MIR-BENCH: BENCHMARKING LLM'S LONG-CONTEXT INTELLIGENCE VIA MANY-SHOT IN-CONTEXT INDUCTIVE REASONING

Kai Yan *
Siebel School of Computing and Data Science
University of Illinois Urbana-Champaign
Urbana, IL 61820, USA
kaiyan3@illinois.edu

Zhan Ling, Kang Liu, Yifan Yang, Ting-Han Fan, Lingfeng Shen, Zhengyin Du, Jiecao Chen
ByteDance Seed
San Jose, CA 95110, USA
jiecao.chen@bytedance.com

ABSTRACT

Inductive Reasoning (IR), the ability to summarize rules from examples and apply on new ones, has long been viewed as a primal ability for general intelligence and widely studied by cognitive science and AI researchers. Many benchmarks have been proposed to measure such ability for Large Language Models (LLMs); however, they focus on few-shot (usually <10) setting and lack evaluation for aggregating many pieces of information from long contexts. On the other hand, the ever-growing context length of LLMs have brought forth the novel paradigm of many-shot In-Context Learning (ICL), which addresses new tasks with hundreds to thousands of examples without expensive and inefficient fine-tuning. However, many-shot evaluations are mostly focused on classification (a very limited aspect of IR), and popular long-context LLM tasks such as Needle-In-A-Haystack (NIAH) seldom require complicated intelligence for integrating many pieces of information. To fix the issues from both worlds, we propose MIR-Bench, the first many-shot in-context inductive reasoning benchmark that asks LLM to induce output via input-output examples from underlying functions with diverse data format. Based on MIR-Bench, we study many novel problems for inductive reasoning and many-shot ICL, including robustness against erroneous shots and the effect of Chain-of-Thought (CoT), and acquired insightful findings.

1 INTRODUCTION

The tremendous success of Large Language Models (LLMs) in recent years (Ouyang et al., 2022; Hurst et al., 2024; Jaech et al., 2024) has finally brought us to the extent where human-level Artificial General Intelligence (AGI) becomes seemingly within reach (Jaech et al., 2024). With such success, researchers have shifted their focus from syntax- and word-level traditional Natural Language Processing (NLP) tasks such as named entity recognition (Mohit, 2014; Li et al., 2020), sentiment classification (Socher et al., 2013; Tang et al., 2016) and translation (Lopez, 2008; Stahlberg, 2020) onto those which measure abilities once considered unique to humans. Inductive Reasoning (IR) (Hayes et al., 2010), which is the ability to summarize general high-level rules from existing examples, thus comes into attention (Chollet, 2019); instead of math or coding which require knowledge and experience in particular fields, inductive reasoning measures the abstract generalization power of an intelligence (Chollet, 2019) and is considered as a primal mental ability (Kinshuk & McNab, 2006). Thus, it is long studied by the cognitive science community (Bisanz et al., 1994; Heit, 2000), adopted in IQ tests for human (Ferrara et al., 1986), and is recently used as a measurement for the

*Work done at ByteDance Seed.

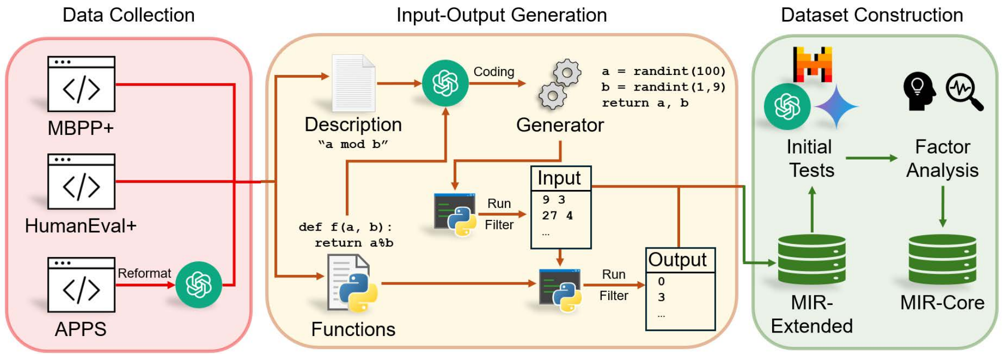


Figure 1: A high-level illustration of our data generation pipeline. We first collect functions from existing coding benchmarks, then let GPT-4o-0806 write data generator for each function; we then run the data generator to produce input shots, and combine them with ground truth function to produce output shots. With input and output shots, we concatenate them and build MIR-extended; then, with initial tests on several models, we study the factor for what makes an inductive reasoning problem benefit from many-shot, and build MIR-core based on selection with the factors.

state-of-the-art LLMs such as o1 (Jaech et al., 2024) to show their intelligence level. While many IR benchmarks (Banatt et al., 2024; Ma et al., 2024; Li et al., 2024a) for LLMs have been proposed, such as ARC (Chollet, 2019) and its variants (Xu et al., 2023; Kim et al., 2022), they all focused on few-shot In-Context Learning (ICL) with typically $<$10 examples. While induction from fewer examples may imply stronger reasoning ability, some underlying rules for IR problems are inherently too complicated or ambiguous for a few examples. For instance, consider a quadratic curve with clipping; with three examples, it is unknown whether the curve is sampled from a circle or a quadratic curve, let alone a clipped one; however, with 300 examples, not only the quadratic function is clear, but the special clipping rule are also very likely to be retrieved. LLM should handle such long-context, many-example cases as well as few-shot inductive reasoning.

In fact, the scaling of the amount of ICL data is in line with the trend of the LLM community striving to expand the context length (Peng et al., 2024; Su et al., 2024) for super-human problem-solving efficiency. It is with this trend that a new paradigm emerged recently: Many-Shot ICL, which typically uses hundreds to thousands of examples for test-time task learning without expensive and relatively data-inefficient fine-tuning (Agarwal et al., 2024). However, many-shot evaluations are mostly focused on classifications (Li et al., 2023; Bertsch et al., 2024; Zhao et al., 2024b; Li et al., 2024c; Zou et al., 2024), which is a very limited area for inductive reasoning. Other standard long-context LLM tasks, such as needle-in-a-haystack (NIAH) (Kamradt, 2023), are more of a retrieval problem than gathering understanding from many pieces of clues. With all these blanks in LLM evaluation (see Tab. 1 for a comparison with the most related works, and Tab. 2 in Appendix A for a more complete version), we must ask: *How to evaluate the LLM's ability to aggregate many pieces of information from many examples to perform inductive reasoning on various complicated problems?*

To address the problem above and fix the limitation of existing LLM evaluation from both inductive reasoning and the many-shot/long-context community, we propose MIR-Bench, a large and diverse **M**any-shot **I**nductive **R**easoning benchmark, in which LLMs are given examples of input-output data generated by an underlying unknown function with diverse input-output forms , and need to predict the output for new input.

The benchmark is generated by the following pipeline as illustrated in Fig. 1: 1) we collect functions from introductory-level coding benchmarks including HumanEval+ (Liu et al., 2023), MBPP+ (Liu et al., 2023) and APPS (Hendrycks et al., 2021); 2) we use GPT-4o-0806 to write code as data generators that produces input-output pairs, and execute them to generate ICL shots and test input; 3) run ground truth function with generated inputs for ground-truth outputs; 4) use scripts to build prompts for target problem, and filter out problems with too long shot length or insufficient input-output diversity. With such procedure, we propose two sets of problems: **MIR-Core** and **MIR-Extended**, which contains 3000 problems (300 functions $\times$ 10 test cases), and 6930 problems (693 functions $\times$ 10 test cases) respectively, and can be easily supplemented by generating more test cases.

Table 1: Comparison on our benchmark with the most related prior many-shot / long-context benchmarks (first part) and inductive reasoning benchmarks (second part). See Tab. 2 in Appendix A for a complete comparison. To save space, we abbreviate "Many Shot" as MS, "Inductive Reasoning" as IR (△ represents "classification only"), "Prob." as problems, and "I/O Div." as "Input/Output Diversity" (**having at least** 2 **different input-output types, e.g., given an array and output an integer, or given a pair of strings and output a choice**). "Gen." means "Generative", which means whether new test cases can be easily generated without much human effort. "LB" means whether a leaderboard is available, and "EE" means "Easy Evaluation", i.e., whether a pipeline for evaluating any given new model exists. "New Data" means whether the input-output data never appears in existing benchmarks; if so, the benchmark is secured against data contamination. Note, the counting of #IR Problems and "Gen." take different target input-output for the same function into account, but **do not take different sets of shots into account**.

Benchmarks	MS	IR	# IR Prob.	I/O Div.	Max # Shots	Gen.	LB	EE	New Data
HELMET (Yen et al., 2024)	✓	△	500	✓	∼10K	×	×	✓	×
LongICLBench (Li et al., 2024c)	✓	△	3000	×	∼2000	×	✓	✓	×
ManyICLBench (Zou et al., 2024)	✓	△	1000	✓	7252	×	×	×	×
LMAct (Ruoss et al., 2024)	✓	×	0	×	256	✓	×	✓	✓
LongBench (Bai et al., 2024)	✓	△	400	✓	600	×	✓	✓	✓
KORBench (Ma et al., 2024)	×	✓	50	✓	3	×	✓	✓	✓
ARC (Chollet, 2019)	×	✓	800	×	3	×	✓	✓	✓
WILT (Banatt et al., 2024)	×	✓	50	×	30	×	✓	✓	✓
LogicVista (Xiao et al., 2024)	×	✓	107	✓	10	×	×	✓	✓
MIRAGE (Li et al., 2024a)	×	✓	2000	✓	8	✓	×	×	✓
MIR-Bench (Ours)	✓	✓	6930	✓	2048	✓	✓	✓	✓

The former is selected from the latter and contains the problems that LLM benefits the most from many-shot (see Sec. 4.2 for details).

To evaluate the long-context intelligence level of different LLMs, we test a variety of the cutting-edge LLMs on our benchmark, and find our benchmark to be both challenging and discriminative, as model performance vary greatly, but none saturates on our benchmark. We have also conducted the following important empirical studies with our benchmark, all of which are novel, important yet largely overlooked by previous works:

1. What are the factors that determine ICL performance change with respect to number of shots (Sec. 4.2, 4.3)?
2. Can Chain-of-Thought (CoT) (Wei et al., 2022) help inductive reasoning? If so / if not, why (Sec. 4.4)?
3. How robust is LLM's inductive intelligence against errorneous input-output pairs (Sec. 4.5)?
4. Does the paradigm of first coding, then executing code for results (Cheng et al., 2024) work for many-shot in-context inductive reasoning (Appendix D.5)?

In conclusion, our key contributions can be summarized as: 1) We propose MIR-Bench, a large-scale and diverse many-shot inductive reasoning benchmark, which fills in the blank for both many-shot and inductive reasoning community; 2) We build a novel automatic pipeline for generating new many-shot inductive reasoning problems from existing coding benchmarks without using existing corpus as input/output (i.e., no data leakage issues); 3) We perform empirical study on many important problems overlooked by previous works and gained important insights on LLM's many-shot / long-context intelligence.

2 Related Work

Long context LLMs. Recent remarkable success of LLMs have given rise to expectations for LLMs to complete more difficult tasks, such as summarization of a whole book (Chang et al., 2024), modification over a complex code repository (Jimenez et al., 2024), test-time improvement (Zhou et al., 2024) and journey learning (Qin et al., 2024). To make sufficient room for related context and meet such demands, researchers have scaled up LLM models and data (Zhang et al., 2024a; Pearce et al., 2024), and proposed novel encoding methods such as Rotational Position Embedding

(RoPE) (Su et al., 2024), YaRN (Peng et al., 2024) and LongRoPE (Ding et al., 2024). With such designs, LLMs have entered the long-context era where the LLM context lengths can reach 128K (Hurst et al., 2024; Yang et al., 2024a; Dubey et al., 2024), 2M (Team et al., 2024), or even an infinite number of tokens (Munkhdalai et al., 2024), enabling the novel many-shot ICL (Agarwal et al., 2024) paradigm. To evaluate such models, many benchmarks have been proposed to evaluate LLM's long-context ability (Wang et al., 2024a; Li et al., 2024b; Zhang et al., 2024b), such as Question-Answering (Shaham et al., 2023; Li et al., 2024b), coding (Zhang et al., 2024b; Dong et al., 2024), math (An et al., 2023; Zhang et al., 2024b), retrival (Kamradt, 2023; Hsieh et al., 2024; Wang et al., 2024a) and summarization (Shaham et al., 2023; An et al., 2023). However, very few long-context benchmarks consider inductive reasoning tasks. Among them, LongBench (Bai et al., 2024) only contains two many-shot classification tasks and few-shot summarization / QA tasks with existing dataset, while BABILong (Kuratov et al., 2024) only considers simple inductive reasoning from a few examples scattering in the long context. In contrast, our benchmark is a more diverse and large-scale evaluation for long-context inductive reasoning from many pieces of information.

