From the holdings of the Department of Special Collections , Princeton University Library, Princeton, NJ

These documents are either in the public domain, or can only be used for educational and research purposes (Fair use) as per U.S. Copyright law (text below). By accessing this file, all users agree that their use falls in the public domain or within fair use as defined by the copyrightlaw. We do not charge any permission or usage fees for the publication of images of material inour collections, including those provided by Princeton University Library, via our website, catalog, or directly from staff.For more information regarding copyright, credit, and citation please review the information onthe Copyright, Credit and Citation Guidelines page of our website.
https://library.princeton.edu/special-collections/policies/copyright-credit-and-citation-guidelines. If you have any questions, including how to order higher-quality images from the collection, contact Special Collections Public Services via our Ask Us! Form. https://library.princeton.edu/special-collections/ask-us

---

Master's Theses, 2022-2026

Computer Science, 2023

2024

Exploring LLM Insight Discovery: InsightBench and MCMC Prompting

Chhabria, Sabhya

Princeton, NJ : Princeton University

---

http://arks.princeton.edu/ark:/88435/dsp01mg74qq49s

U.S. Copyright law: The copyright law of the United States (Title 17, §108, United States Code) governs the making of photocopies or other reproductions of copyrighted material. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of fair use that user may be liable for copyright infringement. This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve a violation of copyright law. https://www.copyright.gov/title17/title17.pdf

Exploring LLM Insight Discovery: InsightBench and MCMC Prompting

Sabhya Chhabria

Advisor: Danqi Chen

Submitted in partial fulfillment

of the requirements for the degree of

Master's of Science in Engineering

Department of Computer Science

Princeton University

May 2024

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

_____

Sabhya Chhabria

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Sabhya Chhabria

Abstract

Large Language Models (LLMs) have become increasingly adept at existing benchmarks that seek to evaluate their abilities on traditional tasks such as language understanding, reasoning, programming and math questions. In this work, we introduce InsightBench, a new benchmark that seeks to evaluate LLMs on insight problems. Insight problems are those problems where the path to a solution is not immediately obvious. They require some form of creative thinking and the answer usually strikes in an "aha" moment. In this work, we evaluate several popular LLMs (both open and closed sourced) of varying sizes on InsightBench. Our results are able to establish that insight problems (creative problem solving) remain a critical failure point of language models. We further try to bolster LLM performance on certain InsightBench tasks by using Markov Chain Monte Carlo prompting schemes, allowing LLMs to explore large search spaces to solve insight problems.

Acknowledgements

First and foremost, I would like to thank you family, to whom I owe everything. Mom, Dad, and Hriday, thank you for all your love, support, and everything you do for me. I am forever grateful and indebted to you all.

I would like to thank my advisor Professor Danqi Chen, who guided me through the world of NLP over the last two years. Thank you for all your time, effort, and patience. I would also like to give a huge shout out to other members of the Princeton NLP group. Being a part of this esteemed group has been a privilege, and has given me access to opportunities beyond my imagination. More specifically, I would like to thank Dan for guiding me through this project and always being available when I needed help.

To all my dear friends, thank you for being my anchors here at Princeton. Your presence in my life has made my time at Princeton incredibly meaningful, and without a doubt, the best two years of my life. I am incredibly lucky and honored to call you my friends.

To my friends in the COS Master's program. Rish and Ani, thank you for being the most wonderful roommates. Thank you for making 718 Hibben Magie feel like home. I will always cherish our dinner table discussions and midnight Wawa runs. Anika and Emma, thank you for being the for always accompanying me in our search for free and midday sugar injections at Murray Dodge. In addition, I would like to thank Nathan, Prachi, Creston, and Shruthi for our Friday shenanigans.

The most defining part of my Princeton experience has been being a part of the cycling club. Thank you for giving me the opportunity to push myself beyond levels I could never have imagined two years ago. Adem and Brendan, thank you for joining me in my stupidity every night. From foraging at D-halls to Saturday night calorie hunts, thank you for providing the carbs to keep me going. To the rest of my cycling friends, Ben, Soleil, Masi, Kim, Caroline, Sebastian, Rosie, and Aaron, thank you

for making cycling more than just an individual undertaking. It has been a pleasure riding with you.

And finally, Ryo. Thank you for being my bestest friend. You were as much a part of this journey of mine at Princeton as I was myself. Thank you for being with me every step of the way and pushing me to better every day. No part of my experience would have any meaning at all if I could not share it with you. Thank you :)

Contents

Abstract . . . iii

Acknowledgements . . . iv

List of Tables . . . viii

List of Figures . . . ix

1 Introduction **1**

2 Background **3**

2.1 Benchmarks . . . 3

2.2 Insight Problems . . . 5

2.2.1 Insight . . . 5

2.2.2 Insight Problems . . . 6

2.2.3 System 1 & System 2 Thinking . . . 7

2.3 Existing Prompting Techniques . . . 8

2.4 Markov Chain Monte Carlo and Metropolis-Hastings . . . 9

2.4.1 Markov Chains . . . 9

2.4.2 Markov Chain Monte Carlo . . . 10

2.4.3 Metropolis-Hastings . . . 11

3 Insight Bench **13**

3.1 Remote Associates Test . . . 13

3.2 Spelling Bee and Pangram . . . 14

3.3 Connections and Linking 16

4 Experimental Setup **18**

5 Benchmark Results **20**

5.1 Remote Associates Test Results 20

5.2 Spelling Bee Results 22

5.3 Pangram Results 24

5.4 Connections Results 25

5.5 Linking Results 28

6 MCMC Setup and Experimentation **31**

6.1 Pangram MCMC 32

7 Future Work & Conclusion **35**

7.1 Future Work 35

7.2 Conclucion 36

List of Tables

4.1 Models used to Evaluate InsightBench 18

4.2 Prompting Methods 19

4.3 Tasks in InsightBench 19

5.1 Model performances on RAT using various prompting strategies. . . . 21

5.2 Model performances on Spelling Bee using various prompting strategies. 22

5.3 Spelling Bee Performance by Word Length 24

5.4 Model performances on Pangram using different prompting strategies. 24

5.5 Model performances on Connections across different prompting strategies. Each cell contains values for FMS, ARI, V Score, %Groups, and %Walls. 27

5.6 Model performances on Linking across different prompting strategies. 29

5.7 Asking GPT 3.5 if the Connection Matches 29

6.1 Performance of GPT 3.5 base prompt with $k = 5$ across varying numbers of MCMC iterations. 34

List of Figures

3.1 Remote Associates Test (RAT) 14
3.2 NYT Spelling Bee [23] 16
3.3 NYT Connections [22] 17

5.1 Remote Associates Test (RAT) Prompt 21
5.2 Spelling Bee Prompt 22
5.3 Spelling Bee Performance by Varying Word Length 25
5.4 Connections Prompt 26

6.1 MCMC Prompt 33
6.2 Performance of GPT 3.5 over varied iterations of MCMC prompts . . 34

Chapter 1

Introduction

Insight problems are those problems where the solution or a path to a solution may not be immediately obvious. The solution to such problems can usually occur us human beings in an "aha" or "eureka" moment. Examples of insight problems include, the Remote Associates Test (RAT), NYT Spelling Bee Game, and NYT Connections game, which are described Section 3. Insight problems make for an interesting set of problems to test LLMs against because they require departure from conventional thinking patterns and they engage several cognitive processes within human beings. In this work, we present InsightBench, which is a benchmark that thoroughly evaluates the capabilities of LLMs as well as popular prompting methods on insight problems. The motivation behind this benchmark is that insight problems are under explored failure point of LLMs. Existing benchmarks are slowly becoming increasingly saturated since they fail to explore stronger failure points of language models. We believe insight problems provide a good scenario for investigating and analyzing the problem solving capabilities and mechanisms of when exposed to problems with straightforward solutions or clear paths to solutions. As we show in further parts of the paper, existing prompting methods such as Chain of Thought (CoT) [26] and Tree of Thought [27] do not help improve on insight problems. Insight

problems can be used as a means to enhance creativity in in the future, across tasks beyond insight problems. We also explore the integration of LLMs within Markov Chain Monte Carlo (MCMC) frameworks as a guided generation strategy to improve performance on InsightBench. We incorporate LLMs within an MCMC framework as well as come up with a resampling strategy that explores encapsulating MCMC-like sampling behaviour within the prompt itself. Both approaches are explained in greater detail in Section **??**.

Chapter 2

Background

In this section of the paper, we hope to explore three broad areas of related works. The first broad area of related work will be work within the field of language modelling that explores existing benchmarks that assess several capabilities of LLMs. The second broad area of related work will look more at some more foundational work in the field of cognitive science, which has served as a source of inspiration for InsightBench. Finally, the third broad area of background we explore is that of Markov Chain Monte Carlo methods and the use of LLMs within such framworks.

2.1 Benchmarks

There exist several popular benchmarks that aim to evaluate various capabilities of language models such their general knowledge understanding, coding abilities, math skills, proficiency across language tasks and more. In this section, we will take a look at some of the most popular benchmarks and how popular open and closed sourced language models do on them. We consider OpenAI's GPT-3.5 [16] and GPT-4 [17] models in this section for closed source models and Meta's LLama-2 7B [24] and Mistral's 7B model [8] as the most popular and smaller sized open source models. We choose these models since we choose to evaluate our benchmark on them further on.

