Generating Diverse Hypotheses for Inductive Reasoning

Kang-il Lee[1] **Hyukhun Koh**[2] **Dongryeol Lee**[1]
Seunghyun Yoon[3] **Minsung Kim**[1] **Kyomin Jung**[1,2*]
[1]Dept. of ECE, Seoul National University [2]IPAI, Seoul National University
[3]Adobe Research
{4bkang,hyukhunkoh-ai,drl123,kms0805,kjung}@snu.ac.kr
syoon@adobe.com

Abstract

Inductive reasoning — the process of inferring general rules from a small number of observations — is a fundamental aspect of human intelligence. Recent works suggest that large language models (LLMs) can engage in inductive reasoning by sampling multiple hypotheses about the rules and selecting the one that best explains the observations. However, due to the IID sampling, semantically redundant hypotheses are frequently generated, leading to significant wastage of compute. In this paper, we 1) demonstrate that increasing the temperature to enhance the diversity is limited due to text degeneration issue, and 2) propose a novel method to improve the diversity while maintaining text quality. We first analyze the effect of increasing the temperature parameter, which is regarded as the LLM's diversity control, on IID hypotheses. Our analysis shows that as temperature rises, diversity and accuracy of hypotheses increase up to a certain point, but this trend saturates due to text degeneration. To generate hypotheses that are more semantically diverse and of higher quality, we propose a novel approach inspired by human inductive reasoning, which we call Mixture of Concepts (MoC). When applied to several inductive reasoning benchmarks, MoC demonstrated significant performance improvements compared to standard IID sampling and other approaches.

1 Introduction

Inductive reasoning, inferring general rules that explain a small number of observations (Figure 1), is a key factor of human intelligence (Lake et al., 2015; Chollet, 2019). Recently, there has been exploration into automatic inductive reasoning using large language models (LLMs). Trained with vast corpora and instruction tuning, LLMs can function as zero-shot rule proposers when given an inductive reasoning problem (Qiu et al., 2024). By leveraging this capability, a new paradigm in automatic inductive reasoning has emerged: feeding observations (often referred to as train examples) as prompts into the LLM, sampling multiple hypotheses about the rule, and selecting the one that best explains the observations (Wang et al., 2024b; Greenblatt, 2024; Brown et al., 2024). However, when the hypotheses are generated by LLMs are fundamentally IID, it often leads to redundant sampling of semantically identical hypotheses. Such redundancy reduces the diversity of hypotheses and results in a significant waste of compute.

To address aforementioned problem, we begin by increasing the temperature, which is often considered a diversity parameter for LMs, in IID sampling setup. Our findings indicate that while both diversity and accuracy improve up to a certain point with increased temperature, they quickly reach saturation. This is because raising the temperature deteriorates the quality of hypotheses, and increases the rate of text degeneration (Holtzman et al., 2020).

To generate diverse hypotheses while maintaining their quality, we propose a simple yet effective method called Mixture of Concepts (MoC). Inspired by human inductive reasoning (Hofstadter, 1979; Mitchell, 2021), our approach consists of two stages: *concept proposal* and *hypothesis generation*. In the concept proposal, we instruct the LLM to generate "a list of K concepts" that are likely to aid in the formation of hypotheses, and semantically non-redundant concepts are generated *sequentially*. In the hypothesis generation, these concepts are parsed and each concept is provided as a hint for generating hypotheses. Our approach allows for the *parallel* generation of semantically diverse hypotheses without hurting the quality of hypotheses (Figure 1).

We conduct experiments on four inductive reasoning datasets from distinct domains. Our methodology significantly improves performance across four LLMs by generating diverse hypotheses while

[*]Corresponding authors.

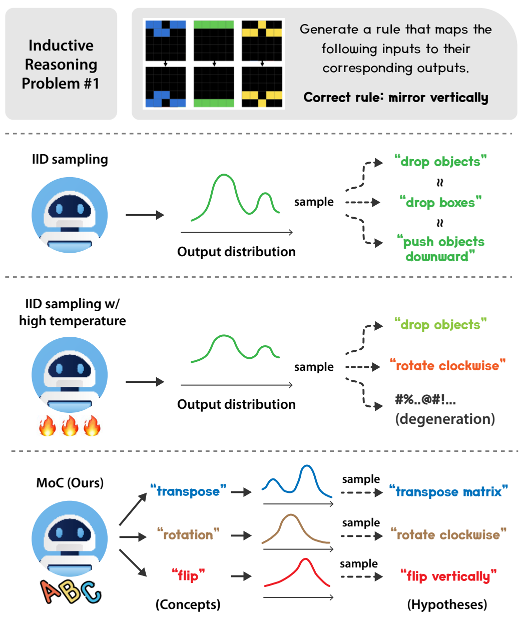


Figure 1: A motivation for MoC approach. IID sampling frequently generates redundant hypotheses (top). Increasing the temperature leads to frequent occurrences of text degeneration (middle). MoC allows for the generation of diverse hypotheses without a decline in hypothesis quality (bottom).

maintaining their quality. Compared to vanilla IID sampling with the same number of generated hypotheses, MoC boosts average accuracy by 4.5%p for GPT-4o-mini and 5.0%p for Llama-3.1-70B-Instruct as base LLM, respectively. Our analysis also shows that MoC enables the LLMs to crack challenging problems that are impossible to solve through IID sampling within a practical compute budget.

Our contributions are as follows:

- We analyze how the diversity and accuracy of IID hypotheses generated from LLMs vary with a change in temperature.
- We propose Mixture of Concepts (MoC), a methodology enabling more diverse and parallel hypothesis generation in LLMs.
- Our experimental results on four datasets and four models show that MoC significantly enhances both the efficiency and performance of inductive reasoning capability of LLMs.

2 Hypothesis Diversity and Temperature

In this section, we provide a description of the problem statement and the baseline method. Subsequently, we analyze the impact of temperature on the diversity and accuracy of hypotheses generated by an LLM.

2.1 Problem Statement and Baseline

Problem Statement We consider inductive reasoning problem with n train examples $D_{train} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ and m test examples $D_{test} = \{(x'_1, y'_1), (x'_2, y'_2), ..., (x'_m, y'_m)\}$. There is a function f that maps all the inputs to corresponding outputs: $f(x_i) = y_i$ and $f(x'_j) = y'_j$ for all $i \in [n]$ and $j \in [m]$, where $[n] = \{1, 2, ..., n\}$. The goal of this task is to predict $\{y'_1, y'_2, ..., y'_m\}$, given the D_{train} and $\{x'_1, x'_2, ..., x'_m\}$.

