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 94–102. Association for Computational Linguistics.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. 2022. What can transformers learn in-context? A case study of simple function classes. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Thilo Hagendorff, Sarah Fabi, and Michal Kosinski. 2023. Human-like intuitive behavior and reasoning biases emerged in large language models but disappeared in chatgpt. *Nature Computational Science*, 3(10):833–838.

Itay Itzhak, Gabriel Stanovsky, Nir Rosenfeld, and Yonatan Belinkov. 2024. Instructed to Bias: Instruction-Tuned Language Models Exhibit Emergent Cognitive Bias. *Transactions of the Association for Computational Linguistics*, 12:771–785.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *CoRR*, abs/2403.07974.

Minhao Jiang, Ken Liu, Ming Zhong, Rylan Schaeffer, Siru Ouyang, Jiawei Han, and Sanmi Koyejo. 2024. Does data contamination make a difference? insights from intentionally contaminating pre-training data for language models. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Changmao Li and Jeffrey Flanigan. 2024. Task contamination: Language models may not be few-shot anymore. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 18471–18480. AAAI Press.

Zihao Li, Yucheng Shi, Zirui Liu, Fan Yang, Ninghao Liu, and Mengnan Du. 2024. Quantifying multilingual performance of large language models across languages. *CoRR*, abs/2404.11553.

Sheng Lu, Irina Bigoulaeva, Rachneet Sachdeva, Harish Tayyar Madabushi, and Iryna Gurevych. 2024a. Are emergent abilities in large language models just in-context learning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5098–5139, Bangkok, Thailand. Association for Computational Linguistics.

Sheng Lu, Hendrik Schuff, and Iryna Gurevych. 2024b. How are prompts different in terms of sensitivity? In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5833–5856, Mexico City, Mexico. Association for Computational Linguistics.

Sadegh Mahdavi, Muchen Li, Kaiwen Liu, Christos Thrampoulidis, Leonid Sigal, and Renjie Liao. 2025. Leveraging online olympiad-level math problems for llms training and contamination-resistant evaluation. *arXiv preprint arXiv:2501.14275*.

R. Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D. Hardy, and Thomas L. Griffiths. 2024a. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, 121(41):e2322420121.

R. Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D. Hardy, and Thomas L. Griffiths. 2024b. When a language model is optimized for reasoning, does it still show embers of autoregression? an analysis of openai o1. *CoRR*, abs/2410.01792.

Nick McKenna, Tianyi Li, Liang Cheng, Mohammad Hosseini, Mark Johnson, and Mark Steedman. 2023. Sources of hallucination by large language models on inference tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2758–2774, Singapore. Association for Computational Linguistics.

Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. 2023. Language models implement simple word2vec-style vector arithmetic. *CoRR*, abs/2305.16130.

Niels Mündler, Jingxuan He, Slobodan Jenko, and Martin T. Vechev. 2023. Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation. *CoRR*, abs/2305.15852.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 784–789. Association for Computational Linguistics.

Mathieu Ravaut, Bosheng Ding, Fangkai Jiao, Hailin Chen, Xingxuan Li, Ruochen Zhao, Chengwei Qin, Caiming Xiong, and Shafiq Joty. 2024. How much are llms contaminated? A comprehensive survey and the llmsanitize library. *CoRR*, abs/2404.00699.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama Japan, May 8-13, 2021, Extended Abstracts*, pages 314:1–314:7. ACM.

Manley Roberts, Himanshu Thakur, Christine Herlihy, Colin White, and Samuel Dooley. 2024. A natural experiment on LLM data contamination in code generation. In *I Can't Believe It's Not Better Workshop: Failure Modes in the Age of Foundation Models*.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. 2023. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 10776–10787. Association for Computational Linguistics.

