Reprompting: Automated Chain-of-Thought Prompt Inference Through Gibbs Sampling

Weijia Xu [1] **Andrzej Banburski-Fahey** [1] **Nebojsa Jojic** [1]

Abstract

We introduce Reprompting, an iterative sampling algorithm that automatically learns the Chain-of-Thought (CoT) recipes for a given task without human intervention. Through Gibbs sampling, Reprompting infers the CoT recipes that work consistently well for a set of training samples by iteratively sampling new recipes using previously sampled recipes as parent prompts to solve other training problems. We conduct extensive experiments on 20 challenging reasoning tasks. Results show that Reprompting outperforms human-written CoT prompts substantially by +9.4 points on average. It also achieves consistently better performance than the state-of-the-art prompt optimization and decoding algorithms.

1. Introduction

Few-shot prompting with large language models (LLMs) has revolutionized the landscape of natural language processing. Given natural language instructions and a few demonstrations as in-context examples, LLMs can quickly adapt to new tasks, approaching or even surpassing the performance of models fine-tuned on larger datasets on a wide range of tasks (Brown et al., 2020). However, such prompting techniques fall short on tasks that require multi-step reasoning and constraint propagation (Wei et al., 2022), such as *logical deduction* in the Big-Bench Hard benchmark (Suzgun et al., 2022). To address these limitations, prior works proposed to teach LLMs to reason step by step like humans by prompting them with chain-of-thought (CoT) reasoning steps for a few example problems (Wei et al., 2022). Despite the improved performance, such a method requires human experts with not only the task knowledge but also an understanding of how prompting works to craft the CoT prompt for each task (Zamfirescu-Pereira et al., 2023), which limits the scalability and generalizability of the method. Furthermore, a problem can be reasoned in many different ways, and some of them may work well on some LLMs but not on others. To fairly compare the performance of various LLMs on each task, we need to find the CoT prompt that works best for each model in a feasible way, which remains a challenge.

In this paper, we propose *Reprompting*, an iterative sampling algorithm that **automatically** finds effective CoT prompt for each model given a few question-answer pairs without human intervention. Specifically, the algorithm aims to infer a set of CoT recipes that perform consistently well as in-context examples for a set of training problems. We frame it as a problem of sampling from a joint distribution of CoT recipes given the training question-answer pairs, which is infeasible to characterize directly but can be approached using Gibbs sampling – we initially sample a set of recipes through zero-shot prompting, expand the set with new recipes sampled iteratively by using previously sampled recipes as parent prompts to solve a different training problem, and weed out the least-fit recipes that lead to wrong answers. Thus, the algorithm will eventually converge to a set of recipes that share similar chains of thought for effectively solving the training problems. These CoT recipes optimized on the training set then serve as effective CoT prompts for solving unseen test problems.

We evaluate *Reprompting* on 20 tasks from three reasoning benchmarks including Big-Bench Hard (BBH) (Suzgun et al., 2022), GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) using ChatGPT (OpenAI, 2023) and InstructGPT (Ouyang et al., 2022) as LLMs. Compared with human-written CoT prompts, *Reprompting* achieves +9.4 higher accuracy on average. It also consistently outperforms self-consistency decoding (Wang et al., 2022b), Auto-CoT (Zhang et al., 2022) and Automatic Prompt Optimization (Pryzant et al., 2023) by 11–33 points on average. Furthermore, *Reprompting* facilitates model combination by using different LLMs for initializing and sampling new recipes. Empirically, leveraging ChatGPT to sample initial recipes for InstructGPT brings up to +71 point improvements over using InstructGPT alone and even outperforms ChatGPT alone on certain tasks. Lastly, our results confirm that the CoT recipes that work well on one model

[1]Microsoft Research, Redmond, USA. Correspondence to: Weijia Xu <weijiaxu@microsoft.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

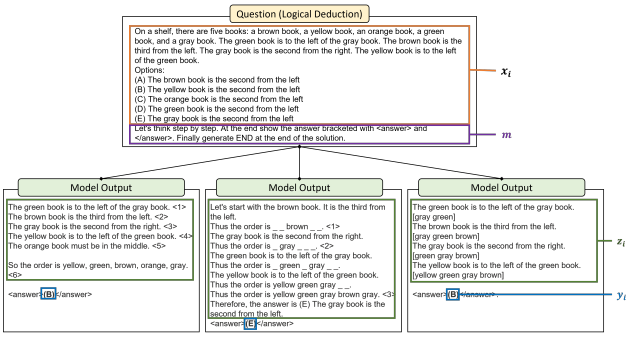


Figure 1: An example that ChatGPT can propose various different solutions to the same problem in zero-shot.

may work poorly on another, even when the latter may approach the best performance using prompts optimized for itself. These findings emphasize the need to optimize the prompt for each model for fair comparisons.

2. *Reprompting*: Prompt Inference Through Gibbs Sampling

2.1. In-Context Learning

In-context learning has become the cornerstone of evaluating large language models (LLMs) (Brown et al., 2020; Srivastava et al., 2022). To facilitate this evaluation approach, data is provided for a large number of different tasks, with each task consisting of dozens or, more often, hundreds of instances with varying problem setup and question texts x_i and their corresponding text answers y_i, where $i \in [1..N]$ and N is the number of problem instances for the task. Formally, in-context learning infers the answer for a given test question x by prompting an LLM with a set of demonstration examples $\{x_i, y_i\}_{i=1}^K$:

$$\hat{y} \sim p_{LLM}(y|\{x_i, y_i\}_{i=1}^K, x) \tag{1}$$

The performance of in-context learning can be significantly enhanced by incorporating auxiliary knowledge or human-written instructions in a prompt (Shwartz et al., 2020; Zelikman et al., 2022; Nye et al., 2021), particularly in the form of Chain-of-Thought (CoT) reasoning (Wei et al., 2022; Wang et al., 2022b; Zhou et al., 2022; Creswell et al., 2022; Wang et al., 2022a; Liu et al., 2022; Kojima et al., 2022; Li et al., 2022).

In-context learning with CoT (Wei et al., 2022) can be seen in a similar light, statistically. In addition to the question-answer pairs $\{x_i, y_i\}$, the CoT prompt also contains worked out step-by-step reasoning "recipes" z_i in text, which are inserted between the question and answer: $\{x_i, z_i, y_i\}$. These recipes can play two roles. First, they further explain the intent of the question x_i, as a small collection of question-answer pairs alone may be insufficient to disambiguate among different patterns an LLM might detect. The second role is more important: it provides step-by-step guidance on one problem and thus teaches an LLM to solve similar problems following the same routine as it continues the text conditioned on the previous tokens. In the extreme, with prompts that strictly regiment self-attention, GPT models can be turned into Turing Machines to execute standard computer algorithms (Jojic et al., 2023). In practice, the CoT prompts commonly used in prior work fall somewhere between colloquial explanations and regimented recipes. Formally, **in-context learning with CoT** infers the answer for a given test question x by prompting an LLM with an optional instruction message m and a set of demonstration examples with step-by-step solutions $\{x_i, z_i, y_i\}_{i=1}^K$:

$$\hat{z}, \hat{y} \sim p_{LLM}(z, y|\{x_i, z_i, y_i\}_{i=1}^K, x, m) \tag{2}$$

Here, m is a textual message that instructs the model to generate the step-by-step solution z_j before the answer text y_j and the specific format to present the answer.[1] It can be task-specific or generic, as in the case of our experiments. Such an instruction message can trigger instruction-tuned LLMs to generate step-by-step solutions given $[x_j, m]$ alone without any demonstration examples (i.e. $K = 0$), as illustrated in Figure 1. These solutions follow varying styles and often lead to incorrect answers. However, we argue that good recipes for solving the set of problems on a given task can evolve from these zero-shot solutions. In the next section, we introduce *Reprompting*, an iterative sampling algorithm that automatically produces the CoT recipes for a given set of problems without human intervention.