Simple Baseline Here, we describe a simple baseline method that we use in our experiments. We adopt the setup defined in recent inductive reasoning and Programming-by-Example (PBE) literature, where the function $\hat{f}$ is first inferred and the test outputs are obtained as $\{\hat{f}(x'_j)\}_{j\in[m]}$. The function $\hat{f}$ is implemented as a Python function, unlike traditional PBE works using domain-specific lanuguages (DSLs).

We prompt LLMs with instruction to generate a hypothesis about f in natural language form and then implement it in Python, following the recent LLM inductive reasoning works (Qiu et al., 2024; Wang et al., 2024b). We sample a fixed number of K responses and extract the Python function from these responses, forming a hypothesis pool $\{\hat{f}_1, \hat{f}_2, ..., \hat{f}_K\}$. Among these hypotheses, the one that perfectly explains the train examples, i.e. $y_i = \hat{f}_k(x_i)$ for all $i \in [n]$, is picked and submitted to be validated on the test examples. If there is no such hypothesis, we regard the problem as not being solved correctly. We consider that a problem is solved correctly if $\hat{f}_k$ passes every test cases: $y'_j = \hat{f}_k(x'_j)$ for all $j \in [m]$.

Duplicate Hypotheses The issue with the aforementioned simple baseline is that, since it samples K hypotheses from IID distributions, a significant portion of these hypotheses may be semantically redundant. If the hypothesis is correct, this redundancy does not pose a problem; however, if an incorrect hypothesis is sampled multiple times, it results in a serious waste of the generation budget.

To investigate whether this redundancy is problematic, we analyze how many semantically unique hypotheses are among the K hypotheses, for the instances that the baseline failed to solve correctly. We implement the baseline using GPT-4o-mini (`gpt-4o-mini-2024-07-18`) with temperature value of 1 and conduct the experiment in the List Functions dataset (Rule, 2020). In order to count the number of semantically unique hypotheses, a method to determine whether two hypotheses were semantically identical or different is needed. Since we represent the hypotheses as Python programs, we consider two hypotheses to be identical if both functions return the same outputs for the same inputs.

K	2	4	8	16	32
N	1.40	1.79	2.92	4.56	7.89

Table 1: Average number of unique programs (N) in K generated hypotheses for failed instances on List Functions dataset.

As shown in Table 1, we can observe that among the incorrect hypotheses, the majority of hypotheses are semantically redundant, especially when K is large. This indicates the need to generate a more diverse set of hypotheses that are semantically distinct for a more efficient hypothesis search.

2.2 Impact of Temperature on Diversity and Accuracy

Experiment on Temperature One of the most straightforward ways to increase diversity when sampling from a language model is by raising the temperature parameter. In this section, we analyze how temperature affects hypothesis diversity and accuracy in inductive reasoning tasks. We focus on the changes that occur when varying the temperature in the baseline method, within the datasets of List Functions and MiniARC (Kim et al., 2022).

We conduct experiments with a total of seven temperature settings, ranging from 0 to 2 in increments of 0.33. For temperatures of 1.67 and 2.0, there are substantial portion of responses that are degenerated (i.e. the Python function cannot be parsed from the model's response) (Figure 2).

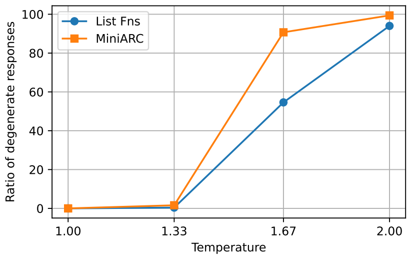


Figure 2: Ratio (%) of degenerate responses.

Therefore, we use top-p sampling (Holtzman et al., 2020) with $p = 0.95$ for these temperature settings, which eliminates degenerate responses entirely. Additionally, by varying the number of sampled responses K, we observe how both K and temperature affect the diversity of hypotheses.

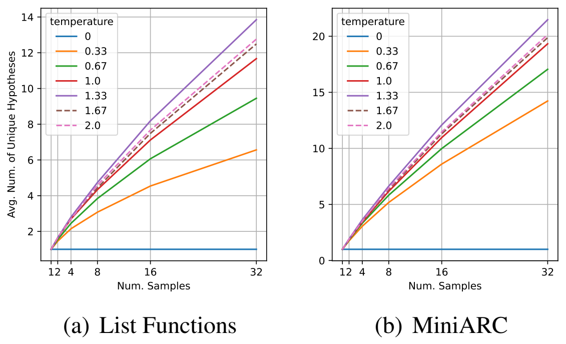


Figure 3: GPT-4o-mini hypothesis diversity on two domains. For the temperature 1.67 and 2.0, we used top-p sampling with $p = 0.95$. Results are averaged over 5 runs.

In Figure 3, it is observed that larger K and higher temperature correspond to larger number of semantically unique programs. However, beyond the temperature of 1.67, no significant difference compared to 1.33 is observed. This plateau may be attributed to the side effect of using top-p sampling. This dilemma, whether to use top-p sampling or not, is the primary reason why higher temperature values do not lead to additional diversity improvements.

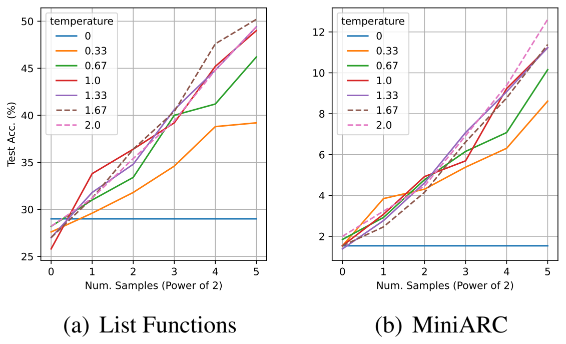


Figure 4: GPT-4o-mini performance on two domains. For the temperature 1.67 and 2.0, we used top-p sampling with $p = 0.95$. Results are averaged over 5 runs.