Many-Shot In-Context Learning (ICL). Many-Shot ICL (Agarwal et al., 2024) is an emerging in-context learning paradigm where LLMs learn to complete new task with hundreds to thousands of examples (instead of the usual < 10 examples for ICL works (Li et al., 2024a; Chollet, 2019; Xu et al., 2023)) given in its context. Compared to Supervised Fine-Tuning (SFT), many-shot ICL makes full use of the current models' long-context capability, and is much more flexible with higher computational and data efficiency (Agarwal et al., 2024). There are a large number of many-shot ICL empirical studies (Bertsch et al., 2024; Zhao et al., 2024b; Song et al., 2024; Zhao et al., 2024a) with several benchmarks (Yen et al., 2024; Li et al., 2024c; Zou et al., 2024; Ruoss et al., 2024) containing many-shot ICL components; however, most of them only focused on classification (Zhao et al., 2024b; Li et al., 2024c; 2023; Jiang et al., 2024; Bertsch et al., 2024), a very limited type of problems. While there are several works that studies decision-making (Ruoss et al., 2024), math (Agarwal et al., 2024), instruction following (Zhao et al., 2024a) and LLM judges (Song et al., 2024), none of the existing works has studied general inductive reasoning, the important measurement of intelligence level (Chollet, 2019). Also, most of the existing many-shot ICL evaluations are not diverse enough, which means they only have one pair of input-output types (Ruoss et al., 2024; Jiang et al., 2024). Our work, on the contrary, measures LLM's intelligence level using inductive reasoning with diverse input-output types.

Inductive reasoning. Inductive reasoning (Hayes et al., 2010) has been widely studied as a primal mental ability of human intelligence (Kinshuk & McNab, 2006) in IQ tests (Ferrara et al., 1986) and cognitive science (Bisanz et al., 1994; Heit, 2000) long before LLMs exist, as it represents an abstract generalizability of rules from given examples without prior knowledge. As LLMs approaches human-level intelligence recently, many inductive reasoning-based approaches (Qiu et al., 2024; Wang et al., 2024b) and benchmarks have been proposed (Chollet, 2019; Ma et al., 2024; Li et al., 2024a; Xiao et al., 2024; Banatt et al., 2024), among which the most representative is the Abstract Reasoning Corpus (ARC) (Chollet, 2019) and its variants (Kim et al., 2022; Xu et al., 2023), which is recently used to demonstrate the intelligence level of OpenAI o3 models (OpenAI, 2024). However, none of the existing inductive reasoning benchmarks is designed for many-shot scenario (There is only one small mini-SCAN (Qiu et al., 2024; Lake et al., 2019) dataset appearing in Qiu et al. (2024) satisfies the definition of many-shot). By filling in this gap, our many-shot inductive reasoning benchmark not only enables the LLM inductive reasoning community to catch up with the long-context era, but also tests the ability of LLMs to gather information from thousands of pieces of data, a much larger number than existing inductive reasoning problems (Chollet, 2019; Li et al., 2024a).

3 MIR-Bench

In this section, we will introduce our MIR-Bench in details, with Sec. 3.1 discussing the formulation of the problems evaluated in our benchmark and Sec. 3.2 introducing the pipeline with which we build our benchmark.

3.1 Problem Formulation

The goal of the problems in our benchmark is for LLMs to predict the output for a new input given a list of examples. More specifically, assume we have an underlying function $y = f(x)$, where x and y can be arbitrary data. Assume for f we have a set of n known example input-output pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, and a new input x_{new}; then, the LLMs' input will be $[c_1, \text{str}(x_1), \text{str}(y_1), \text{str}(x_2), \text{str}(y_2), \ldots, \text{str}(x_n), \text{str}(y_n), c_2, x_{\text{new}}]$, where $[\cdot, \cdot, \ldots, \cdot]$ is a string concatenation, c_1 and c_2 are general context prompts (e.g. "You are an expert in reasoning", or "Here is the target input"; see Sec. B for details), and $\text{str}(\cdot)$ is the string representation plus an "Input: " prefix for x and "Output: " prefix for y. LLMs can output arbitrary rationale; however, they must end their answer with $\text{str}(y_{\text{new}})$, where $y_{\text{new}} = f(x_{\text{new}})$. The answer is extracted with rule-based scripts, and exact match will be performed to determine the LLM's performance in accuracy. See Appendix B.6 for details.

3.2 Benchmark Construction

The construction of our benchmark can be decomposed into four steps: function collection, input generation, output generation, and prompt building. The following boxes illustrate a 4-shot example from our benchmark:

`# Prompts (shown to LLM)`
You are given some function that takes something as input and output something. You need to predict the output for the target input of that function. Remember always end your answer with 'Output: your answer', with your answer in strict python format. Here are some examples:
Input: {'words': 'gSldgGeHRmBdbUPQ'}
Output: Golf Sierra Lima Delta Golf Golf Echo Hotel Romeo Mike Bravo Delta Bravo Uniform Papa Quebec
Input: {'words': 'aJV mwJTALeGYOkEhf'}
Output: Alfa Juliett Victor Mike Whiskey Juliett Tango Alfa Lima Echo Golf Yankee Oscar Kilo Echo Hotel Foxtrot
Input: {'words': 'UNjAIRFaZZtEhbiSRRml'}
Output: Uniform November Juliett Alfa India Romeo Foxtrot Alfa Zulu Zulu Tango Echo Hotel Bravo India Sierra Romeo Romeo Mike Lima
Input: {'words': 'nvILpZ'}
Output: November Victor India Lima Papa Zulu
Input: 'words': 'pvFOSPcBGcJqKewLOury'
`# Answer (hidden to LLM)`
Papa Victor Foxtrot Oscar Sierra Papa Charlie Bravo Golf Charlie Juliett Quebec Kilo Echo Whiskey Lima Oscar Uniform Romeo Yankee
`# Code (hidden to LLM)`
```
def solution(word):
    letters = "A": "Alpha", "B": "Bravo", "C": "Charlie", "D": "Delta", "E": "Echo", "F": "Foxtrot", "G": "Golf", "H": "Hotel", "I": "India", "J": "Juliett","K": "Kilo", "L": "Lima", "M": "Mike", "N": "November","O": "Oscar", "P": "Papa", "Q": "Quebec", "R": "Romeo", "S": "Sierra", "T": "Tango", "U": "Uniform", "V": "Victor", "W": "Whiskey", "X": "X-ray", "Y": "Yankee", "Z": "Zulu"  # Convert each letter to its corresponding NATO word and join with spaces
    return ' '.join(letters[c] for c in word.upper())
```

Function collection. We begin by collecting introductory-level coding problems from three coding benchmarks: Humaneval+ (Liu et al., 2023), MBPP+ (Liu et al., 2023), and APPS (Hendrycks et al., 2021). We use the whole Humaneval+ and MBPP+ dataset (164 and 378 problems respectively); for APPS dataset, we select problems from its training dataset with difficulty level "introductory" (2640 problems)[1]. We ensure that each solution code is a single function without wrapping solution class or test statement; for codes in APPS that do not conform to this standard, we ask GPT-4o-0806 to rewrite the code given problem input and the solution code (See Appendix B.3 for prompts).

Input generation. We use GPT-4o-0806 to automatically generate inputs for each function acquired in the last step, for which prior works (Shao et al., 2024; Li & Ellis, 2024) usually directly generate input data. However, such method is not only non-scalable, but also prone to errors such as input format mismatch. To address this issue, we prompt GPT-4o-0806 to first generate "data generators" for each problem (See Appendix B.4 for prompts), then run each generator in Python interpreter to generate data. We generate 20000 shots and 10 test cases for each problem, which is impossible to acquire with prior methods. We wrote the prompt such that the test case is supposed to be slightly

[1]Solutions from other levels, such as "interview" and "competition", are generally too difficult for LLMs to induce and too complicated to be compressed in a single function

harder (e.g. with larger numbers / longer lists) than the shots. In this step, we filter out problems with the generated input too identical (≤ 4096 different shots out of 20000), duplicate test cases, or test cases appearing in the shots.

Output generation. With input generated, we write a script to stitch generated input and ground truth function f in the same Python script, and run them in the intepreter to acquire ground-truth output. In this step, we filter out problems with floating number output, unless the precision is fixed across all shots by rounding, given by input, or unimportant for exact matching (e.g. the function is to output absolute value). We also filter out problems with too low output diversity (no less than 50% of the shots having the same answer), and problems with invalid output due to code error.

Prompt building. In this step, we use Python scripts to automatically stitch input-output pairs with task description to generate final input for LLMs. Finally, we also filter out problems that are unsolvable (either too difficult or data coverage are insufficient) for current state-of-the-art LLMs, which are the problems that have 0 accuracy for all five models {GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2} across $\{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$ shots in 10 test cases. After this step, we have 693 valid functions, each with 10 test cases; these problems are the content of our benchmark version MIR-Extended. Within this version, we select 300 problems that are challenging and can largely benefit from many-shot; See Sec. 4.2 for details.

4 EXPERIMENTS

In this section, we will introduce general performance of existing models on our benchmark and a series of exploratory experiments which gives novel insights. More specifically, we first introduce the main results on our MIR-Extended benchmark in Sec. 4.1; then, we explore factors that indicate whether a problem can benefit from many-shot, and build MIR-Core in Sec. 4.2. We further conduct more in-depth analysis on important properties of LLM's many-shot intelligence in several aspects on MIR-Core in Sec. 4.3, 4.4 and 4.5. See more empirical studies in Appendix D.5.

4.1 MIR-EXTENDED

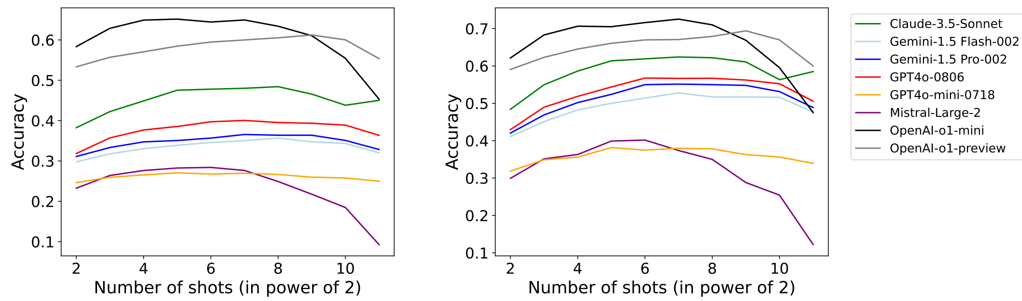
Evaluation setup. We evaluate a set of 15 cutting-edge LLMs with context window $\geq$ 128K tokens on our MIR-Extended benchmark with 693 different function and 10 test cases per function (a total of 6930 problems). The evaluated LLMs are: {o1-preview-0912, o1-mini-0912, GPT-4o-0806, GPT-4o-mini-0708, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Gemini-Flash 2.0, Claude-3.5-Sonnet, Claude-3.5-Haiku, Claude-3-Haiku, Claude-3-Sonnet, Qwen2-72B-Instruct, Mistral-Large-2, Moonshot-128K, GLM-4-Plus} by invoking official APIs; see Appendix B.1 for detailed prompts. We use greedy decoding (with temperature 0) for evaluation (See Appendix D.1 for ablations on the robustness of evaluation), and use exact match accuracy as the metric with rule-based extraction of the answer from LLM's response (See Appendix B.6). Each model is evaluated with $\{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$-shot with shots uniformly randomly sampled from 20000 shots generated in Sec. 3.2. The upper limit 2048 is selected such that most data falls within the mainstream 128K context window (see Tab. 5 for out-of-context-window rate).