2.2. Prompt Inference Through Gibbs Sampling

We introduce the *Reprompting* algorithm, which aims to find a set of CoT recipes z_i that work **consistently** well as few-shot in-context examples for a dataset $\{x_i, y_i\}_{i=1}^N$. Specifically, we formulate it as the problem of sampling from a joint distribution

$$p(z_1, z_2, ...z_N|\{x_i, y_i\}_{i=1}^N, m) \tag{3}$$

such that $z_{1...N}$ are generalized enough so that given any test question x, the distribution over z and y is approximately invariant to the choice of the K-shot CoT recipes:

$$\begin{aligned} &p_{LLM}(z, y|\{x_i, z_i, y_i\}_{i=1}^N, x, m) \\ \approx &p_{LLM}(z, y|\{x_i, z_i, y_i\}_{i\in S}, x, m), \quad \forall S \subset [1, N], |S| = K \end{aligned} \tag{4}$$

Without characterizing the joint distribution, we can use Gibbs sampling (Geman & Geman, 1984) to generate such

[1] This enables us to separate the generated answer y_j from the step-by-step solution z_j and forces the model to stop after generating the answer.

samples $\{z_1, z_2, ...z_N\}$ by first sampling $\{z_1, z_2, ...z_N\}$ independently from the distributions $p(z_j|x_j, y_j)$, and then iteratively drawing samples from the conditional distributions $p(z_j|z_1, ..., z_{j-1}, z_{j+1}, ...z_N, \{x_i, y_i\}_{i=1}^N, m)$. Based on the property (4) of the joint distribution, we have the following approximation:

$$\begin{aligned} & p(z_j|z_1, ..., z_{j-1}, z_{j+1}, ...z_N, \{x_i, y_i\}_{i=1}^N, m) \\ =& p_{LLM}(z_j|\{x_i, z_i, y_i\}_{i\neq j}, x_j, y_j, m) \\ \propto& p_{LLM}(z_j, y_j|\{x_i, z_i, y_i\}_{i\neq j}, x_j, m) \\ \approx& p_{LLM}(z_j, y_j|\{x_i, z_i, y_i\}_{i\in S_j}, x_j, m), \\ & \forall S_j \subset [1, N]\backslash\{j\}, |S_j| = K \end{aligned} \tag{5}$$

Thus, we can sample z_j by randomly picking K data points (excluding j) and then sampling z_j with weights proportional to the conditional probability

$$\begin{aligned} & p_{LLM}(z_j, y_j|\{x_i, z_i, y_i\}_{i\in S_j}, x_j, m) \\ =& p_{LLM}(z_j|\{x_i, z_i, y_i\}_{i\in S_j}, x_j, m) \\ & \cdot p_{LLM}(y_j|\{x_i, z_i, y_i\}_{i\in S_j}, x_j, m, z_j) \end{aligned} \tag{6}$$

One way to approximate it is to sample several $\hat{z}_j$ from the LLM conditioned on $\{x_i, z_i, y_i\}_{i\in S_j}$, x_j and m, compute the weight for each $\hat{z}_j$ using the model's probability of the correct answer y_j conditioned on $\{x_i, z_i, y_i\}_{i\in S_j}$, x_j, m and $\hat{z}_j$, and sample a z_j from $\{\hat{z}_j\}$ based on the weights. In practice, however, the model likelihood of a given text may be inaccessible. Thus, we approximate it using rejection sampling – we sample z_j by sampling $\hat{z}_j$ and $\hat{y}_j$ from $p_{LLM}(z, y|\{x_i, z_i, y_i\}_{i\in S_j}, x_j, m)$ and then reject $\hat{z}_j$ with a probability of p_{rej} if $\hat{y}_j \neq y_j$. Otherwise, we accept $\hat{z}_j$ and update the sample. Algorithm 1 shows the complete *Reprompting* algorithm consisting of the initialization and iterative sampling steps. Note that we set the rejection probability p_{rej} in a way that allows solutions that lead to incorrect answers to be kept occasionally, as these solutions may still contain useful segments that evolve into good recipes through *Reprompting*.

Based on the properties of Gibbs sampling (Casella & George, 1992; Roberts & Smith, 1994), the algorithm should converge to the point where the probability $p_{LLM}(z_j, y_j|\{x_i, z_i, y_i\}_{i\in S_j}, x_j, m)$ is high and agnostic to the choice of S_j, which leads to a set of $\{z_j\}$ that work well as a prompt for solving similar problems in a separate test set.

The algorithm can also be viewed as a variant of evolutionary algorithms: 1) First, we generate the initial population of individuals (where each individual is a CoT recipe given a problem). 2) Next, we repeat the following regeneration steps iteratively: 2a) we first evaluate the fitness of each CoT recipe by comparing the answer that follows the recipe with the correct answer and weed out the least-fit recipes; 2b) we then breed new individuals through crossover and mutation by randomly selecting K recipes from the population as parent recipes, which are then used to prompt the LLM to generate recipes for a new problem. By repeating the 2a and 2b steps, initial recipes can be recombined (Figure 4) and evolve into better recipes (Figure 3) through iterations. And eventually, the fittest recipes (i.e. ones that can be followed to solve similar problems) will survive.

During testing, we select K tuples $\{x_i, z_i, y_i\}$ from the inferred $\{z_j\}$ based on the training accuracy when using each tuple individually in a prompt.

Algorithm 1: *Reprompting* algorithm

Input : Training set $\{x_i, y_i\}_{i=1}^N$, number of examples in the prompt K, number of iterations M, rejection probability p_{rej}, the initialization model LLM_1 and the sampling model LLM_2

Initialization:
for *each* j **do**
  $z_j \leftarrow \emptyset$
  Sample $\hat{z}_j, \hat{y}_j \sim p_{LLM_1}(z, y|x_j, m)$
  Sample $u \sim Uniform([0, 1])$
  if $\hat{y}_j = y_j$ *or* $u > p_{rej}$ **then**
    $z_j \leftarrow \hat{z}_j$
  end
end
Sampling:
repeat
  Randomly select $j \in [1, N]$
  Randomly select $S_j \subset [1, N]\backslash\{j\}$ of size K
  Sample $\hat{z}_j, \hat{y}_j \sim p_{LLM_2}(z, y|\{x_i, z_i, y_i\}_{i\in S_j}, x_j, m)$
  Sample $u \sim Uniform([0, 1])$
  if $\hat{y}_j = y_j$ *or* $u > p_{rej}$ **then**
    $z_j \leftarrow \hat{z}_j$
  end
until *convergence or M iterations are reached*

3. Experimental Setup

We evaluate the *Reprompting* algorithm against various baselines including zero-shot, few-shot, Chain-of-Thought (CoT), Chain-of-Thought combined with self-consistency decoding (Wang et al., 2022b), Auto-CoT (Zhang et al., 2022) and Automatic Prompt Optimization (Pryzant et al., 2023) on 20 challenging reasoning tasks, including 12 challenging tasks in the Big-Bench Hard (BBH) benchmark (Suzgun et al., 2022),[2]

[2]The BBH tasks include *Logical Deduction*, *Geometric Shapes*, *Object Counting*, *Penguins in a Table*, *Temporal Sequences*, *Date Understanding*, *Formal Fallacies*, *Movie Recommendation*, *Reasoning About Colored Objects*, *Ruin Names*, *Salient Translation Error Detection*, and *Word Sorting*.

GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). We choose both tasks that have been shown to benefit substantially from human-written CoT recipes, such as Logical Deduction, Geometric Shapes, Temporal Sequences, GSM8K and MATH, and tasks on which CoT does not improve much or does not improve consistently over zero-shot prompting, such as Formal Fallacies, Movie Recommendation and Word Sorting.

3.1. *Reprompting* Setup

For each task, we randomly select 20 training examples from the Big-Bench dataset excluding the test examples in the BBH benchmark.[3] We experiment with having $k \in \{1,3\}$ clones of the same training example in the set $\{x_i, y_i\}_{i=1}^{N}$ to allow for more diverse recipe samples (so the number of recipes we need to sample from the joint distribution (3) is $N = 20 * k$) and choose k that obtains the highest training accuracy. We set the number of examples in the prompt by $K = 5$. We run *Reprompting* for a maximum of $M = 20,000$ iterations. We allow for early stopping if the average training accuracy stops increasing for $1,000$ iterations. For the rejection probability, we experiment with $p_{rej} \in \{0.95, 0.99\}$ and choose $p_{rej} = 0.99$ as it leads to higher training accuracy on various tasks.

3.2. Baselines

Prompting Baselines For **zero-shot prompting**, we only include the test question x_i and the special message m in the prompt, which triggers the model to generate a step-by-step solution prior to the answer text. For **few-shot prompting**, we randomly select 20 training examples in the same way as in *Reprompting* and concatenate these examples in the form of question-answer pairs in the prompt, followed by the test question. For **CoT prompting**, we use the human-written CoT prompts from Suzgun et al. (2022). For **CoT with self-consistency decoding**, we use the same CoT prompts and follow Wang et al. (2022b) by sampling 10 reasoning paths per question and taking the majority vote on the answer. For both approaches, we randomly select 20 training examples in the same way as in *Reprompting*.[4]

Prompt Optimization Baselines We also compare *Reprompting* with two previous state-of-the-art prompt optimization algorithms, including **Auto-CoT** (Zhang et al., 2022) and **APO** (Pryzant et al., 2023). For **Auto-CoT**, since the original Auto-CoT algorithm differs from our setting as it focuses on the unsupervised setting without exploiting any labeled examples, we adapt the algorithm to our few-shot setting where it follows the original algorithm to generate diverse CoT recipes through zero-shot prompting but selects the demonstration examples based on the training accuracy when used individually in a prompt.[5] We also evaluate **APO**, a recently proposed nonparametric prompt optimization algorithm that uses LLMs to generate "textual gradient" – criticism of the current prompt – based on training samples and edit the prompt accordingly. The algorithm has been shown to outperform other prompt optimization methods, such as TEMPERA (Zhang et al., 2023), Automatic Prompt Engineering (Zhou et al., 2023), and AutoGPT.[6]

3.3. Large Language Models (LLMs)

We experiment with two powerful LLMs including ChatGPT (gpt-3.5-turbo; OpenAI (2023)) and InstructGPT (text-davinci-003; Ouyang et al. (2022)). We also experiment with a combo model for *Reprompting* where we use ChatGPT as LLM_1 for initialization and InstructGPT as LLM_2 for sampling. For both LLMs, we set the maximum number of output tokens to 500, $top\_p = 0.5$, zero frequency and presence penalty. Additionally, we include "END" as the stop word. We set the temperature to 1.0 for *Reprompting* and 0.0 for testing.

3.4. Evaluation Protocol

We extract the final answer from the model output by extracting the text between "<answer>" and "</answer>", except for the CoT baseline where we extract the final answer in the same way as in Suzgun et al. (2022). We measure accuracy based on exact match by comparing the extracted answer with the ground truth.

4. Results

4.1. Main Results

We first compare the performance of *Reprompting* with all the baselines on five BBH tasks. As shown in Table 1, results confirm the previous finding that few-shot in-context prompting improves the performance over zero-shot (Brown et al., 2020) and that CoT prompting outperforms both zero-shot and few-shot prompting by a large margin. However, human-written CoT prompting requires costly prompt en-

[3] Except for *Penguins in a Table* where there are only three samples in the Big-Bench dataset that are excluded from BBH, so we randomly select 17 more examples from BBH into the training set.

[4] Recent prompting methods that are more annotation-intensive, such as Complex-CoT (Fu et al., 2022) and Progressive-Hint Prompting (Zheng et al., 2023), are shown to outperform *Reprompting* by 3.3–5.6 points on GSM8K. However, these methods leverage substantially more human-annotated examples (e.g. 7.5K annotated examples on GSM8K) than *Reprompting*, thus they are not directly comparable.

[5] The original Auto-CoT algorithm selects the demonstration examples based on the diversity of the demonstration questions.

[6] https://news.agpt.co/

BBH Task	SOTA	ZS	FS	CoT	CoT+SC ChatGPT	APO	AutoCoT	*Reprompting* ChatGPT	*Reprompting* InsGPT	*Reprompting* Chat+Ins
Logical	60.4	35.1	46.4	63.1	62.7	28.0	53.2	**66.3**	53.7	60.0
Geometric	56.0	13.6	20.0	58.0	60.0	52.0	52.4	**72.8**	40.8	64.4
ObjectCount	93.2	52.4	46.8	95.6	95.2	74.8	88.8	97.2	42.8	**99.6**
Penguins	81.5	50.7	60.3	67.1	71.2	45.2	**85.6**	**85.6**	78.1	82.9
Temporal	96.8	38.4	41.2	66.8	66.8	50.4	80.8	93.2	28.4	**99.2**
Average	77.6	38.0	42.9	70.1	71.2	50.1	72.2	**83.0**	48.8	81.2

Table 1: Performance of several large language models (LLMs) using *Reprompting* versus the baseline prompting and prompt optimization methods on Big-Bench Hard (BBH) tasks. *SOTA* refers to the state-of-the-art performance among InstructGPT (text-davinci-002; Ouyang et al. (2022)), Codex (Chen et al., 2021), and PaLM 540B (Chowdhery et al., 2022) using CoT prompting from Suzgun et al. (2022). We also compare *Reprompting* with ChatGPT using *ZS* (zero-shot), *FS* (few-shot), *CoT*, *CoT+SC* (CoT prompting combined with self-consistency decoding (Wang et al., 2022b)), *APO* (automatic prompt optimization using textual gradient (Pryzant et al., 2023)), and *AutoCoT* (the few-shot version of Auto-CoT (Zhang et al., 2022)). For *Reprompting*, we show the performance of various LLMs – including *ChatGPT* (gpt-3.5-turbo; OpenAI (2023)), *InstructGPT* (text-davinci-003), and *Chat+Instruct* (a combo version that uses ChatGPT for initialization and InstructGPT at sampling steps).

gineering, as not all CoT recipes work equally well on LLMs (Madaan & Yazdanbakhsh, 2022; Jojic et al., 2023). Crucially, we show that using *Reprompting*, LLMs can achieve better performance compared to the existing CoT prompts, but without requiring any human guidance on how to solve problems step by step. Specifically, comparing the performance of ChatGPT using *Reprompting* versus the best human-written CoT prompts from Suzgun et al. (2022), *Reprompting* achieves consistently higher scores on all tasks.

