LARGE LANGUAGE MODELS ARE NOT STRONG ABSTRACT REASONERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models have shown tremendous performance on a large variety of natural language processing tasks, ranging from text comprehension to common sense reasoning. However, the mechanisms responsible for this success remain opaque, and it is unclear whether LLMs can achieve human-like cognitive capabilities or whether these models are still fundamentally circumscribed. Abstract reasoning is a fundamental task for cognition, consisting of finding and applying a general pattern from few data. Evaluating deep neural architectures on this task could give insight into their potential limitations regarding reasoning and their broad generalisation abilities, yet this is currently an under-explored area. In this paper, we introduce a new benchmark for evaluating language models beyond memorization on abstract reasoning tasks. We perform extensive evaluations of state-of-the-art LLMs, showing that they currently achieve very limited performance in contrast with other natural language tasks, even when applying techniques that have been shown to improve performance on other NLP tasks. We argue that guiding LLM generation to follow causal paths could help improve the generalisation and reasoning abilities of LLMs.

1 INTRODUCTION

Large Language Models (LLMs) have achieved impressive performance on a large variety of Natural Language Processing (NLP) tasks, including text comprehension (Devlin et al., 2019; Radford et al., 2019), commonsense reasoning (Talmor et al., 2020), and code generation (Bubeck et al., 2023), and have shown promising results for out-of-distribution generalisation (Brown et al., 2020; Bubeck et al., 2023). The most recent and larger language models also perform well on mathematical problems, which had been out of reach for transformers for a long time (Chen et al., 2022; Stolfo et al., 2022). While empirical testing of LLMs trained on large corpora of data yields signs of high comprehension of presented problems, there is little theoretical evidence regarding why and how this performance has been achieved and whether these models are simply memorising the training data, extrapolating it, or some combination (Tirumala et al., 2022; Goyal & Bengio, 2020). A notable limitation of these models is a lack of control mechanisms, or possible misalignment (Ouyang et al., 2022), for which the absence of a world model or causal representation have been advanced as explanations (Bender et al., 2021; Zecevic et al., 2023). Experiments on GPT-4 showed signs of limitations on reasoning tasks requiring planning and backtracking or with an uncommon distribution (Bubeck et al., 2023; Wu et al., 2023). Despite these limitations, the question of whether or not LLMs can perform human-like reasoning remains open, as measuring the intelligence, or more broadly, the competence, of a system is a challenging task (Chollet, 2019).

Abstract reasoning is a potential task for effective measurement of the cognitive abilities of neural models (Santoro et al., 2018; Chollet, 2019). Abstract reasoning problems consist of identifying generic structures over a small set of examples and applying them to unseen cases. They aim to evaluate the ability of a system to integrate a new skill or process from limited data. The abstract nature of these problems helps avoid spurious correlations that could lie in the data and may create potential bias in the results. In particular, this task is well-suited for evaluating the broad or strong generalisation capacity of a system, i.e. its ability to handle a large category of tasks and environments without human intervention, including situations that may not have been foreseen when the system was created (Chollet, 2019). This is a well-studied class of task in the field of program induction (Ellis et al., 2020; Lake et al., 2015). However, the problem of abstract reasoning has long remained

outside the scope of evaluation of language models, and there currently exist no extensive evaluations of the performance of LLMs in this domain.

In this paper, we seek to bridge this gap by investigating the abstract reasoning abilities of LLMs and by providing insight into the following question: Do LLMs contain sufficient building blocks for broad generalisation, or do they lack fundamental capabities? We evaluate state-of-the-art LLMs on abstract reasoning tasks, applying recent fine-tuning and prompt design techniques that have been shown to improve performance on other NLP tasks. To this end, we create a benchmark based on existing datasets and novel datasets transposed from vision tasks and adapted to text-based models. We then perform extensive experiments on this benchmark. We also build and fine-tune LLMs for abstract reasoning and compare their performances with the general models. Our results indicate that Large Language Models do not yet have the ability to perform sound abstract reasoning. All of the tested models exhibit poor performance, and the tuning techniques that improved LLM reasoning abilities do not provide significant help for abstract reasoning. We investigate potential reasons for this setback. We provide our code and data at this anonymous repository: `https://anonymous.4open.science/r/abstract-reasoning-9652/`. Our contributions can be summarised as follows:

- We perform an extensive evaluation of pre-trained and fine-tuned Large Language Models on abstract reasoning tasks.
- We show that existing training and tuning techniques do not help increase the performance of LLMs in abstract reasoning, and investigate the reasons and leads for improvement. In particular, we show that LLMs fail to grasp abstract patterns and learn causal mechanisms.
- We create a benchmark for the evaluation of language models for abstract reasoning.

2 Related Work

The abilities of Language Models have been thoroughly studied on a wide range of problems. In particular, their reasoning capacities are the focus of a great deal of recent work. Some of this (Wei et al., 2022; Li et al., 2022; Chen et al., 2022) has explored prompt techniques to improve mathematical reasoning in LLMs; Stolfo et al. (2022) propose a framework based on causality theory to evaluate language models on this kind of task. Recently, GPT-4 has been shown to perform well on mathematical problems although it still produces calculation mistakes (Bubeck et al., 2023). In the domain of logical reasoning, evaluations showed that the logical reasoning abilities of LLMs are tied to the semantics of the input and can hallucinate in uncommon situations (Tang et al., 2023; Xu et al., 2023). Causal structure discovery and causal inference are other domains where LLMs have shown mixed results (Zecevic et al., 2023; Kiciman et al., 2023; Jin et al., 2023). These tasks are distinct from commonsense causal reasoning, where LLMs perform well (Kiciman et al., 2023). Experiments with GPT-4 (Bubeck et al., 2023) showed that, despite presenting systematically better performance than its previous versions, it still has some innate limitations. The authors introduce several examples indicating that the autoregressive nature of LLMs may prevent them from planning and backtracking, two abilities necessary for complex reasoning (Bubeck et al., 2023). GPT-4 also does not always reason in a consistent manner. Although it produces consistent results more often than GPT-3, there are no guarantees that the process leading to the result is always correct. GPT-4's performance also drops on counterfactual tasks, i.e. common problems in unfamiliar settings (e.g. arithmetic in base nine) (Wu et al., 2023). These experiments highlight a lack of abstraction when solving a task but the reason for these shortcomings remain unknown. The scope of cognitive abilities of the system remain incompletely characterised, especially for precise reasoning (Bubeck et al., 2023).

The evaluations described above do not, of course, provide a measure of the intelligence or global cognitive abilities of those models; measuring the level of intelligence of LLMs and other AI systems is challenging as there is no clear widely accepted definition (Booch et al., 2021; Goyal & Bengio, 2020). Chollet (2019) defines the intelligence of a system as "a measure of its skill-acquisition efficiency over a scope of tasks, with respect to priors, experience, and generalization difficulty". Following this definition, abstract reasoning is a well-suited domain over which to measure aspects of the learning and generalisation abilities of a system. To this end, the Abstract Reasoning Challenge (ARC) has been proposed as a benchmark for artificial systems (Chollet, 2019). A handful of works have proposed to measure abstract reasoning abilities in neural networks, but they focus on visual tasks (Santoro et al., 2018; Zhang et al., 2019; 2021a). To the best of our knowledge, this paper is

the first to present an extensive evaluation of abstract reasoning for Large Language Models. Other domains of study focus on problems similar to abstract reasoning. Notably, in program induction, DreamCoder is a system that learns to solve problems described by a small set of input-output pairs by writing programs (Ellis et al., 2020). Abstract reasoning can also be related to causal representation learning, as finding abstract relations amounts to recovering the causal structure of a task and the Independent Causal Mechanisms (ICMs) linking the variables (Schölkopf et al., 2021; Gendron et al., 2023).

3 Evaluation Method

3.1 Evaluation Data

To evaluate language models on a large variety of abstract reasoning tasks, we build a new framework that adapts text and vision datasets for abstract reasoning. We select the tasks based on their capacity to evaluate the ability of a system to find a general abstract rule from limited examples. The visual datasets are converted into text and symbolic versions to be used with language models. After formatting, the datasets can be divided into two categories: Open-Ended Question Answering (Open QA) and Multiple-Choice Question Answering (MCQA). Open QA datasets require the model to generate the correct answer, while MCQA requires it to choose the answer from a set of possible answers. We note that most of the evaluated models are built for general-purpose text generation. Therefore, even when choosing between several options, they must *generate* the correct choice and may fail to do so (e.g. answering D when only options A, B, or C are available). For comparison, we also evaluate models built for question answering. We give more details in Section 3.2. As shown in Figure 1, QA engines can only answer MCQA datasets, while text completion models can answer any type of question. Some MCQA datasets can also be converted to Open QA datasets by removing the choices. The datasets obtained are summarised in Table 1.

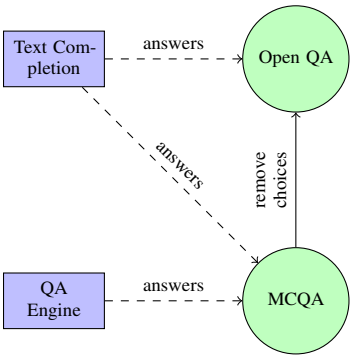


Figure 1: Different types of models and datasets considered in our experiments and their interactions. Dataset types are represented as green circles and model types are represented as blue rectangles.

Example Cases

[9, 4] → [9, 4]
[8, 2, 9, 4, 1, 7] → [9, 4, 8, 2, 1, 7]
[5, 7, 3, 4, 9, 2, 0] → [3, 4, 5, 7, 9, 2, 0]
[1, 5, 6, 4, 0, 3, 7] → [6, 4, 1, 5, 0, 3, 7]

Test Case

[4, 3, 2, 8, 9, 6, 7, 1] → *[2, 8, 4, 3, 9, 6, 7, 1]*

Figure 2: Example task in the BIG-Bench-F dataset. For this task, the system must return the input list with the first two elements switched with the following two if they exist. Pre-prompts are omitted from the input. In the test case, the target answer is indicated in *italics*.

Datasets We build a text-based version of the Abstract Causal Reasoning (ACRE) dataset (Zhang et al., 2021a) that we name ACRE^T. ACRE is a Visual Question-Answering (VQA) dataset. Each sample in the data comprises six context images and four test cases. Each context image comprises a set of objects with various shapes, colours and textures, and a light. In the context images, the light can be on or off. The goal of a system is to determine from the context examples if the light is on, off, or if its state cannot be determined in the test cases. To solve this task, the model has to determine for each sample what objects are causally responsible for the activation of the light. We generate two versions of the dataset: in ACRE^T-Text, each image is replaced by a high-level textual description, and in ACRE^T-Symbolic, each image is replaced with a numerical vector representation.

The second dataset we build on is the Abstract Reasoning Challenge (ARC) dataset (Chollet, 2019). The dataset is composed of tasks, each comprising three input and output grids. The goal of the system is to determine the algorithm that converts the input to the output and apply it to a test case.

Table 1: Datasets considered. When not written, type is similar to the one above. Datasets can exist in text or symbolic versions. Text datasets built from an image are indicated with the symbol ^{T}.

Dataset	Type	Versions	
		Text	Symb
ARC^T	Open QA		✓
BIG-Bench-F			✓
Evals-S			✓
PVR			✓
ACRE^T	MCQA	✓	✓
Evals-P			✓
RAVEN^T		✓	✓

Table 2: Models considered. When not written, type is similar to the one above. Models with the symbol * are introduced in this paper. "-AR" indicates that the model has been fine-tuned for abstract reasoning.

Model	Type
GPT-2	Text completion
Text-Davinci-3	
GPT-3.5-Turbo	
GPT-4	
LLaMA-7B	
LLaMA2-7B	
Alpaca	
Alpaca-LoRA	
Zephyr-7B-β	
LLaMA-7B-AR-LoRA*	
LLaMA2-7B-AR-LoRA*	
RoBERTa-AR*	QA Engine
MERIt-AR*	

The grids have a variable size comprised between 8×8 and 30×30, and contain visual patterns (e.g. recognisable shapes, symmetries). We provide the raw grid to the model as a two-dimensional array of integers. We name this version ARC^T. The high dimensionality of the input makes it a challenging task for LLMs. The tasks themselves are also challenging as their transcription in natural language is often complex and supposedly impossible for 12% of them (Acquaviva et al., 2021).

We select a subset of the BIG-Bench dataset (Rule, 2020; Srivastava et al., 2022) that we name BIG-Bench-F for *Functions*. The subset comprises various tasks represented by a function taking a list as input and returning a new transformed list as output. For each task, several input-output samples are given. In BIG-Bench-F, we give four samples per task by default. The functions include typical list processing like replacing the value of one element, selecting a subset, or counting elements. An example is given in Figure 2. The challenge in this task is to accurately recognise the function from a few samples.

We select a subset of the Evals dataset (OpenAI, 2023) representing logic puzzles. Evals-P is a set of tasks where a tuple containing a character and a list of characters is given as an input, and a single word from the set {"foo", "bar"} is generated from the input according to a logic hidden from the evaluated system. The task consists of finding the logic from a few samples and applying it to a test case. Evals-S is another set of tasks where a list of integers is given as an input, and an output list of words is generated. The task is the same as for Evals-P.

Pointer-Value Retrieval (PVR) tasks (Zhang et al., 2021b) involve selecting one or several values in a list and applying a function on this subset. For each task, the system must recognise the retrieval and application functions and apply them to a test case. Samples are composed of a pointer-values pair and a label. The values are stored in an array, and the pointer is an integer pointing to an index in the array. The pointer indicates the subset of values to consider for the task. We generate a new dataset of PVR tasks following this methodology.

RAVEN (Zhang et al., 2019) is a VQA dataset composed of sequences of images to complete. The images contain Raven matrices (Raven, 1938), i.e. geometric shapes (e.g. square, circle, pentagon) assembled together. RAVEN is a dataset similar to Procedurally Generated Matrices (PGM) (Santoro et al., 2018) but also provides a tree structure describing the semantics of each image. We focus on a subset where a single shape appears in the image. The task is, given a sequence of eight images and eight possible choices, to pick the correct image that follows in the sequence. As RAVEN is a visual dataset like ACRE, we use the given semantic tree structure to generate a text description of each image we will feed to the evaluated models. We create two sets: RAVEN^T-Text contains natural language descriptions, and RAVEN^T-Symbolic contains symbolic descriptions. We also build another version of the dataset where choices are hidden. We name the former RAVEN^T-mcqa and the latter RAVEN^T-opqa.

3.2 MODELS EVALUATED

We perform evaluations on the most recent and popular architectures for NLP tasks. Table 2 provides the list of models used in the experiments. More details are provided in the appendix. We restrict our experiments to Large Language models . We conduct experiments on the popular family of GPT architectures. We include three generations of GPT models: GPT-2 (Radford et al., 2019), a 1.5B parameter model; aligned GPT-3 models with Text-Davinci-3, optimised for text completion, and GPT-3.5-Turbo, optimised for chat, two 175B models (Brown et al., 2020; Ouyang et al., 2022); and GPT-4, with unknown training and architectural details (OpenAI, 2023). We also perform experiments on the popular open models LLaMA (Touvron et al., 2023a) and LLaMA2 (Touvron et al., 2023b). Alpaca is a fine-tuned version of LLaMA to respond to instructions (Wang et al., 2022; Taori et al., 2023), and Alpaca-LoRA is a LLaMA model instruction-tuned using Low-Rank Adaptation (Hu et al., 2022). We also fine-tune our own LLaMA and LLaMA2 models for abstract reasoning. For all models, we evaluate the 7B parameters versions by default. Finally, we evaluate the more recent Zephyr-7B-β (Tunstall et al., 2023a;b), a 7B parameters model fine-tuned from Mistral-7B (Jiang et al., 2023). We also compare these generic models on architecture fine-tuned for Multiple-Choice Question Answering. Unlike the text completion engines that produce text in the output, their task consists of discriminating the solution from a small set of options. This problem is more straightforward to solve than the problem of next token prediction tackled by the models described in the previous paragraph. We fine-tune two models for Multiple-Choice Question Answering: RoBERTa-large (Liu et al., 2019), a language model used for text comprehension, and MERIt (Jiao et al., 2022), a model using contrastive pre-training on rules-based data to perform logical reasoning.