Results. Fig. 2(a) illustrates the performance of all 15 LLMs on our MIR-Extended benchmark. The performance of the LLMs varies greatly; among all models, o1-mini-0912 and o1-preview-0912 clearly outperform all other models, followed by Claude-3.5-Sonnet and GPT-4o-0806. However, all LLMs evaluated are far from addressing our inductive reasoning task; the best model, o1-mini-0912, only reaches an accuracy of less than 0.7, while most models such as GPT-4o-0806 only achieve less than 0.4 accuracy. Such performance indicates that different from the conclusion in Cheng et al. (2024), LLMs' inductive reasoning abilities still limited in complicated tasks. Claude-3.5-Haiku achieves surprisingly low accuracy; upon checking examples, we find that the model often do not understand our prompt and see the target input as part of an incomplete data, thus refusing to answer the problem.

Interestingly, scaling up the number of shots is not always beneficial, similar to many tasks in Agarwal et al. (2024). For models other than Gemini, the performance drop over 512 shots can be partly attributed to exceeding the 128K context limit [2]; however, for most language models evaluated

[2]Which only happens in $\leq 1\%$ case for 1024 shots but more common for 2048 shots. See Tab. 5 for details.

(including GPT and o1-mini), the performance growth often stops at no more than 256 shots, where the context limit is not reached. Such issue stems from attention dispersion as stated in Yuan et al. (2024); as the number of examples increases, the attention weights which should be cast on the most informative shots is distracted by the less informative ones instead of lack of information retrieval ability. We validate this via ablation in Sec. 4.3.



(a) MIR-Extended (b) MIR-Core

Figure 2: The performance of all LLMs on MIR-Extended (panel (a)) and MIR-Core (panel (b)). For better readability, we only show the most representative models; see Fig. 9 in Appendix D.2 for the rest. The benchmark poses challenge to all models tested including o1-preview and o1-mini. Most models will "saturate" at a particular number of shots, i.e., their performances stop to improve when more shots are given due to limited information integration capability.

4.2 MIR-Core: Problems Requiring Many-Shot

Ablation on possible factors. While we have obtained many inductive reasoning problems, not all of them necessarily benefits from many-shot ICL; for example, a simple function such as adding two numbers or absolute value can be induced in a few shots. To study the inductive reasoning problems whose difficulties are *distinctive* between few-shot and many-shot, and curate a high-quality many-shot benchmark, we perform a detailed ablation study on possible factors for such distinctiveness. To better study such property, we define the following metric D:

$$D = \frac{D_1 + D_2}{2}, \text{ where } D_1 = \left[\frac{\text{acc@64} + \text{acc@128}}{2}\right] - \left[\frac{\text{acc@16} + \text{acc@32}}{2}\right], \\ D_2 = \left[\frac{\text{acc@32} + \text{acc@64} + \text{acc@128}}{3}\right] - \left[\frac{\text{acc@4} + \text{acc@8} + \text{acc@16}}{3}\right]. \tag{1}$$

In Eq. (1), acc@x, $x \in \{4, 8, 16, 32, 64, 128\}$ is the average accuracy of {GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2} at x-shot, each over 10 test cases. Intuitively, D is a combination of two components D_1 and D_2, each measures average performance growth from different few-shot to many-shot ranges; The range of x is based on prior inductive reasoning work (Chollet, 2019) and the number of shots where performances saturate on MIR-Extended. Ideally, we want to identify the factors which are positively related to D, and curate MIR-Core with problems having higher values of D.

With such metric D, we consider the following factors that are potentially relevant to the distinctiveness between few-shot and many-shot: **1) Ground truth function complexity:** 64-shot accuracy, function code length, LLM-evaluated function difficulty level[3] and problem topics; **2) Answer complexity:** number of different answers across 20000 shots, and the ratio of the most common answer out of 20000 shots; **3) Input complexity:** input length per shot.

As we aim to ensure the diversity of our evaluation, we did not select problems based on problem topics (See Appendix C.2 for ablation on problem topics). For the rest of the factors, we fit the ground-truth metric D using a quadratic function with these factors (after normalization) as self-variables. We use quadratic function as we found some factors (e.g. # different answers), are roughly

[3]A reliable evaluation is non-trivial; see Appendix B.5.

raised at both ends and concave in the middle, while others are roughly monotonic (e.g. code length); see Fig. 7 in Appendix C.3 for details.

The coefficients are illustrated as Fig. 3. as the result shows, ground truth function complexity is the dominating factor for distinctiveness between few-shot and many-shot performance, among which LLM-labeled difficulty is a leading, positive factor (i.e. more difficult problem will require more shots). Answer diversity and input complexity are relatively less important. See Appendix C.3 for single-factor analysis.

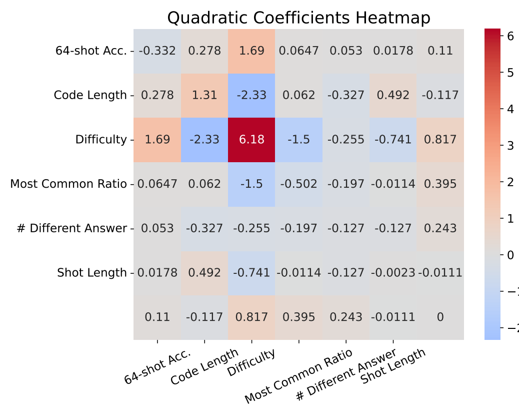


Figure 3: The coefficients of the quadratic function fitting D with the aforementioned factors normalized between $[0, 1]$. The blank row and column are for constant factors. LLM-labeled difficulty is the leading factor for D, while answer diversity and shot length are less important.

Selection of data for MIR-Core. We adopt the quadratic function's fitting result and select the 300 problems with the highest predicted D-value as MIR-Core. Such design is based on a balance between achieving higher D-value for MIR-Core and unbiased evaluation for the LLMs involved in computing D-value (thus we do not use the problems with highest ground truth D).

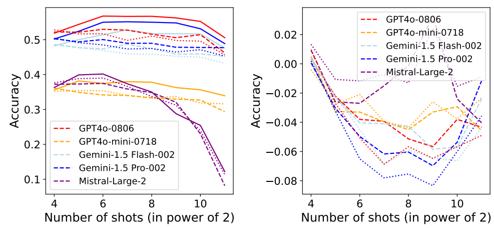
Results on MIR-Core. We again evaluate all 15 LLMs in Sec. 4.1 on our MIR-Core with 300 problems with 10 test case each. The results are illustrated in Fig. 2(b). while the performance difference between few-shot and many-shot are more distinctive as expected, the relative performance and many-shot saturation phenomenon between 15 models remain unchanged.

4.3 Results with Duplicated Few-shots

To study whether the saturation of many-shot in Sec. 4.1 and 4.2 comes from the inability of retrieving the most useful shots for induction or the inability of aggregating many pieces of different, useful information, we conduct an ablation where we test {GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2} on MIR-Core with 16-shot, but with the following two settings: **1) one shot duplicated** until total shots number of reach $\{16, 32, 64, 128, 256, 512, 1024, 2048\}$, while other 15 shots only appear once; and **2) each of the** 16 **shot reused** for $\{1, 2, 4, 8, 16, 32, 64, 128\}$ times.

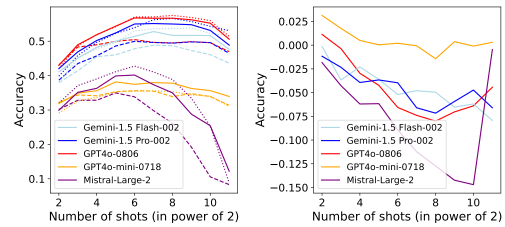
We ensure the examples in test cases with more shots are supersets of those with less shots, i.e., the information given in the input is strictly increasing with more shots. The result is shown in Fig. 4, where solid lines are for original resutls on MIR-Core from Sec. 4.2, dashed lines are for scenario 1 (one shot duplicate), and dotted lines are for scenario 2 (all shots duplicate).When the number of shots increase, as shown in panel (b), the performance difference between normal many-shot and both scenario 1 and 2 increases, which indicates that LLMs can indeed aggregate many pieces of information from more shots and acquire performance gain (which is almost not the case for Mistral-Large-2, and thus its "saturation point" of performance with more shots is the lowest). However, the difference diminishes when there are more than 512 shots (note this also applies for Gemini with 2M context length, thus this is not a problem of hard context limit). Such result indicates that too many pieces of information may actually harm LLMs' performance by distraction. Also, the performance of the dotted line (all shots duplicate) is in general not higher than that of the dashed line (one shot duplicate), which indicates that the problem is not in information retrieval as the two scenarios contain the same amount of information but the latter has higher difficulty for information retrieval. Here we summarize the insight:

Insight 1: The saturation of many-shots does not come from information retrieval, but from distraction when aggregating too many information.



(a) Accuracy (b) Accuracy difference

Figure 4: Results of duplicating shots, where solid lines are for original results on MIR-Core from Sec. 4.2, dashed lines are for scenario 1 (one shot duplicate), and dotted lines are for scenario 2 (all shots duplicate); i.e., panel (b) is the result of dashed and dotted line subtracting solid line in panel (a).



(a) Accuracy (b) Accuracy difference

Figure 5: Panel (a) shows the result of LLMs with forced CoT (dashed lines), no CoT (dotted lines) and no CoT specification (solid line; same as that in Fig. 2) on MIR-Core, and panel (b) shows the difference between forced CoT and no CoT. Surprisingly, forced CoT works generally worse than no CoT, and the gap increases with more shots.

4.4 The Effectiveness of CoT

Chain of Thought (CoT) (Wei et al., 2022) is a foundamental LLM technique proved to be of great help for LLMs in breaking down complex reasoning problem into step-by-step rationale. In this section, we will explore whether CoT helps many-shot inductive reasoning in our task.

Statistics in main results. We first count the number of answers with and without CoT[4] in MIR-Core results (Sec. 4.2) and their respective correct rate; surprisingly, we find that in all 15 models, including thinking models such as o1, answers without CoT have significantly higher accuracy than those with CoT (see Tab. D.4 for results).

Evaluation setup. To further validate whether the performance difference comes from CoT or from problems with different difficulty levels (e.g. LLMs only apply CoT on difficult problems), we further test MIR-Core with different prompts under two settings: 1) the "Direct" setting, where the models are required to **not** write CoT; 2) the "CoT" setting, where the models are **forced** to write CoT. We evaluate all 15 models in Sec. 4.1. See Appendix B.2 for prompts.

Results. Fig. 5 shows the result of {GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2} (see Sec. D.4 for other LLMs) on MIR-Core. Surprisingly, for most models, forced CoT works much worse than no CoT, and such performance gap increases with the number of shots (See Fig. 10 in Appendix D.4 for more models).[5] We hypothesize such phenomenon comes from the complicated nature of CoT. For example, consider a problem with two integers a and b as input and $\max(a, b)$ as output; transformers can easily duplicate the mapping relation between the three sets of tokens a, b and $\max(a, b)$ as if going through a gradient descent with regression loss on examples as the training set, as suggested by many theoretical works in ICL (Dai et al., 2023; Von Oswald et al., 2023; Mahankali et al., 2024). However, a mapping from input to CoT makes the equivalent of gradient descent much more opaque. Here we summarize the insights:

Insight 2: CoT does not necessarily help many-shot inductive reasoning.