Next, we compare *Reprompting* with self-consistency (SC) decoding (Wang et al., 2022b). CoT+SC improves over CoT on two of the five tasks, but the improvements are not consistent. By contrast, *Reprompting* consistently outperforms CoT+SC by 2–26 points on all five tasks.

Additionally, we compare *Reprompting* with existing prompt optimization algorithms. APO improves over zero-shot prompting on three out of five tasks but underperforms it on the two tasks where the model needs to search through a wide range of strategies to find effective solutions. By contrast, *Reprompting* consistently outperforms zero-shot and CoT prompting, and improves over APO by 20–43 points on all five tasks. When compared against Auto-CoT (Zhang et al., 2022), *Reprompting* also archives higher accuracy by +11 points on average. In summary, *Reprompting* outperforms strong decoding and prompt optimization baselines by 11–33 points on average.

Comparing the performance of *Reprompting* on different LLMs, we observe that InstructGPT underperforms ChatGPT on most tasks. However, we show that by using ChatGPT just as the initialization model LLM_1 to bootstrap InstructGPT as LLM_2 in *Reprompting*, we can improve performance over InstructGPT alone by 5–71 points and achieve competitive or even better performance than ChatGPT alone on two of the five tasks. We show in the Appendix why that is: while InstructGPT can follow a given recipe and even be used for recombining and evolving them, it is less capable of generating diverse initial solutions in a zero-shot manner. However, through *Reprompting*, we can use ChatGPT to "teach" InstructGPT diverse strategies for solving the training problems, which are then recombined and evolved by InstructGPT into better CoT prompts for itself.

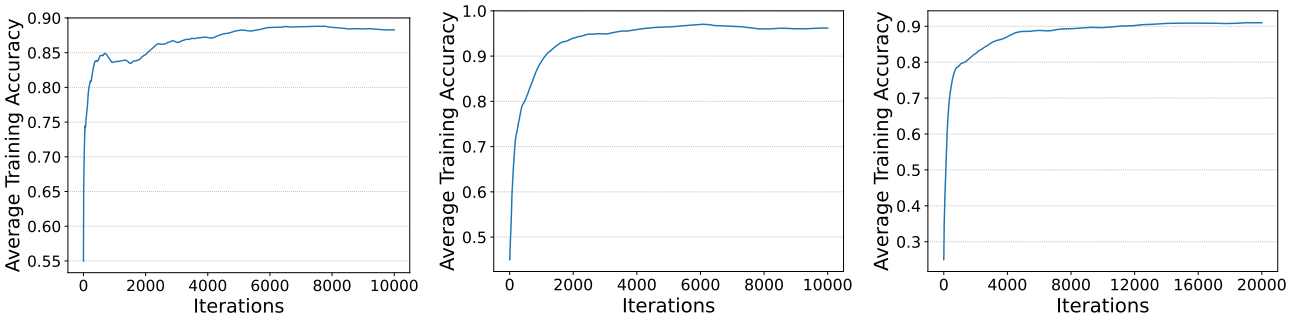
Furthermore, Table 2 shows the performance of *Reprompting* against zero-shot, few-shot and CoT prompting (all using ChatGPT) on the remaining 15 tasks.[7] *Reprompting* still outperforms zero-shot and few-shot prompting consistently and substantially by 14-15 points on average. Compared with CoT, *Reprompting* achieves better performance on 11 out of 15 tasks. On average, *Reprompting* outperforms CoT by +8.2 points. Interestingly, on tasks where CoT even underperforms zero-shot prompting, such as Movie Recommendation, Salient Translation Error Detection, and Word Sorting, *Reprompting* still improves over zero-shot prompting by large margins. This suggests that not all CoT recipes improve model performance, and some may even lead to degradation. This further emphasizes the need for algorithms like *Reprompting* for discovering and optimizing the CoT prompt to best exploit and compare LLMs.

Overall, these findings highlight the potential of *Reprompting* as a powerful method for automating CoT prompting on a wide range of tasks.

4.2. Quantitative Analysis

Ablation Study We conduct an ablation study on the rejection sampling and recombination process. Results in Table 3 show that, without rejection sampling, the test performance degrades substantially by 25 point on average. Always

[7] Based on the main results in Table 1, CoT+SC and Auto-CoT are more complicated than CoT but only slightly improves over CoT. Thus, we select CoT as a baseline here.



(a) InstructGPT (b) ChatGPT (c) ChatGPT+InstructGPT

Figure 2: Learning curves of the *Reprompting* algorithm using InstructGPT, ChatGPT, and the combo ChatGPT+InstructGPT models on the *Logical Deduction* task. The y-axis shows the accuracy on training samples averaged over the current and all previous iterations.

	ZS	FS	CoT	*Reprompting*
BBH				
Date	63.6	46.4	**76.8**	76.4
Formal	49.2	53.6	48.4	**56.8**
Movie	59.2	72.4	25.6	**78.4**
ColoredObj	66.8	48.8	**76.0**	74.0
Ruin	53.2	66.8	60.8	**74.8**
Salient	43.2	53.2	32.8	**54.8**
WordSort	58.0	72.0	46.0	**73.2**
GSM8K	45.6	26.5	75.6	**79.5**
MATH				
Algebra	37.6	23.7	52.0	**53.1**
Counting	17.1	19.8	26.6	**32.3**
Geometry	12.4	16.2	28.5	**29.2**
IntAlgebra	9.4	12.1	**18.0**	16.8
Number	20.8	17.1	32.9	**33.3**
Prealgebra	31.4	33.2	**54.0**	43.8
Precalculus	7.4	18.4	19.0	**19.3**
Average	38.3	38.7	44.9	**53.0**

Table 2: Performance of ChatGPT using *Reprompting* versus *ZS* (zero-shot), *FS* (few-shot), and *CoT* prompting methods on seven additional tasks from Big-Bench Hard (BBH) (Suzgun et al., 2022), GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021).

	$p_{rej} = 0$	$p_{rej} = 1$	NoRec	Orig.
Logical	56.3	61.9	54.7	**66.3**
ObjectCount	52.0	**97.2**	95.6	**97.2**
Temporal	74.8	74.4	90.4	**93.2**
Average	61.0	77.8	80.2	**85.6**

Table 3: Ablation study on rejection sampling (including no rejection ($p_{rej} = 0$) and always rejecting ($p_{rej} = 1$)) and recombination (*NoRec* represents *Reprompting* without recombination of previously sampled recipes) on Logical Deduction, Object Counting, and Temporal Sequences from Big-Bench Hard (BBH) (Suzgun et al., 2022). The *Orig.* column represents the standard *Reprompting* algorithm without ablation.