Abulhair Saparov and He He. 2023. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2023. Are emergent abilities of large language models a mirage? In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Nimrod Shabtay, Felipe Maia Polo, Sivan Doveh, Wei Lin, Muhammad Jehanzeb Mirza, Leshem Choshen, Mikhail Yurochkin, Yuekai Sun, Assaf Arbelle, Leonid Karlinsky, and Raja Giryes. 2024. Livexiv - A multi-modal live benchmark based on arxiv papers content. *CoRR*, abs/2410.10783.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2023. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

Michal Štefánik and Marek Kadlčík. 2023. Can in-context learners learn a reasoning concept from demonstrations? In *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, pages 107–115, Toronto, Canada. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3261–3275.

Xinyi Wang, Wanrong Zhu, and William Yang Wang. 2023. Large language models are implicitly topic models: Explaining and finding good demonstrations for in-context learning. *CoRR*, abs/2301.11916.

Taylor Webb, Keith J Holyoak, and Hongjing Lu. 2023. Emergent analogical reasoning in large language models. *Nature Human Behaviour*, 7(9):1526–1541.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022b. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, et al. 2024. Livebench: A challenging, contamination-free LLM benchmark. *CoRR*, abs/2406.19314.

Jian Xie, Kai Zhang, Jiangjie Chen, Renze Lou, and Yu Su. 2024. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Dylan Slack, Qin Lyu, Sean Hendryx, Russell Kaplan, Michele Lunati, and Summer Yue. 2024. A careful examination of large language model performance on grade school arithmetic. *CoRR*, abs/2405.00332.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. 2023a. How language model hallucinations can snowball. *CoRR*, abs/2305.13534.

Ruiqi Zhang, Spencer Frei, and Peter L. Bartlett. 2023b. Trained transformers learn linear models in-context. *CoRR*, abs/2306.09927.

Shen Zheng, Jie Huang, and Kevin Chen-Chuan Chang. 2023. Why does chatgpt fall short in answering questions faithfully? *CoRR*, abs/2304.10513.

A More on the benchmark

Table 10 shows examples from our benchmark. There are 25 plain texts in natural language English words and 25 in random non-sense words. We chose 4 shift values $[3, 6, 9, 12]$, which results in a total of 100 data for each type of plain text.

type	example
natural	{“plain_text”: “good deeds bring joy”, “cipher_text”: “jrrg ghhgv eulqj mrb”, “shift”: 3}
	{“plain_text”: “good deeds bring joy”, “cipher_text”: “muuj jkkjy hxotm pue”, “shift”: 6}
	{“plain_text”: “good deeds bring joy”, “cipher_text”: “pxxm mnnmb karwp sxh”, “shift”: 9}
	{“plain_text”: “good deeds bring joy”, “cipher_text”: “saap pqqpe nduzs vak”, “shift”: 12}
random	{“plain_text”: “olksad twuqwej”, “cipher_text”: “ronvdg wzxtzhm”, “shift”: 3}
	{“plain_text”: “olksad twuqwej”, “cipher_text”: “urqygj zcawckp”, “shift”: 6}
	{“plain_text”: “olksad twuqwej”, “cipher_text”: “xutbjm cfdzfns”, “shift”: 9}
	{“plain_text”: “olksad twuqwej”, “cipher_text”: “axwemp figciqv”, “shift”: 12}

Table 10: Examples of the benchmark data. See here for the complete dataset. Return to main text.

B More on the experimental setup

For GPT-4o, we set `max_new_tokens=1024` for experiments using the open prompt, `max_new_tokens=64` for `base`, `max_new_tokens=256` for `dict`, and `max_new_tokens=512` for `code`. We set `temperature=0` and `seed=2266` for all experiments.

For locally run models (LLaMA3.1-8B/70B, Qwen2.5-7B/32B, and QwQ-32B), we set `max_new_tokens=1024`, `temperature=0.01`, and `seed=2266`. We use `4bit` quantization, and all experiments were run on an NVIDIA H100.

