Cognitive, Behavioral and Social Data

Analogical reasoning of GPT-4

Mikhail Kolobov[1], **Maksim Kokot**[2]
Yelnur Shauketbek[3]

Department of Mathematics, Università degli Studi di Padova

Date: January 10, 2024

1 Introduction

This project explores the application of GPT-4 to analogical reasoning problems. It builds upon a foundational article [1] that evaluates semantic relational similarity by measuring the prototypicality of word pairs within specific semantic relation classes. The primary objective is to assess GPT-4's effectiveness in this domain and to compare its performance with the results presented in the original study [1].

2 Dataset

The dataset consists of graded relational similarity ratings for 3218 word pairs across 79 categories. It was split into training and testing sets, with 10 relation categories designated for training and the remaining 69 for testing. The experiment unfolded in two phases:

Phase 1: Participants, through Amazon Mechanical Turk, generated new word pairs mirroring the relations of given paradigm examples. This phase yielded a diverse collection of word pairs across different relation categories.

Phase 2: These word pairs were evaluated by participants for their prototypicality within their respective categories. The evaluation employed MaxDiff questions [2], prompting participants to identify the most and least representative examples. The outcomes of this phase provided prototypicality scores for each word pair.

3 Experiments

3.1 Our Approach

For this project, we utilized OpenAI's API [3], offering comprehensive research tools while avoiding the limitations of ChatGPT's messaging constraints (a maximum of 50 messages every 3 hours). This approach enabled autonomous analysis. We focused on data from the second phase of the original experiment, specifically the MaxDiff questions involving the selection of the most and least representative word pairs from a set of four. Data was extracted from multiple files, processed, and then input into GPT. The model's responses were recorded and analyzed to determine the accuracy and calculate the MaxDiff scores as well as Spearman's rank correlation coefficient (Spearman's ρ). The detailed methodology is outlined below. The Python code for the steps mentioned can be found at the end of the report.

3.2 First Method

In this method, we clearly stated the relationship's name in the instructions to provide GPT with a direct context for its analysis.

The experiment began by extracting files from the "Testing/Phase2Questions" folder, which contained several .txt files. Each file consisted of about 100 lines, with each line comprising four word pairs, for example:

yell:anger punch:hatred sigh:exhaustion discourse:relationship.

These pairs were used to prompt GPT to identify the most and least illustrative pair. Meanwhile, correct answers were obtained from the "Testing/Phase2Answers" folder. Here, each line from the question files appeared multiple times (5-6 times on average) with three additional columns: "least_illustrative", "most_illustrative", and "user_selected_relation". This was to account for the variations in the choices made by the participants (Turks) in the experiment. To be more precise, for each of the questions several answers from 3-5 people were provided, and answer corresponds to exact line in txt files stored in "Testing/Phase2Answers" folder. The frequency of these choices determined the correct answer.

After gathering all necessary information (pairs and relationships), we prepared to pose our questions to GPT. To ensure uniformity and facilitate later parsing, we crafted specific instructions and standardized the output format. Following various trials with different prompts, we selected the most effective one to use for all files. All lines were then processed through the model, and the responses were meticulously recorded. Our formatting approach closely resembled the dataset's structure, maintaining a consistent style for easy comparison and analysis. For example, a standard line from the dataset might be formatted as:

yell:anger, punch:hatred, sigh:exhaustion, discourse:relationship, discourse:relationship, yell:anger

where the order is the following: *pair1, pair2, pair3, pair4, least_illustrative, most_illustrative.* GPT's response, reflecting its selection of the least and most illustrative pairs, could be formatted similarly:

yell:anger, punch:hatred, sigh:exhaustion, discourse:relationship, punch:hatred, sigh:exhaustion

This consistent formatting was key to efficiently comparing GPT's selections with the dataset, thereby enabling a thorough evaluation of its performance.

3.3 Second Method

Rather than stating the relationship's name, this method presented the number of pairs sharing the relationship, challenging the model to infer the relationship based on this information.