Here's a broad overview of the popular benchmarks that dominate the LLM research space:

- **MMLU** [7] - Massive Multitask Language Understanding is a benchmark that tests the knowledge acquisition abilities of across a diverse range of topics. GPT-4 achieves 0.864 success rate. Mistral achieves 0.601 and LLama-2 achieves 0.556 on this benchmark.

- **Big-Bench Hard** [21] - is a benchmark that focuses on 23 difficult tasks from the original Big Bench benchmark [20] on which did not outperform the average human error rate. GPT-4 achieves 83.1% success rate.

- **HumanEval** [4] and **Natural2Code** are both benchmarks that measure the coding capabilities of LLMs. GPT-4 achieves 0.67% success rate on HumanEval. Mistral achieves 0.305% on HumanEval and a fine-tuned version of LLama2 achieves 0.311%. Furthermore, GPT-4 achieves 0.739% success rate on Natural2Code.

- **GSM8K** [5] and **MATH** [7] are benchmarks measure the mathematical abilities of LLMs. GPT-4 achieves 0.92% success rate on GSM8K, whereas on the MATH benchmark Gemini achieves 0.532% and GPT-4 achieves 0.529%. Mistral and LLama2 have a lot of room to catch up on the math base benchmark with Mistral achieving 0.131% and LLama2 achieving 0.039% on Math, and Mistral achieving 0.521% and LLama2 achieving 0.16% on GSM8K.

As seen in the above benchmarks, large closed source models such as Gemini and GPT-4 are becoming increasingly proficient at solving most tasks, especially tasks that simply rely on retrieving knowledge from the training data or tasks that could be solved with simple paths to solutions. They are becoming more and more proficient at solving harder tasks that involve coding and mathematical reasoning abilities as

well. This highlights the need for more powerful benchmarks that evaluate language models at more difficult tasks such as insight problems.

The need for more human-like benchmarking is pushing the language modelling community towards evaluation of langauge models on more realistic tasks. In [9], the authors look at extending LLM evaluation from simply just solving simple programming tasks to fixing bugs in large repositories, a job more akin to that of a real-world software engineer. While LLMs have become increasingly proficient at solving coding tasks as seen earlier, they find that LLM performance on their benchmark SWE-Bench leaves much to be desired.

With the latest state of the art LLMs such as GPT-4, Gemini, etc giving rise to increasing talk about the advent of Artificial General Intelligence (AGI), there has been a push to evaluate LLMs on tasks that measure their creativity and problem-solving abilities. In [14], LLMs are evaluated on the Only Connect Wall dataset, testing the LLMs ability to draw semantic connections from groups of words. The results show that even state of the art LLMs are significantly worse than humans at this task. They find that LLMs are often tripped up by red herrings (misleading stimuli) that result in poor performance. Inspired by their exploration, we extend this work by using a similarly constructed dataset / their dataset and evaluating several more LLMs (including open sourced models) with different prompting frameworks as well our MCMC based Metropolis-Hastings setting.

2.2 Insight Problems

2.2.1 Insight

In the field of cognitive psychology, insight is defined as a sudden change in / formation of a concept or knowledge representation [10]. They are considered thoughts that are often considered having a flurry of emotions, or in other words, and "aha" moment.

Insights are also thoughts that help an individual overcome an impasse in their thinking, often times overcoming an incorrect direction of thought or strategy.

2.2.2 Insight Problems

Subsequently, we can think of insight problems as problems that require some form of insight to solve. Insight problems are designed to stimulate an initial impasse as the path to the solution is not immediately obvious and problem space seems ill-defined. Moreover, insight problems are also such where prior knowledge may not necessarily benefit a problem solver. In some problems prior knowledge may not be of any use and in others it could be a red herring [14].

A classic example of an insight problem is the nine dot problem, wherein the objective is to connect nine dots (arranged in 3×3 square) with four straight lines using a pen without lifting it from the paper. There is no strategy one could employ that would help define a clear path to the solution of this problem. All of the solutions of the nine dot problem require some form of out of the box thinking. It perfectly fits the definition of an insight problem as it first presents an impasse via an ill-defined problem space, solving it would require some form of insight, and the solution would illicit an "aha" emotion. The nine dot problem and one of its known solutions are shown below.

Insight problems are an interesting test bed for exploring thinking patterns and problem solving mechanisms within LLMs. A study conducted on undergraduate mathematics students on the effects of solving insight problems found that the solving of insight problems resulted in increased recall and problem solving abilities among students [12]. These findings motivates the question as to whether we can extend similar results in LLMs by designing a framework that would allow for the solving of insight problems and having its benefits transfer to other tasks.

2.2.3 System 1 & System 2 Thinking

It is also worthwhile looking at insight problems from the various lenses of human thinking. In his bestselling novel, Thinking, fast and slow, Daniel Kahneman defines the concepts of System 1 and System 2 thinking [11]. He defines System 1 thinking as the fast thinking mechanisms that allows us to make quick and automatic decisions. He defines System 2 thinking as the slow thinking mechanism that requires more deliberate thought and planning. The behaviour of language modelling can be looked at under the lens of these two systems of thought. When prompted, a language model simply returns the most likely sequence of words based on its procedural memory. This response can be considered a way of eliciting System 1 thinking. However, there are prompting mechanisms that make the language model deliberate further about its response. While several such prompting methodologies exist, the two most popular prompting techniques that have surfaced over the past year are Chain-of-Thought Prompting [26] and Tree-of-Thought Prompting [27]. These prompting methods provide the language model with more intermediate steps and ask it to evaluate its decision making at each step, eliciting System 2 thinking from the language model.

What makes these problems extremely interesting is the thinking that they elicit from human beings cannot be directly classified into either System 1 or System 2 thinking. It could be argued that insight problems require humans to make use of both our System 1 and System 2 ways of thinking. For example, when playing the New York Times Connections Games, we employ our pattern recognition skills to immediately group words together (System 1), however, we also consider the rest of the board to see whether or not the patterns we recognize comply with all the other words on the board (System 2).

As seen in this section, it could justifiably be argued that existing benchmarks do not test at the critical failure point that of insight problems, Moreover, existing prompting methods discussed above steer the language model towards System 2

thinking in addition to its System 1 thinking. However, poor performance of state of the art LLMs on our InsightBench as shown in Section **??** suggest that there is room for improvement and these two prompting methodologies do not help the models much when faced with insight problems.

2.3 Existing Prompting Techniques

Prior work has shown that LLMs cannot match up to human-like standards when evaluated on more human-centric tasks such as fixing bugs in repositories [9] and drawing semantic connections and grouping words [14]. However, they do not explore ways in which more nuanced prompting methods such as Chain of Thought [26] or Tree of Thought [27] can help bridge the gap between human and model performance. We evaluate our benchmark using these advanced prompting techniques as well as define a novel prompting strategy as shown in Section X. We first explore existing prompting strategies and why they help model performance.

Cain of Thought (CoT) [26] - this is a prompting technique that requires us to demonstrate the intermediate steps to the when trying to solve a problem. This was challenging for us to do since insight problems don't have clearly defined paths to solutions, however, we tried to give the as much context as possible in the form of intermediary steps to solve the insight problems.

Tree of Thought (ToT) [27] - this is a prompting method that was developed after CoT to overcome some of the limitations of CoT prompting. This method of prompting asks a model to maintain intermediary steps in the form of "thoughts" and then evaluate each thought and choose the most likely one. Once again, this prompting method was hard to difficult to verbalize for insight problems because of the undefined nature of the intermediary steps for them that an LLM could traverse. Hence, we choose to exclude this prompting method in our experimentation for the

time being.

2.4 Markov Chain Monte Carlo and Metropolis-Hastings

The Metropolis-Hastings algorithm is a popular Markov Chain Monte Carlo (MCMC) method. The algorithm allows us to sample from some probability distribution that is otherwise difficult to sample from. In this section, we first look at explaining MCMC algorithms and then more specifically look at the Metropolis-Hastings algorithm.

Using LLMs within an MCMC framework was also recently explored by [29] by using LLM outputs as key components of various sampling strategies. Their work was inspired by studies in cognitive science that explore simulating sampling algorithms on human beings to analyze mental representations held by humans. By incorporating LLMs within various MCMC sampling strategies, they are able to recover mental representations for human beings by approximating certain distributions and then comparing them to known distributions from similar experiments done with human beings. Similar to their method, we use adaptive methods to dynamically alter prompts for subsequent sampling steps for LLMs. Similar to their method of recovering mental representations for color, our work aims to recover mental representations for certain insight problems by using LLMs an in attempt to approximate the distributions of these problems. In other words, we would be trying to approximate the distribution $P(x|c)$, where x would represent the final correct answer of an insight game and c would be the context we are made aware of at the beginning of the game.

2.4.1 Markov Chains

In this section, we look at defining Markov Chains. To understand Markov Chains, we must first understand the Markov Property, which refers to the memory less state

of stochastic system. The Markov Property claims that the probability of moving from one state to another depends on the current state only [13]. It does not depend on any of the states prior to the current one. This can be formalized as follows. For a given process $X_n, n \in T$ for time domain T, for any states $X_1, X_2, \ldots, X_n$ we have:

$$P(X_{n+1} = s | X_n = s_n, \ldots, X_1 = s_1) = P(X_{n+1} = s | X_n = s_n) \tag{2.1}$$

A Markov Chain can be hence simply be defined as a stochastic process that satisfies the aforementioned Markov Property.