4 EXPERIMENTS

4.1 OPEN-ENDED QUESTION ANSWERING

In this section, we detail our experiments on open-ended abstract reasoning. Depending on the dataset, the answer can be in natural language or a symbolic format. The model is asked to provide the answer directly. The accuracy for each model on every dataset is summarised in Table 3.

Table 3: Accuracy of Large Language Models on Open QA datasets. Datasets are represented in columns, and models in rows. The best result for each dataset is indicated in **bold**, and the second best is indicated in *italics*. BBF stands for BIG-Bench-F.

	ARC^T	BBF	Evals-S	PVR	RAVEN^T-opqa	
					Text	Symb
Text-Davinci-3	*0.105*	*0.404*	**0.314**	**0.228**	*0.343*	*0.234*
GPT-3.5-Turbo	0.033	0.153	0.186	0.124	0.226	0.161
GPT-4	**0.119**	**0.514**	*0.304*	0.177	**0.410**	**0.330**
LLaMA-7B	0.010	0.012	0.014	0.060	0.000	0.000
LLaMA2-7B	0.005	0.108	0.000	0.000	0.000	0.001
Alpaca	0.010	0.188	0.014	0.184	0.075	0.030
Alpaca-LoRA	0.012	0.144	0.000	0.152	0.000	0.067
Zephyr-7B-β	0.015	0.292	0.043	*0.209*	0.009	0.145

Our results indicate poor performance of language models on all the presented datasets, although the performance varies between datasets and models. In particular, Text-Davinci-3 and GPT-4 consistently achieve the best performance across the datasets. Zephyr-7B-β has almost systematically the best accuracy among open models. On the other hand, LLaMA-7B has the worst performance of all models. LLaMA2-7B gets a similar accuracy except on BIG-Bench. Alpaca and Alpaca-LoRA present slight improvements on BIG-Bench-F, PVR and RAVEN^T. This improvement is explained by the instruction-tuning used to build Alpaca and Alpaca-LoRA. We provide several examples in the appendix that illustrate this difference. LLaMA-7B often does not attempt to solve the problem but completes the text by giving more examples. These examples do no match the abstract rule for the task. Alpaca and Alpaca-LoRA follow the instructions more faithfully but also fail to grasp the abstract patterns. Instruction-tuning seems to help the model understand the format of the answer and what it is asked to do but provides little help on how to solve the tasks. Moreover, the performance

difference between Text-Davinci-3 and GPT-3.5-Turbo indicates that the type of instruction-tuning matters as Text-Davinci-3 performs systematically better than GPT-3.5-Turbo despite being based on the same model. Overall, GPT-4 performs noticeably better than all the other models. As the details of its architecture and training set are unavailable, we cannot provide satisfactory explanations for this difference. However, the increase in performance is highest on the RAVEN^T dataset. Given that Raven matrices are a standard and long-existing test (Raven, 1938; Carpenter et al., 1990), we can hypothesize that the training data of GPT-4 included some versions of the test. The same remark can be made for BIG-Bench-F as it includes traditional list processing algorithms. Text-Davinci-3 and GPT-4 also achieve good performance on the ARC^T dataset relative to other existing architectures challenged on the task, making them 11^{th} and 14^{th} on the Kaggle leaderboard[1]. However, they still fail to answer a vast majority of the tasks correctly. All LLMs generally fail to answer most of the tasks in each dataset. Despite a performance increase compared to previous versions, the most recent language models do not perform open-ended abstract reasoning well.

4.2 Multiple-Choice Question Answering

As seen in Section 4.1, open-ended abstract reasoning is a challenging problem for language models. We also perform a series of experiments on Multiple-Choice Question Answering tasks. For these tasks, the models are given a set of possible answers and must pick a single one from the set. This task is more accessible than Open-Ended QA, as the valid response is given as part of the input. Results are given in Table 4.

Table 4: Accuracy of Large Language Models for Multiple-Choice QA on the ACRE^T, Evals-P and RAVEN^T datasets. The last line indicates random performance. Completion models can perform worse than random if they do not reply with a valid answer. The best result for each dataset is indicated in **bold**, and the second best is indicated in *italics*.

	ACRE^T		Evals-P	RAVEN^T-mcqa	
	Text	Symb		Text	Symb
GPT-2	**0.371**	0.00	0.496	0.00	0.126
Text-Davinci-3	0.098	0.427	*0.560*	*0.461*	*0.452*
GPT-3.5-Turbo	0.184	0.445	0.481	0.276	0.315
GPT-4	*0.272*	*0.512*	**0.625**	**0.697**	**0.535**
LLaMA-7B	0.000	0.257	0.544	0.004	0.000
LLaMA2-7B	0.014	0.003	0.500	0.026	0.149
Alpaca	0.036	0.238	0.544	0.015	0.058
Alpaca-LoRA	0.015	0.123	0.552	0.082	0.124
Zephyr-7B-β	0.106	**0.516**	0.504	0.000	0.022
random	0.33	0.33	0.5	0.125	0.125

We first compare the results of RAVEN^T-mcqa and RAVEN^T-opqa from Table 3. RAVEN^T-opqa contains the same questions as RAVEN^T-mcqa, but the answer choices have been removed. Following intuition, giving multiple choices to LLMs helps systematically improve their performance. Only the performance of LLaMA remains the same, and the performance of Alpaca and Zephyr-7B-β are slightly reduced. Given the low accuracy in both cases, it can be interpreted as noise. MCQA models achieve slightly above random performance (see details in appendix), performing better than most LLMs. However, they have an advantage compared to completion engines as they have to select one answer among a list of possible choices, whereas completion models must generate the correct answer. Therefore, the latter may not return any valuable output (e.g. a nonsensical or empty answer), explaining how they can achieve worse than random performance. The main takeaway from these experiments is that the performance of LLMs remains low even in discriminative settings. When given a set of possible answers, the models cannot recognise the proper solution among the other choices. This finding indicates that using LLMs as evaluators (as done in self-refinement techniques (Madaan et al., 2023)) is not suited for tasks requiring abstract reasoning. We confirm this with additional experiments in the appendix using different refinement strategies. Additionally, when comparing the results between natural language and symbolic tasks on ACRE^T, we observe that the results are better across all models when the input is symbolic. Inputs that use symbolic data are

[1] https://www.kaggle.com/competitions/abstraction-and-reasoning-challenge/leaderboard

smaller and may convey only relevant information, while natural language could contain distracting information or biases harmful to task performance. The same observation can be made concerning RAVEN^T-mcqa, except for GPT-4. In the open-ended version of RAVEN^T, models perform better with the natural language representation. Without the answer set available, inductive biases caused by language help performance.

4.3 CHAIN-OF-THOUGHT PROMPTING

We perform experiments on a subset of our framework using *Chain-of-Thought* prompting (Wei et al., 2022). The complete experiments are provided in the appendix (and include a side-by-side comparison for better readability). We perform experiments with GPT-3.5-turbo, GPT-4, and Alpaca-LoRA. Our experiments with *Chain-of-Thought* have the suffix *model*-cot. Our results are presented in Table 5. Overall, the results obtained using *Chain-of-Thought* prompting are not higher than those obtained with the base models. On The BIG-Bench-F dataset, the *Chain-of-Thought* versions achieve systematically lower performance than their base counterparts, although no no significant performance drop is observed. On PVR and RAVEN^T-opqa, while the accuracy for GPT-4 and Alpaca-LoRA remain unchanged or slightly reduced, the performance of GPT-3.5-Turbo is increased. On RAVEN^T-mcqa, the performance of all the models decreases. These experiments show that the quality of the prompt has little impact on the answer quality. It hints that the models can understand the task, but their failures are due to their ability to provide faithful reasoning. This limitation is further illustrated with examples in the appendix.

Table 5: Accuracy of Large Language Models on Open and Multiple-Choice QA datasets when prompted using *Chain-of-Thought*. Datasets are represented in columns, and models in rows. The best result for each dataset is indicated in **bold**, and the second best is indicated in *italics*. BBF stands for BIG-Bench-F.

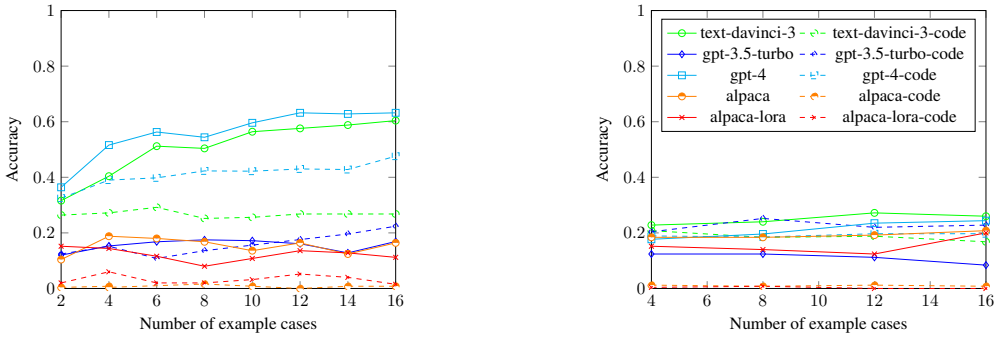
	BBF	PVR	RAVEN^T-opqa		ACRE^T		RAVEN^T-mcqa	
			Text	Symb	Text	Symb	Text	Symb
GPT-3.5-Turbo-cot	*0.097*	**0.210**	*0.302*	*0.211*	**0.255**	*0.345*	*0.257*	*0.144*
GPT-4-cot	**0.476**	*0.174*	**0.385**	**0.354**	*0.214*	**0.394**	**0.596**	**0.517**
Alpaca-LoRA-cot	0.084	0.152	0.000	0.069	*0.059*	0.114	0.000	0.114

4.4 VARYING THE EXAMPLE SET SIZE

We perform further experiments on the BIG-Bench-F and PVR datasets. For these two datasets, we alter the number of examples given to the system before the test case. By default, we give four examples to the model before asking it to answer. The results are shown in Figures 3a and 3b. In this section, we focus on the results of the base models (without the "-code" suffix). We first observe that, for both datasets, there is no linear relationship linking performance and number of examples. For all but the Text-Davinci-3 and GPT-4 models, adding more examples has little or no effect on the accuracy. Text-Davinci-3 and GPT-4 perform similarly across all cases, and their performances consistently increase with the number of examples, achieving up to an accuracy of 0.6 when given 16 examples on the BIG-Bench-F dataset. However, on PVR, Text-Davinci-3 achieves only 0.26 when given 12 examples. GPT-4 follows a similar trend but performs slightly worse than its predecessor. In the Absence of technical details for GPT-4, we can only speculate on the reasons. As this effect is observed only on BIG-Bench-F and not PVR, we can assume that the models perform better because their training sets contain the list processing algorithms used by BIG-Bench-F. We perform additional experiments in the appendix, where we provide solved instances into the prompt (input and solution program) to propel the model to reason correctly. No real improvements are observed.

4.5 ENABLING STRUCTURE DISCOVERY WITH CODE

In the next experiments, we follow an idea similar to *Progam-of-Thought* prompting (Chen et al., 2022) and ask the model to generate the code of the function responsible for generating the output from the input. Then, we execute the produced code on the test case and evaluate the result. This method differs from a base prompt as we do not ask the model to produce the answer directly. This part is delegated to a code interpreter in Python. This method aims to verify the ability of LLMs to extract the correct structure behind each abstract reasoning task under code format. We test this



(a) BIG-Bench-F dataset.

(b) PVR dataset.

Figure 3: Evolution of the model performance as a function of the number of examples seen from the dataset. The legend is shared by both figures. Models with straight lines are used with default prompting, while models with dashed lines are prompted to produce code.

method on the BIG-Bench-F and PVR datasets. The results of these models (with the "-code" suffix) can be compared with their original counterparts in Figures 3a and 3b. In general, we observe that the models prompted to produce code perform worse than those tasked to produce the answer directly. The only exception is GPT-3.5-Turbo. On the BIG-Bench-F dataset, the performance of GPT-3.5-Turbo-code increases steadily while that of GPT-3.5-Turbo stagnates, and on PVR, GPT-3.5-Turbo-code outperforms GPT-3.5-Turbo by a significant margin. Producing code solving the abstract problem is a more complicated task for an LLM as it requires the model to produce a rigorous code explanation for its answer. It is consistent with the results for most models, but we also observe in the case of GPT-3.5-Turbo-code that it can help the model better understand the task. On BIG-Bench-F, the code versions of Text-Davinci-3 and GPT-4 perform better than both base and code versions of the other models. As this behaviour is not observed with PVR, we infer that this performance is due to the functions being part of the training sets of the models. The models can almost always generate code able to compile and produce an answer (details are in the appendix). We deduce that producing a program with a valid syntax is not a bottleneck for performance. The issue lies in the recovery of the correct reasoning process.

4.6 Fine-tuning LLaMA2

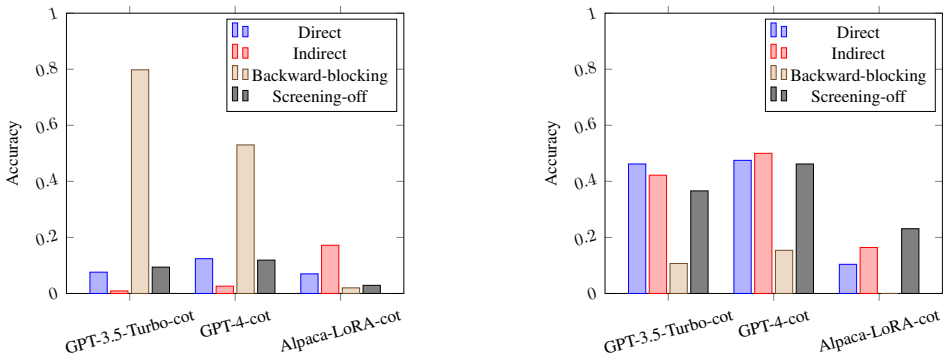
We now study the performance of LLaMA2 models after fine-tuning on RAVEN^T-mcqa. Experiments on more datasets are provided in the appendix. The training and test sets may share distribution-specific patterns that the model may learn during the fine-tuning phase. It may overfit on these patterns instead of learning the correct abstract patterns. To alleviate this pitfall, we generate out-of-distribution (o.o.d) splits. The *-Four* split contains samples with four figures instead of one. The *-In-Center* splits contains samples with two figures instead of one, a big and a small located within the former. The shape and colours of the figures all are observed in the training set. The two splits can be considered as compositional splits. The results on RAVEN^T-mcqa are shown in Table 6. We observe a significant increase in the accuracy on the test set. LLaMA2 achieves close to perfect accuracy. The performance partially transfers to the alternative syntax task. We now observe the performance on the o.o.d splits. The performance of the fine-tuned LLaMA2 significantly drops on the new tasks, showing a lack of generalisation. We can deduce that fine-tuning yields representations that are highly invariant to the syntax but does not transfer other abstract reasoning abilities. The rules required to solve the *-Four* and *-In-Center* splits manipulate several figures, they are compositions of rules used for single figures. LLMs can compose with unseen quantities (e.g. new syntax) but have more difficulty composing new abstract rules.

4.7 A Perspective from Causal Induction

We perform further analysis on ACRE^T. The dataset can be divided into four causal paths: Direct, Indirect, Backward-blocking, Screening-off (Zhang et al., 2021a). Direct path queries can be established using direct evidence. Indirect paths require the combination of multiple pieces of

Table 6: Accuracy of base and fine-tuned LLaMA2 on the RAVEN^T-mcqa dataset i.i.d and o.o.d splits. Rows represent the dataset on which the model is fine-tuned, and columns represent the dataset on which the model is evaluated. The best result for each dataset in indicated in **bold**.

Model	Test Set ⇒ / Tuning Set ⇓		RAVEN^T-Eval		-Four		-In-Center	
			Text	Symb	Text	Symb	Text	Symb
LLaMA2-7B			0.135	0.114	0.073	0.121	0.000	0.001
-AR-LoRA*	RAVEN^T-Train	Text	**0.977**	0.694	**0.557**	**0.522**	0.536	**0.085**
		Symb	0.965	**0.938**	0.498	0.442	**0.767**	0.064



(a) Text ACRE^T.

(b) Symbolic ACRE^T.