As shown in Figure 4, the accuracy shows a similar trend with that of diversity. Higher temperature values have advantage over lower values, but only up to temperature 1.0. It suggests that the quality of generated hypotheses starts to decline before text

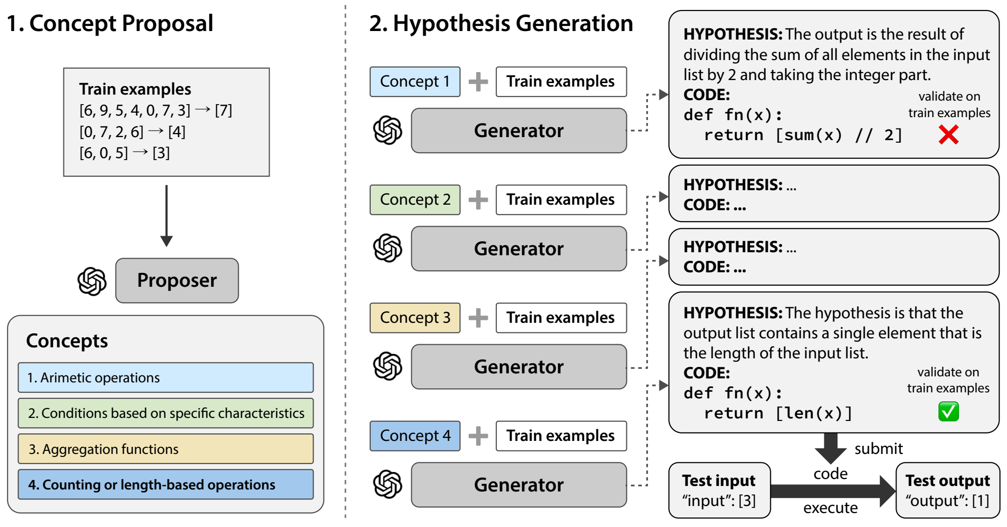


Figure 5: An overview of our Mixture of Concepts approach. We generate K distinct concepts (left) and feed them into the LLM separately for hypothesis generation (right).

degeneration occurs, making additional hypotheses no longer helpful to the accuracy (Peeperkorn et al., 2024). Therefore, in order to further improve performance, it is necessary to find a reliable method to increase semantic diversity while maintaining the quality of the hypotheses.

3 Mixture of Concepts

To achieve aforementioned goal, we propose automatic inductive reasoning framework called Mixture of Concepts (MoC).

When humans engage in inductive reasoning, they begin by carefully observing examples and considering which *concept* or *template* might be effective in identifying the underlying rule (Mitchell, 2021). For instance, if presented with integer lists as depicted in Figure 5, one might initially contemplate concepts such as "arithmetic operations." Using this concept, human would re-examine the given examples to discern what the hidden rule might be. However, in this instance, the correct rule is to output the length of the list, and the concept of "arithmetic operations" do not aid in discovering this rule. In such scenarios, the best strategy is to revisit the given examples from a different point of view and to contemplate novel concepts and rules. This process can be repeated until the correct rule is identified (Hofstadter, 1979).

Inspired by this cognitive process, we propose a framework composed of two stages: *concept proposal* and *hypothesis generation*.

Concept Proposal Through empirical investigation, we discovered that when instructing an LLM to generate a list of items, the generated items are rarely semantically redundant. This capability can be attributed to the autoregressive nature of the LLM. Each item generated by the LLM is used as context for generating the subsequent item, and the LLM tends to produce an item different from those it has already generated.

By leveraging this property, an LLM is instructed to list K elementary concepts that are likely to help the formulation of a hypothesis for given observations. For instance, given an inductive reasoning problem shown in Figure 5, the LLM generates a list of 4 distinct concepts (e.g. Arithmetic operations, Aggregation functions, etc.). Additionally, we instructed the LLM to generate the concepts in JSON format, ensuring that the concepts can be parsed more reliably in the subsequent stage.

Hypothesis Generation Next, we parse the concepts and feed each concept to the LLM. The LLM creates a natural language hypothesis and a Python code implementation based on the given concept. Here, a concept is provided in the instruction prompt as hint, so that the hypothesis can be conditioned on the given concept.

For instance, the 1st concept in Figure 5 (Arithmetic operations) conditions the generation to the wrong path and results in incorrect hypothesis. Meanwhile, the 4th concept (Counting or length-based operations) guides the generation to seman-

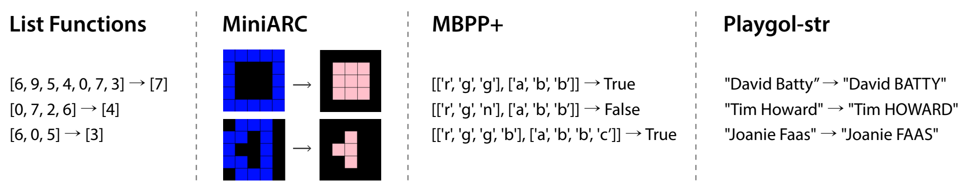

List Functions
[6, 9, 5, 4, 0, 7, 3] → [7]
[0, 7, 2, 6] → [4]
[6, 0, 5] → [3]
MiniARC
MBPP+
[['r', 'g', 'g'], ['a', 'b', 'b']] → True
[['r', 'g', 'n'], ['a', 'b', 'b']] → False
[['r', 'g', 'g', 'b'], ['a', 'b', 'b', 'c']] → True
Playgol-str
"David Batty" → "David BATTY"
"Tim Howard" → "Tim HOWARD"
"Joanie Faas" → "Joanie FAAS"

Figure 6: Example problems in each of four datasets we study. We graphically display the MiniARC examples to help the reader understand.

tically distinct direction, with correct output of returning the length of given input list (for full prompt details, see Appendix A).

Finally, one of the hypotheses (codes) that satisfies all the given train examples is submitted as the final hypothesis. The submitted hypothesis is considered correct if it satisfies all the test examples.

4 Experiments

4.1 Datasets

We validate our method on four datasets from distinct domains as shown in Figure 6.

List Functions The List Functions dataset (Rule, 2020) evaluates inductive reasoning ability in inferring list transform functions. The inputs and outputs are integer list, possibly an empty list. The transformation function covers broad range of list and arithmetic operators such as indexing, slicing, counting, sorting, etc. We randomly sample a subset of 100 instances from the 250 instances of the dataset for evaluations in our work.

MiniARC MiniARC (Kim et al., 2022) is a simplified version of Abstraction and Reasoning Corpus (ARC) dataset (Chollet, 2019). In ARC, the inputs and outputs are 2D grids following specific transformation pattern. The transformation functions are explicitly grounded on the cognitive priors of humans, such as objectness, goal-directedness, arithmetic and basic geometry (Spelke and Kinzler, 2007). Compared to original ARC, MiniARC's inputs and outputs are always 5x5 grids, greatly reducing the complexity of the problems. Nonetheless, MiniARC is still extremely challenging for state-of-the-art LLMs (Qiu et al., 2024). Following the previous works, we input the MiniARC grid into the LLMs in the form of a nested list.