2.4.2 Markov Chain Monte Carlo

Consider a given Markov Process with $D(t, s)$ being the distribution of values over X_t. We know that $X_0 = s$ for some start state s. We now want to be able to estimate $D(t, s)$. There are two ways to do this. The first method being direct computation, this is possible when our underlying distribution $D(t, s)$ is not very complicated and easy to run computations with. The second way is via simulation or sampling, which is referred to as Markov Chain Monte Carlo.

Monte Carlo methods refer to a class of algorithms that involve some form of repeated sampling [13]. This class of computational algorithms generally samples repeatedly from a probability distribution in order to approximate some desired properties. Monte Carlo simulation lends itself naturally to the world of language modelling since the underlying probability distributions behind are extremely complicated.

In order to estimate the distribution $D(t, s)$, we start with $X_0 = s$ and then for all $i = 1, \ldots, t$, we simulate values for X_i using the known probability distribution P that define the transition probabilities and the previous state X_{i-1}. Repeating this multiple times gives us multiple samples from the distribution $D(t, s)$, eventually allowing us to approximate the distribution $D(t, s)$ [25].

The MCMC framework allows us to solve difficult problems that may or may not have any randomness, such as our insight problems. Moreover, as mentioned in [3], MCMC methods allow us to explore problem spaces that would otherwise be computationally infeasible to explore. In Section **??**, we take each insight problem and reframe it by designing a probabilistic process around it that allows us to run the subsequently explained Metropolis-Hastings algorithm using to solve the problem.

2.4.3 Metropolis-Hastings

We now define the Metropolis-Hastings algorithm, building upon the definitions in the previous subsections. The Metropolis-Hastings algorithm is a way to set up a Markov chain in order to sample from a probability distribution that is otherwise difficult to sample from. As defined above, in the Markov Chain setup, we are trying to estimate X_{n+1} using only X_n. We define the Metropolis-Hastings algorithms as defined in the cited blog post [15]. The Metropolis-Hastings algorithm breaks this down into two stages:

1. **Proposal Step** - the first step in the Metropolis-Hastings algorithm to estimate X_{n+1} is to generate a candidate X^* using the proposal distribution we know how to sample. We donate this proposal distribution by $Q(X^*|X_n)$.

2. **Accept-Reject Step** - the second step involves calculating the acceptance probability, which we denote by $A(X_n \longrightarrow X^*)$. This acceptance probability can be defined by:

$$A(X_n \longrightarrow X^*) = min\left(1, \frac{P(X^*)}{P(X_n)} \times \frac{Q(X^n|X^*)}{Q(X^*|X_n)}\right) \tag{2.2}$$

   The second term on the right hand side of the above equation allows us to correct for biases by considering the transition going both ways. Now, in order to make a decision on whether or not we are going to accept the generated candidate

X^*, we sample a number u from a uniform distribution between 0 and 1 and choose based on the following condition:

$$X_{n+1} = \begin{cases} X^* & \text{if } u \leq A(X_n \longrightarrow X^*) \\ X_n & \text{otherwise} \end{cases} \tag{2.3}$$

The setup for reframing our insight problems in the form of MCMC problems and incorporating LLMs within the Metroplois-Hastings algorithm is defined in Section **??**.

Chapter 3

Insight Bench

InsightBench is a benchmark that aims to evaluate the capabilities of on set of problems known as insight problems. In this section of the paper, we look at how insight problem based games for the benchmark were selected and how datasets for each game were constructed and used within the benchmark. As defined earlier, insight problems are those where the solution or the path to one is not immediately obvious. Our benchmark consists of three main tasks or insight problems. We describe each game, why it is suitable to our benchmark, and how we incorporated it into our benchmark in the following subsections.

3.1 Remote Associates Test

The Remote Associates Test (RAT) is the first insight problem that we chose to include in the InsightBench benchmark. It is a test of creative potential that was developed in 1962 by Martha Mednick [18]. Each RAT question consists of three clue words that are seemingly disconnected. Each of the three clue words is linked together by a fourth word semantically. This fourth word may not be immediately obvious and there seems to be no clear path that would lead us to this solution. For example, consider three clue words **Swiss, Cottage, Cake** as seen in Figure 3.1.

These words are seemingly disconnected, however, they all make sense when paired together with the word cheese: Swiss **Cheese**, Cottage **Cheese**, and **Cheese** Cake. In order to use the Remote Associate Test in our benchmark, we scrape 144 sets of tests `https://www.remote-associates-test.com`. We then use these to build prompts to prompt the , giving them the first three words and and asking them to produce the fourth word.

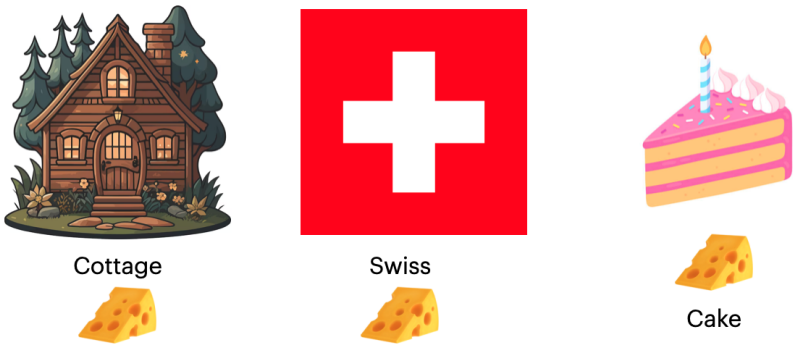


Figure 3.1: Remote Associates Test (RAT)

3.2 Spelling Bee and Pangram

The next two tasks that we include in InsightBench are SpellingBee and Pangram. The Spelling Bee is a popular online game that is managed by the New York Times [23]. The game is presented as shown in Figure 3.2 with seven lettered tiles in the shape of a honeycomb. The objective of the game is to come up with as many words as possible using some combination of the seven letters. The words must be at least four letters long. The main constraint of the game is that all words that a user comes up with must contain the center letter. Any word that contains all seven letters at least once is the called a pangram. The game calls upon players to employ both their memory, vocabulary, and creativity to come up with words that satisfy the constraints. It is clear to see that there is no straightforward path to a solution. One possible strategy a player could employ is to mix and match letters stochastically in order to

come up with words, however, this strategy fails to yield any dividend as there are simply too many possible permutations that one would have to consider. The New York Times asked its readers for tips and tricks for conquering the Spelling Bee [2]. All of the responses are strategies that are abstract and cannot be considered well defined steps that would allow a player to formulate a path to an answer. Since there exists no public dataset of Spelling Bee games, we decided to retrieve all of the past pangrams from a website: `https://www.sbsolver.com` that maintains an archive of past games. At the time of creation of this dataset, there were 1508 unique pangrams that had been used in Spelling Bee games. We scrape these 1508 pangrams to create our games by extracting the seven letters and then choosing a center letter at random. We chose to use pangrams that have previously been seen in real spelling bee games in order to stay as truthful to the original game as possible. We then search through a list of valid English words that satisfies the constraints for each game. This gives us a dataset of size 1059, ideal for experimentation. Using this dataset, we formulate two distinct tasks. The first task is simply the Spelling Bee game as where in we prompt a language model to generate words given the constraints. We score the response the same way as the traditional Spelling Bee game, where in, for each four letter word you make you get 1 point, for each word you make that is greater than length four you get the same number of points as the length of the word. If you make a "pangram" word (a word that includes every letter on the puzzle board) you receive 7 extra points. The second task is much less complicated and can be considered a subtask for the first. Here we ask the language model to generate one of the possible pangrams. We refer to this task as Pangram. We consider this task a classification task of whether or not a Language Model was able to correctly generate a valid pangram.

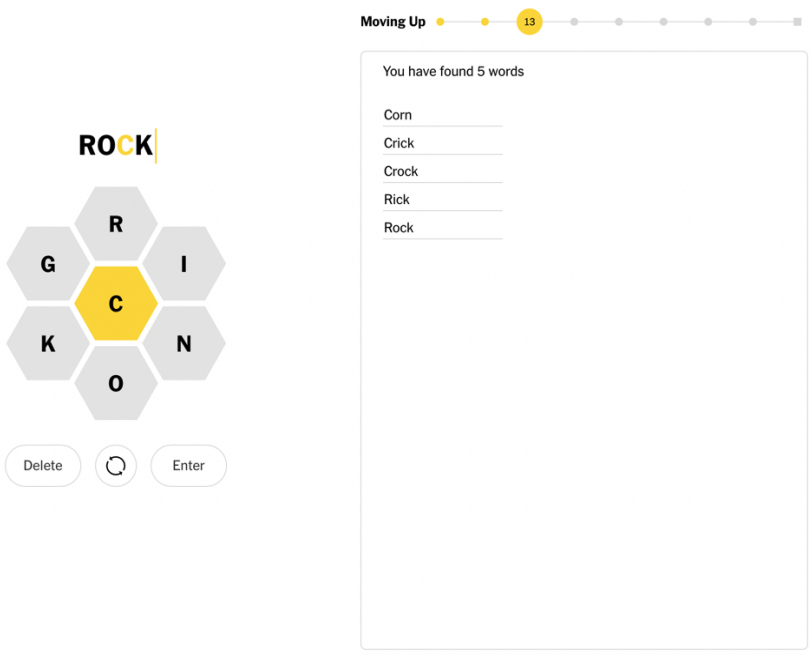


Figure 3.2: NYT Spelling Bee [23]

3.3 Connections and Linking

Connections is a new game that was introduced by the New York Times this year [22]. It is similar to the Remote Associates Test, however, instead of finding a fourth connecting word, we are giving all the words and we have to discover a common theme in between them. The game presents us with a board of sixteen worded tiles as shown in Figure 3.3. The game requires us to group these into four groups of four, where each group contains words that are related to each other semantically. For example, in Figure 3.3, the words "BUD", "CHUM", "MATE", "PAL" are all slang for friend and are hence connected. Similarly, there exists three more sets of four words within the game. While playing the game, the user gets a set number of mistakes it can make. However, we choose to evaluate just one LLM response, in essence giving the models just one try.