4.5 Robustness of LLM Inductive Intelligence

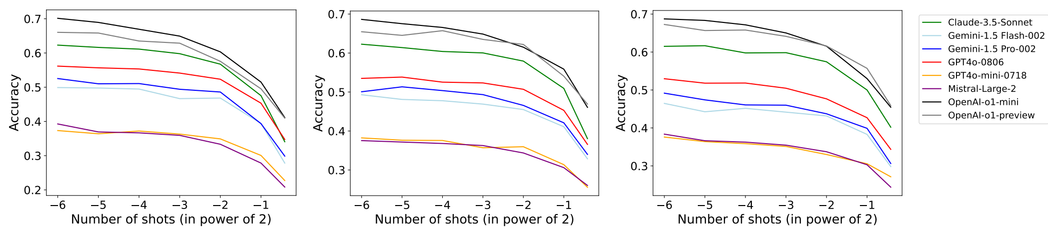
While many works (Agarwal et al., 2024) have studied LLM's many-shot ICL performance, the robustness of LLM's many-shot ICL ability (Zhao et al., 2024a), i.e. the accuracy given incorrect

[4]We count answers with ≥ 20 characters before the final "Output:" as the ones with CoT.

[5]The only exception is that the performance gap of Mistral-Large-2 sharply decreases at 2048-shot, which is due to its inability to deal with long-context problems (Kuratov et al., 2024).

examples, is still largely underexplored. In this section, we explore the performance change with increasing number of shots with incorrect answers.

Evaluation Setup. We test all 15 models in Sec. 4.1 on MIR-Core with 3 different settings: 1) the "unaware" setting, where the models do not know there are incorrect answers in the provided examples; 2) the "aware-error" setting, where the models know that some (unknown number of) examples are incorrect; and 3) the "aware-ratio" setting, where the models know exactly how many shots are incorrect out of all given shots. The three settings are mostly the same, with slight difference in prompt; see Appendix B.2 for details. We test $\{64, 256, 1024\}$ shots $\times$ error ratio of $\{1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 3/4\}$ respectively. See Appendix B.7 for data generation details.



(a) Unaware (b) Aware-Error (c) Aware-Ratio

Figure 6: The accuracy of representative models with erroneous shots under different prompt settings with 64 shots (see Fig. 11 in Appendix D.6 for full results). The result shows LLMs are generally quite robust against erroneous shots.

Results. The results for 64-shot are illustrated in Fig. 6 (see Appendix D.6 for the rest). Surprisingly, we found that LLMs are generally quite robust against erroneous shots; their performance are only slightly harmed below $1/8$, and can maintain decent performance even with $3/4$ error rate. We find that generally, there is generally no significant performance difference in different awareness level of erroneous shots; some exceptions are Gemini-2.0 Flash and Claude-3.5-Haiku (see Fig. 11 in Appendix D.6), where the answering paradigm of the former remains the same, and the latter accepts the target input as part of the "incomplete" data and rejects answering questions less frequently. We summarize the most important insight of this experiment in the box below:

Insight 3: LLMs are quite robust against erroneous shots in many-shot inductive reasoning tasks.

5 Discussion and Conclusion

In this paper, we propose MIR-Bench, a novel, large-scale many-shot in-context inductive reasoning benchmark and poses a difficult challenge for even the state-of-the-art LLMs. We test 15 cutting-edge LLMs from 4-shot to 2048-shot on our benchmark, and conduct extensive ablations on many aspects such as CoT, robustness and coding paradigm in addressing inductive reasoning problems. With many important insights concluded from our experiments, we believe our work provides a unique way of understanding LLM's intelligence level under long-context scenario.

Limitations and future works. To curate MIR-Core with problems that requires many-shot ICL, we studied many related factors such as types of problem and difficulty of the problems; however, they are not decisive enough. A more explainable rule for determining whether a problem needs many-shot would be an interesting avenue for future many-shot ICL works. Also, it would be interesting for future work to explore the intelligence of multimodal models (Sun et al., 2024; Man et al., 2024; Choudhuri et al., 2024).

References

Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. Many-shot in-context learning. In *NeurIPS*, 2024.

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. *arXiv preprint arXiv:2307.11088*, 2023.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *ACL*, 2024.

Eryk Banatt, Jonathan Cheng, Skanda Vaidyanath, and Tiffany Hwu. Wilt: A multi-turn, memorization-robust inductive logic benchmark for llms. *arXiv preprint arXiv:2410.10998*, 2024.

Amanda Bertsch, Maor Ivgi, Uri Alon, Jonathan Berant, Matthew R Gormley, and Graham Neubig. In-context learning with long-context models: An in-depth exploration. *arXiv preprint arXiv:2405.00200*, 2024.

Jeffrey Bisanz, Gay L Bisanz, and Connie A Korpan. Inductive reasoning. In *Thinking and problem solving*. 1994.

Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. Booookscore: A systematic exploration of book-length summarization in the era of llms. In *ICLR*, 2024.

Salman Cheema, Sarah Buchanan, Sumit Gulwani, and Joseph J LaViola Jr. A practical framework for constructing structured drawings. In *IUI*, 2014.

Kewei Cheng, Jingfeng Yang, Haoming Jiang, Zhengyang Wang, Binxuan Huang, Ruirui Li, Shiyang Li, Zheng Li, Yifan Gao, Xian Li, et al. Inductive or deductive? rethinking the fundamental reasoning abilities of llms. *arXiv preprint arXiv:2408.00114*, 2024.

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

Anwesa Choudhuri, Girish Chowdhary, and Alexander G Schwing. Ow-viscap: Open-world video instance segmentation and captioning. In *NeurIPS*, 2024.

Allen Cypher and Daniel Conrad Halbert. *Watch what I do: programming by demonstration*. MIT press, 1993.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. In *ACL Findings*, 2023.

Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*, 2024.

Zican Dong, Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. Bamboo: A comprehensive benchmark for evaluating long text modeling capacities of large language models. In *LREC-COLING*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Roberta A Ferrara, Ann L Brown, and Joseph C Campione. Children's learning and transfer of inductive reasoning rules: Studies of proximal development. *Child development*, 1986.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 2011.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Brett K Hayes, Evan Heit, and Haruka Swendsen. Inductive reasoning. *Wiley interdisciplinary reviews: Cognitive science*, 2010.

Evan Heit. Properties of inductive reasoning. *Psychonomic bulletin & review*, 2000.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. In *NeurIPS*, 2021.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? In *COLM*, 2024.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Yixing Jiang, Jeremy Irvin, Ji Hun Wang, Muhammad Ahmed Chaudhry, Jonathan H Chen, and Andrew Y Ng. Many-shot in-context learning in multimodal foundation models. *arXiv preprint arXiv:2405.09798*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

G. Kamradt. Needle in a haystack- pressure testing llms, 2023. URL `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`.

Subin Kim, Prin Phunyaphibarn, Donghyun Ahn, and Sundong Kim. Playgrounds for abstraction and reasoning. In *NeurIPS 2022 Workshop on Neuro Causal and Symbolic AI (nCSI)*, 2022.

Taiyu Lin Kinshuk and Paul McNab. Cognitive trait modelling: the case of inductive reasoning ability. *Innovations in Education and Teaching International*, 2006.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. In *NeurIPS Datasets and Benchmarks Track*, 2024.

Brenden M Lake, Tal Linzen, and Marco Baroni. Human few-shot learning of compositional instructions. In *Conference of the Cognitive Science Society*, 2019.

Tessa Lau, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. Programming by demonstration using version space algebra. *Machine Learning*, 2003.

Alan Leung, John Sarracino, and Sorin Lerner. Interactive parser synthesis by example. *ACM SIGPLAN Notices*, 2015.

Jiachun Li, Pengfei Cao, Zhuoran Jin, Yubo Chen, Kang Liu, and Jun Zhao. Mirage: Evaluating and explaining inductive reasoning process in language models. *arXiv preprint arXiv:2410.09542*, 2024a.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts? In *ACL*, 2024b.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *IEEE transactions on knowledge and data engineering*, 2020.

Mukai Li, Shansan Gong, Jiangtao Feng, Yiheng Xu, Jun Zhang, Zhiyong Wu, and Lingpeng Kong. In-context learning with many demonstration examples. *arXiv preprint arXiv:2302.04931*, 2023.

Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024c.

Wen-Ding Li and Kevin Ellis. Is programming by example solved by llms? In *NeurIPS*, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *NeurIPS*, 2023.

Adam Lopez. Statistical machine translation. *ACM Computing Surveys*, 2008.

Kaijing Ma, Xinrun Du, Yunran Wang, Haoran Zhang, Zhoufutu Wen, Xingwei Qu, Jian Yang, Jiaheng Liu, Minghao Liu, Xiang Yue, et al. Kor-bench: Benchmarking language models on knowledge-orthogonal reasoning tasks. *arXiv preprint arXiv:2410.06526*, 2024.

Arvind Mahankali, Tatsunori B Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. In *ICLR*, 2024.

Yunze Man, Liang-Yan Gui, and Yu-Xiong Wang. Situational awareness matters in 3d vision language reasoning. In *CVPR*, 2024.

Aditya Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, and Adam Kalai. A machine learning framework for programming by example. In *ICML*, 2013.

Behrang Mohit. Named entity recognition. In *Natural language processing of semitic languages*. 2014.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.

Brad A Myers. Visual programming, programming by example, and program visualization: a taxonomy. *ACM sigchi bulletin*, 1986.

OpenAI. Early access for safety testing, 2024. URL `https://openai.com/index/early-access-for-safety-testing/`.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

Tim Pearce, Tabish Rashid, Dave Bignell, Raluca Georgescu, Sam Devlin, and Katja Hofmann. Scaling laws for pre-training agents and world models. *arXiv preprint arXiv:2411.04434*, 2024.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *ICLR*, 2024.

Yiwei Qin, Xuefeng Li, Haoyang Zou, Yixiu Liu, Shijie Xia, Zhen Huang, Yixin Ye, Weizhe Yuan, Hector Liu, Yuanzhi Li, et al. O1 replication journey: A strategic progress report–part 1. *arXiv preprint arXiv:2410.18982*, 2024.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, et al. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. In *ICLR*, 2024.

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

Anian Ruoss, Fabio Pardo, Harris Chan, Bonnie Li, Volodymyr Mnih, and Tim Genewein. Lmact: A benchmark for in-context imitation learning with long multimodal demonstrations. *arXiv preprint arXiv:2412.01441*, 2024.

Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. Zeroscrolls: A zero-shot benchmark for long text understanding. In *EMNLP Findings*, 2023.

Yunfan Shao, Linyang Li, Yichuan Ma, Peiji Li, Demin Song, Qinyuan Cheng, Shimin Li, Xiaonan Li, Pengyu Wang, Qipeng Guo, et al. Case2code: Learning inductive reasoning with synthetic data. *arXiv preprint arXiv:2407.12504*, 2024.

Kensen Shi, Hanjun Dai, Wen-Ding Li, Kevin Ellis, and Charles Sutton. Lambdabeam: Neural program search with higher-order functions and lambdas. In *NeurIPS*, 2023.

Kensen Shi, Joey Hong, Yinlin Deng, Pengcheng Yin, Manzil Zaheer, and Charles Sutton. Exedec: Execution decomposition for compositional generalization in neural program synthesis. In *ICLR*, 2024.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. Clutrr: A diagnostic benchmark for inductive reasoning from text. In *EMNLP*, 2019.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

Mingyang Song, Mao Zheng, and Xuan Luo. Can many-shot in-context learning help long-context llm judges? see more, judge better! *arXiv preprint arXiv:2406.11629*, 2024.

Felix Stahlberg. Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 2020.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Zhiqing Sun, Sheng Shen, Shengcao Cao, Haotian Liu, Chunyuan Li, Yikang Shen, Chuang Gan, Liang-Yan Gui, Yu-Xiong Wang, Yiming Yang, et al. Aligning large multimodal models with factually augmented rlhf. In *ACL Findings*, 2024.

Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. Effective lstms for target-dependent sentiment classification. In *COLING*, 2016.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *ICML*, 2023.

Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. In *EMNLP*, 2024a.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D Goodman. Hypothesis search: Inductive reasoning with language models. In *ICLR*, 2024b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Yijia Xiao, Edward Sun, Tianyu Liu, and Wei Wang. Logicvista: Multimodal llm logical reasoning benchmark in visual contexts. *arXiv preprint arXiv:2407.04973*, 2024.