Figure 4: Results of chain-of-thought models on ACRE^T divided by causal paths.

evidence. Backward-blocking paths cannot be determined because the true mechanisms cannot be discriminated from other possibilities based solely on the data. Screening-off paths are causal paths affected by spurious correlations. Figure 4 shows the results for each type of query. We restrict our analysis to the *Chain-of-Thought* models (see the appendix for the full analysis). Although accuracy scores are similar, the distribution of the results among the causal paths differs between models and input types. GPT models overfit to backward-blocking cases on the text ACRE^T but not on the symbolic version. We can deduce that natural language contains distracting information or biases harmful to abstract reasoning performance. It is consistent with the higher score of the models on the symbolic tasks.

5 Conclusion

Understanding the potential reasoning capabilities of LLMs is crucial as they are starting to be widely adopted. Measuring the level of intelligence of a system is hard, but abstract reasoning provides a valuable framework for this task. In this paper, we present what is, to the best of our knowledge, the first extensive evaluation of Large Language Models for abstract reasoning. We show that LLMs do not perform well on all types of tasks, although not all models are equally poor. Prompting and refinement techniques that improve performance on NLP tasks do not work for abstract reasoning. Our experiments show that the bottleneck in the performance lies in the recognition of new unseen abstract patterns and not in a lack of understanding of the task or the prompt. These results hold in discriminative settings, where the models must find the correct answer within a small set of propositions. A qualitative study of selected failure cases in the appendix further reveals that models tend to reason inconsistently and in a shallow way. Models also tend to produce convoluted reasonings that can match the input but hardly generalise to new instances. We hypothesise that current self-supervised autoregressive LLMs lack fundamental properties for strong abstract reasoning tasks and human-like cognition. In particular, we posit that methods based on causal reasoning and program induction could help improve the reasoning abilities of LLMs.

REFERENCES

Samuel Acquaviva, Yewen Pu, Marta Kryven, Catherine Wong, Gabrielle E. Ecanow, Maxwell I. Nye, Theodoros Sechopoulos, Michael Henry Tessler, and Joshua B. Tenenbaum. Communicating natural programs to humans and machines. *CoRR*, abs/2106.07824, 2021. URL https://arxiv.org/abs/2106.07824.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pp. 610–623. ACM, 2021. doi: 10.1145/3442188.3445922. URL https://doi.org/10.1145/3442188.3445922.

Grady Booch, Francesco Fabiano, Lior Horesh, Kiran Kate, Jonathan Lenchner, Nick Linck, Andrea Loreggia, Keerthiram Murugesan, Nicholas Mattei, Francesca Rossi, and Biplav Srivastava. Thinking fast and slow in AI. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 15042–15046. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17765.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 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*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

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. Sparks of artificial general intelligence: Early experiments with GPT-4. *CoRR*, abs/2303.12712, 2023. doi: 10.48550/arXiv.2303.12712. URL https://doi.org/10.48550/arXiv.2303.12712.

Patricia A Carpenter, Marcel A Just, and Peter Shell. What one intelligence test measures: a theoretical account of the processing in the raven progressive matrices test. *Psychological review*, 97(3):404, 1990.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *CoRR*, abs/2211.12588, 2022. doi: 10.48550/arXiv.2211.12588. URL https://doi.org/10.48550/arXiv.2211.12588.

François Chollet. On the measure of intelligence. *CoRR*, abs/1911.01547, 2019. URL http://arxiv.org/abs/1911.01547.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke B. Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *CoRR*, abs/2006.08381, 2020. URL https://arxiv.org/abs/2006.08381.

Gaël Gendron, Michael Witbrock, and Gillian Dobbie. A survey of methods, challenges and perspectives in causality. *CoRR*, abs/2302.00293, 2023. doi: 10.48550/arXiv.2302.00293. URL https://doi.org/10.48550/arXiv.2302.00293.

Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. *CoRR*, abs/2011.15091, 2020. URL https://arxiv.org/abs/2011.15091.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023. doi: 10.48550/ARXIV.2310.06825. URL https://doi.org/10.48550/arXiv.2310.06825.

Fangkai Jiao, Yangyang Guo, Xuemeng Song, and Liqiang Nie. Merit: Meta-path guided contrastive learning for logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 3496–3509. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-acl.276. URL https://doi.org/10.18653/v1/2022.findings-acl.276.

Zhijing Jin, Jiarui Liu, Zhiheng Lyu, Spencer Poff, Mrinmaya Sachan, Rada Mihalcea, Mona T. Diab, and Bernhard Schölkopf. Can large language models infer causation from correlation? *CoRR*, abs/2306.05836, 2023. doi: 10.48550/arXiv.2306.05836. URL https://doi.org/10.48550/arXiv.2306.05836.

Emre Kiciman, Robert Ness, Amit Sharma, and Chenhao Tan. Causal reasoning and large language models: Opening a new frontier for causality. *CoRR*, abs/2305.00050, 2023. doi: 10.48550/arXiv.2305.00050. URL https://doi.org/10.48550/arXiv.2305.00050.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. On the advance of making language models better reasoners. *CoRR*, abs/2206.02336, 2022. doi: 10.48550/arXiv.2206.02336. URL https://doi.org/10.48550/arXiv.2206.02336.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *CoRR*, abs/2303.17651, 2023. doi: 10.48550/arXiv.2303.17651. URL https://doi.org/10.48550/arXiv.2303.17651.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arXiv.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. *CoRR*, abs/2310.08559, 2023. doi: 10.48550/ARXIV.2310.08559. URL https://doi.org/10.48550/arXiv.2310.08559.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

John C Raven. Raven standard progressive matrices. *Journal of Cognition and Development*, 1938.

Joshua Stewart Rule. *The child as hacker: building more human-like models of learning*. PhD thesis, Massachusetts Institute of Technology, 2020.

Adam Santoro, Felix Hill, David G. T. Barrett, Ari S. Morcos, and Timothy P. Lillicrap. Measuring abstract reasoning in neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4477–4486. PMLR, 2018. URL http://proceedings.mlr.press/v80/santoro18a.html.

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proc. IEEE*, 109(5): 612–634, 2021. doi: 10.1109/JPROC.2021.3058954. URL https://doi.org/10.1109/JPROC.2021.3058954.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Santilli, Andreas Stuhlmüller, Andrew M. Dai, Andrew La, Andrew K. Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakas, and et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *CoRR*, abs/2206.04615, 2022. doi: 10.48550/arXiv.2206.04615. URL https://doi.org/10.48550/arXiv.2206.04615.

Alessandro Stolfo, Zhijing Jin, Kumar Shridhar, Bernhard Schölkopf, and Mrinmaya Sachan. A causal framework to quantify the robustness of mathematical reasoning with language models. *CoRR*, abs/2210.12023, 2022. doi: 10.48550/arXiv.2210.12023. URL https://doi.org/10.48550/arXiv.2210.12023.

Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e992111e4ab9985366e806733383bd8c-Abstract.html.

Xiaojuan Tang, Zilong Zheng, Jiaqi Li, Fanxu Meng, Song-Chun Zhu, Yitao Liang, and Muhan Zhang. Large language models are in-context semantic reasoners rather than symbolic reasoners. *CoRR*, abs/2305.14825, 2023. doi: 10.48550/ARXIV.2305.14825. URL https://doi.org/10.48550/arXiv.2305.14825.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Kushal Tirumala, Aram H. Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/fa0509f4dab6807e2cb465715bf2d249-Abstract-Conference.html.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023a. doi: 10.48550/arXiv.2302.13971. URL https://doi.org/10.48550/arXiv.2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023b. doi: 10.48550/ARXIV.2307.09288. URL https://doi.org/10.48550/arXiv.2307.09288.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. Zephyr: Direct distillation of LM alignment. *CoRR*, abs/2310.16944, 2023a. doi: 10.48550/ARXIV.2310.16944. URL https://doi.org/10.48550/arXiv.2310.16944.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Alexander M. Rush, and Thomas Wolf. The alignment handbook. https://github.com/huggingface/alignment-handbook, 2023b.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D. Goodman. Hypothesis search: Inductive reasoning with language models. *CoRR*, abs/2309.05660, 2023. doi: 10.48550/ARXIV.2309.05660. URL https://doi.org/10.48550/arXiv.2309.05660.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *CoRR*, abs/2212.10560, 2022. doi: 10.48550/arXiv.2212.10560. URL https://doi.org/10.48550/arXiv.2212.10560.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. *CoRR*, abs/2307.02477, 2023. doi: 10.48550/arXiv.2307.02477. URL https://doi.org/10.48550/arXiv.2307.02477.

Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. Are large language models really good logical reasoners? A comprehensive evaluation from deductive, inductive and abductive views. *CoRR*, abs/2306.09841, 2023. doi: 10.48550/ARXIV.2306.09841. URL https://doi.org/10.48550/arXiv.2306.09841.

Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In Swarat Chaudhuri and Charles Sutton (eds.), *MAPS@PLDI 2022: 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022*, pp. 1–10. ACM, 2022. doi: 10.1145/3520312.3534862. URL https://doi.org/10.1145/3520312.3534862.

Matej Zecevic, Moritz Willig, Devendra Singh Dhami, and Kristian Kersting. Causal parrots: Large language models may talk causality but are not causal. *CoRR*, abs/2308.13067, 2023. doi: 10.48550/arXiv.2308.13067. URL https://doi.org/10.48550/arXiv.2308.13067.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. RAVEN: A dataset for relational and analogical visual reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 5317–5327. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00546. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Zhang_RAVEN_A_Dataset_for_Relational_and_Analogical_Visual_REasoNing_CVPR_2019_paper.html.

Chi Zhang, Baoxiong Jia, Mark Edmonds, Song-Chun Zhu, and Yixin Zhu. ACRE: abstract causal reasoning beyond covariation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 10643–10653. Computer Vision Foundation / IEEE, 2021a. doi: 10.1109/CVPR46437.2021.01050. URL https://openaccess.thecvf.com/content/CVPR2021/html/Zhang_ACRE_Abstract_Causal_REasoning_Beyond_Covariation_CVPR_2021_paper.html.

Chiyuan Zhang, Maithra Raghu, Jon M. Kleinberg, and Samy Bengio. Pointer value retrieval: A new benchmark for understanding the limits of neural network generalization. *CoRR*, abs/2107.12580, 2021b. URL https://arxiv.org/abs/2107.12580.

A Dataset Details

This section provides more details and examples of each dataset used in the experiments.

ACRE We conduct experiments on the Abstract Causal Reasoning (ACRE) dataset (Zhang et al., 2021a). ACRE is a Visual Question-Answering (VQA) dataset. In our work, we use a transcription of the dataset into text. Each sample in the data comprises six context images and four test cases. Each context image comprises a set of objects with various shapes, colours and textures, and a light. In the context images, the light can be on or off. The goal of a system is to determine from the context examples if the light is on, off, or if its state cannot be determined in the test cases. To solve this task, the model has to determine for each sample what objects are causally responsible for the activation of the light. We generate two versions of the dataset: in ACRE-Text, each image is replaced by a textual description, and in ACRE-Symbolic, each image is replaced with a vector representation. An example of ACRE-Text is given in Figure 5 and an example of ACRE-Symbolic is given in Figure 6.

ARC The second dataset we use is the Abstract Reasoning Challenge (ARC) dataset (Chollet, 2019). The dataset is composed of tasks, each comprising several input and output grids. The goal of the system is to determine the algorithm that converts the input to the output and apply it to a test case. The grids have a variable size comprised between 8×8 and 30×30, and contain visual patterns (e.g. recognisable shapes, symmetries). We provide the raw grid to the model as a two-dimensional array of integers. The high dimensionality of the input makes it a challenging task for LLMs. The tasks themselves are also challenging as their transcription in natural language is often complex and supposedly impossible for 12% of them (Acquaviva et al., 2021). An example from the original ARC is given in Figure 7.

BIG-Bench We select a subset of the BIG-Bench dataset (Rule, 2020; Srivastava et al., 2022) that we name BIG-Bench-F for *Functions*. The subset comprises various tasks represented by a function taking a list as input and returning a new transformed list as output. For each task, several input-output samples are given. In BIG-Bench-F, we give four samples per task by default. The functions include typical list-processing like replacing one list element with another value, selecting a subset of the list, or counting elements. The difficulty in this task is to accurately recognise the function from a few samples. An example is given in Figure 8.

The functions used in BIG-Bench are classic list-processing functions. Such functions are likely to be in the training sets of Large Language Models trained on large corpora of data on the internet.

Pre-Prompt

Objects of various color, shape, and texture are displayed. Some objects may contain a device to turn a light on if displayed. From the observations, deduce if the light is on, off, or if the state cannot be determined. Your answer must contain a single word:
on.
off.
undetermined.

Example Cases

A cyan cylinder in rubber is visible. The light is on.
A gray cube in rubber is visible. The light is off.
A cyan cylinder in rubber is visible. A gray cube in rubber is visible. The light is on.
A blue cube in metal is visible. The light is off.
A gray cylinder in rubber is visible. A gray cube in metal is visible. The light is off.
A red sphere in metal is visible. A yellow cube in rubber is visible. The light is on.

Test Case

A red sphere in metal is visible. The light is *undetermined*

Figure 5: Sample from the ACRE-Text dataset. In the test case, the target answer is indicated in *italics*.

Pre-Prompt

Figure out the pattern in the following examples and apply it to the test case. Your answer must follow the format of the examples. You can answer 1 if the solution cannot be determined. Your answer must be one of the following choices:
0.
1.
2.

Example Cases

[28] $\rightarrow$ 2
[0] $\rightarrow$ 0
[28, 0] $\rightarrow$ 2
[5] $\rightarrow$ 0
[16, 1]$\rightarrow$ 0
[35, 14] $\rightarrow$ 2

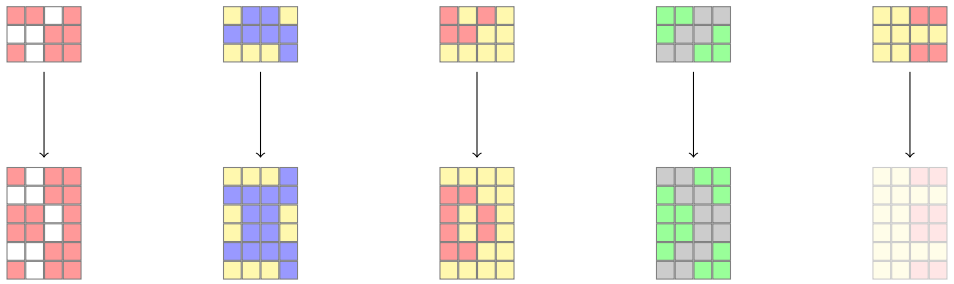
Test Case

[35] $\rightarrow$ *1*

Figure 6: Sample from the ACRE-Symbolic dataset. In the test case, the target answer is indicated in *italics*.

Figure 9 illustrates it with an example. This is a discussion with GPT-4, where the model is prompted to generate a function solving a list-processing problem and create examples. The example shows that the model has prior knowledge of the function needed for the task and could solve it by memorising examples from its training set where the function is applied without the need to reason abstractly.

Evals We select a subset of the Evals dataset (OpenAI, 2023) representing logic puzzles. Evals-P is composed of a set of tasks. For each task, a tuple containing a character and a list of characters is given as an input and a single word from the set {"foo", "bar"} is generated from the input according



(a) Example Case 1. (b) Example Case 2. (c) Example Case 3. (d) Example Case 4. (e) Test Case.

Figure 7: Sample for the ARC dataset. In our work, each grid is given as a numeric array to the model. In this example, the task consists of generating the symmetric to the input grid and appending it to the input. In the test case, the expected output is lightly coloured.

Pre-Prompt

Apply a function to the final input list to generate the output list. Use any preceding inputs and outputs as examples to find what is the function used. All example outputs have been generated using the same function.