Since this game is relatively new, there are not sufficient enough games for us to build a dataset. We also chose not to use the dataset used in [14] since its knowledge

was skewed geographically towards the United Kingdom. We decided to scrape a sets of four from the website called `https://puzzgrid.com`. This gave us around 2000 groups of 4, which we use as a separate task called Linking. This task requires LLMs to find the semantic relation between four given words or phrases. We further divide these 2000 groups of 4 into 500 connection walls, which we use for the connections task described above.



Figure 3.3: NYT Connections [22]

All five of the insight above require the usage of both System 1 and System 2 type of thinking. We believe that these five insight problems make for a promising start to our benchmark. We hope to include more insight problems within the benchmark in the future, which we discuss in the Section **??**.

Chapter 4

Experimental Setup

In order to evaluate our benchmarks we decided to prompt the following models using two different prompting techniques for each model. We decided to use models of varying model size and as well as a mix of closed-source and open source models. For our open source models we chose Llama 2 7B, Mistral 7B, and OLMo 7B as these are the two most popular open source models that can easily run on a single GPU instance. For our closed-source models we use OpenAI's API to query GPT-3.5 and GPT-4. Note that for each of the aformentioned models, we used the instruction tuned version of these models. Exact details about our model choice can be found in the table 4.1 below.

Table 4.1: Models used to Evaluate InsightBench

Models	**Size (Parameters)**
Llama-2-7b-chat-hf	7 billion
OLMo-7B-Instruct	7 billion
Mistral-7B-Instruct-v0.2	7 billion
gpt-3.5-turbo-0613	Not specified
gpt-4-0613	Not specified

We decided to prompt these models and ask them to solve the five insight problems that we described earlier using two different prompt formats - basic prompts and

chain-of-thought prompts [26]. We also experiment with the number of few-shot examples to prompt the model with. For our experimentation we either use 5 or 10 few-shot examples. In all of the results tables in Section **??**, we abbreviate our prompting strategies as follows in table 4.2.

Table 4.2: Prompting Methods

Prompting Method	**Abbreviation**
Basic 5-shot	Basic (k=5)
Basic 10-shot	Basic (k=10)
Chain-of-Thought 5-shot	CoT (k=5)
Chain-of-Thought 10-shot	CoT (k=10)

We evaluate all of the above models as listed in table 4.1 with all of the prompting strategies as listed in table 4.2 on InsightBench. We summarize the tasks, their and sizes in table 4.3. Note that for each task, we further split the dataset into a train and test set. We reserve 10% of the dataset for the train set to use as in-context learning examples in our prompts, and we use 90% of the examples to test the models capabilities. Along with the benchmark, we release our datasets publicly as well.

Table 4.3: Tasks in InsightBench

Tasks	**Dataset Size**
Remote Associates Test (RAT)	144
Spelling Bee	1508
Pangram	1508
Linking	2444
Connections	611

Chapter 5

Benchmark Results

We now present the results of our experimentation in this section of the paper. We present the results of all of the models on each of the fives tasks of our benchmark.

5.1 Remote Associates Test Results

An example base and Chain-of-Thought prompt for the Remote Associate Test (RAT) can be seen in Figure 5.1. As seen, we have a system prompt that grounds the LLM to the task that it is solving by giving it some information and instructions about the task. This is followed by a user prompt that contains in-context learning examples and the sample that we want the model to attempt to solve. We build the CoT prompt on top of the base prompt to the best of our abilities by trying to define a sequence of logical steps the LLM can take in order to reach an answer. However, this is rather counter intuitive since the nature of insight problems is such that we cannot really define a clear path to a solution.

The results for the Remote Associate Test (RAT) can be see in the 5.1. The metric displayed here **top-3 accuracy**, that is the model was able to guess the answer within three tries.

As we can see, LLama 2, OLMo, and Mistral struggle with the RAT task, with

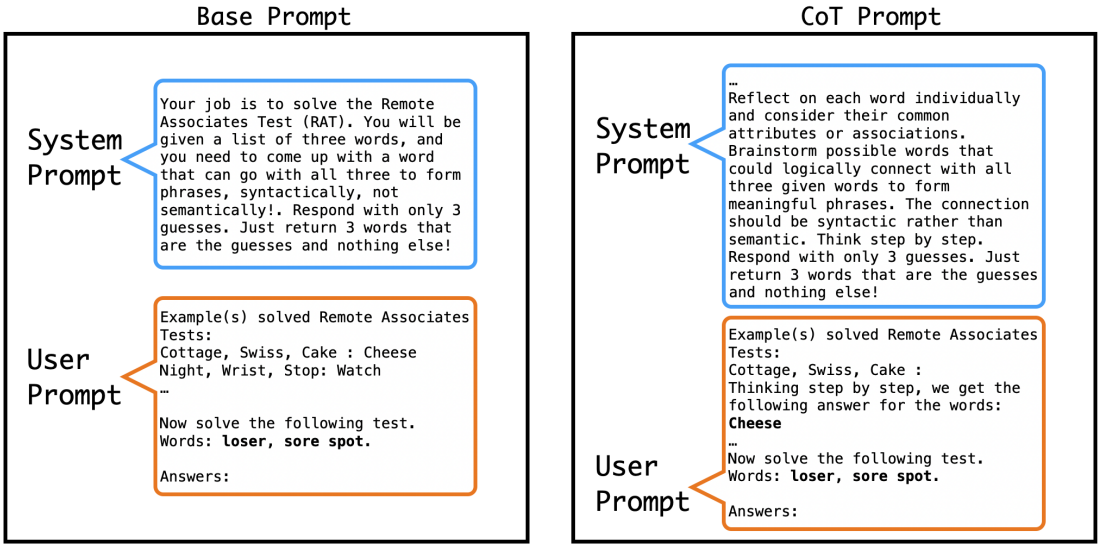


Figure 5.1: Remote Associates Test (RAT) Prompt

Table 5.1: Model performances on RAT using various prompting strategies.

Model / Prompt	Basic (k=5)	Basic (k=10)	CoT (k=5)	CoT (k=10)
LLama 2 7B	0.132	**0.186**	0.140	0.163
OLMo 7B	0.147	0.124	**0.163**	0.132
Mistral 7B	0.202	0.209	0.140	**0.217**
GPT 3.5	0.465	0.496	0.496	**0.504**
GPT 4	0.860	**0.868**	0.822	0.806

the best LLama 2 performance being 0.186 (Basic, k=10), OLMo being 0.163 (CoT, k=5) Mistral (CoT, k=10) being 0.217 (CoT). GPT 3.5 and GPT 4 are significantly better at the RAT task. The best result for GPT 3.5 is 0.504 (CoT, k=10) and for GPT 4 is 0.868 (Basic, k=10). As hypothesized in the earlier sections, we can see that there is no clear benefit to Chain-of-Thought prompting. In the case of LLama 2 and GPT 4, CoT prompting hurts model performance. After manually inspecting some of the model generations, we can see that the LLMs, especially some of the smaller ones in instances when they are unable to solve the task will default to their pretraining objective of next-word prediction and simply return words that are likely to be seen in conjunction with just one of the given choices in the questions.

5.2 Spelling Bee Results

An example base and Chain-of-Thought prompt for the Spelling Bee task can be seen in Figure 5.2. Similar to the RAT task, we build the CoT prompt to the best of our abilities by trying to define a sequence of logical steps the LLM can take in order to generate words that satisfy the criteria of the Spelling Bee task.