Yudong Xu, Wenhao Li, Pashootan Vaezipoor, Scott Sanner, and Elias B Khalil. Llms and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations. *TMLR*, 2023.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

Zonglin Yang, Li Dong, Xinya Du, Hao Cheng, Erik Cambria, Xiaodong Liu, Jianfeng Gao, and Furu Wei. Language models as inductive reasoners. In *EACL*, 2024b.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. Helmet: How to evaluate long-context language models effectively and thoroughly. *arXiv preprint arXiv:2410.02694*, 2024.

Peiwen Yuan, Shaoxiong Feng, Yiwei Li, Xinglin Wang, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Heda Wang, Yao Hu, and Kan Li. Focused large language models are stable many-shot learners. *arXiv preprint arXiv:2408.13987*, 2024.

Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets llm finetuning: The effect of data, model and finetuning method. In *ICLR*, 2024a.

Chi Zhang, Baoxiong Jia, Mark Edmonds, Song-Chun Zhu, and Yixin Zhu. Acre: Abstract causal reasoning beyond covariation. In *CVPR*, 2021.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, et al. ∞ bench: Extending long context evaluation beyond 100k tokens. *arXiv preprint arXiv:2402.13718*, 2024b.

Hao Zhao, Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Is in-context learning sufficient for instruction following in llms? *arXiv preprint arXiv:2405.19874*, 2024a.

Siyan Zhao, Tung Nguyen, and Aditya Grover. Probing the decision boundaries of in-context learning in large language models. In *NeurIPS*, 2024b.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. In *ICML*, 2024.

Kaijian Zou, Muhammad Khalifa, and Lu Wang. Retrieval or global context understanding? on many-shot in-context learning for long-context evaluation. *arXiv preprint arXiv:2411.07130*, 2024.

APPENDIX: MIR-BENCH: BENCHMARKING LLM'S LONG-CONTEXT INTELLIGENCE VIA MANY-SHOT IN-CONTEXT INDUCTIVE REASONING

The appendix is organized as follows. First, we include extended related work in Sec. A which we put in the appendix due to page limit, and conduct an extended comparison to all related many-shot ICL or inductive reasoning works to further illustrate the position of our work. Then, in Sec. B, we introduce more details in our experiments, including the prompts we adopted in our curation of dataset and ablation experiments and the regex rule we used for extracting the answer. After these, we provide statistical features of MIR-Bench in Sec. C, and more experiment results in Sec. D.

We hereby summarize the important novel insights obtained from experiments in the appendix:

1. LLMs tend to underestimate inductive reasoning difficulty during evaluation given a concise ground truth. A better choice is to do a multi-round evaluation where LLMs can better evaluate difficulty by self-reflection on its attempt for solving the problem. (Sec. B.5)

2. While generally adding more shots increases LLM's inductive performance, the performance change varies with problem types. LLMs improve the most on string manipulation tasks where each character in the input serves as a "shot" inside each example, and will not improve if the functions are too straightforward or too difficult. (Sec. C.2)

3. The evaluation on our benchmark is robust across different random seeds; i.e., the standard deviation of the performance is low. (Sec. D.1)

4. While some suggests that first let LLMs to write code, and then run code to get inductive results are helpful (Cheng et al., 2024), we find such paradigm are generally not scalable to many-shot case. Many-shot in-context inductive reasoning remains an open problem. (Sec. D.5)

5. The performance of LLMs against erroneous shot largely depends on the ratio of errorneous shots; under the same ratio, the total number of shots does not change much. (Sec. D.6)

A EXTENDED RELATED WORK

Programming-by-Examples (PbE). PbE (Myers, 1986; Cypher & Halbert, 1993) is a classic programming paradigm where programs are automatically written with user-provided input-output pairs as examples; it can be seen as an application of inductive reasoning in coding, and has wide application in sheet processing (Gulwani, 2011), data parsing (Leung et al., 2015), and systematic drawing (Cheema et al., 2014). It is traditionally addressed by symbolic-based approaches, such as heuristic search (Gulwani, 2011; Cheema et al., 2014), version space algebra (Lau et al., 2003) and learning weights for rule probabilities (Menon et al., 2013); this symbolic formulation has largely limited the generalizability of PbE. Recently, as LLMs have proved themselves to be strong coders (Guo et al., 2024), several works tried to address general-purpose PbE with LLMs (Shi et al., 2023; 2024; Shao et al., 2024; Li & Ellis, 2024). None of them, however, considers many-shot scenario with more than 10 shots. Compared to existing works, Our benchmark is organized in a way that resembles many-shot PbE paradigm, but for most of the evaluations, the LLMs we tested are not required to write code; instead, they only need to directly predict output for new input. That being said, with minimal adaptation, our proposed benchmark can fill in the blank of many-shot PbE study (and we explored this in Sec. D.5).

A complete comparison with prior works. Tab. 2 shows a detailed comparison of our work to existing works (including empirical study and benchmarks) in the field of many-shot and inductive reasoning. As shown in the table, our work is indeed unique among all the many-shot ICL and inductive reasoning works.

Table 2: Comparison on our benchmark with the most related prior many-shot / long-context benchmarks or empirical studies (first part) and inductive reasoning benchmarks or empirical studies (second part). To save space, we abbreviate "Many Shot" as MS, "Inductive Reasoning" as IR (△ represents "classification only"), "Prob." as problems, and "I/O Div." as "Input/Output Diversity" (**having at least** 2 **different input-output types, e.g., given an array and output an integer, or given a pair of strings and output a choice**). "Gen." means "Generative", which means whether new test cases can be easily generated without much human effort. "LB" means whether a leaderboard is available, and "EE" means "Easy Evaluation", i.e., whether a pipeline for evaluating any given new model exists. "New Data" means whether the input-output data never appears in existing benchmarks (△ means partly); if so, the benchmark is secured against data contamination. Note, the counting of #IR Problems and "Gen." take different target input-output for the same mapping into account, but **do not take different sets of shots into account**.

Evaluations	MS	IR	# IR Prob.	I/O Div.	Max # Shots	Gen.	LB	EE	New Data
Classifications (Li et al., 2023)	✓	△	∼25K	×	2000	×	×	×	×
Many-Shot ICL (Agarwal et al., 2024)	✓	✓	450	✓	2048	✓	×	×	△
Classifications (Bertsch et al., 2024)	✓	△	1250	×	2000	×	×	△	×
Visual Classifications (Jiang et al., 2024)	✓	△	4010	×	∼2000	×	×	✓	×
Instruction Following (Zhao et al., 2024a)	✓	×	0	✓	300	✓	×	✓	×
2D Classifications (Zhao et al., 2024b)	✓	△	100	×	256	✓	×	△	✓
LLM Judge (Song et al., 2024)	✓	×	0	×	512	✓	×	×	✓
HELMET (Yen et al., 2024)	✓	△	500	✓	∼10K	×	×	✓	×
LongICLBench (Li et al., 2024c)	✓	△	3000	×	∼2000	×	✓	✓	×
ManyICLBench (Zou et al., 2024)	✓	△	1000	✓	7252	×	×	×	×
LMAct (Ruoss et al., 2024)	✓	×	0	×	256	✓	×	✓	✓
LongBench (Bai et al., 2024)	✓	△	400	✓	600	×	✓	✓	✓
BABILong (Kuratov et al., 2024)	×	✓	unknown	✓	unknown	✓	✓	✓	✓
KORBench (Ma et al., 2024)	×	✓	50	✓	3	×	✓	✓	✓
SolverLearner (Cheng et al., 2024)	×	✓	1300	✓	16	✓	×	×	✓
Case2Code (Shao et al., 2024)	×	✓	1.3M	✓	10	×	×	×	✓
DEER (Yang et al., 2024b)	×	✓	1250	×	3	×	×	×	✓
List functions (Rule, 2020)	×	✓	4000	×	5	✓	×	✓	✓
SyGus (Wang et al., 2024b)	×	✓	89	✓	3	×	×	✓	✓
ARC (Chollet, 2019)	×	✓	800	×	3	×	✓	✓	✓
1D-ARC (Xu et al., 2023)	×	✓	900	×	3	×	×	✓	✓
Mini-ARC (Kim et al., 2022)	×	✓	150	×	3	×	×	✓	✓
WILT (Banatt et al., 2024)	×	✓	50	×	30	×	✓	✓	✓
LogicVista (Xiao et al., 2024)	×	✓	107	✓	10	×	×	✓	✓
CLUTRR (Sinha et al., 2019)	×	✓	70K	×	N/A	✓	×	✓	✓
MIRAGE (Li et al., 2024a)	×	✓	2000	✓	8	✓	×	×	✓
ACRE (Zhang et al., 2021)	×	✓	30K	×	10	✓	×	×	✓
Mini-SCAN (Qiu et al., 2024)	✓	✓	400	×	100	✓	×	✓	✓
Ours	✓	✓	6930	✓	2048	✓	✓	✓	✓

B More Experiment Details

B.1 Prompts for Main Results

We provide the prompt for the main results in Sec. 4.1 and Sec. 4.2 in the box below (the first commented line is not a part of the prompt):

```
# prompt for main results

You are given some function that takes something as input and output something. You need to predict the output for the target input of that function. Remember always end your answer with 'Output: your answer', with your answer in strict python format. Here are some examples:
Input: <example input 1>
Output: <example output 1>
Input: <example input 2>
Output: <example output 2>
... (omitting more shots)
Input: <target input>
```

B.2 Prompts for Ablations

Effectiveness of CoT. The following boxes demonstrate the prompt for the result used in Sec. 4.4 with forced CoT and no CoT respectively (the first commented line is not a part of the prompt):

```
# prompt for forced CoT

You are given some function that takes something as input and output something. You need to predict the output for the target input of that function. You need to first analyze it after 'Analysis:', then give your answer after 'Output:'. Remember always end your answer with 'Output: your answer', with your answer in strict python format. Here are some examples:
Input: <example input 1>
Output: <example output 1>
Input: <example input 2>
Output: <example output 2>
... (omitting more shots)
Input: <target input>
```

```
# prompt for no CoT

You are given some function that takes something as input and output something. You need to predict the output for the target input of that function. Your answer should always be 'Output: your answer', with your answer in strict python format. DO NOT OUTPUT ANYTHING ELSE INCLUDING YOUR THOUGHTS. Here are some examples:"
Input: <example input 1>
Output: <example output 1>
Input: <example input 2>
Output: <example output 2>
... (omitting more shots)
Input: <target input>
```

Robustness of LLM inductive intelligence. The following box demonstrates the prompt for the result used in Sec. 4.5. For the "unaware" setting, we use the same prompt as that in the main results; for the "aware error" and "aware ratio" setting, we use the following prompts respectively:

prompt for “aware error”

You are given some function that takes something as input and output something. You need to predict the output for the target input of that function. Remember always end your answer with ’Output: your answer’, with your answer in strict python format. Here are some examples. Note that not all shots are correct; there are a small portion of shots that are incorrect:
Input: <example input 1>
Output: <example output 1>
Input: <example input 2>
Output: <example output 2>
. . . (omitting more shots)
Again, note that not all shots are correct; there are a small portion of shots that are incorrect. Use your caution and think wisely.
Input: <target input>

prompt for “aware ratio”

You are given some function that takes something as input and output something. You need to predict the output for the target input of that function. Remember always end your answer with ’Output: your answer’, with your answer in strict python format. Here are some examples. Note that not all shots are correct; there are <number of error shots> out of <total number> shots that are incorrect:
Input: <example input 1>
Output: <example output 1>
Input: <example input 2>
Output: <example output 2>
. . . (omitting more shots)
Again, note that not all shots are correct; <number of error shots> out of <total number> shots that are incorrect. Use your caution and think wisely.
Input: <target input>

B.3 PROMPTS FOR REFORMATTING APPS PROBLEMS (SEC. 3.2)