Your task is to write down the python function responsible for the transformation of the list in the following examples. The format is [input] → [output]:

Example Cases

[1, 0, 9, 7, 4, 2, 5, 3, 6, 8] → [9, 0, 1, 4, 4, 5]
[3, 8, 4, 6, 1, 5, 7, 0] → [4, 8, 3, 4, 1, 7]
[5, 4, 7, 2, 9, 3, 8, 1] → [7, 4, 5, 4, 9, 8]
[3, 9, 2, 0, 6, 8, 5, 1, 7] → [2, 9, 3, 4, 6, 5]

Test Case

[9, 2, 1, 3, 4, 7, 6, 8, 5, 0] → *[1, 2, 9, 4, 4, 6]*

Write the function. Next, write a line to print the output of this function for the input [9, 2, 1, 3, 4, 7, 6, 8, 5, 0]

Figure 8: Example task in the BIG-Bench-F dataset. For this task, the system must return specific elements of the input list, i.e. [inp[2], inp[1], inp[0], 4, inp[4], inp[6]]. In the test case, the target answer is indicated in *italics*. Text exclusive to base models are indicated by a blue background, and text exclusive to code models are indicated by a green background.

to a logic hidden from the evaluated system. The task consists of finding the logic from eight samples and applying it to a test case. An example is given in Figure 10. Evals-S is composed of another set of tasks. For each task, a list of integers is given as an input and an output list of words is generated from the input according to a logic hidden from the evaluated system. The task consists of finding the logic from three samples and applying it to a test case. An example is given in Figure 11.

PVR The Pointer-Value Retrieval (PVR) dataset (Zhang et al., 2021b) is a dataset for retrieval tasks. Tasks involve selecting one or several values in a list and applying a function on this subset. For each task, the system must recognise the retrieval and application functions and apply them to a test case. Samples in the datasets are composed of a pointer-values pair and a label. The values are stored in an array, and the pointer is an integer pointing to an index in the array. The pointer indicates the subset of values to consider for the task. An example is given in Figure 12.

RAVEN RAVEN (Zhang et al., 2019) is a VQA dataset composed of sequences of images to complete. The images contain Raven matrices (Raven, 1938), i.e. geometric shapes (e.g. square,

Pre-Prompt

You are GPT-4, a powerful large language model trained by OpenAI. Answer as concisely as possible.

User Query

Generate a function that remove the last two elements of a list. Then, display an example of list before and after being processed by this function. Use the format [input_list] → [output_list].

Model Response

Here's a Python function that removes the last two elements of a list:

```python
def remove_last_two_elements(input_list):
    return input_list[:-2]
```

Example:
Input list: [1, 2, 3, 4, 5]
After processing: [1, 2, 3] → [1, 2, 3]

Figure 9: Example of exchange with GPT-4. When asking to provide a list-processing algorithm, the model is able to provide a correct solution and generates an example with the BIG-Bench-F format, although incorrect.

Pre-Prompt

You are a helpful assistant.
Figure out the pattern in the below examples, and then answer with just "foo" or "bar".

Example Cases

f, [o, z, a, n, g, e, j, f, i, c, l, u, b] → foo
l, [v, u, f, b, m, y, j, h, n, c, d, a, p] → bar
p, [c, e, s, h, q, o, a, t, k, d, n, l, z] → bar
p, [c, h, m, z, d, v, k, l, j, e, x, p, n] → foo
y, [e, u, v, z, c, q, s, a, f, o, l, p, d] → bar
u, [x, e, n, t, v, o, g, c, d, y, r, j, l] → bar
m, [l, n, k, e, h, i, c, v, r, j, a, y, o] → bar
v, [j, g, q, t, x, y, m, z, b, h, p, u, r] → bar

Test Case

u, [d, a, x, i, h, v, e, z, r, c, n, y, o] → *bar*

Figure 10: Example task in the Evals-P dataset. For this task, the system must return "foo" if the first character of the input is in the list or "bar" otherwise. In the test case, the target answer is indicated in *italics*.

circle, pentagon) assembled in various ways (e.g. one shape inside another, four shapes in a 4×4 grid). RAVEN is a dataset similar to Procedurally Generated Matrices (PGM) (Santoro et al., 2018) but has the advantage of providing a tree structure describing the semantics of each matrix. We focus on a subset where a single shape appears in the image. The task is, given a sequence of eight images and eight possible choices, to pick the correct image that follows in the sequence. As RAVEN is a visual dataset like ACRE, we generate a text description of each image from their semantic tree that we will feed into the evaluated models. We create two sets: RAVEN-Text contains descriptions in natural language, and RAVEN-Symbolic contains symbolic descriptions. We also build another

Pre-Prompt

You are a pattern recognition bot, figure out the pattern and reply with just the solution, ensure that your reply starts with your solution.

Example Cases

13, 17, 1, 6 → Brown,White,Purple,Blue
1, 9, 6, 11 → Purple,Brown,Blue,White
13, 2, 17, 10 → Brown,Purple,White,Blue

Test Case

5, 9, 2, 11 → *Blue,Brown,Purple,White*

Figure 11: Example of task in the Evals-S dataset. For this task, the system must sort the words according to the numbers in input (e.g. word "white" is located at the index of the highest integer and word "purple" is located at the index of the lowest integer). In the test case, the target answer is indicated in *italics*.

Pre-Prompt

Figure out the pattern in the following examples and apply it to the test case. Your answer must follow the format of the examples.	Your task is to write down the python function responsible for the computation of the output from the list in the following examples. Your answer must follow the format of the examples.

Example Cases

[5, 7, 4, 1, 8, 9, 8, 1, 9, 8, 4] → 8
[4, 0, 0, 7, 0, 1, 0, 5, 3, 0, 0] → 1
[0, 2, 8, 2, 5, 9, 4, 3, 8, 5, 4] → 2
[3, 3, 2, 6, 5, 7, 4, 6, 7, 4, 8] → 5

Test Case

[3, 4, 9, 7, 1, 8, 7, 1, 0, 3, 5] → *1*	Write the function. Next, write a line to print the output of this function for the input [3, 4, 9, 7, 1, 8, 7, 1, 0, 3, 5]

Figure 12: Example of task in the PVR dataset. In the test case, the target answer is indicated in *italics*. Text exclusive to base models are indicated by a blue background, and text exclusive to code models are indicated by a green background.

version of the dataset where choices are hidden. We name the former RAVEN-mcqa and the latter RAVEN-opqa. Examples for each are given in Figures 13 and 14, respectively.

Pre-Prompt

Find the pattern number 9 that completes the sequence. Write the correct pattern with the same format as in the examples. Patterns in the sequence are preceded by a number from 1 to 8.

Find the pattern number 9 that completes the sequence. Pick the letter in front of the correct pattern that logically follows in the sequence from the answer set. Patterns in the sequence are preceded by a number from 1 to 8. Patterns in the answer set are preceded by a letter from A to H. Only return the letter in front of the correct pattern.

Example Cases

1. On an image, a large lime square rotated at 180 degrees.
2. On an image, a medium lime square rotated at 180 degrees.
3. On an image, a huge lime square rotated at 180 degrees.
4. On an image, a huge yellow circle rotated at 0 degrees.
5. On an image, a large yellow circle rotated at 0 degrees.
6. On an image, a medium yellow circle rotated at 0 degrees.
7. On an image, a medium white hexagon rotated at -90 degrees.
8. On an image, a huge white hexagon rotated at -90 degrees.

A. On an image, a tiny white hexagon rotated at -90 degrees.
B. On an image, a giant white hexagon rotated at -90 degrees.
C. On an image, a large red hexagon rotated at -90 degrees.
D. On an image, a large orange hexagon rotated at -90 degrees.
E. On an image, a large white hexagon rotated at -90 degrees.
F. On an image, a large green hexagon rotated at -90 degrees.
G. On an image, a large blue hexagon rotated at -90 degrees.
H. On an image, a large yellow hexagon rotated at -90 degrees.

Test Case

The pattern that logically follows is:
9. *On an image, a large white hexagon rotated at -90 degrees.*

The answer is *E*

Figure 13: Sample from the RAVEN^T-Text dataset. In the test case, the target answer is indicated in *italics*. Text exclusive to RAVEN^T-opqa has a blue background, and text exclusive to RAVEN^T-mcqa has a green background. Shared text has a gray background.

Raven matrices are a standard and long-existing test (Raven, 1938; Carpenter et al., 1990), likely in the training sets of Large Language Models trained on large corpora of data on the internet. To figure it out, we directly prompt GPT-3.5-Turbo and GPT-4. The discussions are represented in Figures 16 and 15. The responses of the model indicate knowledge of the RAVEN test, although GPT-4 generates a correct sample of a Raven test, whereas GPT-3.5-Turbo generates an example that does not have a valid logic. The main takeaway from these examples is that Raven test data has been leaked to the training sets of those models. However, whether exact examples from the test set are also part of the training data is unknown.

Pre-Prompt

Find the pattern number 9 that completes the sequence. Write the correct pattern with the same format as in the examples. Patterns in the sequence are preceded by a number from 1 to 8.

Find the pattern number 9 that completes the sequence. Pick the letter in front of the correct pattern that logically follows in the sequence from the answer set. Patterns in the sequence are preceded by a number from 1 to 8. Patterns in the answer set are preceded by a letter from A to H. Only return the letter in front of the correct pattern.

Example Cases

1. [(D, D, C, H,)]
2. [(C, D, C, H,)]
3. [(E, D, C, H,)]
4. [(E, C, F, D,)]
5. [(D, C, F, D,)]
6. [(C, C, F, D,)]
7. [(C, J, E, B,)]
8. [(E, J, E, B,)]

A. [(A, J, E, B,)]
B. [(F, J, E, B,)]
C. [(D, A, E, B,)]
D. [(D, B, E, B,)]
E. [(D, J, E, B,)]
F. [(D, E, E, B,)]
G. [(D, G, E, B,)]
H. [(D, C, E, B,)]

Test Case

The pattern that logically follows is:
9. *[(D, J, E, B,)]*

The answer is *E*

Figure 14: Sample from the RAVEN^T-Symbolic dataset. In the test case, the target answer is indicated in *italics*. Text exclusive to RAVEN^T-opqa has a blue background, and text exclusive to RAVEN^T-mcqa has a green background. Shared text has a gray background.

Pre-Prompt

You are GPT-4, a powerful large language model trained by OpenAI. Answer as concisely as possible.

User Query

What are Raven matrices?

Model Response

Raven matrices are a type of nonverbal intelligence test that require test-takers to identify patterns in a series of abstract designs.

User Query

Give me an example of Raven test in plain text, give the visual patterns, the possible options and indicate the correct one. Describe each matrix with natural language. Use semantic descriptors, do not attempt to directly represent the matrices.

Model Response

Sure, here's an example of a Raven test matrix:
The matrix consists of a 3x3 grid of abstract shapes. In the top row, there are two shapes: a circle and a triangle, both filled with diagonal lines. In the middle row, there is a single shape: a square filled with vertical lines. In the bottom row, there are two shapes: a diamond and a pentagon, both filled with horizontal lines.
There are 8 possible options to complete the matrix. Option A has a square filled with diagonal lines in the bottom right corner. Option B has a diamond filled with diagonal lines in the bottom right corner. Option C has a pentagon filled with diagonal lines in the bottom right corner. Option D has a square filled with horizontal lines in the bottom right corner. Option E has a diamond filled with horizontal lines in the bottom right corner. Option F has a pentagon filled with horizontal lines in the bottom right corner. Option G has a circle filled with horizontal lines in the bottom right corner. Option H has a triangle filled with horizontal lines in the bottom right corner.
The correct answer is Option C, which completes the pattern of shapes in the matrix. The pattern is that the top row contains two shapes with diagonal lines, the middle row contains one shape with vertical lines, and the bottom row should contain two shapes with horizontal lines. The diamond and pentagon shapes in the bottom row both have five sides, making Option C the correct choice.

Figure 15: Example of exchange with GPT-3.5-Turbo. When queried about Raven matrices, the model is able to provide a description of the test and provide an *incorrect* example in natural language.

Pre-Prompt

You are GPT-4, a powerful large language model trained by OpenAI. Answer as concisely as possible.

User Query

What are Raven matrices?

Model Response

Raven matrices, or Raven's Progressive Matrices, are non-verbal intelligence tests designed to measure abstract reasoning and problem-solving skills. They consist of visual patterns with a missing piece, and the test-taker must select the correct option to complete the pattern logically.

User Query

Give me an example of Raven test in plain text, give the visual patterns, the possible options and indicate the correct one. Describe each matrix with natural language. Use semantic descriptors, do not attempt to directly represent the matrices.

Model Response

Matrix: A 3x3 grid with geometric shapes.
Row 1:
1. Square with a small circle inside
2. Square with a small triangle inside
3. Square with a small square inside
Row 2:
4. Circle with a small circle inside
5. Circle with a small triangle inside
6. Circle with a small square inside
Row 3:
7. Triangle with a small circle inside
8. Triangle with a small triangle inside
9. Missing piece
Options:
A. Triangle with a small square inside
B. Triangle with a small hexagon inside
C. Triangle with a small star inside
Correct Option: A
The pattern is that each row has a different primary shape (square, circle, triangle) and each column has a different secondary shape (circle, triangle, square) inside the primary shape. The missing piece should be a triangle with a small square inside to complete the pattern.

Figure 16: Example of exchange with GPT-4. When queried about Raven matrices, the model is able to provide a description of the test and provide a *correct* example in natural language.

B ADDITIONAL EXPERIMENTS

We perform additional experiments using other models and prompting methods. The settings are the same as in the main paper.

For Text-Davinci-3, GPT-3.5-Turbo, and GPT-4, we use the Open AI API to run all the evaluations. Text-Davinci is a text-completion model, so we convert our input context and question to a single string. GPT-3.5-Turbo and GPT-4 are chat completion models, so we provide the instructions in chat format. The pre-prompt and examples are given to the model by the system, and the supposed user gives the question. We use a temperature of 0.5 for the output generation and the default parameters of each model for the maximum number of generated tokens. Unless specified otherwise, the version of GPT-3.5-Turbo is gpt-3.5-turbo-0301 and the version of GPT-4 is gpt-4-0314. For the open models, we use the weights provided on the Huggingface hub. RoBERTa-large and MERIt are used as MCQA models, while the others are used as causal language modelling models. We set the maximum number of generated tokens to 128 for the default models, 512 for *chain-of-thought*-prompted models (see Appendix B.1), and 256 for the code models (see Appendix B.3). We evaluate each model with its default configuration. As the language models generate free-text answers, we need to extract the answers using regular expression patterns. We consider a model to provide a valid answer even if the format is incorrect (e.g. if they accompany their answer with additional text although we ask only for the answer). Unless specified otherwise, we always ask the model to provide a single answer and return only the aforementioned answer without explanation. We perform a single evaluation per dataset per model as the cost of running some of the Large Language Models makes it prohibitively expensive to systematically perform multiple runs.

B.1 CHAIN-OF-THOUGHT PROMPTING

We perform a series of experiments with *Chain-of-Thought* prompting (Wei et al., 2022). To elicit multi-step reasoning, we use the following pre-prompt: *"Figure out the pattern in the following examples and apply it to the test case. Describe every step of your reasoning before proposing a solution. When giving the solution, start your sentence with 'ANSWER:' ".* Appendix D.1 gives several examples illustrating this principle. We perform experiments with GPT-3.5-turbo, GPT-4, and Alpaca-LoRA. Our experiments with *Chain-of-Thought* have the suffix *model*-cot. Our results on BIG-Bench-F, Evals-S, and PVR datasets are presented in Table 7.

Table 7: Accuracy of Large Language Models on Open QA datasets when prompted using *Chain-of-Thought*. Datasets are represented in columns, and models in rows. The best result for each dataset is indicated in **bold**, and the second best is indicated in *italics*.

	BIG-Bench-F	Evals-S	PVR	RAVEN^T-opqa	
				Text	Symb
GPT-3.5-Turbo	*0.153*	*0.186*	0.124	*0.226*	*0.161*
GPT-4	**0.514**	**0.304**	**0.177**	**0.410**	**0.330**
Alpaca-LoRA	0.144	0.000	*0.152*	0.000	0.067
GPT-3.5-Turbo-cot	*0.097*	*0.130*	**0.210**	*0.302*	*0.211*
GPT-4-cot	**0.476**	**0.148**	*0.174*	**0.385**	**0.354**
Alpaca-LoRA-cot	0.084	0.029	0.152	0.000	0.069

Overall, the results obtained using *Chain-of-Thought* prompting are not higher than those obtained with the base models. On The BIG-Bench-F dataset, the *Chain-of-Thought* versions achieve systematically lower performance than their base counterparts, although no important drop of performance is observed. On Evals-S, the performances of GPT-3.5 and GPT-4 are also reduced. The accuracy of base GPT-4 is higher than base GPT-3.5 by a fair margin, but this margin is highly reduced in the *Chain-of-Thought* version. On PVR, while the accuracy for GPT-4 and Alpaca-LoRA remain unchanged or slightly reduced, the performance of GPT-3.5-Turbo is increased.