MBPP+ MBPP+ (Liu et al., 2023) expands a program synthesis dataset MBPP (Austin et al., 2021) with 35x more test cases. It covers basic Python programming tasks written by humans. The abundance of test cases avails utilizing MBPP+ as an inductive reasoning dataset. We arrange each instance such that it has 8 train examples and 6 test examples, and randomly sample a subset of 100 instances for evaluation.

Playgol-str The string transformation domain is a practical application of automatic inductive reasoning that many users actively utilize (Gulwani, 2011; Cambronero et al., 2023; Chen et al., 2021). We evaluate our method on the real-world string transformations dataset introduced by Cropper (2019). In our work, we refer it using the name of their proposed method, *Playgol-str*. We randomly sample a subset of 150 instances for evaluation.

4.2 Main Results

We compare Mixture of Concepts (MoC) with simple baseline described in section 2.1. Each method is implemented with two proprietary and two open-source LLMs: GPT-4o-mini (`gpt-4o-mini-2024-07-18`), GPT-4o (`gpt-4o-2024-08-06`), Llama-3.1-70B-Instruct, and Qwen2.5-72B-Instruct. If not otherwise specified, the temperature is set to 1.0 for all experiments, because temperature higher than that does not bring any performance gain as observed in section 2.2. Also, we set the number of samples $K = 8$ here. Results with $K = 4$ and $K = 16$ are in Appendix C.

In Table 2, our MoC approach shows significant improvement in overall test accuracy, demonstrating the effectiveness of our approach. In the MiniARC dataset, all LLMs generally exhibit low performance, which is largely due to the fact that ARC-variants often require visual priors that LLMs are not expected to possess.

Comparison with Hypothesis Refinement Iterative Hypothesis Refinement (IHR) (Qiu et al., 2024) involves validating the generated hypotheses against the training examples, and if none of the

Model		List Fns	MiniARC	MBPP+	Playgol-str	Average
GPT-4o-mini	Baseline	43	5.4	33	75.3	39.2
	IHR (T=2, N=4)	39	5.4	-	-	-
	MoC	**49**	**8.5**	**40**	**77.3**	**43.7 (+4.5)**
GPT-4o	Baseline	47	9.2	53	81.3	47.6
	IHR (T=2, N=4)	52	11.5	-	-	-
	MoC	**53**	**12.3**	**55**	**83.3**	**50.9 (+3.3)**
Llama 3.1 70B	Baseline	40	**9.2**	45	64	39.6
	MoC	**45**	7.7	**55**	**70.7**	**44.6 (+5.0)**
Qwen2.5 72B	Baseline	46	6.9	45	76.7	43.7
	MoC	**53**	**8.5**	**51**	**78.7**	**47.8 (+4.1)**

Table 2: Test accuracy (%) of IID baseline, Iterative Hypothesis Refinement (IHR) (Qiu et al., 2024), and our MoC, with 8 hypotheses generated. The numbers in parentheses indicate the improvement compared to the baseline.

Model		List Fns	MiniARC	MBPP+	Playgol-str	Average
GPT-4o-mini	Baseline	4.26	6.31	3.76	2.51	4.21
	MoC	**4.75**	**6.52**	**4.40**	**2.63**	**4.58 (+0.37)**
GPT-4o	Baseline	4.19	6.19	3.99	2.10	4.12
	MoC	**5.08**	**6.80**	**4.64**	**2.31**	**4.71 (+0.59)**
Llama 3.1 70B	Baseline	4.81	6.35	4.10	3.07	4.58
	MoC	**6.07**	**7.23**	**5.35**	**3.66**	**5.58 (+1.00)**
Qwen2.5 72B	Baseline	4.22	5.72	3.05	2.24	3.81
	MoC	**5.26**	**6.28**	**3.82**	**2.75**	**4.53 (+0.72)**

Table 3: Average number of unique hypotheses in 8 hypotheses. The numbers in parentheses indicate the improvement compared to the baseline.

hypotheses pass, the hypothesis with highest train accuracy is refined using execution feedback. We conduct experiments of IHR in a setting of $T = 2$ and $N = 4$, similar to the case where $K = 8$. This means there are 2 iterations, with 4 hypotheses generated per iteration. Additionally, when translating the generated hypotheses into Python code, we insert the training examples in the input prompt, resulting in a significant performance improvement.

As shown Table 2, iterative hypothesis refinement performs well with GPT-4o, but with weaker GPT-4o-mini, it does not outperform the baseline. However, our MoC methodology consistently shows performance improvements regardless of the model's base performance.

Hypothesis Diversity To analyze the direct impact of MoC on hypothesis diversity, we observe the number of unique hypotheses under the same conditions as the previous accuracy experiment ($K = 8$). The number of unique hypotheses is measured using the method described in section 2.2. As shown in Table 3, MoC significantly increases the diversity of the hypotheses. This indicates that our methodology effectively minimizes semantically redundant hypotheses, which leads to improvements in inductive reasoning performance.

4.3 Analysis and Discussion

In this section, we raise several research questions related to MoC and address them.

Does the benefit of MoC arise from hypothesis diversity or from an improvement in reasoning? It has been observed that LLMs can improve their reasoning ability by generating intermediate steps before the final answer, a method also known as Chain-of-Thought (CoT) prompting (Wei et al., 2022). From this perspective, MoC, which first generates elementary concepts and then uses them to create hypotheses, shares similarities with such CoT prompting approaches.

To isolate the benefits of the reasoning ability from that on the hypothesis diversity, we set the temperature to 0 and generate only one hypothesis to observe if there is an improvement in accuracy. As shown in Table 4, the MoC method does not show significant gains compared to the greedy baseline. Although there are certain model-dataset pairs where performance significantly improved (e.g., GPT-4o-mini on Playgol-str), no consistent trend was observed. Therefore, we conclude that

Model		List Fns	MiniARC	MBPP+	Playgol-str	Average
GPT-4o-mini	Greedy	29	**1.5**	24	52.7	26.8
	MoC (K=1)	**34**	0.8	**25**	**63.3**	**30.8**
GPT-4o	Greedy	39	**4.6**	**31**	**69.3**	**36.0**
	MoC (K=1)	**42**	3.8	24	**69.3**	34.8
Llama 3.1 70B	Greedy	**32**	2.3	31	**44.7**	27.5
	MoC (K=1)	30	**3.1**	**34**	**44.7**	**28.0**
Qwen2.5 72B	Greedy	30	0.8	29	**58**	29.5
	MoC (K=1)	**31**	**3.8**	**32**	54.7	**30.4**

Table 4: Test accuracy (%) of baseline and MoC using greedy decoding. MoC increases the diversity of hypotheses when generating multiple hypotheses, so it offers little benefit in cases where only a single hypothesis is generated using greedy decoding.