The following box demonstrates the prompt for reformatting APPS problems in the "function collection" part of Sec. 3.2.

```
# prompt for reformatting
You are a coding expert. You will be given a problem and corresponding solution. Rewrite the solution
such that:
1. It becomes a single function named 'solution', which takes parameters as input instead of reading from
input() function if there is any;
2. There is no code out of the solution function and no solution class. All auxiliary functions should be
defined inside the solution function, and all imports should also be in the function.
3. The solution function should not have any print() function. Instead, it should return the result of the
function. If you need to output any rationale, leave them in comments. Your output must be directly
runnable without any change.
4. Just output the rewritten function; do not test it with extra statements.
Here is an example:
[[Problem]]
problem: Given a string, you need to reverse the order of characters in each word within a sentence while
still preserving whitespace and initial word order.
Example 1:
Input: "Let's take LeetCode contest"
Output: "s'teL ekat edoCteeL tsetnoc"
Note:
In the string, each word is separated by single space and there will not be any extra space in the string.
[[Solution]]
class Solution:
    def reverseWords(self, s):
        """
        :type s: str
        :rtype: str
        """
        rev_str = s[::-1]
        rev_arr = rev_str.split()
        final = rev_arr[::-1]
        return ' '.join(map(str, final))
[[Rewrite]]
def solution(s):
    """
    :type s: str
    :rtype: str
    """
    rev_str = s[::-1]
    rev_arr = rev_str.split()
    final = rev_arr[::-1]
    return ' '.join(map(str, final))
```

B.4 PROMPT FOR THE GENERATION OF DATA GENERATOR

The following box demonstrates the prompt for generating data generator:

```
# prompt for generating data generator
You are a coding expert. You will be provided a coding question and corresponding solution. Please write two python function that randomly generates test case for the question. Specifically:
The first function's name is gen1, which generates random data (should be able to generate VERY DIVERSE, i.e., at least 1000 different data points).
The second function's name is gen2, which generates data that is slightly harder than those generated in gen1. (should be able to generate at least 100 different data points).
You shall not define any function outside gen1 or gen2. Should you use any helper function, make them inner functions inside gen1 or gen2. You gen1 and gen2 function should have and only have one int parameter, which is the number of cases.
Finally, the special cases should be designed as informative as possible that reveals the underlying function when looking at the input and corresponding output from the solution.
Here is an example. Note the output of gen1 and gen2 should be a list of dicts describing the parameters, and your special case input should be a dict describing the parameters. Please follow the format, and do not generate cases that are too long. Do not output any other text; put all your thoughts after "# rationale:" as shown in the example.
[[Problem]]
from typing import List
    def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
from typing import List
[[Solution]]
sorted_numbers = sorted(numbers)
for i in range(len(sorted_numbers) - 1):
if sorted_numbers[i + 1] - sorted_numbers[i] < threshold:
    return True
return False
[[Gen1]]
# rationale: none
import random
def gen1(num_cases: int):
    low, high = 5, 10 # generate lists between length 5 to 10
    data = []
    for i in range(num_cases):
        N = random.randint(low, high)
        lst = [round(random.random() * 10, 1) for _ in range(N)]
        threshold = round(random.random(), 1) + 0.1
        data.append('numbers': lst, 'threshold': threshold)
    return data
[[Gen2]]
import random
def gen2(num_cases: int): # rationale: the data is slightly harder as the list is slightly longer
    low, high = 10, 20 # generate lists between length 10 to 20
    data = []
    for i in range(num_cases):
        N = random.randint(low, high)
    lst = [round(random.random() * 10, 1) for _ in range(N)]
    threshold = round(random.random(), 1) + 0.1
    data.append('numbers': lst, 'threshold': threshold)
    return data
```

B.5 LLM Difficulty Labeling

In Sec. 4.2, we discussed several factors for problems which leads to LLM's few-shot and many-shot performance difference, and one of them is LLM-evaluated function difficulty level. While seemingly straightforward, a reliable evaluation of difficulty for LLM is highly non-trivial, as **LLM tend to underestimate inductive reasoning difficulty when given a simple ground truth function.** We provide an example below in the box with GPT-4o-latest as the prompted model:

Prompt
You are an expert in coding and inductive reasoning. You will now be given a function that solves some problems in python and some example input-output pairs.
Please rate the difficulty of guessing the function through input-output pairs. You should judge the difficulty NOT on whether the function itself is easy to write, but on whether it is easy to guess. You should first give your rationale after [[Analysis]], then end your answer with 'Overall Score: [[x]]', where x is your score from 1 to 10.
[[code]]

```
def solution(l):
    return [l[i] for i in range(1, len(l)) if l[i] % i == 0]
```

[[data]]
Input: 'l': [40, -13]
Output: [-13]
Input: 'l': [-87, 29, -27, 90]
Output: [29, 90]
Input: 'l': [48, -4, 16, -63, -32, -5, -32, -45]
Output: [-4, 16, -63, -32, -5]
Input: 'l': [-75, 77, 44, 10, 17, -71, -80, -11, -68, -41]
Output: [77, 44]
Input: 'l': [87, 71]
Output: [71]
LLM's response
[[Analysis]] To solve this puzzle, the key is to identify the rule used to filter the input list 'l'. By observing the given input-output pairs and the function definition, the pattern can be deduced as follows:
Function Logic. The function constructs a new list from the original list 'l', starting from index '1'. For each element at index 'i', it checks if the element is divisible by its **index** (using 'l[i] % i == 0'). If true, it includes the element in the resulting list.
Difficulty Analysis Based on Input-Output Pairs. **Clues that might help guess the function:** 1. **The output never includes the first element of the input list.** This suggests that index-based filtering is involved.
2. **Only specific elements from the input list appear in the output.** A guesser would likely try to find what distinguishes the selected elements from the others. Given the presence of both positive and negative numbers, divisibility by index is a plausible criterion.
3. **The divisibility condition isn't obvious at a glance.** It isn't immediately clear that elements are being checked for divisibility by their **index**, as it's somewhat unconventional. Most people would initially look for a rule involving the values themselves, like even/odd checks, thresholds, or signs.
What Makes It Difficult to Guess?
1. The divisibility rule by index is **non-intuitive**. Divisibility by **element values** is a common guess, but divisibility by **index** is not.
2. The function starts from index '1' instead of '0'. This subtle detail could mislead a guesser into thinking it's a filtering rule based on the **position** in the list rather than the index value.
3. The rule applies to **all elements**, regardless of their sign or magnitude. This broad applicability makes it harder to pin down the exact condition.
What Makes It Easier to Guess?
1. **Simple filtering condition:** Despite the challenge, once a guesser considers index-based filtering, the rule becomes easy to validate.
2. **The input-output pairs are consistent.** There are no exceptions or edge cases that would throw off a guesser.
Overall Score: [[7]] (moderately difficult)

GPT-4o-latest gives an average difficulty score of 7.2 / 10 out of 5 evaluations (which indicates moderate-to-high difficulty), while GPT-4o-0806 gives much lower difficulty score (below 5). This problem, however, is in fact so difficult that we have to exclude it from our benchmark, i.e., have 0 accuracy for all five models $\{$GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2$\}$ across $\{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$ shots in 10 test cases.

To address such issue, we adopt a multi-round conversation approach with self-reflection. We first let the LLM to try to solve the problem without code by itself, and then reveal the ground-truth answer and let LLM to score the difficulty based on self-reflection. The box below demonstrates an example:

Round 1 prompt
You are an expert in inductive reasoning. You will now be given a function that solves some problems and some example input-output pairs. You need to guess the answer for the one last input. After that, you will be given the answer and the code, you need to do a self-reflection on your attempt, and rate a difficulty level for future guessers. Here are some examples:
<few-shot examples of the whole conversation>
Here is the target problem:
[[Problem]]
Input: {'lst': ['V4', 'V10', 'V4', 'V0', 'V9', 'V4', 'V2', 'V14', 'V16', 'V9']}
Output: ['V0', 'V2', 'V4', 'V4', 'V4', 'V9', 'V9', 'V10', 'V14', 'V16']
. . .
Input: {'lst': ['V0', 'V13', 'V1', 'V4', 'V2', 'V10', 'V6', 'V5', 'VB', 'V1']}
Output: ['VB', 'V0', 'V1', 'V1', 'V2', 'V4', 'V5', 'V6', 'V10', 'V13']
Input: {'lst': ['V0+', 'V3', 'V14', 'V13']}
Output: ['V0+', 'V3', 'V13', 'V14']
Input: {'lst': ['V12', 'V6', 'V7', 'V2', 'VB', 'V2', 'V3', 'V0', 'V4', 'V6']}
[[Solution]]
Round 1 response
The solution appears to sort the elements in each list lexicographically, maintaining any duplicates in their respective positions within the sorted order.
Following this rule, for the input ''lst': ['V12', 'V6', 'V7', 'V2', 'VB', 'V2', 'V3', 'V0', 'V4', 'V6']', the output should be:
Output: '['V0', 'V2', 'V2', 'V3', 'V4', 'V6', 'V6', 'V7', 'V12', 'VB']'

Round 2 prompt
[[Response]] Now, I will tell you the answer is ['VB', 'V0', 'V2', 'V2', 'V3', 'V4', 'V6', 'V6', 'V7', 'V12']. Here is the code for the function:

```
def solution(lst):
def grade(v):
        if v == 'VB': return -2
        if v == 'V0': return -1
        if v == 'V0+': return 0
        return int(v[1:])
    return sorted(lst, key=grade)
```

Based on the answer, your guess and the code, do a self-reflection on your attempt, and rate a difficulty level between 1 to 10 inclusive for future guessers with different shots sampled for the same function. You MUST always end your answer with "Difficulty: [[x]]" where x is 1 to 10.
Round 2 response
In my initial attempt, I focused on lexicographical sorting but overlooked the specific priority handling in the function for values like "VB" and "V0." The code clarifies that "VB" should always appear first, followed by "V0," and then "V0+," with all other values ordered by their numeric suffix. This customized grading system is subtle, especially without the code, as it combines both conditional priorities and numeric sorting.
Overall, understanding the need for this specific grading logic adds some complexity, but it's manageable once recognized.
Difficulty: [[4]]

With such paradigm, we found that the evaluation from LLMs are much more accurate.

B.6 Answer Extraction

We use the following way to extract final answer:

1. find the last occurence of "utput:" in LLM's response, which can be either "output" or "Output";
2. remove trailing "**" after "output:" for bold, "' and "'python in the answer for python format, and then remove all leading and trailing whitespaces and linebreaks;
3. conduct an exact match. If succeeds, the LLM is considered to be correct; otherwise, we try to convert the output into a python object by using ast.literal_eval(). If the convertion fails, the LLM is considered to be incorrect.

Table 3: Number of functions extracted from each coding benchmark.

	HumanEval+	MBPP+	APPS	Total
MIR-Core	26	35	239	300
MIR-Extended	53	89	551	693

4. If the output can be converted to a single-key dictionary or single-element set, we will do an exact match between the value of the dictionary / element of the set to the ground truth answer with both converted to string (This is to account for responses similar to {"ans": 3} with ground truth being 3); otherwise, we do an exact match between the whole output and the ground truth answer converted to string.

B.7 Robustness Test: Erroneous Shots

We generate test cases with erroneous shot in the following way:

1. For n-shot with a given error rate ER, randomly sample $ER \times n$ indices to be the "erroneous shots" with incorrect answer. $ER \times n$ is guaranteed to be an integer.
2. for each "erroneous shot", we randomly sample one unused shot as we generate 20000 shots for each function, and substitute the original output with the selected shot's output. We will re-sample the unused shot if its answer is identical with the original shot.

C Statistical Features of MIR-Bench

C.1 Data Source

Tab. 3 shows that out of 693 functions in MIR-Extended and 300 functions in MIR-Core, how many problems are extracted from each coding benchmark (HumanEval+, MBPP+ and APPS).

C.2 Problem Types in Sec. 4.2

To study the effect on the topic of the problems for whether the problem benefits from many-shot, we first try to cluster the 693 problems in MIR-Extended using GPT-4o-0806. More specifically, we first prompt the LLM to generate python-style tags for each problem with the following prompt:

```
# prompt for tags
You are an expert in coding. You will now be given a function that solves some problems and some example input-output pairs. You need to briefly summarize what the function is about in a tag in high-level, with no more than 5 words connected with '_'. DO NOT OUTPUT ANYTHING ELSE. Here are some examples:
<some examples>
[[Code]]
...
[[Input-Output Pairs]]
...
[[Answer]]
```

after acquiring tags for each problem, we prompt the LLM to merge all different tags down to 30 different tags with 6 major types: {List Analysis, List Manipulation, Mathematical Computations, String Analaysis, String Manipulations, Other}. Tab. 4 shows the number of problems, detailed tags and metric D (defined in Eq. (1)) for each problem type.