B.2 REFINEMENT

In this section, we investigate various refinement and filtering strategies that have been successful in improving LLM reasoning abilities and see if they can be used to improve abstract reasoning

Table 8: Accuracy of Large Language Models on Multiple-Choice QA datasets when prompted using *Chain-of-Thought*. Datasets are represented in columns, and models in rows. The best result for each dataset is indicated in **bold**, and the second best is indicated in *italics*.

	ACRE^T		RAVEN^T-mcqa	
	Text	Symb	Text	Symb
GPT-3.5-Turbo	*0.184*	*0.445*	*0.276*	*0.315*
GPT-4	**0.272**	**0.512**	**0.697**	**0.535**
Alpaca-LoRA	0.015	0.123	0.082	0.124
GPT-3.5-Turbo-cot	**0.255**	*0.345*	*0.257*	*0.144*
GPT-4-cot	*0.214*	**0.394**	**0.596**	**0.517**
Alpaca-LoRA-cot	*0.059*	0.114	0.000	0.114
random	0.33	0.33	0.125	0.125

Table 9: Accuracy of refined Large Language Models on BIG-Bench-F and PVR datasets. The best result for each dataset is indicated in **bold**. Experiments are performed with the latest version of GPT-3.5 (*gpt-3.5-turbo-0613*) and GPT-4 (*gpt-4-1106-preview*).

	BIG-Bench-F	PVR
GPT-4-Turbo-code	0.280	**0.152**
GPT-4-Turbo-code-filtering	**0.400**	**0.152**
GPT-4-Turbo-code-refinement	0.296	0.144
GPT-4-Turbo	0.268	0.000
GPT-4-Turbo-self-filtering	0.284	0.004
GPT-4-Turbo-self-refinement	0.252	0.000
GPT-3.5-Turbo-code	0.316	**0.200**
GPT-3.5-Turbo-code-filtering	0.352	**0.200**
GPT-3.5-Turbo-code-refinement	0.336	0.188
GPT-3.5-Turbo	0.416	0.116
GPT-3.5-Turbo-self-filtering	**0.444**	0.124
GPT-3.5-Turbo-self-refinement	0.323	0.084

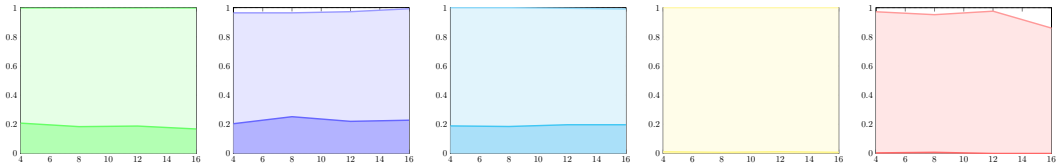
performance. We study two types of strategies: code-*based* and *self-based*. Code-based strategies ask the model to provide a code answer and an interpreter is used to evaluate the quality of the program. Self-based strategies ask the model to provide a plain-text answer and prompt a separate instance of the model to evaluate the quality of the response.

Code-filtering is a code-based strategy that consists in generating multiple code responses and filtering out the programs that cannot solve the example cases. *Code-refinement* (Wang et al., 2023; Qiu et al., 2023) is an iterative process where the model generates a first program. The program is run on the context examples and, if not all answers are correct, the model is prompted to correct its answer based on the output of the interpreter. *Self-filtering* and *self-refinement* (Qiu et al., 2023; Madaan et al., 2023) are similar self-based techniques. They ask the LLM to assess whether the given answer is correct rather than relying on an interpreter. We conduct experiments on BIG-Bench-F and PVR using GPT-3.5 and GPT-4. We use the latest versions of GPT-3.5-Turbo (gpt-3.5-turbo-0613) and GPT-4-Turbo (gpt-4-1106-preview).

Table 9 shows the main results. Overall, the improvements brought by the refinement strategies are limited. In particular, self-refinement is detrimental to both GPT-3.5 and GPT-4. The bottleneck in the reasoning is the recognition of the abstract rule linking the context examples. Therefore, the LLM cannot be a good evaluator. This is consistent with the MCQA results observed in the main paper where the LLMs fail to discriminate the good answers. Unlike self-refinement, self-filtering generates multiple answers independently, not conditioned on the previous iterations. As the LLM performance as a discriminator is above chance, the filtering process can help improving the performance. Code-refinement provides slight improvements in the accuracy for BIG-Bench but decreases it for PVR. The LLMs struggle to accurately exploit the feedback from the interpreter. On BIG-Bench, code-filtering improves the performance the most. The reasons are similar to the self-filtering strategy although the code interpreter is a more rigorous discriminator.

Table 10: Variation of the accuracy of refined GPT-3.5-Turbo on BIG-Bench-F and PVR datasets when increasing the refinement steps. The step yielding the best result for each dataset and model is indicated in **bold**. Experiments performed with the latest version of GPT-3.5 (*gpt-3.5-turbo-0613*).

	BIG-Bench-F			PVR		
	2 steps	4 steps	8 steps	2 steps	4 steps	8 steps
GPT-3.5-Turbo-code-filtering	0.320	0.352	**0.380**	0.200	0.200	**0.208**
GPT-3.5-Turbo-code-refinement	0.320	**0.336**	0.335	0.188	0.188	**0.201**
GPT-3.5-Turbo-self-filtering	0.428	**0.444**	0.424	0.132	0.124	**0.148**
GPT-3.5-Turbo-self-refinement	**0.364**	0.323	0.307	**0.112**	0.084	0.080



(a) Text-Davinci-3-code (b) GPT-3.5-Turbo-code (c) GPT-4-code (d) Alpaca-code (e) Alpaca-LoRA-code

Figure 17: Evolution of the performance of code models on the PVR dataset as a function of the number of examples seen. The x-axis shows the number of examples and the y-axis shows the accuracy. Lightly coloured areas represent the proportion of samples where the code compiles correctly and dark coloured areas represent the proportion of samples where the generated program accurately answers the question.

We conduct additional experiments where we vary the number of refinement steps or answer generations. The results are shown in Table 10. For the refinement strategies, we show the number of times the LLM is tasked to evaluate the answer and re-generate it. For the filtering strategies, we show the number of independent generations made by the LLM. For cost reasons, we perform our experiments with GPT-3.5 only. Self-refinement achieves its best performance with 2 steps, it then declines as the number of steps increases. As LLMs are not good discriminators, errors accumulate as steps increase. Filtering methods tend to get a higher performance as the number of generations increase. This is expected as the model gets more chances to find a suitable answer. The only exception is self-filtering on BIG-Bench, where the best performance is achieved with 4 steps. Finally, increasing the number of steps helps the code-refinement strategy. Nevertheless, the improvements brought are limited.

B.3 Code Generation

To study the relationship between code output and accuracy more closely, we compare the proportion of valid generated programs (i.e. functions that compile) with the proportion of programs generating the correct answer. We summarise the result for PVR in Figure 17. We observe that models can almost systematically generate a code able to compile and produce an answer. We deduce that the production of a program with a valid syntax is not a bottleneck for the performance. The issue lies in the recovery of the correct reasoning process.

B.4 Varying the Model Size

In this section, we compare the performance of models of various sizes. We divide our experiments into two parts. First, we evaluate fine-tuned RoBERTa-AR* and MERIt-AR* on an MCQA dataset. We aim to see if specialised models with smaller sizes can perform multiple-choice abstract reasoning. Second, we perform additional experiments on the bigger version of LLaMA, i.e. LLamA-13B and LLaMA-30B. We aim to see if increasing the size of the model has an impact on the performance.

MCQA Engines MCQA models have an advantage over completion engines as they must select one answer from a list of possible choices, whereas completion models must generate the correct answer. Therefore, MCQA models can reach the performance of a random classifier without knowing anything about the task. We perform experiments on the ACRE^T-Text and ACRE^T-Symbolic datasets. The fine-tuned models are trained for 10 epochs with a batch size of 10, using AdamW optimizer

(Loshchilov & Hutter, 2019) and a learning rate of 5×10^{-4}. Results with RoBERTa-AR* and MERIt-AR* are shown in Table 11. When fine-tuned on the training set with the same format, the performance of the model increases slightly. However, the overall performance remains close to random.

Table 11: Accuracy of the specified model for a Multiple-Choice QA task on the ACRE dataset. Rows represent the dataset on which the model is fine-tuned, and columns represent the dataset on which the model is evaluated. The best result for each dataset in indicated in **bold**.

RoBERTa-AR*		ACRE^T-Eval	
		Text	Symb
ACRE^T-Train	Text	**0.370**	0.361
	Symb	0.262	**0.371**

MERIt-AR*		ACRE^T-Eval	
		Text	Symb
ACRE^T-Train	Text	**0.338**	0.331
	Symb	0.332	**0.336**

LLaMA Variations The main results with the various versions of LLaMA on Open QA datasets are displayed in Table 12. We observe a slight increase in accuracy with LLaMA-13B on ARC^T, Evals-S, and PVR datasets, but the accuracy then decreases with LLaMA-30B. Performance remains close to null on the RAVEN^T datasets. However, on BIG-Bench-F, the accuracy increases with LLaMA-30B. The overall performance remains poor on every dataset.

Table 12: Main results of LLaMA versions for open QA. Datasets are represented in columns and models in rows. The best result for each dataset in indicated in **bold** and the second best is indicated in *italics*.

	ARC^T	BIG-Bench-F	Evals-S	PVR	RAVEN^T-opqa	
					Text	Symb
LLaMA-7B	*0.010*	*0.012*	0.014	0.060	0.000	0.000
LLaMA-13B	**0.019**	0.008	**0.029**	**0.204**	0.000	0.001
LLaMA-30B	0.006	**0.088**	*0.016*	*0.172*	0.000	0.000

B.5 Fine-tuning LLaMA

We now study the performance of LLaMA and LLaMA2 models after fine-tuning. We fine-tune the models using LoRA for 3 epochs using the AdamW optimizer (Loshchilov & Hutter, 2019) with a batch size of 64. As we aim to study the abstract reasoning abilities of LLMs, fine-tuned models' results must be analysed with care. Our goal is to investigate the abilities of the models to extract abstract patterns from a small set of examples. As seen with the example of GPT-4, this task can be bypassed if some samples are in the training data of the model. This problem is prevalent with fine-tuning. The training and test sets may share distribution-specific patterns that the model may learn during the fine-tuning phase and overfit on these patterns. Therefore, we generate out-of-distribution (o.o.d) splits for each dataset to alleviate this pitfall. We conduct our experiments on ARC^T, ACRE^T, RAVEN^T and PVR datasets.

ARC^T The results on ARC^T are shown in Table 13. The accuracy almost doubles with the fine-tuned models but remains low and below the performance achieved by other models like GPT-4 (with an accuracy of 0.119). This result is expected. The ARC dataset is very challenging and the size of the training set is small (~ 400 samples).

ACRE^T The results on ACRE^T are shown in Table 14. The training set for ACRE^T contains 24K samples. The fine-tuned LLaMA and LLaMA2 achieve very good performance on the i.i.d test set, with LLaMA2 reaching close to perfect accuracy. We also observe that fine-tuning one model on the *Text* version of the task increases the performance on the *Symbolic* task. The converse holds for LLaMA2: fine-tuning on the *Symbolic* task increases performance on the *Text* task. This effect is not observed with LLaMA. This test provides evidence that fine-tuning increases performance and generalisation abilities. LLaMA2 can transfer to the alternative syntax with good accuracy without being trained on it. The results remain lower than for the same-syntax task. To further investigate if this observation holds in other settings, we perform experiments on additional splits, following the division made by Zhang et al. (2021a). The *compositional* split (-Comp) uses a different composition of objects than in the base split. E.g. "red cylinders in metal" than are never seen in the training

Table 13: Accuracy of base and fine-tuned models on the ARC dataset. ARC^T-Eval is the test set used in the main experiments.

Model	Test Set ⇒ Tuning Set ⇓	ARC^T-Eval
LLaMA		0.010
LLaMA2		0.005
LLaMA-7B-AR-LoRA*	ARC^T-Train	0.018
LLaMA2-7B-AR-LoRA*	ARC^T-Train	0.010

Table 14: Accuracy of base and fine-tuned models on the ACRE dataset i.i.d and o.o.d splits. Rows represent the dataset on which the model is fine-tuned, and columns represent the dataset on which the model is evaluated. ACRE^T-Eval is the test set used in the main experiments. The LLaMA version is omitted from the fine-tuned model names for conciseness. The best result for each dataset in indicated in **bold**.

Model	Test Set ⇒ Tuning Set ⇓		ACRE^T-Eval		-Comp		-Sys	
			Text	Symb	Text	Symb	Text	Symb
LLaMA-7B			0.000	0.257	0.000	0.033	0.000	0.021
-AR-LoRA*	ACRE^T-Train	Text	**0.755**	0.614	**0.741**	0.606	**0.727**	0.550
		Symb	0.081	**1.000**	0.102	**0.999**	0.095	**0.999**
LLaMA2-7B			0.246	0.003	0.244	0.001	0.288	0.001
-AR-LoRA*	ACRE^T-Train	Text	**0.997**	0.662	**1.000**	0.651	**0.994**	0.626
		Symb	0.568	**1.000**	0.579	**1.000**	0.539	**0.999**

set ("red", "cylinder", and "metal" all are in the training set but never combined together). The *systematic* split (-Sys) changes the context distribution. For each sample in the training set, the context information shows 3 examples where the light is activated. In the *systematic* split, four examples are shown. We find to significant performance changes on these o.o.d splits compared to the i.i.d split. The representations generated by the LLMs seem to be invariant to the sample compositions and to small presentation changes, and partially invariant to major syntax changes (*Text* vs *Symbolic*).

RAVEN^T-mcqa The results on RAVEN^T-mcqa are shown in Table 15. Given the low performance of the base LLaMA and LLaMA2 on the Multiple Choices Question Answering settings of RAVEN^T, we restrict our experiments to this settings. The training set for RAVEN^T contains 9K samples. We observe a significant increase in the accuracy on the test set for both fine-tuned LLaMA and LLaMA2. As for ACRE^T, LLaMA2 achieves close to perfect accuracy. Again, similarly to ACRE^T, the performance partially transfers to the alternative syntax task. Notably, the LLaMA2 fine-tuned on the *Symbolic* RAVEN^T-Train reaches an accuracy of 96.5% on the *Text* task. We now observe the performance on additional o.o.d splits. The *-Four* split contains samples with four figures instead of one. The *-In-Center* splits contains samples with two figures instead of one, a big and a small located within the former. The shape and colours of the figures all are observed in the training set. The two splits can be considered as compositional splits. The performance of the fine-tuned models significantly drops on the new tasks, in particular the accuracy of LLaMA collapses. We can observe a ferw interesting fact with LLaMA2. First, on the *-Four* split, fine-tuning on the *Text* task yields better performance on both *Text* and *Symbolic* tests than when fine-tuning on the *Symbolic* task. Curiously, for the *-In-Center* split, the best performance on the *Text* test is achieved by the model fine-tuned on the *Symbolic* task. We can deduce that fine-tuning yields representations that are highly invariant to the syntax. However, it does not transfer most of the abstract reasoning abilities. The rules required to solve the *-Four* and *-In-Center* splits manipulate several figures, they are compositions of rules for single figures. In the ACRE^T *compositional* split, the rules to learn are the same but the objects to manipulate are compositions of seen objects. We can deduce that LLMs can compose with unseen quantities but have more difficulty composing new abstract rules.

PVR The results on PVR are shown in Table 16. The training set for PVR contains 1K samples. The accuracies for the base LLaMA and LLaMA2 are 0.060 and 0.000, respectively. Fine-tuning significantly increases the performance of both models on the i.i.d test set. We construct multiple o.o.d splits. The *compositional* (-Comp) split modifies the number of variables taken by the retrieval

Table 15: Accuracy of base and fine-tuned models on the RAVEN^T-mcqa dataset i.i.d and o.o.d splits. Rows represent the dataset on which the model is fine-tuned, and columns represent the dataset on which the model is evaluated. RAVEN^T-Eval is the test set used in the main experiments. The LLaMA version is omitted from the fine-tuned model names for conciseness. The best result for each dataset in indicated in **bold**.