Tasks	InsGPT	ChatGPT
Logical	65.9	**66.3***
Geometric	53.6	**72.8***
ObjectCount	**99.6***	96.8
Penguins	82.2	**85.6***
Temporal	**99.2***	81.6

Table 4: Testing the best performing CoT prompt learned on ChatGPT, InstructGPT or InstructGPT+ChatGPT through *Reprompting* on both ChatGPT and InstructGPT. The superscript * denotes the model used as LLM_2 in *Reprompting*.

rejecting solutions that lead to incorrect answers also causes a degradation of 8 point. Additionally, not allowing multiple solutions to be recombined when sampling new solutions at the iterative sampling stage also hurts performance.

Do the generated CoT recipes generalize across models? We test the best-performing CoT recipes optimized with InstructGPT, ChatGPT, or InstructGPT+ChatGPT through *Reprompting* on both InstructGPT and ChatGPT. As shown in Table 4, the CoT recipes optimized for one model may not work as well for other models. Specifically, we observe that on tasks such as *Logical Deduction* and *Object Counting*, the best CoT recipes achieve similar performance on both InstructGPT and ChatGPT. However, on *Geometric Shapes* and *Temporal Sequences*, the best CoT prompts optimized for LLM_2 work well on LLM_2, but poorly with the other LLM – using them on the other LLM leads to 18–19 points lower accuracy than testing with LLM_2 (see examples in Figure A.2). On such tasks, using the prompt optimized for the testing LLM improves accuracy by 11–12 points over the same testing LLM with prompt optimized for other LLMs. These results suggest that, to make a fair comparison

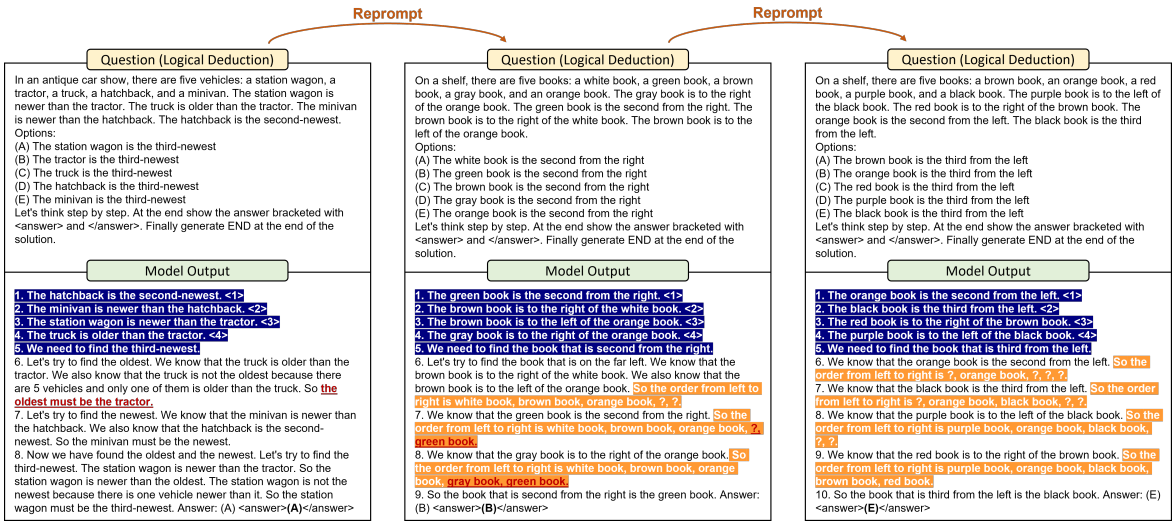


Figure 3: An example of how the CoT recipes evolve through *Reprompting*. In the left-most recipe, the model (ChatGPT) first reorders the constraints so that the ones with absolute ranking positions are considered prior to the ones with relative positions (highlighted in dark blue). Next, the model attempts to deduce the objects at specific positions but makes a mistake (see the red underlined part). Despite the error, this recipe still provides a useful strategy for solving similar problems – when it is used in a prompt to solve another problem, the model first adopts the same strategy to reorder the constraints and then proposes another way to deal with the constraints (highlighted in orange). Although the resulting solution still contains errors, it makes a good recipe for solving this type of problem. Thus, when using it in a new prompt to solve yet another problem, the model can follow the same recipe and deduce the correct answer.

between different LLMs, one needs to optimize the CoT prompt for each model.

***Reprompting* improves CoT recipes over iterations.** In Figure 2, we plot the average training accuracy (averaged over iterations up to the current iteration) over training iterations on *Logical Deduction*. For all three model variants, the initial training accuracy is relatively low, but it gradually increases (with occasional fluctuations) over iterations until convergence. This is the result of evolution and recombination of the recipes associated with training examples.

Compute and Resources We use the OpenAI APIs for all our experiments.[8] Running *Reprompting* costs around $80 (in US dollars) on gpt-3.5-turbo and $800 on text-davinci-003 based on the standard pricing,[9] while being exempted from any human cost. By contrast, CoT prompting requires manual prompt construction and engineering, which costs not only human labor (including the cost for humans to get familiar with the task itself and how LLM prompting works, write down various CoT solutions for each problem, test and optimize the solutions on the LLM) but also LLM queries, but these costs are typically neglected in previous works. In addition, previous works typically compare different LLMs using the same CoT prompt. While this strategy avoids additional costs for custimizing CoT prompt for each LLM (even with *Reprompting*, one can also save the cost by running it with ChatGPT and using the inferred CoT prompt on other LLMs), it risks making unfair comparisons as we have shown in Table 4 that the CoT prompt that works well on one model may be sub-optimal for another.

[8] https://platform.openai.com/docs/api-reference?lang=python

[9] https://openai.com/pricing

4.3. Qualitative Analysis

We observe that **even model outputs containing errors and unreasonable deductions can evolve into a high-quality recipe through *Reprompting*.** This is illustrated by the *Logical Deduction* example in Figure 3, when $K = 1$, where the model initially generates a recipe that is erroneous and contains illogical deductions. However, when this recipe is used as the new prompt for solving a similar problem, the model is able to exploit parts of the recipe and propose an alternative way to continue reasoning. Although the subsequent recipe still contains errors, it aids the model in correctly solving other problems when incorporated into a prompt. As a result, such recipes will be populated on other training samples, while the recipes that lead to low accuracy will eventually die out.