In this case, similar steps were conducted for parsing but different instructions were given to the model. We aimed to ensure that we control the GPT's ability to reason analogically from multiple prospective and in this method, by giving the examples we wanted to give the model a bit more examplish context and not the direct one like before.

This method better complies with the steps performed in original study [1]. During the second phase, the participants were given four examples of relations instead of the names of relations. We assume that the models used in experiment utilized examples of relations for making prediction as well. Thus, the second method is more appropriate for obtaining fair results and conducting comparative analysis with results provided by models in the original paper [1].

3.4 Prompt engineering

In our work, we sought a prompt that would not only solve the task at hand but also yield an output in a specific format crucial for subsequent data parsing.

We trialed multiple prompts, with a few examples listed below:

- *For each line, output the least illustrative and the most illustrative representation of this relation: '+* ***relation*** *+ '. The output should be two pairs: least illustrative and most illustrative.*

  This prompt is clear but lacked details on the output's presentation. We decided to articulate these specifics.

- *I'm going to give you several lines of the same type. Your task is for each line to output the least illustrative and the most illustrative representation of this relation: ' +* ***relation*** *+ ' (the order is important here!). So, the output should be multiple lines with 2 pairs: least illustrative and most illustrative. Output only this information without any other comments.*

  While this prompt is comprehensive, it needed clarity regarding the output structure.

- *In this line, based on the pairs provided, choose among them the least illustrative and the most illustrative representation for this relation: '+* ***relation*** *+ ' (the order of the relation matters). The output should be these four pairs and the least illustrative and the most illustrative as the 5th and 6th column, accordingly. The output should be written in one line, 6 pairs overall in the following format: pair1, pair2, pair3, pair4, least_illustrative, most_illustrative And that's it, no brackets, no quotes, nothing else, it must be in this format.*

  This version proved most effective, offering comprehensive context and explicit output requirements, essential due to GPT's formatting tendencies. This became our standard query format.

- For the second method, we adjusted the instructions to use example pairs instead of explicitly naming the relation:

  In this line, based on the pairs provided, choose among them the least illustrative and the most illustrative representation for the kind of relation demonstrated by these examples: + ***pairs[file_key]*** *+ . The output should be these four pairs and the least illustrative and the most illustrative as the 5th and 6th column, accordingly. The output should be written in one line, 6 pairs overall in the following format: pair1, pair2, pair3, pair4, least_illustrative, most_illustrative. And that's it, no brackets, no quotes, nothing else, it must be in this format. Do not include examples of the relation in your output! They just provide you examples of a certain relation and based on this relation (that you infer from these examples) you need to choose the least and most illustrative among the following 4 pairs:*

3.5 Challenges Encountered

Throughout our research, we encountered several challenges:

- Cost of Tokens: The utilization of the new and computationally intensive GPT-4 model incurs significant expense. For our study, we processed 8 random files from the 69 provided in the test set, which cost approximately 7 euros. A comprehensive analysis of all files would have projected costs between 60 and 70 euros. Nevertheless, the 10% sample of the dataset yielded sufficiently representative data, enabling us to draw meaningful comparisons with the article's [1] findings.

- Fine-tuning GPT-4: undoubtedly, the utilization of training dataset for fine-tuning GPT-4 could bring us better results. Moreover, it would allow us fully replicate all the steps performed in original study [1]. However, we decided to skip this step for following reasons. First, we deal with pretrained large language model (LLM) aimed to provide solutions for various tasks, meanwhile the models used in original paper [1] were trained from a scratch. Based on this, we can argue that our model is already trained. Second, fine-tuning GPT-4 creates additional cost. Third, GPT-4 provides good results without fine-tuning (see Results).

- Imperfect Outputs: Despite the efficacy of our optimized prompt, GPT-4's responses occasionally included extraneous quotations or inserted column names directly into the output, such as

  "least_illustrative: "punch:hatred"",
  "most_illustrative": "sigh:exhaustion"

  instead of just

  "punch:hatred", "sigh:exhaustion".