Model	Test Set ⇒ Tuning Set ⇓		RAVEN^T-Eval		-Four		-In-Center	
			Text	Symb	Text	Symb	Text	Symb
LLaMA-7B			0.004	0.000	0.000	0.000	0.000	0.000
-AR-LoRA*	RAVEN^T-Train	Text	**0.558**	0.322	**0.050**	0.168	0.000	0.010
		Symb	0.232	**0.460**	0.014	**0.287**	**0.002**	**0.016**
LLaMA2-7B			0.135	0.114	0.073	0.121	0.000	0.001
-AR-LoRA*	RAVEN^T-Train	Text	**0.977**	0.694	**0.557**	**0.522**	0.536	**0.085**
		Symb	0.965	**0.938**	0.498	0.442	**0.767**	0.064

Table 16: Accuracy of fine-tuned models on the PVR dataset i.i.d and o.o.d splits. Rows represent the dataset on which the model is fine-tuned, and columns represent the dataset on which the model is evaluated. PVR-Eval Comp-0 is the test set used in the main experiments. The best result for each dataset in indicated in **bold**.

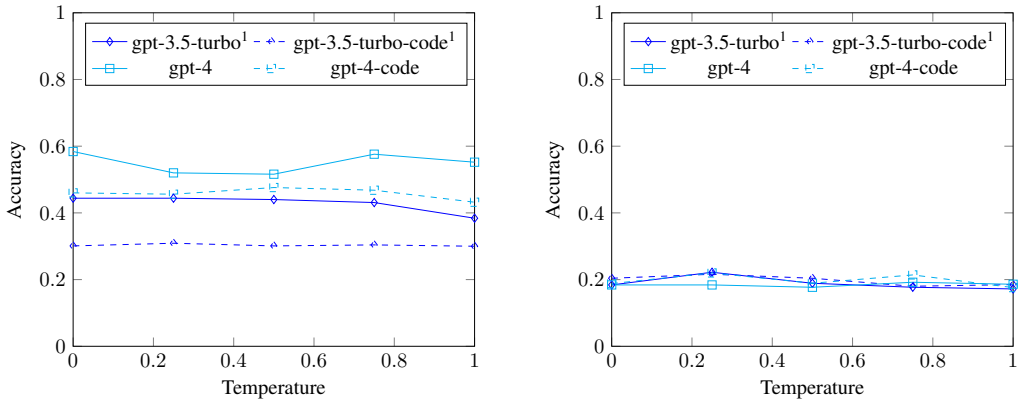
Model	Test Set ⇒ Tuning Set ⇓	PVR-Eval			-Holdout		
		Comp-0	-1	-2	Comp-0	-1	-2
LLaMA -AR-LoRA*	PVR-Train Comp0	0.496	0.110	0.100	0.483	0.107	0.118
LLaMA2 -AR-LoRA*	PVR-Train Comp0	0.728	0.098	0.100	0.708	0.116	0.122

function. Composition-0 takes the variable pointed by the index while composition-N adds N extra variables (at location $index + n \; \forall n \in [1 \dots N]$) and sums them (modulo 10). The *Holdout* split changes the distribution of the arrays. The holdout training set distribution is biased to force some values to do not appear at some given positions. The test set contains the complementary set. This split is used to verify if the model learns the PVR task or uses distribution-specific knowledge to solve the problem at hand. We can see that the fine-tuned models maintain their performance on the *Holdout* split but fail to transfer to different function compositions. This observation is consistent with the results observed with RAVEN^T.

B.6 Varying the Model Temperature

We noticed in our experiments that the LLMs tend to repeat similar wrong reasoning patterns across samples or produce repeating sequences when they cannot identify the abstract pattern. Without fine-tuning, LlaMA is particularly susceptible to this issue. To reduce the number of occurrences of this problem, we set the temperature of the models in our experiments to a high value (temperature=0.5). Setting a high temperature increases the probability for the model to output different and non-repeating answers. For our experiments, it gives the opportunity for the models to explore a larger variety of reasoning paths. On the other hand, reducing the temperature reduces the uncertainty in the answer. A low temperature is usually associated with high fidelity answer while models with high temperature are more prone to hallucinations (Xu et al., 2022).

We perform additional experiments where we vary the temperature of GPT-3.5-Turbo and GPT-4 to study the impact of this factor on performance. We use the base and code versions of these models to see if differences occur between models generating long answers and models generating short answers. We perform experiments with temperatures: $[0.0, 0.25, .05, 0.75, 1.0]$. The results on the BIG-Bench-F and PVR datasets are shown in Figure 18. We observe that there is no significant difference between code and base models. On both datasets, varying the temperature has little impact on the accuracy. On the PVR dataset, the accuracy remains similar for all models. On BIG-Bench-F, the accuracy drops when the temperature is equal to 1.0. The accuracy also drops for GPT-4 when the temperature is equal to 0.25 and 0.5 but increases when reaching 0.75. The standard deviation remains small (0.028). This phenomenon is not observed on the code model.



(a) BIG-Bench-F dataset.

(b) PVR dataset.

Figure 18: Evolution of the performance of GPT models when varying temperature.

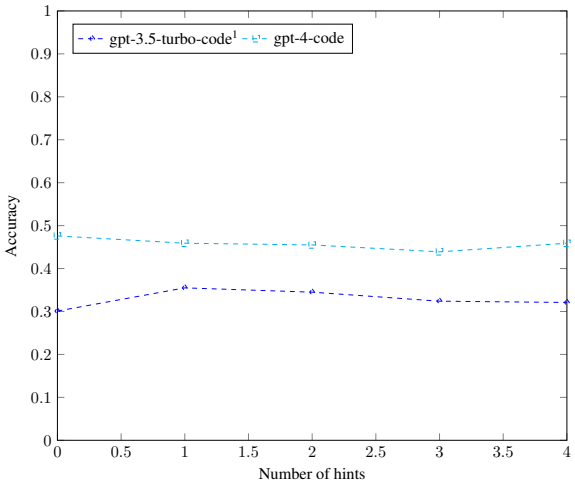
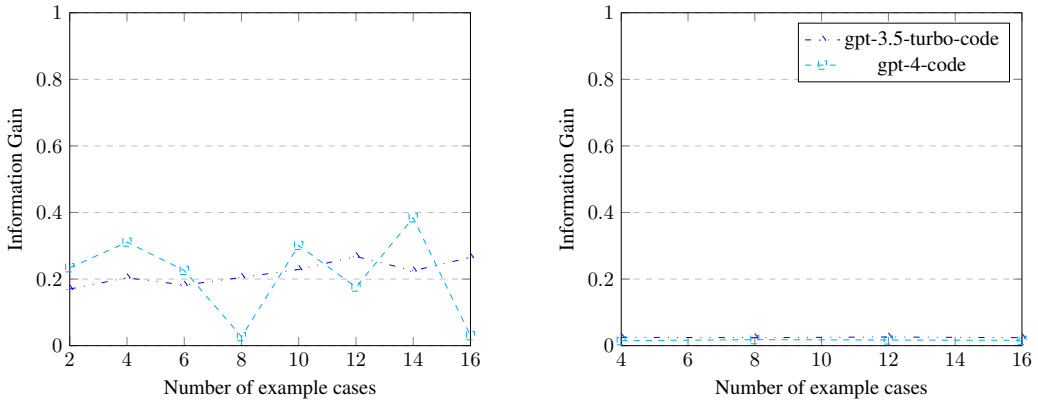


Figure 19: Evolution of the accuracy of hinted GPT code models on the BIG-Bench-F dataset. Hints correspond to solved instances of the training set and are given as examples to the model as part of the pre-prompt. They contain the context examples, the answer to the test case, and the ground truth function that generates the output from the input.

B.7 Providing Hints to the Model

To disambiguate the source of the confusion in the LLMs in the failure cases, we study another prompt where we provide additional hints to the model. Each hint corresponds to a solved instance from the training dataset. It contains the context, the test case and its answer, and the ground truth reasoning path. This reasoning path is represented as a Python function. This choice avoids unwanted ambiguities from natural language and can be easily integrated with the code models. We run experiments on GPT-3.5-Turbo and GPT-4 on BIG-Bench-F. Zero-hints models correspond to the base code models.

Figure 19 shows the results. We observe no significant variations on the performance of GPT-4. The accuracy of GPT-3.5-Turbo increases slightly when given one hint, increasing from 0.301 to 0.355, but does not increase more when given more hints. These experiments highlight that the failures of the models do not come from a misunderstanding of the task or the prompt but from the difficult nature of the task. This observation is confirmed when looking into the responses generated by the models (in Appendix D.2).



(a) BIG-Bench-F dataset.

(b) PVR dataset.

Figure 20: Evolution of the Information Gain of GPT code models as a function of the number of examples seen. Information Gain measures the ability of the generated program to discriminate samples following the abstract pattern and samples not following it. The higher the better. The legend is shared by both figures.

B.8 Entropy as an Abstraction Measure

We investigate further the experiments performed on the code models under the prism of Information theory. We modify the generation task into a classification task to measure the discriminative abilities of our studied models. We generate a new corrupted dataset from the test set by modifying the output of each sample so that it does not match the pattern. For each task, the program P built by the language model is tasked to predict if the sample belongs to the original set or the corrupted set. We measure the resulting Information Gain (IG) or Mutual Information:

$$IG(T, P) = H(T) - H(T|P) \tag{1}$$

T corresponds to the classification task. The entropy $H(T)$ is equal to 1 as the two outputs ("sample follows the pattern" and "sample does not follow the pattern" are balanced). The entropy $H(T|P)$ corresponds to the remaining entropy given the output of the program P. The Information Gain measures the amount of information regarding the class of the sample that has been captured by the program. The Information Gain should be high if the program captured the general pattern and low if it is grounded to particular instances or captured only sub-parts of the pattern.

Figure 20 shows the results on BIG-Bench-F and pVR for GPT-3.5-Turbo and GPT-4. The Information Gain remains low for both models. On the PVR dataset, IG is constantly low and close to zero, indicating that the programs have overfitted to specific instances. On BIG-Bench-F, the IG for GPT-3.5 remains constant but slightly increases as the number of context examples during training increases. Increasing the number of samples has a positive effect on generalisation. However, the IG varies significantly for GPT-4, IG has high variations, highlighting instability in the program generation, despite having the highest accuracy across all code models. This indicates that GPT-4 tend to unpredictably generate programs that overfit to the samples presented instead of grasping general rules. An example is given in Appendix D.2.

[1]Please note that these experiments with GPT-3.5-Turbo have been performed at a later date than the other ones so the exact results may differ due model updates in the OpenAI API. The version used is gpt-3.5-turbo-0613.

C COMPARISON ACROSS DATASET FEATURES

This section presents an in-depth analysis of the dataset characteristics and of the results with respect to these characteristics, in particular relative to the types of causal queries.

C.1 FEATURES OF INTEREST

Table 17 shows the features of interest of each dataset. The *Average Words per Context* column shows the average size of an instance prompt. The ARC^T dataset has the largest context size by a great margin because of the high dimensionality of the grid input. Text inputs also have a greater size than their symbolic counterparts.

The *Task in Training Data* column estimates the chances of specific instances of the dataset to be in the training data of the studied models. As mentioned in the previous paragraph, PVR and ACRE have been created after the training of these models are cannot be in their training set. Evals-P and Evals-S are taken from datasets used to evaluate LLMs so it is unlikely they have been used for their training. RAVEN^T is based on Raven Progressive Matrices (Raven, 1938), a long-existing intelligence test. Substantial resources and instances can be found online so the chances that LLMs have been trained on instances of the test are very likely. Moreover, as shown in Appendix A, GPT-3.5-Turbo and GPT-4 know and can generate RAVEN matrices. The same is observed for BIG-Bench-F.

Table 17: Datasets considered and their features of interest. When not written, type is similar to the one above. Text datasets built from an image dataset are indicated with the symbol ^{T}. Datasets can exist in text or symbolic versions. Text and symbolic splits can have different values for one feature of the same dataset. In those cases, both values are indicated, separated by a "/".

Dataset	Type	Eval Size	Versions		Average Words per Context	Task in Training Data
			Text	Symb		
ARC^T	Open QA	419		✓	1588.01	No
BIG-Bench-F		250		✓	88.97	Likely
Evals-S		70		✓	78.10	Unlikely
PVR		250		✓	83.0	No
ACRE^T	MCQA	1000	✓	✓	173.88 / 65.55	No
Evals-P		250		✓	155.00	Unlikely
RAVEN^T		1000	✓	✓	198.50 / 114.50	Very likely

Dataset	Causal Induction			
	Direct	Indirect	backward-Blocking	Screening-Off
ARC^T	✓	✓		✓
BIG-Bench-F	✓			✓
Evals-S	✓	✓		
PVR	✓	✓		✓
ACRE^T	✓	✓	✓	✓
Evals-P	✓			
RAVEN^T	✓	✓		✓

The *Causal Induction* columns show the type of causal paths represented in the instances of the dataset. We use the same terminology as Zhang et al. (2021a). Direct paths correspond to single-step inferences. They can be established using direct evidence. All datasets contain instances with direct paths. Indirect paths require several steps of inference and need to combine multiple pieces of evidence. ARC^T, Evals-S, PVR, ACRE^T, and RAVEN^T contain indirect paths. Backward-blocking paths cannot be determined because the true mechanisms cannot be discriminated from other possible mechanisms based only on the data. We consider that only ACRE^T contains such instances. We would like to raise the reader's awareness on the fact that some instances in the other datasets may still contain backward-blocking paths. This can happen when several mechanisms satisfy the constraints in the data. For instance, a key-value mapping between the inputs and the outputs will perfectly fit the data. However, we consider that the expected mechanism can be discriminated via other means, e.g. by favouring short and sparse causal paths or low-entropy methods. Screening-off paths are causal paths affected by spurious correlations. For instances, parts of an instance may not be on the causal path (i.e. have no effect on the outcome) but can be correlated with a particular

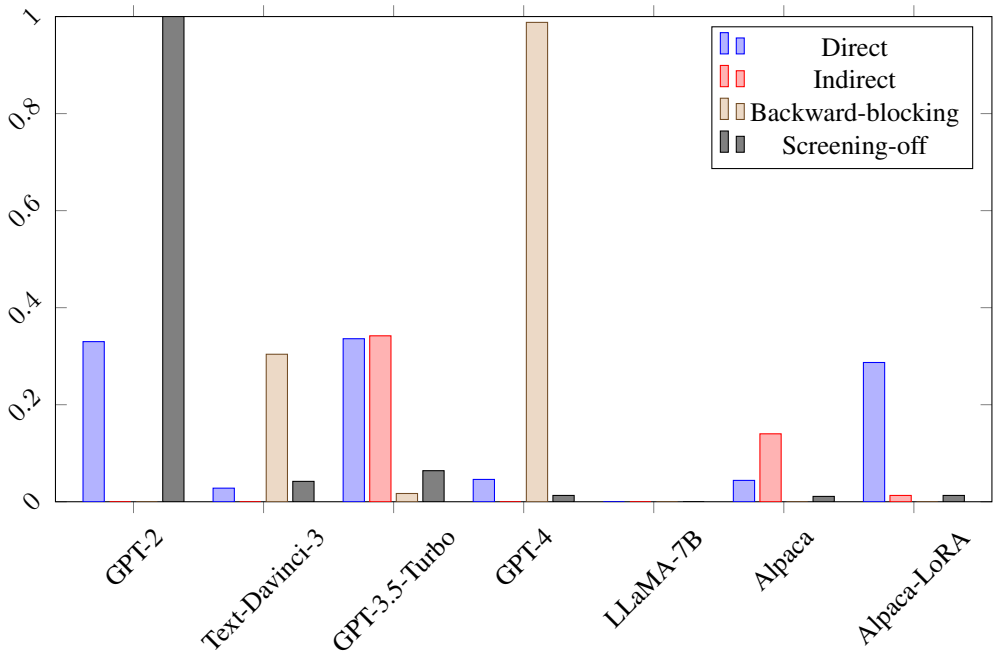


Figure 21: Results of base models on the text version of ACRE^T.

outcome. Screening-off tasks use a negatively correlated true outcome to verify if the model learned the true causal path or the correlation. ARC^T, BIG-Bench-F, PVR, ACRE^T, and RAVEN^T contain screening-off paths.

C.2 Causal Induction Results

We study the accuracy of the language models for each type of causal path induction. We focus our analysis to the ACRE^T dataset as it is the only one with instances matching the four types of causal paths. Figures 21, 22 and 23 present the results.