Based on the results, we find that generally adding more problem will have a positive effect on performance; however, for some types of problem such as geometric calculation and summation, the performance will decrease with more shots included. Upon checking those problems, we found them mostly fall into two categories: 1) the function is relatively straightforward, but the LLM gets confused with more shots due to over-complicated guesses; 2) the function is too hard to guess, and the LLM cannot make reasonable guesses when aggregating many pieces of information. The boxes below give examples for case 1) and 2) respectively:

```
# Case 1: Straightforward Problems
[[Code]]
def solution(num: int) → int:
    steps = 0
    while num > 0:
        if num % 2 == 0: num /= 2
        else: num -= 1
        steps += 1
    qquad return steps
[[Input-Output Pairs]]
Input: {'num': 68037}
Output: 23
...
```

```
# Case 2: Difficult Problems
[[Code]]
def solution(boardSize, initPosition, initDirection, k):
    yq, yr = divmod(initPosition[0] + k * initDirection[0], 2 * boardSize[0])
    xq, xr = divmod(initPosition[1] + k * initDirection[1], 2 * boardSize[1])
    # Calculate the final position considering reflections
    return [min(yr, 2 * boardSize[0] - yr - 1), min(xr, 2 * boardSize[1] - xr - 1)]
[[Input-Output Pairs]]
Input: {'boardSize': [10, 11], 'initPosition': [5, 9], 'initDirection': [1, -1], 'k': 264}
Output: [9, 9]
```

On the other hand, problems such as removing duplicates and string manipulation generally benefit more from many-shot, probably because manipulation on each element / character can be considered a shot by itself, and thus the effective number of "shots" in such types of problems are higher. Note, "Others" problems have relatively high value of D because of one outlier; other than the outlier, it is almost equal to average level of D across MIR-Extended.

C.3 Other Factors Studied in Sec. 4.2

Fig. 7 illustrates the relation between our metric D (see Eq. (1) for definition) for distinctiveness between few-shot and many-shot performance.

D More Experiment Results

D.1 Analysis on the Stability of Evaluation

As we use 0 temperature in the evaluations in our main papers, it is possible that the performance vary across LLM inferences with different random seeds, hence making our evaluation unreliable. To address such concern, we report the mean and standard deviation of the performance across 5 models {GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2} over 5 different inferences with temperature 0.7 in Fig. 8 on MIR-Extended with $4, 8, 16, 32, 64, 128, 256, 512$ shots. The result clearly shows that the standard deviation for all models are very small, and thus our evaluation is reliable.

D.2 Complete Results on MIR-Extended and MIR-Core (Sec. 4.1, 4.2)

For better readability, we only put the performance of part of the models for MIR-Extended and MIR-Core in the main paper; Fig. 9 demonstrates the performance of all models.

Table 4: The tags for problem topics and related statistics; D is the average metric (see Eq. (1) for definition) of the corresponding type of problems in MIR-Extended. We marked entries with $D > 0.1$ **blue** and $D < -0.1$ **red.** While increasing the number of shots generally brings better performance, We find that string manipulation benefits the most from many-shot.

Major Tag	Minor Tag	# Questions in MIR-Extended	# Questions in MIR-Core	D
List Analysis	Counting Elements	4	2	0.075
List Analysis	Counting Occurences	25	9	0.017
List Analysis	Maximum/Minimum Elements	30	5	-0.024
List Analysis	Statistics	2	0	0.041
List Analysis		61	16	0.001
List Manipulation	Filtering Elements	30	14	0.066
List Manipulation	Generating Sequences	15	6	0.077
List Manipulation	Mapping Elements	9	3	-0.022
List Manipulation	Removing Duplicates	6	4	**0.136**
List Manipulation	Sorting Elements	16	7	-0.041
List Manipulation		76	34	0.041
Mathematical Computations	Basic Arithmetic	35	12	0.085
Mathematical Computations	Boolean Determination	7	0	0.033
Mathematical Computations	Calculations Based on Formulas	98	32	0.051
Mathematical Computations	Condition Checking	43	20	**0.114**
Mathematical Computations	Geometric Calculation	4	2	**-0.110**
Mathematical Computations	Number Base Conversions	12	5	0.038
Mathematical Computations	Rounding	7	5	**0.185**
Mathematical Computations	Summation	5	0	**-0.128**
Mathematical Computations		211	76	0.066
String Analysis	Character Code Calculations	14	3	-0.004
String Analysis	Comparison	13	6	**0.255**
String Analysis	Counting Characters	29	10	0.050
String Analysis	Pattern Matching	27	7	0.084
String Analysis		83	26	0.084
String Manipulation	Case Transformation	19	7	0.089
String Manipulation	Encryption/Decryption	8	4	0.061
String Manipulation	Generating Substrings	8	4	-0.027
String Manipulation	Rearranging Characters	48	22	**0.105**
String Manipulation	Substitution	36	30	**0.327**
String Manipulation	Substring Replacement	33	22	**0.205**
String Manipulation	Swapping Parts	6	5	-0.053
String Manipulation	Transformation	34	17	0.083
String Manipulation		192	111	**0.160**
Others		76	37	**0.123**
Total		693	300	0.092

D.3 Out-of-Context Rate for 1024 and 2048 shot in MIR-Extended

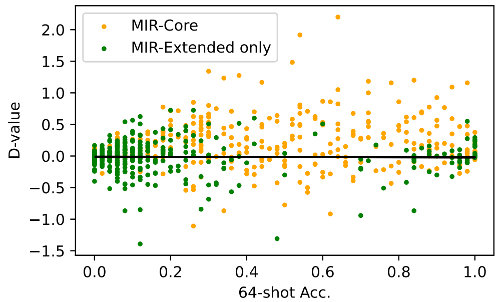
Tab. 5 shows the rate of out-of-context error we received when invoking APIs for MIR-Extended. Some models other than Gemini (which has $> 1M$ context length) have an error rate of 0, which could due to its internal truncation.

D.4 More Results on Many-Shot Inductive Reasoning with CoT

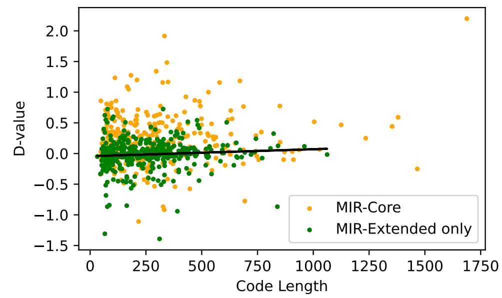
Tab. 6 lists the ratio of CoT / no CoT and their respective performance for more models on MIR-Core using original prompt. Fig. 10 illustrates the performance difference for more models between forced CoT and no CoT. The result shows that results with no CoT are indeed better than those with forced CoTs, and such gap increases with the number of shots.

D.5 SolverLearner

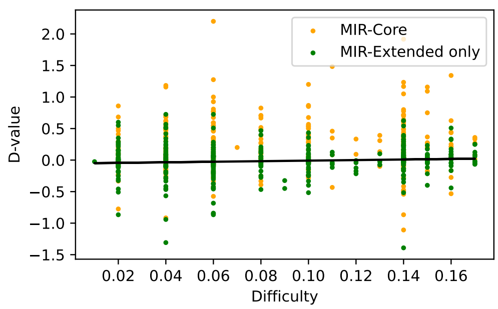
Cheng et al. (2024) proposed SolverLearner, an inductive reasoning framework where LLMs write code first for inductive reasoning problems and then generate answers with python interpreter. With such framework, the authors claim that LLMs demonstrate remarkable inductive reasoning capabilities under their framework. However, their study is limited to a few relatively weak LLMs, (GPT-3.5, GPT-3), limited amount of inductive reasoning problems and few-shot; to check whether such solution also works for the many-shot case, we re-implement their method on MIR-Core.



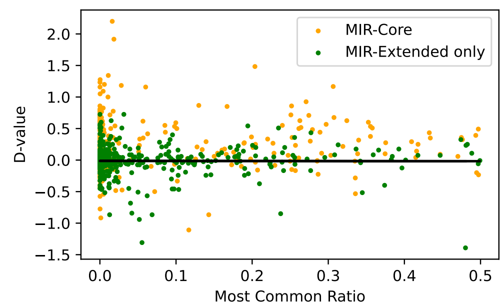
(a) 64-Shot accuracy



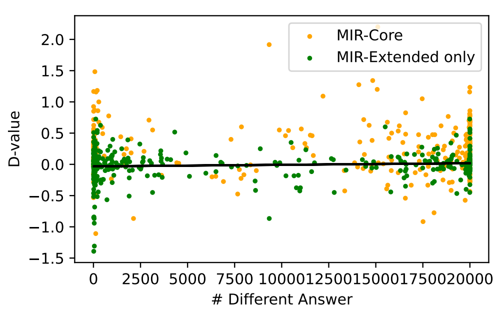
(b) Code length



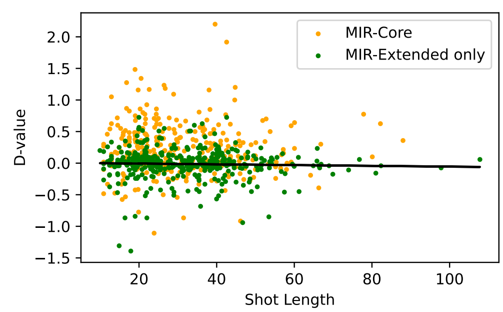
(c) Difficulty level



(d) Most common answer ratio



(e) # Different answer



(f) Shot length

Figure 7: Single-factor analysis between each factor in Sec. 4.2 and our distinctiveness metric D. Each point represents one of the 693 functions in our benchmark. The black line is the linear regression result of all functions in MIR-Extended; it is clearly shown that D is positively related to difficulty level and code length. The 64-shot accuracy is an average of {GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2} over 10 test cases.

Below are the prompt we used for generating code solution:

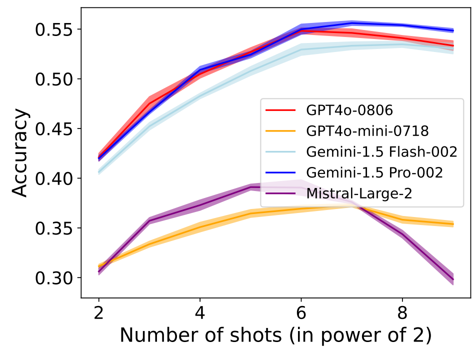
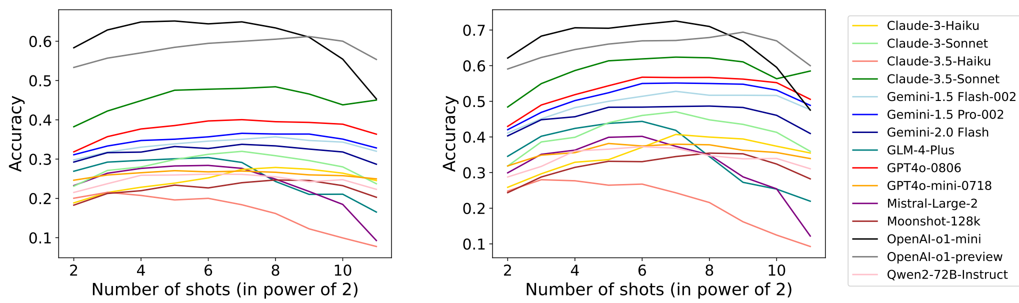


Figure 8: The performance of 5 cutting-edge LLM models on MIR-Extended with temperature 0.7 across 5 runs. The result clearly shows that the standard deviation of accuracy is always below 0.01, and thus the evaluation is highly stable.



(a) MIR-Extended (b) MIR-CoRE

Figure 9: The performance of all LLMs on MIR-Extended (panel (a)) and MIR-Core (panel (b)). As shown in Fig. 2 in the main paper, the benchmark poses challenge to all models tested including o1-preview and o1-mini. Most models will "saturate" at a particular number of shots, i.e., their performances stop to improve when more shots are given due to limited information integration capability.