  These anomalies disrupted the parsing process and necessitated additional manual data cleaning.

- Batch Processing Trials: We explored processing data in batches (about 20 lines at once) to streamline output formatting. However, this approach conflated the lines, leading GPT-4 to mistakenly treat 40 pairs as if they were part of a single line, resulting in incorrect pair selections. Consequently, we reverted to processing single lines individually to maintain data integrity and avoid such complications.

4 Evaluation

As mentioned above, MaxDiff score and Spearman's ρ are two main criteria for measure LLM's analogical reasoning performance. Each of these metrics is calculated separately for each subcategory. As a final evaluation, we take average value across all subcategories.

The procedure of calculating MaxDiff score conducted as follows. First, we take MaxDiff questions belonging to current subcategory and answers given by participants; for each question we compute mapping P_l between pairs and how many times they were selected as the least illustrative as well as mapping P_m between pairs and how many times they were selected as the most illustrative. For each of obtained mappings we extract the largest frequency (amount of time) N_l and P_m respectively. After this, for each answer provided by GPT-4 we frequency V_l from P_l by the key which is given the least illustrative pair from the answer and frequency V_m from P_m by the key which is given the most illustrative pair from the answer. If V_l equals N_l, we set variables N_l^r equal to one and N_l^w equal to zero. Otherwise, we set variables N_l^r equal to zero and N_l^w equal to one. Following the same logic, we set variables N_m^r and N_m^w operating with V_m and N_m. Finally, we compute MaxDiff score by following formula,

$$MaxDiff = 100 * \frac{\sum_{i=1}^{Q} N_m^r(i) + \sum_{i=1}^{Q} N_l^r(i)}{2 * Q}$$

where Q is number of questions in the subcategory.

The calculation of Spearman's ρ is based on computing ratings for each pair in selected subcategory. Two sets of rating are computed: one based on answers provided by participants, and other based on answers given by GPT-4. The procedure of calculating ratings conducted as follows. For each pair in the subcategory, we compute mapping P_l between pair and how many times they were selected as the least illustrative example, but this time we compute it over all questions within subcategory unlike we did it while computing MaxDiff score, where we computed similar mapping for each question separately. Following the same logic, we compute P_m and mapping P_q between the pairs and number of questions they present in as an option. The ratings for each pair are calculated by following formula.

$$Rank(pair) = 100 * (\frac{P_m(pair) - P_l(pair)}{P_q})$$

Given two rank sets, we can compute Spearman's rank correlation coefficient ρ between them.

5 Results

5.1 The frequencies of correct answers

The overall results showed an interesting pattern, that might not be surely expected.

In the tables bellow we show the frequencies of correct answers for each category represented by column "ID". Table 1 and Table 2 report results of the first and the second methods respectively.

ID	Least Illustrative	Most Illustrative
1c	51 out of 105	25 out of 105
2a	46 out of 110	44 out of 110
2g	51 out of 108	57 out of 108
4b	26 out of 88	36 out of 88
4d	40 out of 75	43 out of 75
6d	34 out of 113	45 out of 113
10b	45 out of 100	32 out of 100

Table 1: The frequencies of correct answers for each subcategory obtained by the first method.

ID	Least Illustrative	Most Illustrative
1c	49 out of 105	42 out of 105
2a	46 out of 110	37 out of 110
2g	42 out of 108	41 out of 108
4b	26 out of 88	36 out of 88
4d	45 out of 75	40 out of 75
6d	37 out of 113	30 out of 113
10b	50 out of 100	32 out of 100

Table 2: The frequencies of correct answers for each category obtained by the second method.

Further we are going to investigate in detail the results obtained by using the second method.