Table 11 shows fragments of the few-shot prompts used in our experiments.

```
Encode the given text to a Caesar cipher. The shift
is 9. Output the cipher text only. Here are some
examples:

plain text: live and let live
cipher text: uren jwm unc uren

plain text: against all odds
cipher text: jpjrwbc juu xmmb

plain text: beyond the horizon
cipher text: knhxwm cqn qxarixw

plain text: through thick and thin
cipher text: cqaxdpq cqrlt jwm cqrw

plain text: time heals all wounds
cipher text: crvn qnjub juu fxdwmb

plain text: stay true to yourself
cipher text: bcjh cadn cx hxdabnuo

......
```

(a) encode

```
Decode the following Caesar cipher text. The shift
is 9. Output the plain text only. Here are some
examples:

cipher text: uren jwm unc uren
plain text: live and let live

cipher text: jpjrwbc juu xmmb
plain text: against all odds

cipher text: knhxwm cqn qxarixw
plain text: beyond the horizon

cipher text: cqaxdpq cqrlt jwm cqrw
plain text: through thick and thin

cipher text: crvn qnjub juu fxdwmb
plain text: time heals all wounds

cipher text: bcjh cadn cx hxdabnuo
plain text: stay true to yourself

......
```

(b) decode

Table 11: Examples of the 50-shot prompts used in our experiments. See here for the complete prompts. Return to main text.

Table 12 shows that all the tested models "know" what the Caesar cipher is. All of them are able to give a correct definition and provide an example of it. Note that 4 of 6 models use `shift=3` in their examples, which suggests that cipher texts with this shift may be prominent in their pre-training data.

model	output
GPT-4o	Yes, I'm familiar with the Caesar cipher. It's a type of substitution cipher in which each letter in the plaintext is shifted a certain number of places down or up the alphabet. For example, **with a shift of 3**: - A would be replaced by D - B would become E - C would become F - and so on. The Caesar cipher is named after Julius Caesar, who is reputed to have used it to communicate with his officials. It's a simple form of encryption and is easy to both encode and decode. Would you like to see an example or need help with encoding or decoding a message using the Caesar cipher?
LLaMA3.1-8B	It's a simple encryption technique that replaces each letter with a letter a fixed number of positions down the alphabet. For example, if you **shift each letter three positions down the alphabet**, "a" becomes "d", "b" becomes "e", and so on. Here's a simple implementation of the Caesar cipher in Python...
LLaMA3.1-70B	It's a simple encryption technique where each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. Here's a Python function that does this...
Qwen2.5-7B	It is a simple substitution cipher where each letter in the plaintext is shifted by a fixed number of positions in the alphabet. For example, with a shift of 1, 'A' would be replaced by 'B', 'B' would become 'C', and so on. The last letter of the alphabet wraps around to the beginning, so 'Z' would become 'A'. Now, let's apply this concept to a more complex scenario. Imagine you have a message that needs to be encrypted using a Caesar cipher with a shift of 3. However, there's a twist...
Qwen2.5-32B	It's a simple substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet. For example, **with a shift of 3**, 'A' would become 'D', 'B' would become 'E', and so on. The same shift is applied to all letters in the message. Can you write a Python function that encodes a given string using the Caesar cipher with a specified shift value? Certainly! Below is a Python function...
QwQ-32B	It's a simple encryption technique where each letter in the plaintext is shifted a certain number of places down or up the alphabet. For example, with a shift of 1, 'A' would be replaced by 'B', 'B' would become 'C', and so on. The method is named after Julius Caesar, who used it for confidential communication. Now, imagine you have a message encrypted with a Caesar cipher **using a shift of 3**. The encrypted message is...

Table 12: Responses from each of the tested models to the query "Do you know the Caesar cipher?" Note that 4 out of 6 responses use `shift=3` examples (highlighted in **bold**). Return to Experimental setup. Return to Results.

C More on results

Table 13 shows the model performance when `shift=12`. Table 14 and 15 provide examples of model outputs. See here for all model outputs.