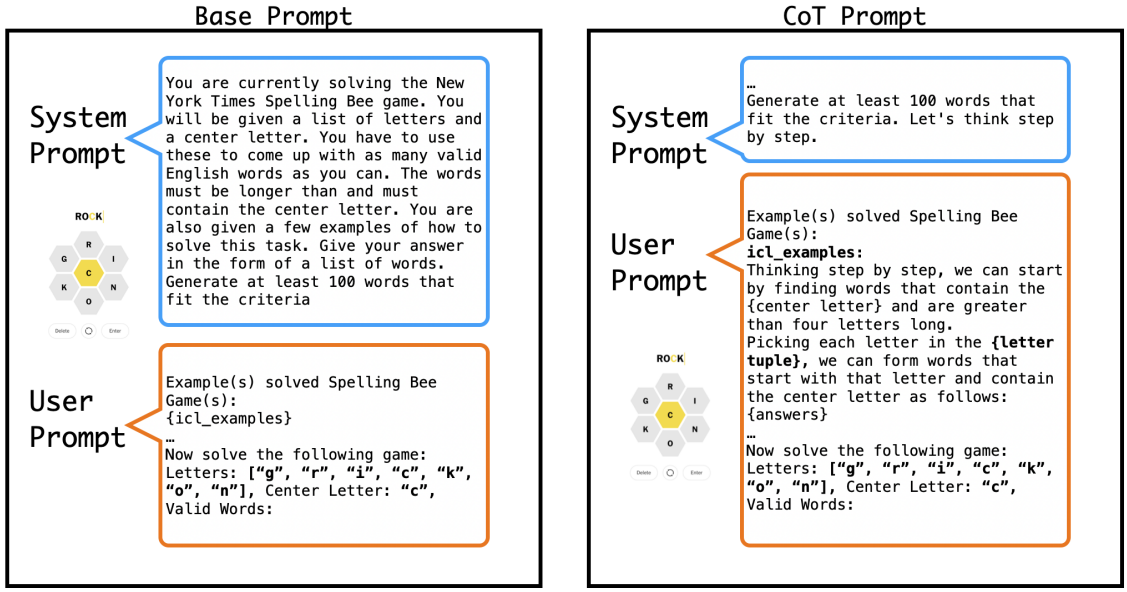


Figure 5.2: Spelling Bee Prompt

The results for the the Spelling Bee game can be seen in 5.2. The metric displayed in the table is the score as a percentage of the total possible score, averaged over all the games. We calculate this metric scoring the model response based on the rules described in Section 3.

Table 5.2: Model performances on Spelling Bee using various prompting strategies.

Model / Prompt	Basic (k=5)	Basic (k=10)	CoT (k=5)	CoT (k=10)
LLama 2 7B	0.018	**0.020**	0.019	0.016
OLMo 7B	0.066	**0.091**	0.066	**0.091**
Mistral 7B	0.126	**0.163**	0.123	0.160
GPT 3.5	0.160	0.180	**0.184**	0.179
GPT 4 (25%)	**0.390**	0.365	0.368	0.360

As seen in the table, Spelling Bee proves to be a rather difficult task. All of the models except GPT-4 had fairly poor performance across the board, especially the smaller models. Llama2 and OLMo have rather poor performance. They fail to reach even 10% of the total possible score for the games. Inspecting their generations manually, we see that they are skewed towards generating only four letter words and are rarely able to guess words that are longer. Moreover, we see that the generations often go astray and the models start coming up with gibberish that does not comply with the rules of the games. Mistral and GPT 3.5 fair slightly better, with the best Mistral model achieving 0.163 (Basic, k=10) and the best GPT 3.5 model achieving 0.184 (CoT, k=5). GPT 4 outperforms all other models significantly, achieving greater than two times the performance of the next best model, with the best instance being 0.390 (Basic, k=5). This is a very impressive result given how difficult the task is. Do note that GPT-4 was only evaluated on 25% of the test set due to budge constraints. Once again, we're able to observe that CoT prompting is not guaranteed help the model and could even hurt the performance as seen with three of the five models above.

We can also analyze how the LLMs perform when generating words of varying lengths. Looking at 5.3 and Figure 5.3, we can see the models are most successful on shorter words and the performance continues to drop for longer words. As the model size increases, so does the number of words guessed of each length. In essence, models get better at generating larger words with increasing size. Another thing that is evident in the from 5.3 is that the gap between the performance on between models grows when the length of the words generated increases.

Table 5.3: Spelling Bee Performance by Word Length

Word Length	Llama 2 7B	OLMo 7B	Mistral 7B	GPT 3.5	GPT 4
4	0.087	0.209	0.317	0.322	0.572
5	0.022	0.108	0.178	0.201	0.428
6	0.007	0.092	0.139	0.119	0.309
7	0.002	0.056	0.100	0.104	0.254
8	0.001	0.038	0.069	0.073	0.222
9	0.000	0.040	0.069	0.053	0.187
10	0.000	0.041	0.051	0.029	0.192
11	0.000	0.014	0.029	0.009	0.071
12	0.000	0.021	0.032	0.031	0.078

5.3 Pangram Results

Similar to the prompts in 5.2, we create prompts asking the LLMs to generate just the pangram (word that uses all of the characters at least once) using a given set of letters. The results for this task can be seen in table 5.4 below. The metric used here is accuracy. We simply judge whether or not the model was able to generate a pangram or not by searching the generated text for an exact match of all the possible pangrams for a particular game.

Table 5.4: Model performances on Pangram using different prompting strategies.

Model / Prompt	Basic (k=5)	Basic (k=10)	CoT (k=5)	CoT (k=10)
LLama2 7B	0.003	**0.006**	0.002	0.005
OLMo 7B	0.000	0.000	0.001	**0.003**
Mistral 7B	0.007	0.005	0.015	**0.018**
GPT 3.5	0.216	0.246	**0.254**	0.233
GPT 4 (25%)	**0.550**	0.525	0.412	0.429

From table 5.4 we can see that generating pangrams proves to be an extremely difficult task for smaller open-sourced models. This result was predictable given the poor performance of these models when generating words equal to longer than seven characters, as seen in figure 5.3. Larger models like GPT 3.5 and GPT 4 significantly outperform smaller models, with the best GPT 4 result being 0.550 (Basic, k=5).

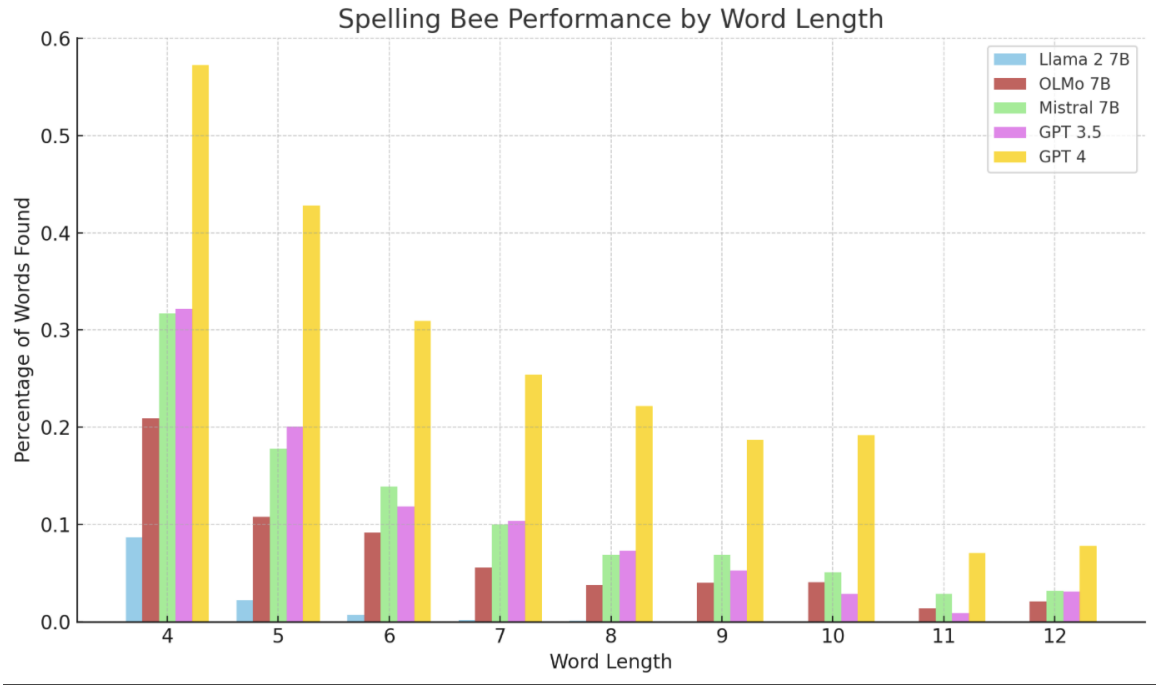


Figure 5.3: Spelling Bee Performance by Varying Word Length

This is an extremely interesting result, given that GPT 4 was only able to generate 0.254 seven letter words on average as seen in table 5.3 and even worse for words with greater length. With this result, we could justifiably argue that with greater prompt optimization, we could further boost scores on the original Spelling Bee task.

5.4 Connections Results

Example prompts for the Connections game can be seen in Figure 5.4. We design a prompt by providing the LLM with a randomized list of 16 words and then we ask the models to group these words into 4 groups of 4 as required by the connections task. The Chain-of-Thought prompts include some additional helpful steps that could help the model better group the words.