Approach	List Fns	MiniARC	MBPP+	Playgol-str	Average
Baseline (K=16)	46	6.9	40	78	42.7
MoC (C=16, S=1)	51	10.8	**48**	78.7	47.1 (+4.4)
MoC (C=8, S=2)	**52**	**12.3**	46	80	**47.6 (+4.9)**
MoC (C=4, S=4)	48	9.2	43	**80.7**	45.2 (+2.5)
Baseline (K=32)	46	9.2	45	81.3	45.4
MoC (C=32, S=1)	51	13.1	47	80	47.8 (+2.4)
MoC (C=16, S=2)	54	**13.8**	**55**	**85.3**	**52.0 (+6.6)**
MoC (C=8, S=4)	**55**	13.1	46	81.3	48.9 (+3.5)

Table 5: Test accuracy (%) of baseline and MoC with varying C (number of concepts) and S (number of hypotheses per concept). Base LLM is GPT-4o-mini. The numbers in parentheses indicate the improvement compared to the baseline.

the performance improvements observed with MoC in Table 2 and 8 are mostly due to the increase in hypothesis diversity rather than improvements in reasoning ability.

What impact does scaling compute have on the MoC? In the sampling-based inductive reasoning discussed in this work, if a larger compute budget is provided, performance improvement is almost guaranteed by sampling additional hypotheses. We investigate whether MoC shows a similar scaling effect, as observed in Figure 4, when the number of generated hypotheses K is set higher. We use GPT-4o-mini as base LLM for the scaling experiment.

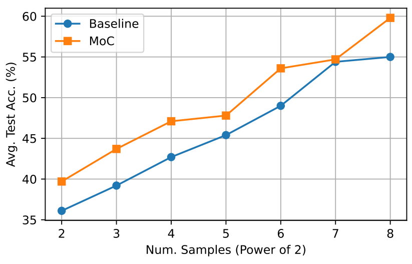


Figure 7: Test accuracy (%) averaged over 4 datasets.

As shown in Figure 7, both the IID baseline and MoC demonstrate higher average accuracy with larger number of samples K. Additionally, MoC achieves similar performance while generating only half the number of hypotheses relative to the baseline, demonstrating a significant improvement in the efficacy of inductive reasoning. Full results of scaling experiment are in Appendix C.

What impact does generating multiple hypotheses per concept have on performance? In previous experiments, we generate only one hypothesis using a single concept as a hint. However, some concepts carry richer meanings than others, which may lead to the generation of multiple hypotheses. Additionally, if there is an error in the process of implementing the Python program, resampling may generate an error-free implementation.

In Table 5, we examine how the number of concepts (C) and the number of hypotheses sampled per concept (S) affect performance. When we compare results while keeping the total number of hypotheses the same, we find that sampling about two hypotheses per concept performs better than other settings. In conclusion, balancing C and S is critical for achieving optimal performance.

Does MoC assist in solving highly challenging problems beyond improving efficiency? The

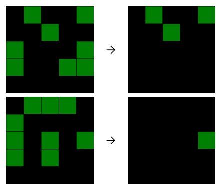
Train Examples	Concepts	Natural Language Hypothesis
 ...	border, corner, **cell adjacency,** identity transformation, scan, ...	The transformation involves detecting continuous segments of **adjacent cells** containing the value '3' within the input grid. If any cell containing '3' has neighboring cells (up, down, left, or right) that also contain '3', the entire connected segment is cleared, meaning all cells in that segment are set to '0'. If a cell is isolated (has no adjacent '3's), it remains '3'. The process is repeated for the entire grid, adjusting the output based on this adjacency rule.
[2] $\rightarrow$ False [10] $\rightarrow$ True [61] $\rightarrow$ False [55] $\rightarrow$ True ...	even numbers, odd numbers, **prime numbers,** composite numbers, divisibility, ...	The transformation checks if the input number is either a composite number or the number 1. If the number is **prime** (greater than 1 and only divisible by 1 and itself), the output is False. For all other inputs, including the number 1, the output is True.

Table 6: Two challenging examples from MiniARC and MBPP+, where the IID baseline fails to generate a valid hypothesis from over 500 hypothesis samples. MoC solves these problems correctly with only 64 hypotheses. The concepts in **boldface** formulate the correct hypothesis.

two examples in Table 6 represent highly challenging problems in which more than 500 IID samples from GPT-4o-mini fail to identify the correct rule. In these problems, MoC effectively discovers the rule using far less compute ($K = 64$). This indicates that challenging problems with a low probability of being solved through IID sampling can be effectively resolved using the MoC approach. In Appendix D, we provide a full list of generated concepts for these problems.

5 Related Work

5.1 Automatic Inductive Reasoning

Automating the process of inductive reasoning has long been a significant challenge in AI, due to its central role in achieving human-level intelligence (Lake et al., 2015; Ellis, 2023; Ellis et al., 2021) and its practical applications, such as Programming-by-Example (PBE) (Menon et al., 2013; Cambronero et al., 2023; Gulwani, 2011). Before the advent of LLMs, neural networks were widely used to guide program search but their search spaces are often limited by domain-specific languages (DSLs) (Chen et al., 2019; Shi et al., 2022; Odena and Sutton, 2020; Odena et al., 2021; Clarke et al., 2010; Lee et al., 2023).

Recently, LLMs have significantly transformed the paradigm of automatic inductive reasoning. They are employed mainly in two ways: directly prompting with training input-output pairs to predict outputs for unseen inputs (Moskvichev et al., 2023; Mirchandani et al., 2023; Webb et al., 2023; Sun et al., 2024), or formulating hypotheses in natural language or executable programs and applying them to unseen test inputs (Li and Ellis, 2024; Shao et al., 2025; Yang et al., 2024).