***Reprompting* combines fragments from different recipes into a better one.** *Reprompting* benefits from having multiple examples in the prompt, which allows the model to integrate various segments from different prompt recipes into a new recipe. As illustrated by the *Object Counting* examples in Figure 4, the model can combine large segments of reasoning steps, as well as small segments that address

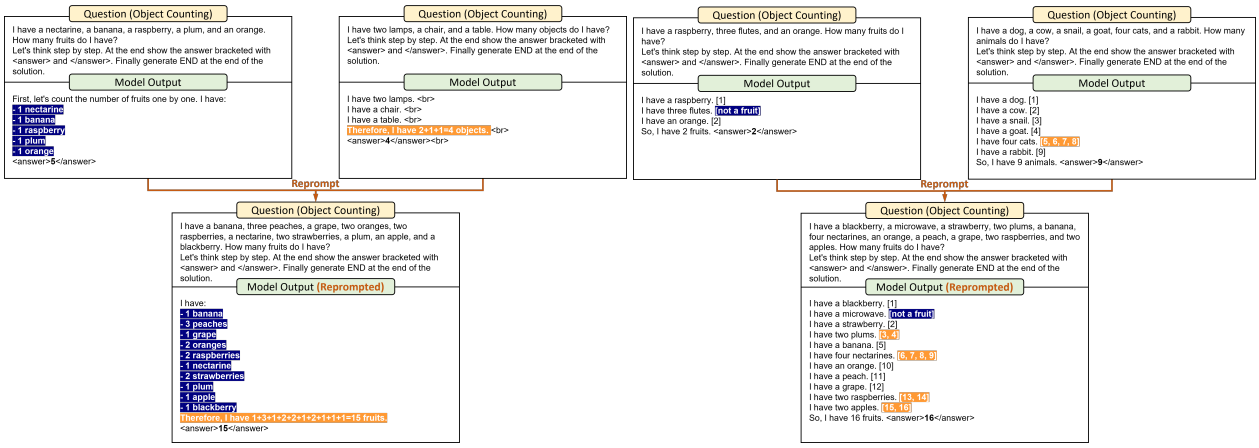


Figure 4: Examples of how fragments from different recipes in a prompt can be (re)combined into a better recipe to solve a new problem through *Reprompting*.

distinct cases to solve a more complex problem. The resulting prompts sometimes, but not always, share similarities with the human-written prompts (See the Appendix).

5. Related Work

In-Context Learning is an emergent ability of LLMs as they scale up in model sizes and training data, where an LLMs can learn to perform a task from a few examples in the context (which is also referred to as few-shot prompting) (Brown et al., 2020). It has been shown to achieve promising few-shot and even zero-shot performance on various natural language processing (Brown et al., 2020; Schick & Schütze, 2020; Perez et al., 2021) and program synthesis (Austin et al., 2021) tasks.

Reasoning via Chain-of-Thought Prompting Chain-of-Thought (CoT) prompting is a technique that enables LLMs to perform complex reasoning tasks by prompting them with a few examples with step-by-step solutions (Wei et al., 2022; Suzgun et al., 2022). CoT prompting has been shown to improve performance on various reasoning tasks, such as arithmetic reasoning (Wei et al., 2022; Zhou et al., 2022), symbolic reasoning (Wei et al., 2022; Zhou et al., 2022), multi-hop question answering (Press et al., 2022; Arora et al., 2022), and natural language inference (Wang et al., 2022b). However, designing effective CoT prompts requires human experts with an understanding of both the task and the prompting technique (Zamfirescu-Pereira et al., 2023), which limits the scalability and generalizability of CoT prompting.

Several works have attempted to **automate the process of CoT prompt discovery**. Zhang et al. (2022) proposed Auto-CoT, which uses LLMs to generate CoT solutions for diverse training questions in zero-shot and integrates the generated CoT solutions in the prompt for solving test questions. This method differs from *Reprompting* in that: 1) it focuses on the unsupervised setting and exploits a large set of example questions without annotated answers, and 2) it relies more heavily on the correctness of the zero-shot recipes as it does not have any iterative algorithm (as in *Reprompting*) to further improve the recipes. In our experiments, we adapted Auto-CoT to the few-shot setting and showed that *Reprompting* outperforms the few-shot version of Auto-CoT.

Deng et al. (2022); Zhang et al. (2023) proposed to train an additional policy model to find the best prompt through reinforcement learning, but their approaches are limited to prompt optimization within a relatively small search space (i.e. it is restricted to the prompts that are either extremely short or within a small edit distance from an initial prompt). Zhou et al. (2023) proposed a method for automatically generating, scoring and selecting effective instruction messages m for zero-shot chain-of-thought reasoning, which is orthogonal and can be potentially combined with our algorithm. Paranjape et al. (2023) introduced a framework that automatically retrieves demonstrations of related tasks from a task library and generates CoT solutions for the new task. However, this framework still requires collective human efforts to write demonstrations for a diverse set of tasks in the task library. In contrast, our *Reprompting* algorithm enables LLMs to solve complex reasoning tasks without any human guidance. Additionally, Yoran et al. (2023) proposed a multi-chain reasoning (MCR) method that prompts LLMs to combine pieces of information from multiple chains of thought to predict the final answer, which differs from our method in two ways: first, MCR combines multiple CoT solutions to the same question at test time, while *Reprompting* combines CoT solutions generated for different training questions before testing; second, MCR combines solutions only once, whereas *Reprompting* iteratively samples new

solutions and recombines them. As a result, *Reprompting* generates effective CoT recipes from only a few training examples, resulting in improved test performance without slowing down test inference.

6. Conclusion

We introduce *Reprompting*, an automated prompt inference algorithm which, without human effort, discovers effective chain-of-thought (CoT) prompts for each task given a few question-answer pairs. Experiments on 20 challenging reasoning tasks show that *Reprompting* achieves +9.4 higher accuracy than human-written CoT on average. It also outperforms self-consistency decoding and the state-of-the-art prompt optimization algorithms by 11–33 points on average. Our results also suggest that LLM comparisons can be highly sensitive to the choice of CoT prompts, further emphasizing the need for automatic prompt discovery and optimization using algorithms such as *Reprompting*.

Acknowledgements

We thank Bill Dolan, Sudha Rao and the reviewers for their valuable feedback.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Arora, S., Narayan, A., Chen, M. F., Orr, L. J., Guha, N., Bhatia, K., Chami, I., Sala, F., and Ré, C. Ask me anything: A simple strategy for prompting language models. *arXiv preprint arXiv:2210.02441*, 2022.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *Neural Information Processing Systems (NeurIPS)*, 2020.

Casella, G. and George, E. I. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Creswell, A., Shanahan, M., and Higgins, I. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*, 2022.

Deng, M., Wang, J., Hsieh, C.-P., Wang, Y., Guo, H., Shu, T., Song, M., Xing, E. P., and Hu, Z. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

Fu, Y., Peng, H., Sabharwal, A., Clark, P., and Khot, T. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.

Geman, S. and Geman, D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:721–741, 1984.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. *CoRR*, abs/2103.03874, 2021. URL https://arxiv.org/abs/2103.03874.

Jojic, A., Wang, Z., and Jojic, N. Gpt is becoming a turing machine: Here are some ways to program it, 2023.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

Li, Y., Lin, Z., Zhang, S., Fu, Q., Chen, B., Lou, J.-G., and Chen, W. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*, 2022.

Liu, Z., Patwary, M., Prenger, R., Prabhumoye, S., Ping, W., Shoeybi, M., and Catanzaro, B. Multi-stage prompting for knowledgeable dialogue generation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 1317–1337, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.104. URL https://aclanthology.org/2022.findings-acl.104.