5.2 Average Spearman's rank correlation coefficients and MaxDiff scores

In order to asses GPT-4 performance and compare it to models mentioned in original study [1], we computed Average Spearman's rank correlation coefficient and MaxDiff score for GPT-4 answers as well as number of subcategories with statistically significant Spearman's rank correlation coefficient at different levels of confidence. The metrics are shown in Table 3 which also contains an information about the scores of previous models. We can conclude that GPT-4 outperforms all previous models and baselines. Obtained average Spearman's ρ and average MaxDiff score are significantly larger than other models' metrics. In addition, columns 6 and 7 in Table 3 show, that the majority of tested subcategories exhibits significant correlation between GPT-4 answers and the gold standard ranking.

5.3 The frequencies of correct answers

The overall results showed an interesting pattern, that might not be surely expected.

In the tables bellow we show the frequencies of correct answers for each subcategory represented by column "ID". Table 1 and Table 2 report results of the first and the second methods respectively.

Team	System	Spearman's ρ	# of Subcategories		Ratio of Subcategories		MaxDiff
			p <0.05	p <0.01	p <0.05	p <0.01	
BUAP	BUAP	0.014	2	0	0.0290	0	31.7
UTD	NB	0.229	22	16	0.3189	0.2319	39.4
	SVM	0.116	11	5	0.1594	0.0725	34.7
Duluth	V0	0.050	9	3	0.1304	0.0435	32.4
	V1	0.039	10	4	0.1449	0.0580	31.5
	V2	0.038	7	3	0.1014	0.0435	31.1
Group 9	GPT-4	**0.550**	7	6	0.8750	0.750	**49.27**
Baselines	Random	0.018	4	0	0.0580	0	31.2
	PMI	0.112	15	7	0.2174	0.1014	33.9

Table 3: Performance of GPT-4 compared to previous models. This table is based on Table 3 in the original study [1]. We complimented this table with ration of subcategories with statistically significant Spearman's rank correlation coefficient at different levels of confidence (columns 6 and 7).

5.4 Performance across categories

In this subsection, we investigate GPT-4 performance across different categories compare it to result shown in original study [1] (Table 4). GPT-4 provides the highest Spearman's ρ among all models for majority of categories. However, GPT-4 fails to outperform PMI baseline .

Relation Class	Random	PMI	BUAP	UTD-NB	UTD-SVM	Duluth-V0	Duluth-V1	Duluth-V2	GPT-4
Class-Inclusion	0.057	0.221	0.064	0.233	0.093	0.045	0.178	0.168	**0.667**
Part-Whole	0.012	0.144	0.066	0.252	0.142	-0.061	-0.084	-0.054	**0.564**
Contrast	-0.049	0.032	0.000	0.206	0.162	0.142	0.120	0.051	**0.577**
Non-Attribute	-0.070	**0.191**	0.009	0.098	0.094	0.079	0.066	0.074	0.128
Case Relations	0.090	0.168	-0.037	0.241	0.187	-0.011	-0.068	-0.115	**0.584**
Reference	0.142	0.125	-0.001	0.346	0.082	0.028	0.074	0.067	**0.739**

Table 4: GPT-4 performance across different categories compared to previous models. This table is based on Table 4 in the original study [1].

5.5 Resilience to reversals

Authors of original paper [1] proposed a way to asses models' ability to be resilient to reversals. For most categories, reversing the order of two words in a pair relevant to this category makes it lose the relevance. This is the reason why the words' order should be taken into account by the models. In order to check this ability, authors took five pairs from each subcategory and sampled a new pairs from the reversed ones. Each model was evaluated by comparing Spearman's ρ calculated on whole dataset and dataset with removed reversals. We conducted the same procedure and present the results in Table 5. We can observe that adding reversals to dataset doesn't decrease score of GPT-4. Moreover, removing reversals slightly decreases the score similarly as for BUAP, NB, SVM. Authors suggest that this is due to the fact that it's easy to identify reversals as least illustrative examples for the models which utilize order information.

Team	System	Spearman's ρ	
		No Reversals	With Reversals
BUAP	BUAP	-0.003	0.014
UTD	NB	0.190	0.229
	SVM	0.104	0.116
Duluth	V0	0.062	0.050
	V1	0.040	0.039
	V2	0.046	0.038
Group 9	GPT-4	0.540	0.550
Baselines	Random	0.004	0.018
	PMI	0.143	0.112

Table 5: GPT-4 resilience to reversals compared to previous models. This table is based on Table 5 of the original study [1].