The results for the Connections game can be seen in 5.5. We employ several metrics to evaluate the results from the connections task. These metrics were selected to evaluate both, the correctness and quality of the clusters. The metrics can be

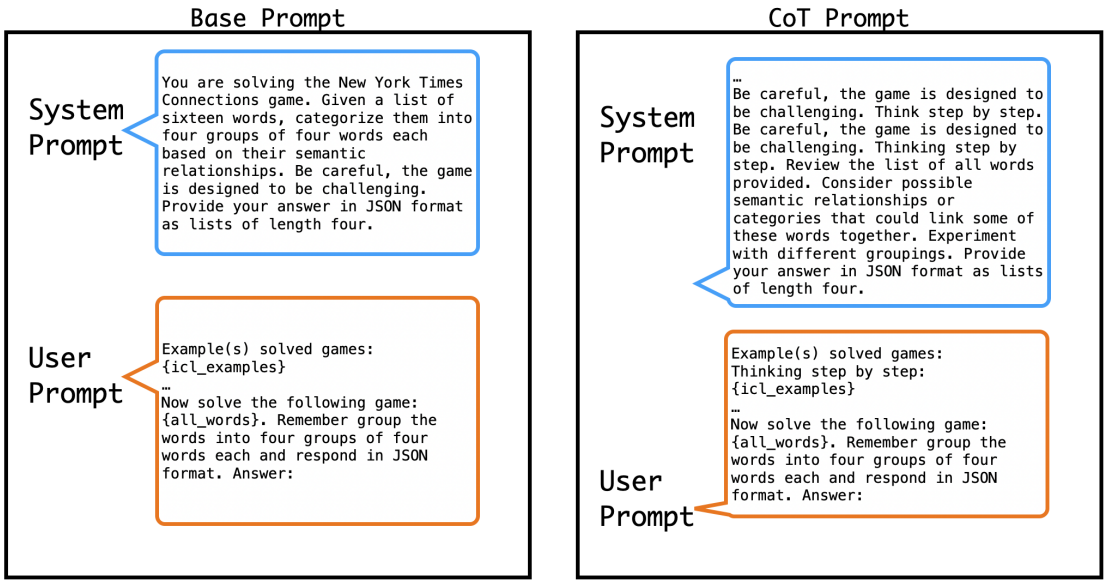


Figure 5.4: Connections Prompt

summarized as follows:

1. **FMS** - Fowlkes-Mallows Score is a method for comparing two heirachical clusterings and is defined as the geometric mean between the precision and recall [6].

2. **ARI** - Adjusted Rand Index is a also a measure of similarity between two clusterings. It is built on top of the Rand Index by further incorporating the fact that two given clusterings can match by chance [1].

3. **V Score** - the v-score [19], which is an average measure between completeness and homogeneity. The V-score is a metric that measures the quality of clustered outputs. The homogeneity of a cluster is defined as the property if all the clusters have data points that are members of just one class, whereas completeness is defined as the property if all data points within a cluster as in fact members of the same cluster. The V-score is hence defined as: $V = \frac{(1+\beta)\times homogeneity \times completeness}{\beta \times homogeneity + completeness}$. β is a variable that controls the tradeoff between homogeneity and completeness. We set $\beta = 1$ for our experiments to value both properties equally.

4. **%Groups** - this metric simply measures, what percentage of groups the model was able to group correctly. That is, how many sets of 4 was the model able to make correctly.

5. **%Walls** - this metric measures how many connections walls the model was able to solve completely. That is, on how many walls was the model able to solve all of the games correctly.

Table 5.5: Model performances on Connections across different prompting strategies. Each cell contains values for FMS, ARI, V Score, %Groups, and %Walls.

Model / Prompt	Metric	Basic (k=5)	Basic (k=10)	CoT (k=5)	CoT (k=10)
LLama 2 7B	FMS	0.231	**0.251**	0.234	0.229
	ARI	0.039	**0.064**	0.042	0.036
	V Score	0.316	**0.337**	0.321	0.314
	%Groups	0.014	**0.027**	0.014	0.013
	%Walls	0.004	**0.009**	0.002	0.002
OLMo 7B	FMS	0.228	0.225	0.231	**0.234**
	ARI	0.035	0.031	0.038	**0.043**
	V Score	0.313	0.309	0.316	**0.317**
	%Groups	0.017	0.013	0.017	**0.027**
	%Walls	0.010	0.006	0.010	**0.021**
Mistral 7B	FMS	**0.320**	0.315	0.313	0.305
	ARI	**0.149**	0.144	0.142	0.132
	V Score	**0.405**	0.402	0.400	0.392
	%Groups	**0.077**	0.073	0.069	0.065
	%Walls	0.023	**0.025**	0.019	0.023
GPT 3.5	FMS	0.466	**0.481**	0.465	0.471
	ARI	0.333	**0.352**	0.332	0.338
	V Score	0.544	**0.559**	0.543	0.548
	%Groups	0.214	**0.235**	0.217	0.220
	%Walls	0.073	**0.079**	0.076	0.073
GPT 4 (50%)	FMS	0.731	0.736	0.731	**0.748**
	ARI	0.664	0.669	0.664	**0.684**
	V Score	0.780	0.781	0.778	**0.793**
	%Groups	0.549	0.556	0.549	**0.576**
	%Walls	0.305	0.330	0.320	**0.358**

Similar trends are observed for Connections as the previous two tasks. The smaller

models, LLama 2, OLMo, and Mistral have extremely low scores, whereas the largest model GPT 4 has significantly higher scores across the board. This pattern follows across all five metrics that we described above. GPT 4, in the best scenario (CoT, k=10), was able to correctly solve an impressive number of walls (0.358) in its first attempt. Given the difficulty of this task, it would be interesting to see how this measures up against human performance. The three metrics that measure the quality of clusters (FMS, ARI, and V-Score) all exhibit a clear pattern once again, where we have the larger closed-sourced models performing significantly better than smaller open-sourced models. GPT 4 (CoT, k=10) has the best results across any model and prompting strategy.

5.5 Linking Results

As described in section 3 we define a subtask of the connections task that requires models to guess the link between groups of four words. We choose to do this instead of asking the model to group first and then find the common link since it is an easier task. The prompts for this task are similar to that of the connections task as seen in 5.4. In order to evaluate the linking task we use three metrics:

1. **Exact Match** - this metric tests if the model was able to exactly generate the string describing the connection between the four words.

2. **Rouge Score** - we consider the ROUGE 1 score that considers the overlap between two strings.

3. **BertScore** - is a metric that scores two strings based on their contextual BERT embeddings [28].

For each of the above three metrics, we pick a small sliding window and pass that over the model generated text and subsequently choose the best scores. This method,

however, could be flawed given that the model may explain its thinking and would hence generate words with similar contextual embeddings to the words in the correct answer, leading to a higher than expected BertScore. Hence, we do not include the BertScore in our analysis further on. The model performances on the linking task can be seen in table 5.6

Table 5.6: Model performances on Linking across different prompting strategies.

Model	**Metric**	**Basic (k=5)**	**Basic (k=10)**	**CoT (k=5)**	**CoT (k=10)**
LLama2 7B	Exact Match	0.011	0.014	0.022	**0.027**
	Rouge Score	0.162	0.164	0.181	**0.238**
	BertScore	0.698	0.684	0.686	**0.768**
OLMo 7B	Exact Match	0.011	0.014	**0.037**	0.033
	Rouge Score	0.133	0.121	**0.220**	0.187
	BertScore	0**.874**	0.872	0.835	0.814
Mistral 7B	Exact Match	**0.021**	0.020	0.011	0.015
	Rouge Score	**0.179**	0.163	0.151	0.146
	BertScore	**0.772**	0.750	0.671	0.659
GPT 3.5	Exact Match	0.015	**0.018**	0.016	0.015
	Rouge Score	**0.205**	0.196	0.144	0.154
	BertScore	**0.639**	0.631	0.492	0.501
GPT 4 (25%)	Exact Match	**0.024**	0.015	**0.024**	0.020
	Rouge Score	0.146	**0.116**	0.140	0.134
	BertScore	0.408	0.400	**0.447**	0.422

Table 5.7: Asking GPT 3.5 if the Connection Matches

Model / Prompt	**Basic (k=5)**	**Basic (k=10)**	**CoT (k=5)**	**CoT (k=10)**
GPT 3.5	0.15	0.155	0.164	**0.177**
GPT 4	0.248	**0.272**	0.221	0.202

As see in table 5.6, the trend of larger models significantly outperforming smaller models is defied by the linking task. While GPT 4 does have the highest Exact Match score of all models (0.024), we can't really look at this as a true measure of model performance given how strict this metric is. Looking at the Rouge Scores, we see that Llama 2 7B has the best score of 0.238 (CoT, k=10). Since the results for the larger

models seemed to completely defy the previous trends, we decided to evalute them in isolation and simply compare them against each other. We took the responses from GPT 3.5 and GPT 4 fed them to GPT 3.5 along with the correct answer and asked if there was a high degree of similarity between the two. The results for this can be seen in table 5.7. The results in this table are far more in line with expectations, with GPT 4 having generating the correct string most frequently (0.272), doing reasonably better than GPT 3.5. We were unable to use this method of evaluation for the smaller models since it was much harder to isolate exactly what the answer was within the response.

Chapter 6

MCMC Setup and Experimentation

As seen in the results above, there is plenty of room for improvement for smaller, open sourced models on each of the tasks that we explored. In the remainder of this work, we explore Markov Chain Monte Carlo methods in conjunction with the Metropolis-Hastings algorithm to try and improve model performance on InsightBench. Our approach here is two-fold. We will first try to show that when used within the Metroplis-Hastings will exhibit stronger creative and critical thinking, thereby displaying empirical evidence of MCMC, similar to work done in [3]. Subsequently, we develop a prompting strategy that tries to incorporate this MCMC behaviour within it to see if we can improve upon the results shown in the previous section.

The idea of using Markov Chain Monte Carlo (MCMC) search strategy within a semantic network to solve the Remotes Associates Test was explored by [3]. This work predates the era of LLMs, so the underlying model and network that was used during the MCMC formulation of the RAT dataset was collected from human responses. In this section, we will look at how we can adapt the the Pangram task to comply with the MCMC framework, using probabilities from open source language models that we looked at earlier. Our formulations build upon the base formulation for the RAT task that was defined in [3] and [29].