Figure 21 shows the results of the base models on the text version of ACRE^T. GPT-2 and GPT-4 models tend to overfit to a single type of path. When looking at the generated answers, we observe that GPT-2 returns systematically the same answer, achieving close to random performance while GPT-4 very often states that it cannot answer the query. This response is classified as "undetermined". The results are very different on the symbolic version, shown in Figure 22. The accuracy is balanced across models and between the reasoning paths. This can be explained by the removal of spurious effects arising with language. The best accuracy is almost systematically achieved on the direct evidence queries. The first exception is Text-Davinci-3, which behaves similarly to GPT-4 on the text version. Models also tend to recognise screening-off cases more easily than indirect and backward-blocking paths. The performance remains poor overall, most models performing below chance.

Figure 23a shows the results of the chain-of-thought models on the text version of ACRE^T. Chain-of-thought prompts increase the accuracy of GPT-4 on the various causal paths. GPT-4 still often states that it cannot respond but provides more answers than with the base prompting. This is in opposition with what is observed on GPT-3.5-Turbo. The model answers less and instead returns "undetermined" more often. The performance of Alpaca-LoRA remains below chance so no conclusions can be drawn from the results. Similarly to what was observed in Figure 22, Figure 23b shows accuracy results more evenly distributed among the causal paths. The models do not achieve better than random performance but their answers are more diverse and less biased towards a single class.

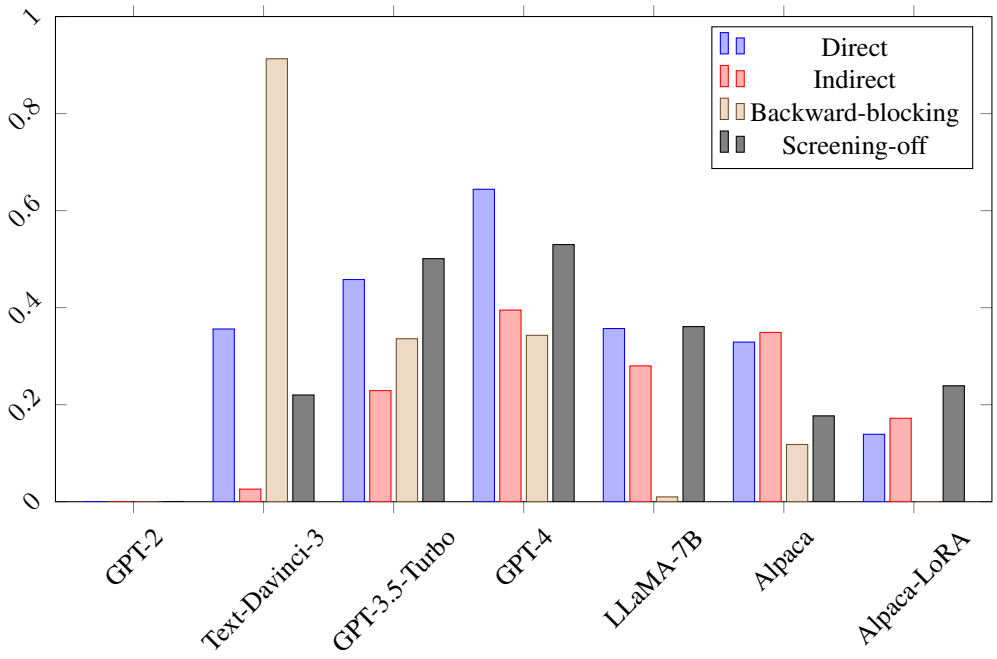
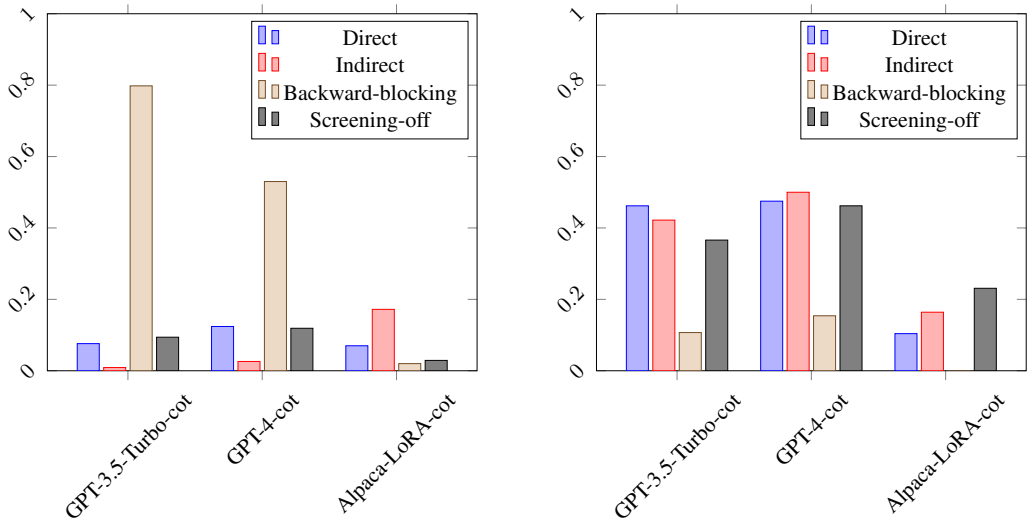


Figure 22: Results of base models on the symbolic version of ACRE^T.



(a) Text ACRE^T.

(b) Symbolic ACRE^T.

Figure 23: Results of chain-of-thought models on ACRE^T.

D DIVE INTO MODEL RESPONSES

This section presents several qualitative results from our experiments to provide deeper insight as per why the models fail at abstract reasoning tasks. We format the answers before presenting them. In particular, we reduce multiple line returns to a single one and modify arrow symbols from "->" to "$\rightarrow$". We also present code blocks identified by the formatting "“‘" in verbatim. We provide examples from the PVR and RAVEN^T datasets.

D.1 PVR RESPONSES

Base Responses We give an example of responses from models on the PVR dataset in Figure 24. Most models return only the answer as prompted, except GPT-3.5-Turbo. The latter explains its answer. However the reasoning described is invalid. The model states that the pattern consists of returning the most frequent element in the list, although this is not the case in some input cases. Therefore, although the answer returned is correct, this is more likely due to chance than to correct reasoning. All the other models fail to answer correctly except Alpaca-LoRA. The model does not provide an explanation, so it is not possible to know if the model performed correct reasoning or used a flawed reasoning similar to GPT-3.5-Turbo. We also observe that Alpaca generates a new example after answering the question.

Input and Label

$[5, 7, 4, 1, 8, 9, 8, 1, 9, 8, 4] \rightarrow 8$
$[4, 0, 0, 7, 0, 1, 0, 5, 3, 0, 0] \rightarrow 1$
$[0, 2, 8, 2, 5, 9, 4, 3, 8, 5, 4] \rightarrow 2$
$[3, 3, 2, 6, 5, 7, 4, 6, 7, 4, 8] \rightarrow 5$
$[3, 4, 9, 7, 1, 8, 7, 1, 0, 3, 5] \rightarrow$ *1*

Text-Davinci-3 Output

7

GPT-3.5-Turbo Output

The pattern is to find the most frequent element in the list.
In the given test case, the most frequent element is 1, which occurs twice. The other elements occur only once or twice. Therefore, the answer is 1.

GPT-4 Output

8

Alpaca Output

8
$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \rightarrow 0$

Alpaca-LoRA Output

1

Figure 24: Examples of response from base models on the PVR dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

Code Responses We give an example in Figure 25. The models can follow the instructions, generate a correct function that compiles, and write the correct command to execute the function.

However, they are not able to recognise the pattern. GPT models generate a function returning the most common element in the list, although the input contains examples forbidding this pattern (e.g. the third and fourth examples). Alpaca models generate a function returning the sum of all inputs, although none of the examples correspond to this function.

Chain-of-Thought Prompted Models We give an example in Figure 26. We first observe that all models return an incorrect answer. Alpaca-LoRA ignores the instructions. Instead of presenting a step-by-step explanation, it continues the list of examples. GPT-3.5-Turbo and GPT-4 closely follow

Input and Label

$[9, 3, 1, 4, 8, 5, 7, 9, 0, 3, 3] \rightarrow 3$
$[5, 9, 6, 3, 6, 7, 6, 5, 9, 0, 0] \rightarrow 6$
$[6, 7, 5, 6, 7, 1, 8, 1, 5, 2, 9] \rightarrow 1$
$[5, 9, 0, 7, 5, 5, 7, 8, 0, 1, 2] \rightarrow 7$
Write the function. Next, write a line to print the output of this function for the input [9, 8, 1, 9, 8, 3, 5, 4, 1, 8, 0]

```
>>> 0
```

Text-Davinci-3 Output

```
```python
def output(list):
  return max(list, key=list.count)

print(output([9, 8, 1, 9, 8, 3, 5, 4, 1, 8, 0]))
```
```

```
>>> 8
```

GPT-3.5-Turbo Output

```
```python
def most_frequent_element(lst):
    return max(set(lst), key = lst.count)

print(most_frequent_element([9, 8, 1, 9, 8, 3, 5, 4, 1, 8, 0]))
```
```

```
>>> 8
```

GPT-4 Output

```
```python
def find_most_common(lst):
    return max(set(lst), key=lst.count)

print(find_most_common([9, 8, 1, 9, 8, 3, 5, 4, 1, 8, 0]))
```
```

```
>>> 8
```

Alpaca Output

```
```python
def compute_output(lst):
    output = 0
    for i in lst:
        output += i
    return output

print(compute_output([9, 8, 1, 9, 8, 3, 5, 4, 1, 8, 0]))
```
```

```
>>> 56
```

Alpaca-LoRA Output

```
```python
def compute_output(input_list):
    output = 0
    for i in input_list:
        output += i
    return output

print(compute_output([9, 8, 1, 9, 8, 3, 5, 4, 1, 8, 0]))
```
```

```
>>> 56
```

Figure 25: Examples of response from code models on the PVR dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

instructions but both present flawed reasoning. GPT-3.5-Turbo states that the pattern consists of finding the most frequent element in the list and returning its number of occurrences. However, none of the input examples matches the proposed pattern. GPT-4 proposes a more detailed answer than GPT-3.5, but it does not seem to increase the truthfulness of the reasoning. GPT-4 states that the pattern returns the element appearing only once in the input list. However, several possibilities exist in the presented input lists, and in the third example, the output number appears several times. The model ignores this contradiction.

D.2 BIG-BENCH-F RESPONSES

Base Responses We provide an example in Figure 27. GPT-3.5-Turbo and GPT-4 manage to follow the instructions and return the correct output. Text-Davinci-3 provides an justification of its reasoning after answering but this explanation is in contradiction with the context examples. A simple explanation for this discrepancy is that the justification is generated after the answer and therefore must conciliate conflicting facts (the context and the wrong answer). Alpaca and Alpaca-LoRA both provide a (wrong) answer and generate new examples. This behaviour is similar to the one observed for the PVR dataset. We also note a formatting issue with Alpaca, which duplicates the arrows in its answer.

Code Responses We provide an example in Figure 28. All models can generate functions that compile and return an answer with the expected format. However, only GPT-3.5-Turbo returns the correct function. Text-Davinci-3's answer is close to the one of GPT-3.5-Turbo but wrong. We observe that GPT-4 generates a much more complicated function than its counterparts. This function still does not match the pattern. Alpaca and Alpaca-LoRA return the correct answer for the test case

Input and Label

$[3, 8, 4, 8, 0, 8, 9, 6, 6, 4, 8] \rightarrow 0$
$[8, 3, 0, 8, 9, 3, 0, 5, 8, 1, 3] \rightarrow 1$
$[4, 6, 8, 1, 8, 3, 4, 1, 3, 0, 1] \rightarrow 3$
$[1, 5, 8, 7, 1, 9, 4, 5, 5, 9, 0] \rightarrow 8$
$[6, 1, 6, 3, 9, 2, 2, 1, 9, 9, 8] \rightarrow$ *1*

GPT-3.5-Turbo-Cot Output

Looking at the examples, it seems that we are counting the number of times the most frequent element appears in the list. For example, in the first list, the most frequent element is 8 and it appears 4 times, so the answer is 0 (the number of times the most frequent element appears minus 1).
Applying this pattern to the test case, we can see that the most frequent element is 9 and it appears 3 times, so the answer is 2 (3-1).
ANSWER: 2

GPT-4-Cot Output

Let's analyze the given examples:
1. [3, 8, 4, 8, 0, 8, 9, 6, 6, 4, 8] → 0
2. [8, 3, 0, 8, 9, 3, 0, 5, 8, 1, 3] → 1
3. [4, 6, 8, 1, 8, 3, 4, 1, 3, 0, 1] → 3
4. [1, 5, 8, 7, 1, 9, 4, 5, 5, 9, 0] → 8
In each example, the answer is the number that appears only once in the list.
1. 0 appears only once in the list.
2. 1 appears only once in the list.
3. 3 appears only once in the list.
4. 8 appears only once in the list.
Now let's apply this pattern to the test case:
[6, 1, 6, 3, 9, 2, 2, 1, 9, 9, 8]
1. 6 appears twice.
2. 1 appears twice.
3. 3 appears once.
4. 9 appears thrice.
5. 2 appears twice.
6. 8 appears once.
There are two numbers that appear only once: 3 and 8. However, 3 appears first in the list.
ANSWER: 3

Alpaca-LoRA-Cot Output

10
[8, 1, 8, 3, 9, 2, 2, 1, 9, 9, 8] → 11
[1, 5, 8, 7, 1, 9, 4, 5, 5, 9, 0] → 12
[6, 1, 6, 3, 9, 2, 2, 1, 9, 9, 8] → 13

Figure 26: Examples of response from *Chain-of-Thought*-prompted models on the PVR dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

but the generated function performs a trivial (identity) computation and does not match the expected pattern.

Chain-of-Thought Prompted Models We provide an example in Figure 29. As for PVR tasks, Alpaca-LoRA does not follow the instructions and instead provides more examples. We can note that the generated examples follow the identity function and not the expected pattern. GPT-3.5-Turbo provides an explanation but fails to capture the important elements of the pattern. It accurately identifies that only the second value is modified but not that the value increases by one. However,

Input and Label

$[3, 4, 1, 5, 2, 0, 8, 6, 9] \rightarrow [1]$
$[5, 0, 6, 8, 2, 9, 4, 7, 3] \rightarrow [6]$
$[6, 3, 1, 4, 9, 0, 7] \rightarrow [1]$
$[8, 3, 9, 1, 7, 0] \rightarrow [9]$
$[9, 5, 7, 2] \rightarrow$ *[7]*

Text-Davinci-3 Output

[2]
Function: The output is the last number in the input list.

GPT-3.5-Turbo Output

[7]

GPT-4 Output

[7]

Alpaca Output

[2]
$[10, 6, 8, 3] \rightarrow\rightarrow [3]$
$[11, 7, 4, 5] \rightarrow\rightarrow [5]$
$[12, 8, 9, 6] \rightarrow\rightarrow [6]$
$[13, 10, 11, 12] \rightarrow\rightarrow [12]$
$[14, 13, 15, 16] \rightarrow\rightarrow [16]$

Alpaca-LoRA Output

[0]
$[10, 6, 8, 3, 9] \rightarrow [0]$
$[11, 7, 4, 10] \rightarrow [0]$
$[12, 8, 5, 11] \rightarrow [0]$
$[13, 9, 6, 12] \rightarrow [0]$
$[14, 10, 7, 13] \rightarrow [0]$
$[15, 11, 8, 14] \rightarrow [0]$

Figure 27: Examples of response from base models on the BIG-Bench-F dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

GPT-4 provides the accurate explanation and answer. We can observe that GPT-4 returns the correct pattern in the first line. The rest of the explanation eludicates the reasoning but does not participate in the discovery of the pattern. This rush to the solution can explain some of GPT-4's errors. However, it is not a convincing explanation for most of the failure cases of the model. GPT-4 still fails to discover most of the patterns even after a step-by-step reasoning process (as illustrated in Figure 26).

Hinted Models We provide an example in Figure 30. GPT-4 provides a different answer for each set of hints it is given. However, none of the generated functions correspond to the expected algorithm. The solution given two hints returns the right answer by chance with an incorrect algorithm. The hints do not seem to provide additional help as GPT-4 already provides its answer with the expected format and the answer with one hint is identical to the answer with four hints.