Madaan, A. and Yazdanbakhsh, A. Text and patterns: For effective chain of thought, it takes two to tango. *arXiv preprint arXiv:2209.07686*, 2022.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

OpenAI. Gpt-4 technical report, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Paranjape, B., Lundberg, S., Singh, S., Hajishirzi, H., Zettlemoyer, L., and Ribeiro, M. T. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

Perez, E., Kiela, D., and Cho, K. True few-shot learning with language models. *Advances in neural information processing systems*, 34:11054–11070, 2021.

Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N. A., and Lewis, M. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

Pryzant, R., Iter, D., Li, J., Lee, Y., Zhu, C., and Zeng, M. Automatic prompt optimization with "gradient descent" and beam search. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494.

Roberts, G. O. and Smith, A. F. Simple conditions for the convergence of the gibbs sampler and metropolis-hastings algorithms. *Stochastic processes and their applications*, 49(2):207–216, 1994.

Schick, T. and Schütze, H. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.

Shwartz, V., West, P., Le Bras, R., Bhagavatula, C., and Choi, Y. Unsupervised commonsense question answering with self-talk. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4615–4629, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.373. URL https://aclanthology.org/2020.emnlp-main.373.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., and Zhou, D. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*, 2022a.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Yoran, O., Wolfson, T., Bogin, B., Katz, U., Deutch, D., and Berant, J. Answering questions by meta-reasoning over multiple chains of thought. *arXiv preprint arXiv:2304.13007*, 2023.

Zamfirescu-Pereira, J., Wong, R. Y., Hartmann, B., and Yang, Q. Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394215. doi: 10.1145/3544548.3581388. URL https://doi.org/10.1145/3544548.3581388.

Zelikman, E., Wu, Y., and Goodman, N. D. STaR: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

Zhang, T., Wang, X., Zhou, D., Schuurmans, D., and Gonzalez, J. E. TEMPERA: Test-time prompt editing via reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=gSHyqBijPFO.

Zhang, Z., Zhang, A., Li, M., and Smola, A. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

Zheng, C., Liu, Z., Xie, E., Li, Z., and Li, Y. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*, 2023.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Bousquet, O., Le, Q., and Chi, E. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., and Ba, J. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

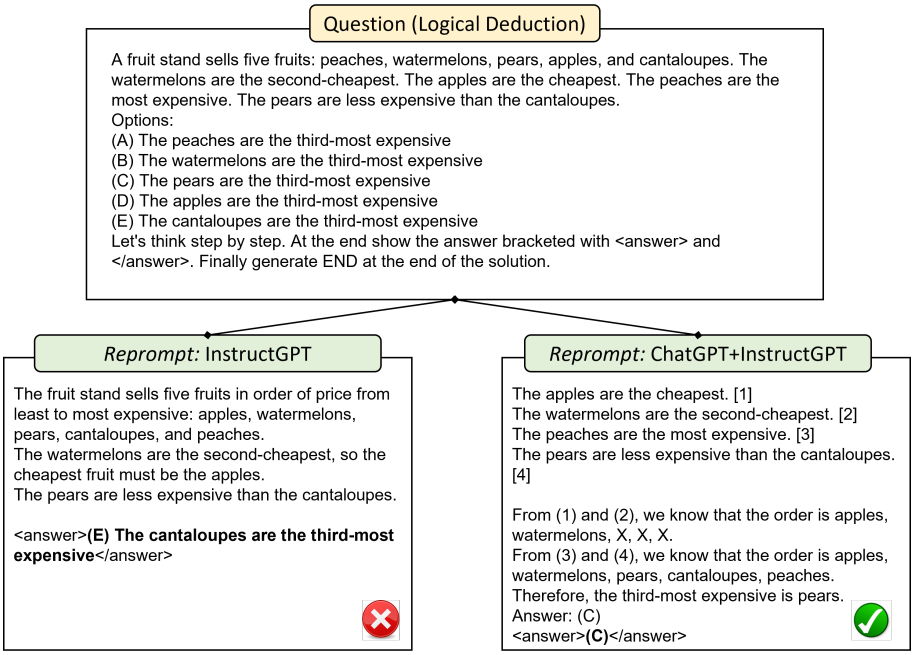


Figure A.1: Comparing the CoT recipes inferred through *Reprompting* using IntructGPT alone versus ChatGPT (for initialization) + InstructGPT (for sampling).

A. Additional Illustrations

On sensitivity to initialization We have shown that *Reprompting* can be sensitive to initial recipe generation. Armed with the optimal prompts discovered with ChatGPT+InstructGPT through *Reprompting*, InstructGPT can reach test accuracy equalling or besting ChatGPT on most challenging reasoning tasks. However, on some tasks, such prompts could not be discovered using InstructGPT itself as the initialization model LLM_1. Figure A.1 points to a likely explanation: ChatGPT can generate a wider range of useful recipes, and whether these initial recipes lead to the correct solution or not, InstructGPT can follow them and, through *Reprompting*, refine and correct them iteratively. Thus, as we have shown in our experiments, with a diverse pool of initial recipes, LLMs that may appear inferior based on their zero-shot performance may end up performing just as well or better than LLMs whose zero-shot performance is more encouraging. It would be interesting to see if *Reprompting* can use a mixture of LLMs in initialization to perform even better, or if humans can be put back into the loop to provide some initial recipes or some generic instructions on how to generate such recipes.

On transferability of discovered recipes The fact that LLM_1 (ChatGPT) can point LLM_2 (InstructGPT) in the right directions for prompt discovery does not mean that the discovered prompts, having been optimized for training performance on LLM_2, will perform well when used to prompt LLM_1. In fact, Table 4 indicates that the discovered CoT recipes that work for one model may not necessarily work for other models. For example, in the case of *Temporal Sequences*, the best performance is achieved with a prompt trained with InstructGPT (after initialization with ChatGPT as LLM_1). But when using that prompt on ChatGPT, the test performance is by 18% lower. Figure A.2 illustrates how ChatGPT and InstructGPT follow the same CoT prompt differently. Following the prompt recipes, the time intervals that need to be reasoned over are sorted, and among the sorted list, the missing interval was inserted as the possible interval when the person in question could have performed an activity. InstructGPT follows this procedure with accuracy over 99%, but ChatGPT sometimes skips the crucial line (for this recipe) with the missing interval within the timeline and therefore obtains suboptimal test accuracy. However, the best performance of ChatGPT (using the CoT prompt optimized for itself through *Reprompting*) is only slightly lower than that of the ChatGPT+InstructGPT combination.

These results suggest that, for a fair comparison between different LLMs, one needs to optimize the CoT prompt for each LLM using prompt optimization algorithms such as *Reprompting*.