Prompt for SolverLearner
You are given some function that takes something as input and output something. You need to write a python code of the function. You need to write your rationale after # (as if it is a python comment), and give your answer after 'Code:'. DO NOT OUTPUT ANYTHING ELSE. Your function name should be 'solution'. You are not allowed to write other custom functions unless it is inside 'solution'. Use imports before using package functions. You must strictly follow python format, especially input / output format (e.g., if it is a dictionary, your param should also be a dictionary). DO NOT ADD ANY STATEMENT FOR EVALUATION AFTER 'solution'. Here are the input-output pairs for the function, with input followed by output:"
Input: <input 1>
Output: <output 1>
...
Input: <input n>
Output: <output n>
Here is your code. Again, do not output anything else; Your function name should be 'solution'. You are not allowed to write other custom functions unless it is inside 'solution'. Use imports before using package functions. You must strictly follow python format, especially input / output format (e.g., if it is a dictionary, your param should also be a dictionary). DO NOT ADD ANY STATEMENT FOR EVALUATION AFTER 'solution'.
Code:

We test SolverLearner with {Claude-35-Sonnet, GPT-4o-0806, GPT-4o-mini-0718, Gemini-Pro 1.5-002, Gemini-Flash 1.5-002, Mistral-Large-2} for $\{16, 64, 256, 1024\}$ shots respectively on MIR-

Table 5: Out-of-context rate for model API calls on MIR-Extended.

Model	1024-shot (%)	2048-shot (%)
Claude-3-Haiku	0	1.67
Claude-3-Sonnet	0	1.67
Claude-3.5-Haiku	0.96	2.4
Claude-3.5-Sonnet	0	1.77
Gemini 1.5-Flash-002	0	0
Gemini 1.5-Pro-002	0	0
Gemini 2.0-Flash	0	0
GLM-4-Plus	0	0
GPT-4o-0806	0.33	5.67
GPT-4o-mini-0718	0.33	5.67
Mistral-Large-2	0.67	10.67
Moonshot-128K	0	0
o1-mini-0912	1	11
o1-preview-0912	1	11
Qwen2-72B-Instruct	0	0

Table 6: The results on MIR-Core of each model with and without CoT. Results are averaged over $\{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$-shot. Error rate include cases where answer cannot be extracted and API error for exceeding context length. Claude-3.5-Haiku often refuses to answer the question due to "incomplete data".

Model	Answer w./ CoT (%)	Accuracy w./ CoT	Answer w./o. CoT (%)	Accuracy w./o. CoT	Error (%)
Claude-3-Haiku	51.03	0.278	47.66	0.441	1.31
Claude-3-Sonnet	20.46	0.233	76.98	0.475	2.56
Claude-3.5-Haiku	65.11	0.317	1.32	0.823	33.57
Claude-3.5-Sonnet	98.73	0.585	1.08	0.775	0.19
Gemini 1.5-Flash-002	20.11	0.306	79.75	0.539	0.14
Gemini 1.5-Pro-002	20.96	0.339	78.90	0.561	0.14
Gemini 2.0-Flash	24.43	0.363	74.59	0.498	0.98
GLM-4-Plus	19.70	0.248	79.33	0.388	0.97
GPT-4o-0806	10.85	0.488	88.34	0.540	0.81
GPT-4o-mini-0718	37.21	0.279	61.77	0.414	1.02
Mistral-Large-2	75.66	0.306	21.64	0.403	2.70
Moonshot-128K	43.40	0.242	53.11	0.398	3.50
o1-mini-0912	56.71	0.588	40.88	0.775	2.41
o1-preview-0912	2.54	0.334	93.63	0.696	3.82
Qwen2-72B-Instruct	1.85	0.130	97.05	0.349	1.10

Core. For each code snippet, generated by LLMs, we set a limit of 1 second for execution, as we need to run 300 functions $\times$ 10 test cases $\times$ 4 different number of shots $\times$ 6 models = 72000 code snippets.

Tab. 7 demonstrates the accuracy of each model (with difference from the standard inductive reasoning paradigm) on MIR-Core, and Tab. 8 demonstrates the error rate when writing code. We found that the effect of SolverLearner varies from model to model; i.e., SolverLearner does not necessarily improve performance on our benchmark. Also, SolverLearner does not seem to utilize many-shot well; the performance increase from 16-shot to 1024-shot is much smaller than that of standard inductive reasoning paradigm. We hypothesize such issue, similar to that in Sec. 4.4, stems from the complicated nature of the code. Also, for models with relatively weaker long-context ability (e.g. Mistral-Large-2), the error rate with many-shot will largely increase beyond its "effective" (Hsieh et al., 2024) context length. Thus, many-shot inductive reasoning is still an open problem and not yet solved by straightforward solutions such as SolverLearner.

D.6 Complete Results on Robustness of LLM Inductive Intelligence

Fig. 11 shows the results of models on 64-shot, 256-shot and 1024-shot with different error rate for the shots, where the solid lines are 256-shot or 1024-shot accuracy respectively. We find that there are no significant performance difference across the same error rate with different number of shots

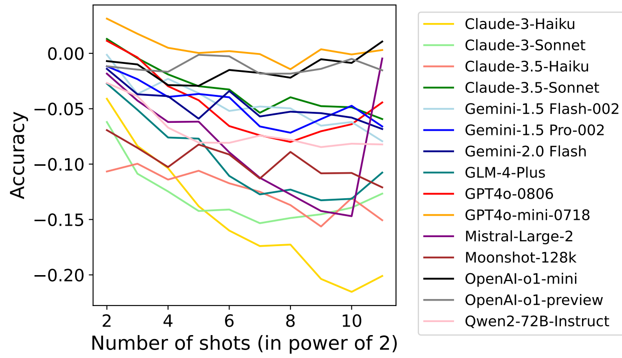


Figure 10: Performance difference for all LLMs on MIR-Core between forced CoT and no CoT (dotted lines). Results forced CoTs almost always works worse (with the exception of GPT4o-mini-0718 and o1 models), and such gap increases with the number of shots. Mistral-Large-2's gap decreases dramatically at 2048-shot as such setting often exceeds its context length and the performance is low under both settings.

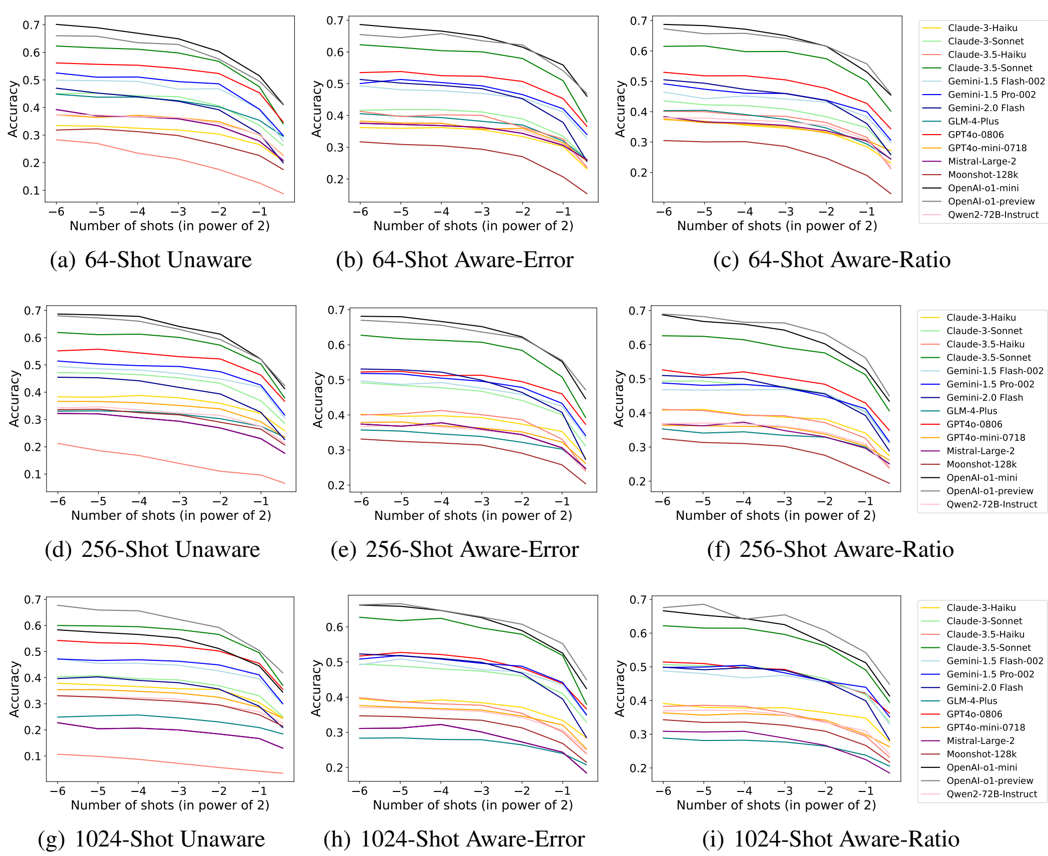
Table 7: The accuracy at 16, 64, 256 and 1024 shots respectively for SolverLearner on MIR-Core, and its performance difference from normal inductive reasoning paradigm. We plot improvements with > 0.02 **blue** and < -0.02 **red**. We find that the performance of SolverLearner varies from model to model, and does not necessarily perform better than normal inductive reasoning paradigm. Also, SolverLearner curves under many-shot are more "flat"; i.e., it does not seem to make good use of extra information from a large number of shots. Such result indicates that LLM many-shot inductive reasoning is still an open problem, and straightforward solutions such as SolverLearner are not suffice yet.

Model	Acc.@16	Acc.@64	Acc.@256	Acc.@1024
Claude-3.5-Sonnet	0.577(-0.009)	0.604(-0.015)	0.605(-0.017)	0.603(+0.04)
GPT4o-0806	0.53(+0.012)	0.534(-0.033)	0.538(-0.029)	0.556(+0.004)
GPT4o-mini-0718	0.350(-0.006)	0.375(+0.003)	0.386(+0.008)	0.370(+0.014)
Gemini-1.5 Flash-002	0.473(-0.009)	0.484(-0.03)	0.479(-0.038)	0.486(-0.03)
Gemini-1.5 Pro-002	0.469(-0.029)	0.495(-0.055)	0.483(-0.067)	0.491(-0.04)
Mistral-Large-2	0.42(+0.057)	0.430(+0.028)	0.428(+0.078)	0.356(+0.102)

(with the exception of o1-mini-0912 with 1024 shots), and the robustness persists across different number of shots.

Table 8: The Do-Not-Finish (i.e., no solution function generated) and Runtime Error (RE, including timeout and exception during running) rate at 16, 64, 256 and 1024 shots respectively for Solver-Learner on MIR-Core. Generally, with more shots, the error rate of LLMs will increase. For models with relatively weaker long-context ability such as Mistral-Large-2, the error rate will wildly increase under long context scenario.

Model	DNF@16	RE@16	DNF@64	RE@64	DNF@256	RE@256	DNF@1024	RE@1024
Claude-3.5-Sonnet	0	0.0027	0	0.0063	0	0.0007	0	0.0037
GPT4o-0806	0	0.009	0	0.0103	0	0.0157	0.0033	0.0137
GPT4o-mini-0718	0	0.0103	0	0.0147	0	0.0167	0.0033	0.017
Gemini-1.5 Flash-002	0	0.0093	0	0.0117	0	0.0087	0	0.011
Gemini-1.5 Pro-002	0	0.0093	0	0.008	0	0.009	0	0.0107
Mistral-Large-2	0	0.008	0	0.0077	0.0047	0.012	0.1163	0.0473



(a) 64-Shot Unaware (b) 64-Shot Aware-Error (c) 64-Shot Aware-Ratio

(d) 256-Shot Unaware (e) 256-Shot Aware-Error (f) 256-Shot Aware-Ratio

(g) 1024-Shot Unaware (h) 1024-Shot Aware-Error (i) 1024-Shot Aware-Ratio

Figure 11: The accuracy of models with errorneous shots under different prompt settings. The performance of the same error rate with different numbers of total shots are similar.