Iterative refinement is a popular methodology in the latter domain (Wang et al., 2024b; Qiu et al., 2024; Greenblatt, 2024; Tang et al., 2024); however, they are sequential in nature and can slow down inference. Furthermore, it has been suggested that LLMs lack the ability to self-correct (Huang et al., 2024), and that such refinement offers no substantial advantages compared to drawing more IID samples (Acquaviva, 2024; Olausson et al., 2024).

5.2 Reasoning in Inference Time of LLMs

Recently, various methodologies have been explored to enhance the reasoning abilities of LLMs in inference time (Wei et al., 2022; Zheng et al., 2024). When allocated an increased inference budget, they are capable of solving complex reasoning tasks (Brown et al., 2024; Snell et al., 2024). By leveraging these properties, LLMs have shown promising reasoning capabilities to solving diverse reasoning tasks within a few iterative processes. For example, Yao et al. (2023); Lanchantin et al. (2023) suggest an iterative generate-and-self-reflect approach to address complex reasoning tasks. Kumar et al. (2024) show the self-correction capabilities of LLMs, and Zhou et al. (2024) illustrate how paraphrasing input prompts can enhance reasoning capabilities. In addition, many concurrent works have proposed methodologies to increase the diversity of responses to perform more efficient search (Wen and Chaudhuri, 2024; Wang et al., 2024a; Light et al., 2024; Chang et al., 2024).

6 Conclusion

In this paper, we study the diversity of hypotheses generated by an LLM during inductive reasoning.

We find that, even if incorrect, these hypotheses are predominantly redundant, leading to wasted compute. Increasing the LLM's temperature initially helps, but benefits saturate at certain point and excessively high temperatures cause text degeneration. Therefore, we propose a method called Mixture of Concepts (MoC); the core idea is to generate non-redundant concepts and using these to formulate hypotheses. Our approach significantly improves performance across various datasets and models.

Limitations

Our MoC methodology requires additional computation during the concept proposal process prior to hypothesis generation. Additionally, during the hypothesis generation phase, each concept is used as a hint to construct prompts, so the model should encode more tokens compared to IID sampling. However, this incurs relatively small overhead compared to refinement or summarization-based methods found in existing works.

Furthermore, the actual process of inductive reasoning in humans is far more complex than that of MoC. Humans are capable of solving difficult problems by transforming concepts and composing multiple concepts in various ways (Lake et al., 2015). These aspects have not been addressed in this study.

Lastly, it is not yet known how our approach affects the safety of LLMs as it may enhance the diversity of LLM responses in unexpected way. While such risks are minimal in the programming tasks we primarily dealt with, in the context of reasoning in natural language, there is a risk that increasing response diversity could amplify issues such as social bias and toxicity (Koh et al., 2024).

Acknowledgments

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP)-ITRC(Information Technology Research Center) grant funded by the Korea government(MSIT)(IITP-2025-RS-2024-00437633, 50%), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) [RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University) & RS-2021-II212068, Artificial Intelligence Innovation Hub (Artificial Intelligence Institute, Seoul National University)], and the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2024. K. Jung is with ASRI, Seoul National University, Korea.

References

Samuel Acquaviva. 2024. Overcoming the expressivity-efficiency tradeoff in program induction. Master's thesis, Massachusetts Institute of Technology.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *Preprint*, arXiv:2407.21787.

José Cambronero, Sumit Gulwani, Vu Le, Daniel Perelman, Arjun Radhakrishna, Clint Simon, and Ashish Tiwari. 2023. Flashfill++: Scaling programming by example by cutting to the chase. *Proc. ACM Program. Lang.*, 7(POPL).

Haw-Shiuan Chang, Nanyun Peng, Mohit Bansal, Anil Ramakrishna, and Tagyoung Chung. 2024. Real sampling: Boosting factuality and diversity of open-ended generation via asymptotic entropy. *arXiv preprint arXiv:2406.07735*.

Xinyun Chen, Chang Liu, and Dawn Song. 2019. Execution-guided neural program synthesis. In *International Conference on Learning Representations*.

Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021. Spreadsheetcoder: Formula prediction from semi-structured context. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1661–1672. PMLR.

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

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 18–27, Uppsala, Sweden. Association for Computational Linguistics.

Andrew Cropper. 2019. Playgol: Learning programs through play. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6074–6080. International Joint Conferences on Artificial Intelligence Organization.

Kevin Ellis. 2023. Human-like few-shot learning via bayesian reasoning over natural language. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. 2021. Dreamcoder: bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 835–850, New York, NY, USA. Association for Computing Machinery.

Ryan Greenblatt. 2024. Getting 50% sota on arc-agi with gpt-4. *Technical Report*.

Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *SIGPLAN Not.*, 46(1):317–330.

Douglas Hofstadter. 1979. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, New York.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

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

Hyukhun Koh, Dohyung Kim, Minwoo Lee, and Kyomin Jung. 2024. Can LLMs recognize toxicity? a structured investigation framework and toxicity metric. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6092–6114, Miami, Florida, USA. Association for Computational Linguistics.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2024. Training language models to self-correct via reinforcement learning. *Preprint*, arXiv:2409.12917.

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

Jack Lanchantin, Shubham Toshniwal, Jason Weston, Arthur Szlam, and Sainbayar Sukhbaatar. 2023. Learning to reason and memorize with self-notes. *Preprint*, arXiv:2305.00833.

Kang-il Lee, Segwang Kim, and Kyomin Jung. 2023. Weakly supervised semantic parsing with execution-based spurious program filtering. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6884–6894, Singapore. Association for Computational Linguistics.

Wen-Ding Li and Kevin Ellis. 2024. Is programming by example solved by llms? *arXiv preprint arXiv:2406.08316*.

Jonathan Light, Yue Wu, Yiyou Sun, Wenchao Yu, Yanchi liu, Xujiang Zhao, Ziniu Hu, Haifeng Chen, and Wei Cheng. 2024. Scattered forest search: Smarter code space exploration with llms. *arXiv preprint arXiv:2411.05010*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Aditya Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, and Adam Kalai. 2013. A machine learning framework for programming by example. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 187–195, Atlanta, Georgia, USA. PMLR.

Suvir Mirchandani, Fei Xia, Pete Florence, brian ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. 2023. Large language models as general pattern machines. In *7th Annual Conference on Robot Learning*.

Melanie Mitchell. 2021. Abstraction and analogy-making in artificial intelligence. *Annals of the New York Academy of Sciences*, 1505(1):79–101.