Since GPT-4 identifies reversals, we can evaluate its ability to correctly order them and compare it with other models. It could be computed as root mean square error (RMSE) between pairs' rank and their reversals' rank in the gold standard. The results are demonstrated in Table 6. RSME of GPT-4 is significantly smaller than errors of other models. Thus, GPT-4 provides the best way to prototypicality order reversals.

Team	System	RMSE
BUAP	BUAP	256.07
UTD	NB	257.15
	SVM	209.95
Group 9	GPT-4	**59.16**
Baseline	Random	227.25

Table 6: GPT-4 ability to correctly order reversals compared to previous models. This table is based on Table 6 of the original study [1].

6 Conclusions

In this project, we explored the performance of modern state of the art large language model GPT-4 in the context of solving analogical reasoning problem. In order to do this, we conducted step by step procedure described in original paper [1] and compared results with ones obtained in the study [1]. According to various qualitative measures, GPT-4 significantly outperforms all previous model utilized in original paper [1]. Moreover, GPT-4 exploits words' order information for making prediction and correctly weights reversals' prototypicality.

References

[1] David A. Jurgens, Saif M. Mohammad, Peter D. Turney, and Keith J. Holyoak. 2012. SemEval-2012 Task 2: Measuring Degrees of Relational Similarity. In First Joint Conference on Lexical and Computational Semantics (*SEM), pages 356–364. Association for Computational Linguistics.

[2] Jordan J. Louviere. 1991. Best-worst scaling: A model for the largest difference judgments. Working Paper.

[3] Kanaries. 2023. An Advanced Guide: How To Use ChatGPT API In Python. https://docs.kanaries.net/topics/ChatGPT/how-to-use-chatgpt-api-python

Appendix

Here we show an example of Python code that was written for sending requests to GPT-4.

This script installs the OpenAI library, sets an API key for OpenAI, and defines a function to process files and query GPT-4. For each file in the input folder, it reads tab-separated values, sends the content along with specific instructions to the GPT-4 API, and saves the responses into a new file in the output folder. The instructions guide GPT-4 to choose the least and most illustrative pairs from the provided data.

```
def process_files_and_query_gpt(input_folder, output_folder, api_key):
    openai.api_key = api_key  # Set the API key for OpenAI

    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    for file in os.listdir(input_folder):
        with open(os.path.join(input_folder, file), ’r’) as infile:
            reader = csv.reader(infile, delimiter=’\t’)
            relation = next(reader, [])[0]  # First row for the relation

            # Update instructions including the relation
            instructions = ("For each line, output the least illustrative "
                            "and the most illustrative representation of this relation: ’"
                            + relation + "’. The output should be two pairs: "
                            "least illustrative and most illustrative.")

            responses = []

            for pairs in reader:
                # Prepare the message for API call, including the instructions and the line
                message = [{"role": "system", "content": instructions},
                           {"role": "user", "content": " ".join(pairs)}]

                # Make API calls for each line
                chat_completion = openai.ChatCompletion.create(
                    model="gpt-4",
                    messages=message
                )
                # Extracting the assistant’s response
                assistant_message = chat_completion[’choices’][0][’message’]
                if assistant_message[’role’] == ’assistant’:
                    responses.append(assistant_message[’content’])

            # Write to new file
            with open(os.path.join(output_folder, file.replace(’.txt’, ’_gpt.txt’)), ’w’, newline=’’)
            as outfile:
                writer = csv.writer(outfile, delimiter=’\t’)
                writer.writerow([relation])
                for response in responses:
                    writer.writerow([response])

# Example usage
input_folder = ’/content/output_no_least_most’
output_folder = ’/content/output_gpt’
api_key = ’’  # Replace with your actual API key
process_files_and_query_gpt(input_folder, output_folder, api_key)
```