As Table 14 shows, GPT-4o outputs a reasoning chain that includes all letter mappings and produces the correct final answer. However, in the encoding task, GPT-4o does not use verbalization and simply outputs "muut jkkjy hxotm puze", which is incorrect.

type	model	exact match accuracy↑			character error rate↓		
		open	base	dict	open	base	dict
natural	GPT-4o	0.3600	0.1100	0.0000	0.2145	0.5171	0.5102
	LLaMA3.1-8B	0.0000	0.0000	0.0000	0.9344	0.9098	0.9723
	LLaMA3.1-70B	0.0000	0.0000	0.0000	0.8940	0.9128	0.9144
	Qwen2.5-7B	0.0000	0.0000	0.0000	0.9435	0.9293	0.9313
	Qwen2.5-32B	0.0000	0.0000	0.0000	0.8401	0.8907	0.8522
random	GPT-4o	0.2000	0.0000	0.0000	0.2804	0.5336	0.4648
	LLaMA3.1-8B	0.0000	0.0000	0.0000	0.9278	0.9172	0.9657
	LLaMA3.1-70B	0.0000	0.0000	0.0000	0.8931	0.9181	0.9229
	Qwen2.5-7B	0.0000	0.0000	0.0000	0.9057	0.9067	0.9113
	Qwen2.5-32B	0.0000	0.0000	0.0000	0.8282	0.8839	0.8633
AVERAGE (w/o GPT-4o)		0.0000	0.0000	0.0000	0.8958	0.9085	0.9167

Table 13: Continual of Table 2. Exact match accuracy and character error rate; `shift=12`. Return to main text.

To decode a Caesar cipher with a shift of 6, you need to shift each letter in the text back by 6 positions in the alphabet.

Let's decode the text "muuj jkkjy hxotm pue":

1. 'm' shifted back by 6 positions is 'g'; 2. 'u' shifted back by 6 positions is 'o'; 3. 'u' shifted back by 6 positions is 'o'; 4. 'j' shifted back by 6 positions is 'd'; So, "muuj" becomes "good".

5. 'j' shifted back by 6 positions is 'd'; 6. 'k' shifted back by 6 positions is 'e'; 7. 'k' shifted back by 6 positions is 'e'; 8. 'j' shifted back by 6 positions is 'd'; 9. 'y' shifted back by 6 positions is 's'; So, "jkkjy" becomes "deeds".

10. 'h' shifted back by 6 positions is 'b'; 11. 'x' shifted back by 6 positions is 'r'; 12. 'o' shifted back by 6 positions is 'i'; 13. 't' shifted back by 6 positions is 'n'; 14. 'm' shifted back by 6 positions is 'g'; So, "hxotm" becomes "bring".

15. 'p' shifted back by 6 positions is 'j'; 16. 'u' shifted back by 6 positions is 'o'; 17. 'e' shifted back by 6 positions is 'y'; So, "pue" becomes "joy".

Putting it all together, the decoded text is: "good deeds bring joy"

Table 14: GPT-4o output for the query `system prompt` *Decode the following Caesar cipher text. The shift is 6.* `user prompt` *muuj jkkjy hxotm pue*. The decoded text is correct. The model explicitly verbalizes all letter mappings. Return to main text.

The Caesar cipher shifts each letter by a certain number of places in the alphabet. In this case, the shift is 3. To decode the text, we need to shift each letter back by 3 places. Let's decode "udnler coptzh": u -> r d -> a n -> k l -> i e -> b r -> o c -> z o -> l p -> m t -> q z -> w h -> e So, "udnler coptzh" decoded with a shift of 3 is "ranker zlmqwe".

Table 15: GPT-4o output for the query `system prompt` *Decode the following Caesar cipher text. The shift is 3.* `user prompt` *udnler coptzh*. The decoded text is incorrect. The correct answer is "rakibo zlmqwe". Note that though the final answer is incorrect, the model generates a correct reasoning chain. Return to main text.