In the MCMC formulations below, we will define two main components. First, we define a proposal distribution to generate new candidates at any given stage of running Metropolis-Hastings algorithm. Second, we define an objective function that allows us to either accept or reject the generated candidate based on some custom criterion. Our objective functions are informed by the results that we observed in Section 5. We re-evaluate all of the LLM responses from the direct prompting scenarios above on our the new objective functions defined below in order to have a fair comparison between the methods.

6.1 Pangram MCMC

We present two formulations for the proposal distribution. They are defined below. In either case, the start state X_0 is initialized with a random shuffling of the letters.

1. **Formulation 1 (Symmetric)** - at each step, generate candidate X_{n+1} by choosing one of the actions with equal probability:

   (a) Swap two letters within the sequence

   (b) Insert a letter somewhere in the sequence

   (c) Shuffle all the letters - this would essentially reset the problem to a new start state X_0

2. **Formulation 2 (Non-symmetric)** - this formulation of the proposal distribution incorporates LLMs. Each prompt would contain the start state X_0 and the current guess or state X_n. At each step, we ask the LLM to generate a new guess X^{n+1}. Essentially sampling from the distribution with probability $P(X_{n+1}|X_n, X_0)$. The LLM is also allowed to shuffle the letters based on the current guess, essentially resetting the problem for itself. While this method can simply be looked as reprompting an LLM if its initial guess is incorrect, we try

to incorporate MCMC-like behaviour within the LLM via the prompt in effort to get it "unstuck" during the generation.

For the symmetric formulation above, we present an LLM judge with the two options X_{n+1} and X_n and ask it to pick the one it thinks is more likely to become a pangram if it were used to sample the symmetric distribution. We try to see if we can eventually reach the correct answer based on this sampling strategy, however, our initial experimentation has not yielded any positive results.

For the non-symmetric formulation, our initial experimentation has yielded some positive results. We pick GPT 3.5 (Base, k=5) as our base model, and random 10% subset of the dataset to work with. We prompt the model regularly. If the response is incorrect, we then reprompt the model with our MCMC Prompt. An example of the MCMC prompt can be seen below in figure 6.1

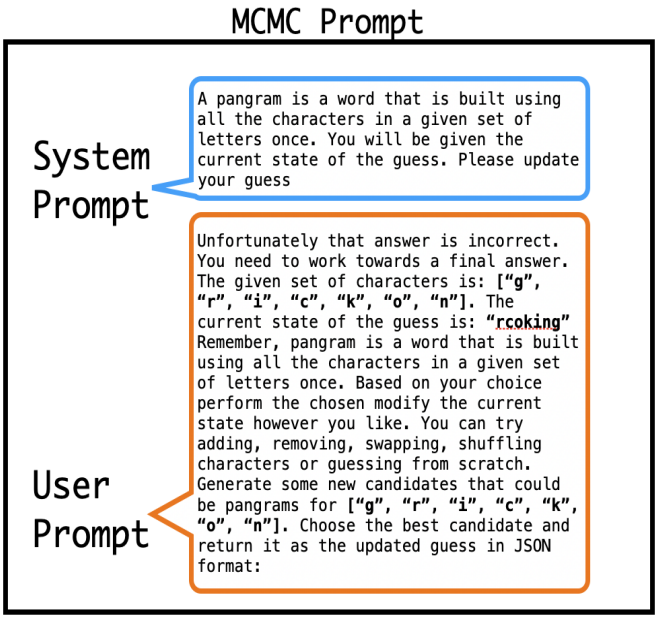


Figure 6.1: MCMC Prompt

The results for our experimentation can be found below in table 6.1. We vary the number of times we reprompt GPT 3.5, with 0 being the baseline. We choose to repropt 1, 5, 10, 25, 50, and 100 times.

Table 6.1: Performance of GPT 3.5 base prompt with $k = 5$ across varying numbers of MCMC iterations.

Num MCMC Iterations*	**Accuracy**
0 (baseline)	0.216
1	0.37
5	0.39
10	0.44
25	0.44
50	0.43
100	**0.47**

As seen in the above table, even reprompting the model one time almost double the performance (0.37) over the baseline. The reprompting continues to increase performance and then starts tapering off after 10 iterations as seen in the graph below 6.2.

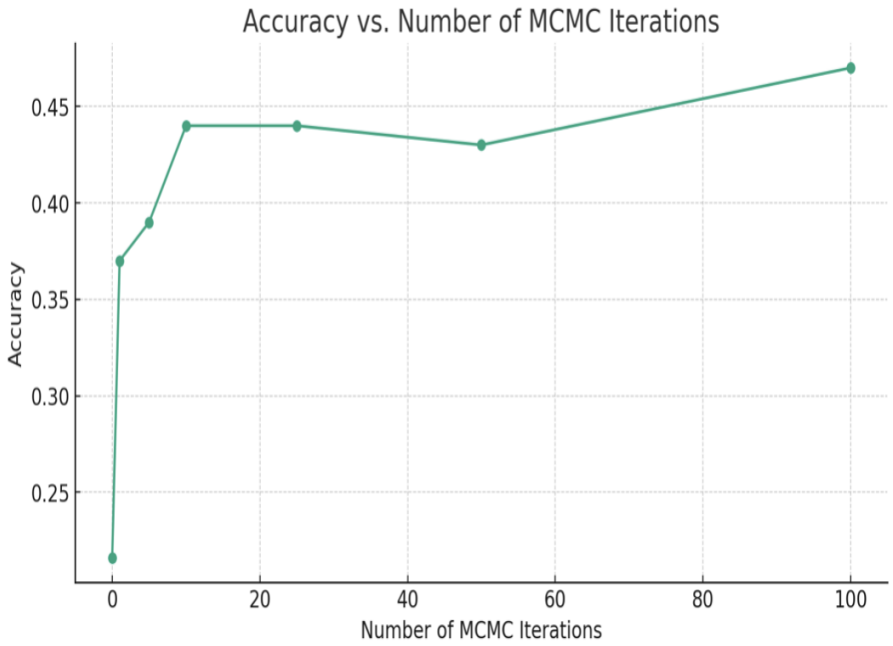


Figure 6.2: Performance of GPT 3.5 over varied iterations of MCMC prompts

Chapter 7

Future Work & Conclusion

7.1 Future Work

While we are proud of the progress that we have made with InsightBench, we believe it is only just the beginning. Here are a few ways which we believe we improve upon InsightBench.

One way in wich we can improve InsightBench is to add support for more insight problems. This would make the benchmark more interesting and robust. An easy addition to InsightBench would be the game Codenames, which involves guessing groups of connected words from a given set of words and clues. We can also take advantage of the latest multimodal APIs of GPT 4 and Gemini to pass in image data and see if that improves performance across all tasks.

Another way in which we can do more thorough experimentation with the MCMC formulation to include more multi-step evaluations that mimic the thinking patterns of human beings. It would be helpful to do more larger scale experiments testing MCMC formulations for the remaining tasks as well.

Another limitation of the project that we hope to assess is that of dataset contamination. Given that the tasks were collected from games for which solutions exist

online, we believe that there is a high likelihood of there being some amount of dataset contamination, which would essentially result in the models having slightly elevated performance.

7.2 Conclucion

We can see that existing prompting methods, while widely successful when solving tasks with clear paths to solutions, don't provide a significant (if at all) boost in performance when used for insight problems. Another conclusion that we can draw is that the ability to solve insight problems might be an emergent property in language models as we see a clear trend in model size and model performance. Smaller models like LLama 2, OLMo, and Mistral performed significantly worse than their larger counterparts GPT 3.5 and GPT 4 on all tasks.

We believe that the LLMs suffer on InsightBench tasks because of their next word prediction training objective. We suspect that this training objective can hinder the ability of an LLM to think outside the box and make connections across words with sound semantic reasoning.

InsightBench is a foray into the interesting combination of the worlds of language modelling and cognitive science. We believe that with InsightBench, we have explored a critical failure point of language model and have introduced a new benchmark that manages to trip up even the most advanced language models.

Bibliography

[1] Adjusted rand index (ari). https://oecd.ai/en/catalogue/metrics/adjusted-rand-index-%28ari%29. Accessed: 2024-04-29.

[2] D. Amlen and J. Frere. How the hivemind conquers spelling bee, Dec 2021.

[3] D. Bourgin, J. Abbott, T. Griffiths, K. Smith, and E. Vul. Empirical evidence for markov chain monte carlo in memory search. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36, 2014.

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

[5] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021.

[6] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.

[7] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding, 2021.

[8] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.

[9] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2023.

[10] M. B. John Kounios. The cognitive neuroscience of insight. *Annual Review of Psychology*, 65:71–93, 2014.

[11] D. Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[12] P. G. Liljedahl. Mathematical discovery and affect: the effect of aha! experiences on undergraduate mathematics students. *International Journal of Mathematical Education in Science and Technology*, 36(2-3):219–234, 2005.

[13] D. R. Miller. *Markov processes*, pages 486–490. Springer US, New York, NY, 2001.

[14] S. Naeini, R. Saqur, M. Saeidi, J. Giorgi, and B. Taati. Large language models are fixated by red herrings: Exploring creative problem solving and einstellung effect using the only connect wall dataset, 2023.

[15] D. Navarro. The metropolis-hastings algorithm, Apr 2023.

[16] OpenAI. Gpt-3.5: Language models are few-shot learners. https://platform.openai.com/docs/models/gpt-3-5, 2021. 11/15/2023.