Arseny Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. 2023. The conceptarc benchmark: Evaluating understanding and generalization in the arc domain. *arXiv preprint arXiv:2305.07141*.

Augustus Odena, Kensen Shi, David Bieber, Rishabh Singh, Charles Sutton, and Hanjun Dai. 2021. {BUSTLE}: Bottom-up program synthesis through learning-guided exploration. In *International Conference on Learning Representations*.

Augustus Odena and Charles Sutton. 2020. Learning to represent programs with property signatures. In *International Conference on Learning Representations*.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*.

Max Peeperkorn, Tom Kouwenhoven, Dan Brown, and Anna Jordanous. 2024. Is temperature the creativity parameter of large language models? *arXiv preprint arXiv:2405.00492*.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. 2024. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. In *The Twelfth International Conference on Learning Representations*.

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

Yunfan Shao, Linyang Li, Yichuan Ma, Peiji Li, Demin Song, Qinyuan Cheng, Shimin Li, Xiaonan Li, Pengyu Wang, Qipeng Guo, Hang Yan, Xipeng Qiu, Xuanjing Huang, and Dahua Lin. 2025. Case2Code: Scalable synthetic data for code generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 11056–11069, Abu Dhabi, UAE. Association for Computational Linguistics.

Kensen Shi, Hanjun Dai, Kevin Ellis, and Charles Sutton. 2022. Crossbeam: Learning to search in bottom-up program synthesis. In *International Conference on Learning Representations*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Elizabeth S Spelke and Katherine D Kinzler. 2007. Core knowledge. *Developmental science*, 10(1):89–96.

Wangtao Sun, Haotian Xu, Xuanqing Yu, Pei Chen, Shizhu He, Jun Zhao, and Kang Liu. 2024. ItD: Large language models can teach themselves induction through deduction. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2719–2731, Bangkok, Thailand. Association for Computational Linguistics.

Hao Tang, Keya Hu, Jin Peng Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. 2024. Code repair with llms gives an exploration-exploitation tradeoff. *arXiv preprint arXiv:2405.17503*.

Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024a. Planning in natural language improves llm search for code generation. *arXiv preprint arXiv:2409.03733*.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah Goodman. 2024b. Hypothesis search: Inductive reasoning with language models. In *The Twelfth International Conference on Learning Representations*.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yeming Wen and Swarat Chaudhuri. 2024. Synthesize, partition, then adapt: Eliciting diverse samples from foundation models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zonglin Yang, Li Dong, Xinya Du, Hao Cheng, Erik Cambria, Xiaodong Liu, Jianfeng Gao, and Furu Wei. 2024. Language models as inductive reasoners. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 209–225, St. Julian's, Malta. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. 2024. Take a step back: Evoking reasoning via abstraction in large language models. In *The Twelfth International Conference on Learning Representations*.

Yue Zhou, Yada Zhu, Diego Antognini, Yoon Kim, and Yang Zhang. 2024. Paraphrase and solve: Exploring and exploiting the impact of surface form on mathematical reasoning in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2793–2804, Mexico City, Mexico. Association for Computational Linguistics.

A Prompts

Here, we provide prompts for concept proposal (MoC) and hypothesis generation (baseline and MoC). These prompts were constructed based on those from Hypothesis Search (Wang et al., 2024b).

Prompt for Hypothesis Generation: Baseline

~~~
You will be given a list of input-
    output pairs. There is a SINGLE
     pattern that transforms each
    input to the corresponding
    output.
First, output a hypothesis for the
     transformation in natural
    language form.
Then, generate a Python function `
    fn` that maps the following
    inputs to their corresponding
    outputs.

Please format your hypothesis and
    Python function as follows:

```hypothesis
HYPOTHESIS
~~~

```python
def fn(x):
    # x is {INPUT_FORMAT}
    # Your code here
    return y # y is {OUTPUT_FORMAT
        }
```

Input-output pairs:
{TRAIN_EXAMPLES}
Hypothesis and Python function:
```

Prompt for Concept Proposal: MoC

```
You will be given a list of input-
    output pairs. There is a SINGLE
     pattern that transforms each
    input to the corresponding
    output.
List {K} elementary concepts that
    may be useful to induce the
    transformation pattern.
Format your response in json
    format (dictionary whose keys
    are indices and values are
    elementary concepts).

Input-output pairs:
{TRAIN_EXAMPLES}

Elementary concepts:
```

Prompt for Concept Proposal: MoC (GPT-4o)

```
You will be given a list of input-
    output pairs. There is a SINGLE
     pattern that transforms each
    input to the corresponding
    output.
List {K} elementary concepts that
    may be useful to induce the
    transformation pattern.
The concepts should be diverse,
    simple and concise.
Format your response in json
    format (dictionary whose keys
    are indices and values are
    elementary concepts).

Input-output pairs:
{TRAIN_EXAMPLES}

Elementary concepts:
```

Prompt for Hypothesis Generation: MoC

```
You will be given a list of input-
    output pairs. There is a SINGLE
     pattern that transforms each
    input to the corresponding
    output.
First, output a hypothesis for the
     transformation in natural
    language form. Use hint: {HINT
    }.
Then, generate a Python function `
    fn` that maps the following
    inputs to their corresponding
    outputs.

Please format your hypothesis and
    Python function as follows:

```hypothesis
HYPOTHESIS
```

```python
def fn(x):
    # x is {INPUT_FORMAT}
    # Your code here
    return y # y is {OUTPUT_FORMAT
        }
```

Input-output pairs:
{TRAIN_EXAMPLES}
Hypothesis and Python function:
```