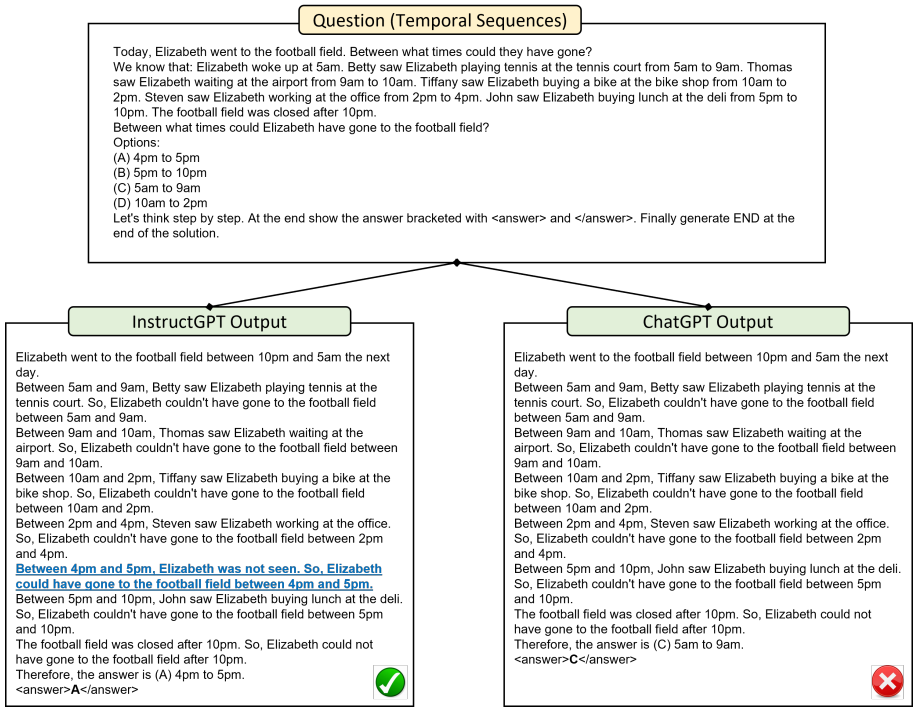
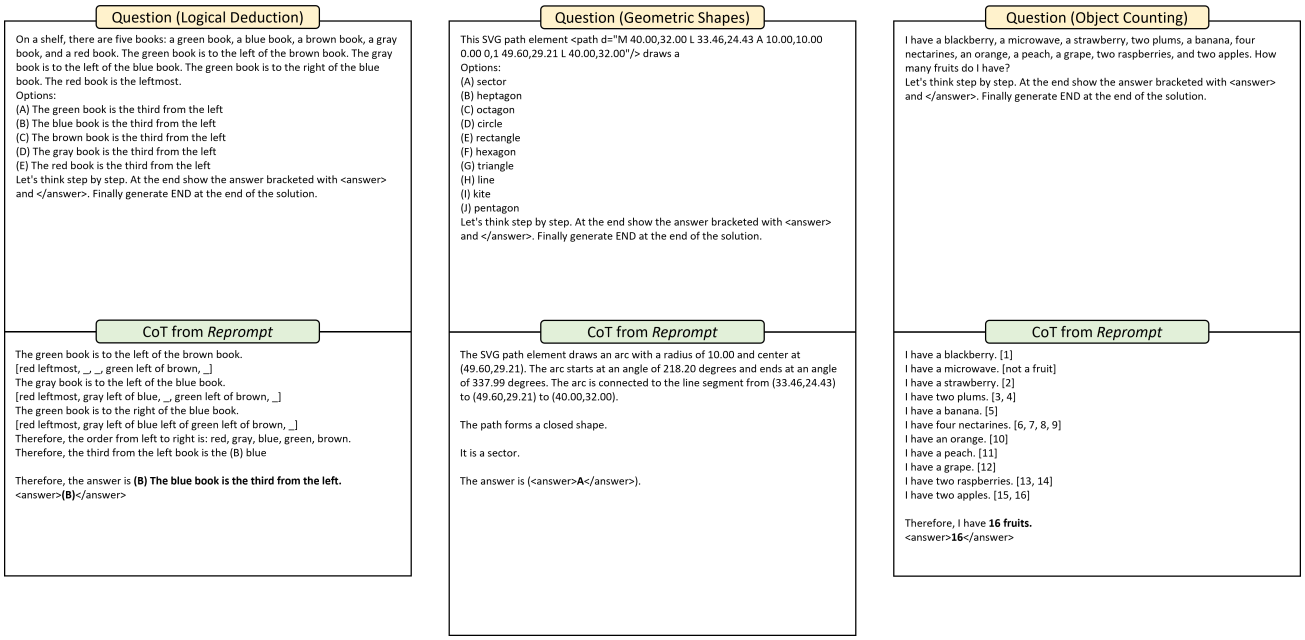
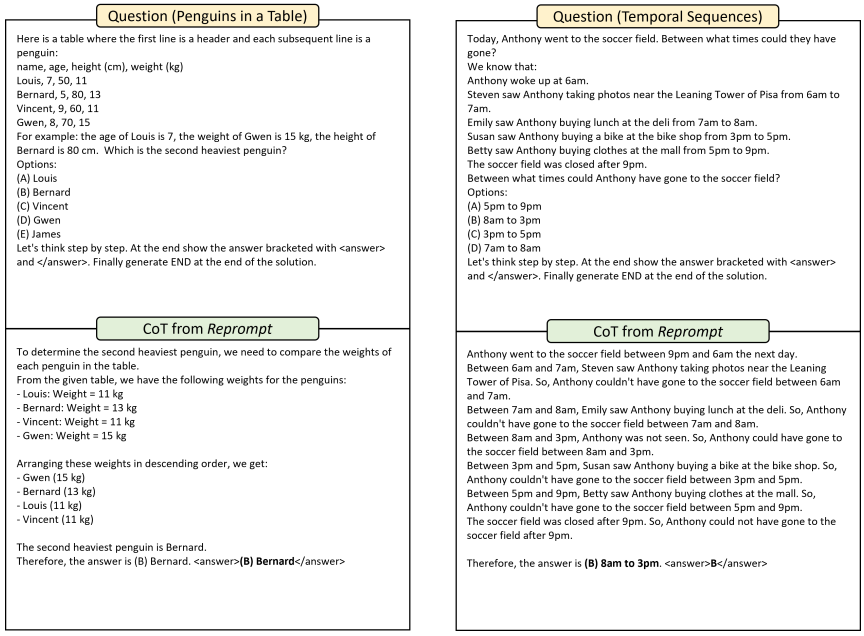


Figure A.2: An example on *Temporal Sequences (BBH)* where ChatGPT underperforms InstructGPT using the same CoT prompt optimized for InstructGPT via *Reprompting* (using ChatGPT+InstructGPT). ChatGPT fails to correctly execute the recipe as it skips a key step (the blue underlined text from InstructGPT) to reach the final answer. (The illustration does not show the full prompt that precedes the puzzle x for brevity; it consists of 5 training examples with worked-out solutions that all follow the same strategy of solving these types of problems.)

How do the model-generated CoT recipes differ from human-written ones? In the paper, We evaluated the performance of the CoT prompt discovered through *Reprompting* and contrasted it with human-written ones. As illustrated by the example recipes in Figure A.3, the automatically discovered CoT recipes share some similarities to human-written ones on some tasks (such as *Logical Deduction*), but differs on other tasks. For instance, on *Object Counting*, the CoT generated using *Reprompting* computes the total number of objects by incrementing the count one by one (e.g. adding 4 to the count 5 by "$[6, 7, 8, 9]$"), while in the human written recipe, it computes the addition through an arithmetic formula at the end.



(a)



(b)

Figure A.3: Examples of the best-performing CoT recipes inferred via *Reprompting* on *Logical Deduction*, *Geometric Shapes*, *Object Counting*, *Penguins in a Table*, and *Temporal Sequences*.