[17] OpenAI. Gpt-3.5: Language models are few-shot learners. https://platform.openai.com/docs/models/gpt-4, 2021. 11/15/2023.

[18] Remote Associates Test. Remote associates test. https://www.remote-associates-test.com, 2023. Accessed on: 12/1/2023.

[19] A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In J. Eisner, editor, *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[20] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, A. Kluska, A. Lewkowycz, A. Agarwal, A. Power, A. Ray, A. Warstadt, A. W. Kocurek, A. Safaya, A. Tazarv, A. Xiang, A. Parrish, A. Nie, A. Hussain, A. Askell, A. Dsouza, A. Slone, A. Rahane, A. S. Iyer, A. Andreassen, A. Madotto, A. Santilli, A. Stuhlmüller, A. Dai, A. La, A. Lampinen, A. Zou, A. Jiang, A. Chen, A. Vuong, A. Gupta, A. Gottardi, A. Norelli, A. Venkatesh, A. Gholamidavoodi, A. Tabassum, A. Menezes, A. Kirubarajan, A. Mullokandov, A. Sabharwal, A. Herrick, A. Efrat, A. Erdem, A. Karakaş, B. R. Roberts, B. S. Loe, B. Zoph, B. Bojanowski, B. Özyurt, B. Hedayatnia, B. Neyshabur, B. Inden, B. Stein, B. Ekmekci, B. Y. Lin, B. Howald, B. Orinion, C. Diao, C. Dour, C. Stinson, C. Argueta, C. F. Ramírez, C. Singh, C. Rathkopf, C. Meng, C. Baral, C. Wu, C. Callison-Burch, C. Waites, C. Voigt, C. D. Manning, C. Potts, C. Ramirez, C. E. Rivera, C. Siro, C. Raffel, C. Ashcraft, C. Garbacea, D. Sileo, D. Garrette, D. Hendrycks, D. Kilman, D. Roth, D. Freeman, D. Khashabi, D. Levy, D. M. González, D. Perszyk, D. Hernandez, D. Chen,

D. Ippolito, D. Gilboa, D. Dohan, D. Drakard, D. Jurgens, D. Datta, D. Ganguli, D. Emelin, D. Kleyko, D. Yuret, D. Chen, D. Tam, D. Hupkes, D. Misra, D. Buzan, D. C. Mollo, D. Yang, D.-H. Lee, D. Schrader, E. Shutova, E. D. Cubuk, E. Segal, E. Hagerman, E. Barnes, E. Donoway, E. Pavlick, E. Rodola, E. Lam, E. Chu, E. Tang, E. Erdem, E. Chang, E. A. Chi, E. Dyer, E. Jerzak, E. Kim, E. E. Manyasi, E. Zheltonozhskii, F. Xia, F. Siar, F. Martínez-Plumed, F. Happé, F. Chollet, F. Rong, G. Mishra, G. I. Winata, G. de Melo, G. Kruszewski, G. Parascandolo, G. Mariani, G. Wang, G. Jaimovitch-López, G. Betz, G. Gur-Ari, H. Galijasevic, H. Kim, H. Rashkin, H. Hajishirzi, H. Mehta, H. Bogar, H. Shevlin, H. Schütze, H. Yakura, H. Zhang, H. M. Wong, I. Ng, I. Noble, J. Jumelet, J. Geissinger, J. Kernion, J. Hilton, J. Lee, J. F. Fisac, J. B. Simon, J. Koppel, J. Zheng, J. Zou, J. Kocoń, J. Thompson, J. Wingfield, J. Kaplan, J. Radom, J. Sohl-Dickstein, J. Phang, J. Wei, J. Yosinski, J. Novikova, J. Bosscher, J. Marsh, J. Kim, J. Taal, J. Engel, J. Alabi, J. Xu, J. Song, J. Tang, J. Waweru, J. Burden, J. Miller, J. U. Balis, J. Batchelder, J. Berant, J. Frohberg, J. Rozen, J. Hernandez-Orallo, J. Boudeman, J. Guerr, J. Jones, J. B. Tenenbaum, J. S. Rule, J. Chua, K. Kanclerz, K. Livescu, K. Krauth, K. Gopalakrishnan, K. Ignatyeva, K. Markert, K. D. Dhole, K. Gimpel, K. Omondi, K. Mathewson, K. Chiafullo, K. Shkaruta, K. Shridhar, K. McDonell, K. Richardson, L. Reynolds, L. Gao, L. Zhang, L. Dugan, L. Qin, L. Contreras-Ochando, L.-P. Morency, L. Moschella, L. Lam, L. Noble, L. Schmidt, L. He, L. O. Colón, L. Metz, L. K. Şenel, M. Bosma, M. Sap, M. ter Hoeve, M. Farooqi, M. Faruqui, M. Mazeika, M. Baturan, M. Marelli, M. Maru, M. J. R. Quintana, M. Tolkiehn, M. Giulianelli, M. Lewis, M. Potthast, M. L. Leavitt, M. Hagen, M. Schubert, M. O. Baitemirova, M. Arnaud, M. McElrath, M. A. Yee, M. Cohen, M. Gu, M. Ivanitskiy, M. Starritt, M. Strube, M. Swedrowski, M. Bevilacqua, M. Yasunaga, M. Kale, M. Cain, M. Xu, M. Suzgun, M. Walker, M. Tiwari, M. Bansal, M. Aminnaseri, M. Geva,

M. Gheini, M. V. T, N. Peng, N. A. Chi, N. Lee, N. G.-A. Krakover, N. Cameron, N. Roberts, N. Doiron, N. Martinez, N. Nangia, N. Deckers, N. Muennighoff, N. S. Keskar, N. S. Iyer, N. Constant, N. Fiedel, N. Wen, O. Zhang, O. Agha, O. Elbaghdadi, O. Levy, O. Evans, P. A. M. Casares, P. Doshi, P. Fung, P. P. Liang, P. Vicol, P. Alipoormolabashi, P. Liao, P. Liang, P. Chang, P. Eckersley, P. M. Htut, P. Hwang, P. Miłkowski, P. Patil, P. Pezeshkpour, P. Oli, Q. Mei, Q. Lyu, Q. Chen, R. Banjade, R. E. Rudolph, R. Gabriel, R. Habacker, R. Risco, R. Millière, R. Garg, R. Barnes, R. A. Saurous, R. Arakawa, R. Raymaekers, R. Frank, R. Sikand, R. Novak, R. Sitelew, R. LeBras, R. Liu, R. Jacobs, R. Zhang, R. Salakhutdinov, R. Chi, R. Lee, R. Stovall, R. Teehan, R. Yang, S. Singh, S. M. Mohammad, S. Anand, S. Dillavou, S. Shleifer, S. Wiseman, S. Gruetter, S. R. Bowman, S. S. Schoenholz, S. Han, S. Kwatra, S. A. Rous, S. Ghazarian, S. Ghosh, S. Casey, S. Bischoff, S. Gehrmann, S. Schuster, S. Sadeghi, S. Hamdan, S. Zhou, S. Srivastava, S. Shi, S. Singh, S. Asaadi, S. S. Gu, S. Pachchigar, S. Toshniwal, S. Upadhyay, Shyamolima, Debnath, S. Shakeri, S. Thormeyer, S. Melzi, S. Reddy, S. P. Makini, S.-H. Lee, S. Torene, S. Hatwar, S. Dehaene, S. Divic, S. Ermon, S. Biderman, S. Lin, S. Prasad, S. T. Piantadosi, S. M. Shieber, S. Misherghi, S. Kiritchenko, S. Mishra, T. Linzen, T. Schuster, T. Li, T. Yu, T. Ali, T. Hashimoto, T.-L. Wu, T. Desbordes, T. Rothschild, T. Phan, T. Wang, T. Nkinyili, T. Schick, T. Kornev, T. Tunduny, T. Gerstenberg, T. Chang, T. Neeraj, T. Khot, T. Shultz, U. Shaham, V. Misra, V. Demberg, V. Nyamai, V. Raunak, V. Ramasesh, V. U. Prabhu, V. Padmakumar, V. Srikumar, W. Fedus, W. Saunders, W. Zhang, W. Vossen, X. Ren, X. Tong, X. Zhao, X. Wu, X. Shen, Y. Yaghoobzadeh, Y. Lakretz, Y. Song, Y. Bahri, Y. Choi, Y. Yang, Y. Hao, Y. Chen, Y. Belinkov, Y. Hou, Y. Hou, Y. Bai, Z. Seid, Z. Zhao, Z. Wang, Z. J. Wang, Z. Wang, and Z. Wu. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023.

[21] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, and J. Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022.

[22] The New York Times. Connections puzzle game. `https://www.nytimes.com/puzzles/connections`, 2023. 12/7/2023.

[23] The New York Times. Spelling bee puzzle game. `https://www.nytimes.com/puzzles/spelling-bee`, 2023. Accessed on: 11/15/2023.

[24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[25] T. R. . G. Valiant. Cs168: The modern algorithmic toolbox lecture 14: Markov chain monte carlo. Stanford University Course CS168: The Modern Algorithmic Toolbox, May 2023. Available online at `https://web.stanford.edu/class/cs168/l/l14.pdf`.

[26] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[27] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[28] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. Bertscore: Evaluating text generation with bert, 2020.

[29] J.-Q. Zhu, H. Yan, and T. L. Griffiths. Recovering mental representations from large language models with markov chain monte carlo, 2024.