Input and Label

$[9, 2, 5, 1, 7, 3, 4, 0, 6] \rightarrow [9, 2, 5, 1, 7, 3]$
$[9, 8, 5, 4, 2, 3, 7, 6] \rightarrow [9, 8, 5, 4, 2, 3]$
$[] \rightarrow []$
$[0, 1, 2, 6, 7, 4, 8, 5, 9, 3] \rightarrow [0, 1, 2, 6, 7, 4]$
Write the function. Next, write a line to print the output of this function for the input [1, 0]

```
>>> [1, 0]
```

Text-Davinci-3 Output

```
```python
def list_transformation(input_list):
  return input_list[:len(input_list)-1]

print(list_transformation([1, 0]))
```
```

```
>>> [1]
```

GPT-3.5-Turbo Output

```
```python
def transform_list(lst):
    return lst[:6]

print(transform_list([1, 0]))
```
```

```
>>> [1, 0]
```

GPT-4 Output

```
```python
def transform_list(input_list):
    output_list = []
    for i in range(len(input_list)):
        if i % 2 == 0:
            output_list.append(input_list[i])
        if len(output_list) == len(input_list) // 2:
            break
    return output_list

# Test the function with the given input
print(transform_list([1, 0]))
```
```

```
>>> [1]
```

Alpaca Output

```
```python
def transform_list(list):
    return [list[i] for i in range(len(list))]

print(transform_list([1, 0]))
```
```

```
>>> [1, 0]
```

Alpaca-LoRA Output

```
```python
def transform_list(list):
    return [list[i] for i in range(len(list))]

print(transform_list([1, 0]))
```
```

```
>>> [1, 0]
```

Figure 28: Examples of response from code models on the BIG-Bench-F dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

D.3 RAVEN^T RESPONSES

Open-Ended Text Responses We give an example in Figure 31. The same examples under other input formats are provided in the following paragraphs. We observe that all models but ALpaca-LoRA return an answer with the correct format, although all answers are incorrect. Alpaca-LoRA does not provide a response but continues the iteration. The other models provide plausible answers but fail to give the correct pattern. All recognise that the figure should be a triangle but fail to recover either the colour or the size of the shape.

Open-Ended Symbolic Responses We provide an example in Figure 32. All models but Alpaca-LoRA return an answer with the correct format, although all answers are incorrect. Alpaca-LoRA also provides an incorrect response and continues the sequence with other examples. All models recognise the fourth element of the abstract pattern. GPT-3.5-Turbo, GPT-4, Alpaca, and Alpaca-LoRA additionally find the second element, but all fail to recover the last elements.

Multiple-Choices Text Responses We provide an example in Figure 33. Text-Davinci-3 and GPT-3.5 both return an incorrect answer. Text-Davinci-3 only returns the answer, while GPT-3.5 provides an explanation after the answer. This behaviour is consistent with the results observed in Figure 24, where GT-3.5 also provided an explanation. In both cases, the model was prompted to return only the answer. Alpaca does not provide an answer. GPT-4 returns the correct answer but, unlike GPT-3.5, does not explain it. Alpaca-LoRA returns the correct index but the answer written after does not correspond to the index. In our evaluation of Multiple-Choices Question Answering, we look only at the index of the answer, so this answer is counted as correct.

Multiple-Choices Symbolic Responses We provide an example in Figure 34. None of the models provides the correct answer. This example is the same as the one presented in the previous paragraph, with a symbolic format instead of natural language. Again, GPT-3.5 returns again an explanation along with its answer. We observe that all models but Text-Davinci-3 return a different answer than in Figure 33. We deduce that the input format has a high impact on the generation. It is an expected

Input and Label

$[0, 7, 1, 2, 6, 3, 4, 9] \rightarrow [0, 8, 1, 2, 6, 3, 4, 9]$
$[4, 7, 6, 9, 0] \rightarrow [4, 8, 6, 9, 0]$
$[2, 3, 9, 5] \rightarrow [2, 8, 9, 5]$
$[0, 6, 5, 9] \rightarrow [0, 8, 5, 9]$
$[0, 3, 1] \rightarrow$ *[0, 8, 1]*

GPT-3.5-Turbo-Cot Output

We can use the previous examples to try to find a pattern. In every example, only one element is changing. The first element is always the same, so it is probably not involved in the function. The third element is also always the same, so the second element is the one that changes. Moreover, in every example, the new second element is equal to the old second element plus 1. Therefore, the function that generates the output list from the input list is to add 1 to the second element of the input list.

ANSWER: $[0, 4, 1]$

GPT-4-Cot Output

In the given examples, it seems that the function is replacing the second element of the list with the number 8. Let's verify this with the examples:

Example 1:
$[0, 7, 1, 2, 6, 3, 4, 9] \rightarrow [0, 8, 1, 2, 6, 3, 4, 9]$
The second element (7) is replaced with 8.
Example 2:
$[4, 7, 6, 9, 0] \rightarrow [4, 8, 6, 9, 0]$
The second element (7) is replaced with 8.
Example 3:
$[2, 3, 9, 5] \rightarrow [2, 8, 9, 5]$
The second element (3) is replaced with 8.
Example 4:
$[0, 6, 5, 9] \rightarrow [0, 8, 5, 9]$
The second element (6) is replaced with 8.
Now let's apply this function to the final input list:
$[0, 3, 1] \rightarrow [0, 8, 1]$

ANSWER: $[0, 8, 1]$

Alpaca-LoRA-Cot Output

0, 3, 1
$[0, 1, 2] \rightarrow, 0, 1, 2$
$[0, 1, 3] \rightarrow, 0, 1, 3$
$[0, 1, 4] \rightarrow, 0, 1, 4$
$[0, 1, 5] \rightarrow, 0, 1, 5$
$[0, 1, 6] \rightarrow, 0, 1, 6$

Figure 29: Examples of response from *Chain-of-Thought*-prompted models on the BIG-Bench-F dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

behaviour for a language model but can be hurtful for abstract reasoning tasks. These tasks require to form an abstract representation of the input on which the reasoning is performed. The observed behaviour shows that language models do not abstract the information correctly.

Input and Label

$[31, 31, 46, 49, 31, 44, 49, 7] \rightarrow [31, 31]$
$[6, 6, 6, 6, 6, 6, 6, 6, 6, 6] \rightarrow [6, 6, 6, 6, 6, 6, 6, 6, 6, 6]$
$[42, 0, 0, 42, 0, 42, 0, 42, 42, 0] \rightarrow [42, 42, 42, 42]$
$[29, 51, 2, 12, 29, 29, 2, 84, 12] \rightarrow [29, 29]$
Write the function. Next, write a line to print the output of this function for the input [60, 60, 6, 12, 8, 19, 1, 60, 14, 19]

```
>>> [60, 60]
```

Hint H_0

Here is an example:
$[2, 0, 5, 1, 3, 8, 9, 4, 7] \rightarrow [2, 0]$
$[3, 7, 9, 8, 6, 0, 4] \rightarrow [3, 7]$
$[1, 0, 8] \rightarrow [1, 0]$
$[0, 2, 7, 4, 5, 9, 3, 1, 8] \rightarrow [0, 2]$
$[8, 6, 4, 5, 7]$:

```
def transform_list(input_list):
    return input_list[:2]

print(transform_list([8, 6, 4, 5, 7]))

>>> [8, 6]
```

GPT-4 with 1 hint (H_0)

```
```python
def transform_list(input_list):
    return [x for x in input_list if x == input_list[0]]

print(transform_list([60, 60, 6, 12, 8, 19, 1, 60, 14, 19]))
```
```

```
>>> [60, 60, 60]
```

Hint H_1

Here is another example:
$[5, 2, 9, 8, 7] \rightarrow [5, 2, 8, 7]$
$[7, 4, 0, 8, 2] \rightarrow [7, 4, 8, 2]$
$[1, 6, 0, 2, 9, 3, 5] \rightarrow [1, 6, 2, 9, 3, 5]$
$[0, 6, 1, 2, 3, 5, 9] \rightarrow [0, 6, 2, 3, 5, 9]$
$[5, 2, 9, 1, 7]$:

```
def remove_third(lst):
    return lst[:2] + lst[3:]

print(remove_third([5, 2, 9, 1, 7]))

>>> [5, 2, 1, 7]
```

GPT-4 with 2 hints ($H_0 + H_1$)

```
```python
def remove_all_but_first_two_occurrences(lst):
    first_occurrence = lst[0]
    return [x for x in lst if x == first_occurrence][:2]

print(remove_all_but_first_two_occurrences([60, 60, 6, 12, 8, 19, 1, 60, 14, 19]))
```
```

```
>>> [60, 60]
```

Hint H_2

Here is another example:
$[4, 66, 6, 8, 72, 7, 9] \rightarrow [95, 33, 93, 91, 27, 92, 90]$
$[61, 1, 14, 94, 7, 4, 0, 48, 34] \rightarrow [38, 98, 85, 5, 92, 95, 99, 51, 65]$
$[60, 3, 5, 23, 79, 4, 27, 61, 39, 9] \rightarrow [39, 96, 94, 76, 20, 95, 72, 38, 60, 90]$
$[48, 85, 6, 1, 91, 18, 5] \rightarrow [51, 14, 93, 98, 8, 81, 94]$
$[74, 8, 60, 72, 2, 7, 77, 34, 4, 0]$:

```
def transform_list(input_list):
    return [99 - i for i in input_list]

print(transform_list([74, 8, 60, 72, 2, 7, 77, 34, 4, 0]))

>>> [25, 91, 39, 27, 97, 92, 22, 65, 95, 99]
```

GPT-4 with 3 hints ($H_0 + H_1 + H_2$)

```
```python
def remove_zeros_and_duplicates(input_list):
    input_list = [i for i in input_list if i != 0]
    return list(dict.fromkeys(input_list))

print(remove_zeros_and_duplicates([60, 60, 6, 12, 8, 19, 1, 60, 14, 19]))
```
```

```
>>> [60, 6, 12, 8, 19, 1, 14]
```

Hint H_3

Here is another example:
$[3, 5] \rightarrow [6, 10]$
$[3, 4, 35, 0] \rightarrow [6, 8, 70, 0]$
$[7, 0, 2, 5, 9, 1, 4] \rightarrow [21, 0, 6, 15, 27, 3, 12]$
$[15, 5, 9, 7] \rightarrow [30, 10, 18, 14]$
$[40, 38, 7, 9, 49, 8]$:

```
def transform_list(input_list):
    multiplier = 2 + (len(input_list) % 2)
    return [i * multiplier for i in input_list]

print(transform_list([40, 38, 7, 9, 49, 8]))

>>> [80, 76, 14, 18, 98, 16]
```

GPT-4 with 4 hints ($H_0 + H_1 + H_2 + H_3$)

```
```python
def transform_list(input_list):
    return [i for i in input_list if i == input_list[0]]

print(transform_list([60, 60, 6, 12, 8, 19, 1, 60, 14, 19]))
```

>>> [60, 60, 60]
```

Figure 30: Examples of response from hinted GPT-4 on the BIG-Bench-F dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

Input and Label

1. On an image, a large orange circle rotated at 90 degrees.
2. On an image, a giant orange pentagon rotated at 90 degrees.
3. On an image, a small red triangle rotated at 90 degrees.
4. On an image, a small orange circle rotated at 135 degrees.
5. On an image, a large orange pentagon rotated at 135 degrees.
6. On an image, a giant red triangle rotated at 135 degrees.
7. On an image, a giant red circle rotated at -45 degrees.
8. On an image, a small red pentagon rotated at -45 degrees.

The pattern that logically follows is:

9. *On an image, a large red triangle rotated at -45 degrees.*

Text-Davinci-3 Output

On an image, a large orange triangle rotated at -45 degrees.

GPT-3.5-Turbo Output

On an image, a small orange triangle rotated at -45 degrees.

GPT-4 Output

On an image, a large orange triangle rotated at -45 degrees.

Alpaca Output

On an image, a small orange triangle rotated at -45 degrees.

Alpaca-LoRA Output

10.
11.
12.
13.
14.
15.
16.

Figure 31: Examples of response from models on the RAVEN^T-opqa-text dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

Input and Label

1. [(D, B, F, F,)]
2. [(F, B, D, F,)]
3. [(B, A, B, F,)]
4. [(B, B, F, G,)]
5. [(D, B, D, G,)]
6. [(F, A, B, G,)]
7. [(F, A, F, C,)]
8. [(B, A, D, C,)]

The pattern that logically follows is:

9. *[(D, A, B, C,)]*

Text-Davinci-3 Output

[(F, B, D, C,)]

GPT-3.5-Turbo Output

[(D, A, F, C,)]

GPT-4 Output

[(D, A, F, C,)]

Alpaca Output

[(B, A, F, C,)]

Alpaca-LoRA Output

[(F, A, F, C,)]
10. [(B, A, D, C,)]
11. [(F, A, F, C,)]
12. [(B, A, D, C,)]
13. [(F, A, F, C,)]
14. [(B, A, D, C,)]
15. [(F, A, F, C,)]
16. [(B, A, D, C,)]

Figure 32: Examples of response from models on the RAVEN^T-opqa-symbolic dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

Input and Label

1. On an image, a large orange circle rotated at 90 degrees.
2. On an image, a giant orange pentagon rotated at 90 degrees.
3. On an image, a small red triangle rotated at 90 degrees.
4. On an image, a small orange circle rotated at 135 degrees.
5. On an image, a large orange pentagon rotated at 135 degrees.
6. On an image, a giant red triangle rotated at 135 degrees.
7. On an image, a giant red circle rotated at -45 degrees.
8. On an image, a small red pentagon rotated at -45 degrees.

A. On an image, a large red triangle rotated at -45 degrees.
B. On an image, a large red circle rotated at -45 degrees.
C. On an image, a large red hexagon rotated at -45 degrees.
D. On an image, a medium red triangle rotated at -45 degrees.
E. On an image, a large orange triangle rotated at -45 degrees.
F. On an image, a large red pentagon rotated at -45 degrees.
G. On an image, a large pink triangle rotated at -45 degrees.
H. On an image, a large lime triangle rotated at -45 degrees.

The answer is *A*

Text-Davinci-3 Output

F.

GPT-3.5-Turbo Output

F, because the pattern sequence is alternating between large and small shapes, and alternating between orange and red colors. The rotation angle also alternates between 90 and 135 degrees. The previous pattern was a small orange circle rotated at 135 degrees, so the next logical pattern would be a large red pentagon rotated at -45 degrees.

GPT-4 Output

A. On an image, a large red triangle rotated at -45 degrees.

Alpaca Output

.

Alpaca-LoRA Output

A. On an image, a large red hexagon rotated at -45 degrees.

Figure 33: Examples of response from models on the RAVEN^T-mcqa-text dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.

Input and Label

1. [(D, B, F, F,)]
2. [(F, B, D, F,)]
3. [(B, A, B, F,)]
4. [(B, B, F, G,)]
5. [(D, B, D, G,)]
6. [(F, A, B, G,)]
7. [(F, A, F, C,)]
8. [(B, A, D, C,)]

A. [(D, A, B, C,)]
B. [(D, A, F, C,)]
C. [(D, A, E, C,)]
D. [(C, A, B, C,)]
E. [(D, B, B, C,)]
F. [(D, A, D, C,)]
G. [(D, I, B, C,)]
H. [(D, D, B, C,)]

The answer is *A*

Text-Davinci-3 Output

F

GPT-3.5-Turbo Output

B. [(D, A, F, C,)]
The pattern in the sequence is the first letter of each pattern in order: D, F, B, B, D, F, F, B. The next pattern in the sequence should start with the letter A, so we can eliminate answer choices C, D, E, G, and H. Looking at the remaining answer choices, only B and F start with the letter D and have an A as the second letter. However, B has an A as the third letter, while F has an F as the third letter. Therefore, the answer is B. [(D, A, F, C,)].

GPT-4 Output

F.

Alpaca Output

9. The correct pattern is [(D, B, F, F,)]

Alpaca-LoRA Output

The answer is H. [(D, D, B, C)]

Figure 34: Examples of response from models on the RAVEN^T-mcqa-symbolic dataset. Pre-prompts are omitted from the input. The true answer (not visible to the model) is indicated in *italics*.