In above prompts, TRAIN_EXAMPLES represents all the train inputs and outputs of given inductive reasoning problem. INPUT_FORMAT and OUTPUT_FORMAT specify input and output formats. For instance, in List Functions dataset,
```

| Model | $K$ | | List Fns | MiniARC | MBPP+ | Playgol-str | Average |
|---|---|---|---|---|---|---|---|
| GPT-4o | 4 | Baseline | 46 | 8.5 | 50 | 78 | 45.6 |
| | | MoC | 52 | 10.8 | 48 | 80 | 47.7 (+2.1) |
| | 16 | Baseline | 53 | 14.6 | 54 | 83.3 | 51.2 |
| | | MoC | 62 | 16.9 | 59 | 83.3 | 55.3 (+4.1) |
| Llama 3.1 70B | 4 | Baseline | 36 | 7.7 | 40 | 60.7 | 36.1 |
| | | MoC | 37 | 5.4 | 42 | 64 | 37.1 (+1.0) |
| | 16 | Baseline | 44 | 10.8 | 54 | 65.3 | 43.5 |
| | | MoC | 52 | 8.5 | 52 | 74 | 46.6 (+3.1) |
| Qwen2.5 72B | 4 | Baseline | 39 | 5.4 | 37 | 70.7 | 38.0 |
| | | MoC | 49 | 7.7 | 44 | 70 | 42.7 (+4.7) |
| | 16 | Baseline | 48 | 11.5 | 46 | 80.7 | 46.6 |
| | | MoC | 51 | 14.6 | 51 | 79.3 | 49.0 (+2.4) |

Table 7: Test accuracy (%) with varying $K$. The numbers in parentheses indicate the improvement compared to the baseline.

| $K$ | | List Fns | MiniARC | MBPP+ | Playgol-str | Average |
|---|---|---|---|---|---|---|
| 4 | Baseline | 34 | 4.6 | 33 | 72.7 | 36.1 |
| | MoC | 43 | 7.7 | 36 | 72 | 39.7 |
| 8 | Baseline | 43 | 5.4 | 33 | 75.3 | 39.2 |
| | MoC | 49 | 8.5 | 40 | 77.3 | 43.7 |
| 16 | Baseline | 46 | 6.9 | 40 | 78 | 42.7 |
| | MoC | 51 | 10.8 | 48 | 78.7 | 47.1 |
| 32 | Baseline | 46 | 9.2 | 45 | 81.3 | 45.4 |
| | MoC | 51 | 13.1 | 47 | 80 | 47.8 |
| 64 | Baseline | 53 | 13.1 | 46 | 84 | 49.0 |
| | MoC | 57 | 15.4 | 58 | 84 | 53.6 |
| 128 | Baseline | 59 | 16.9 | 55 | 86.7 | 54.4 |
| | MoC | 58 | 17.7 | 59 | 84 | 54.7 |
| | MoC ($C$=64, $S$=2) | 63 | 18.5 | 60 | 86.7 | 57.1 |
| 256 | Baseline | 62 | 16.2 | 55 | 86.7 | 55.0 |
| | MoC | 64 | 20.8 | 69 | 85.3 | 59.8 |
| | MoC ($C$=64, $S$=4) | 64 | 20.8 | 66 | 84.7 | 58.9 |

Table 8: Test accuracy (%) with varying $K$ with GPT-4o-mini as a base LLM.

both `INPUT_FORMAT` and `OUTPUT_FORMAT` are `List[int]`. `K` represents the number of concepts $K$ in the main text. `HINT` represents one of the concepts generated in the concept proposal stage.

We added the instruction "The concepts should be diverse, simple and concise." in the concept proposal prompt for GPT-4o, because of excessively long and specific concepts.

## B Experimental Details

In this paper, we use 2 Nvidia A100 GPUs for running experiments with open-source LLMs. The evaluation on four datasets takes approximately 1 to 2 days. We conduct experiments using Llama 3.1[1] and Qwen2.5[2] via Huggingface.

[1]https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct

[2]https://huggingface.co/Qwen/Qwen2.5-72B-Instruct

## C Additional Results

Here, we provide additional experimental results. Table 7 shows test accuracy of various models in various $K$. In Table 8, we report full results of scaling experiment in section 4.3.

Also, we evaluate the performance of MoC when applied to GPT-4-0613 and compare it with the performance of Hypothesis Search on List Functions domain reported in their paper (Wang et al., 2024b). According to Figure A.5 of the Hypothesis Search paper, the performance when generating 8 hypotheses and programs per problem appears to be 59 (although this is based on reading the graph). In comparison, the performance of MoC with $K = 8$ using GPT-4-0613 was 62, indicating that MoC outperforms under similar conditions.

## D Generated Concepts

We provide a complete list of 64 concepts for the MiniARC and MBPP+ problems shown in Table 6.

**64 Concepts Proposed by MoC: MiniARC**

```
"matrix",
"grid",
"element",
"zero",
"non-zero",
"transpose",
"row",
"column",
"diagonal",
"sum",
"element removal",
"max",
"min",
"count",
"identity matrix",
"row operation",
"column operation",
"boolean logic",
"pattern recognition",
"transformation",
"filter",
"morhological operations",
"average",
"median",
"mode",
"conditions",
"local neighborhood",
"border",
"corner",
"cell adjacency",
"identity transformation",
"scan",
"order of operations",
"data structure",
"selectivity",
"placement",
"swap",
"binary logic",
"iteration",
"mappings",
"shift",
"constant pattern",
"symmetry",
"rotation",
"reflection",
"submatrix",
"duplicates",
"translational symmetry",
"identification",
"data comparison",
"sequence",
"position",
"inversion",
"overlay",
"region selection",
"objective spacing",
"dynamic structure",
"permutation",
"normalization",
"clustering",
"threshold",
"classification",
"consolidation",
"extraction"
```

64 Concepts Proposed by MoC: MBPP+

```
"even numbers",
"odd numbers",
"prime numbers",
"composite numbers",
"divisibility",
"factors",
"square numbers",
"cube numbers",
"binary representation",
"decimal values",
"number properties",
"natural numbers",
"integer numbers",
"numerical sequences",
"mathematical reasoning",
"truth values",
"falsehood",
"high values",
"low values",
"parity",
"logical operators",
"mathematical operations",
"set theory",
"constructive proof",
"algorithmic thinking",
"binary classification",
"discrete mathematics",
"logical expressions",
"conditions",
"case analysis",
"function evaluation",
"numerical properties",
"mathematical inductions",
"problem-solving techniques",
"classification systems",
"set membership",
"theoretical foundations",
"logic gates",
"boolean values",
"truth tables",
"coordinate systems",
"rounding numbers",
"fibonacci sequence",
"modular arithmetic",
"graph theory",
"probability theory",
"prime factorization",
"algebra",
"geometry",
"calculus",
"number theory",
"functional mathematics",
"mathematical logic",
"distributions",
"linear equations",
"asymptotic analysis",
"complex numbers",
"real numbers",
"irrational numbers",
"even-odd tests",
"theoretical number systems",
"order of magnitude",
"negation",
"quantitative reasoning"
```

E Information about Use of AI Assistants

In writing this paper, we utilized ChatGPT[3] for paraphrasing.

[3] https://chatgpt.com/