Supplementary Materials

S1 Code Availability

Code for all simulations can be downloaded from:

https://github.com/taylorwwebb/counterfactual_analogies

Most code was written in Python v3.11.4, using the following packages: NumPy v1.25.0 [1], SciPy v1.11.1 [2], statsmodels v0.14.0 [3], and Matplotlib v3.7.1 [4]. Logistic regression analyses were carried out in R v3.6.3 [5]. Experimental stimuli for human behavioral experiments were written in JavaScript using jsPsych v7.2.1 [6].

S2 Data Availability

Data for the human behavioral experiment and GPT-4 evaluations, along with the letter string problem sets, can be downloaded from:

https://github.com/taylorwwebb/counterfactual_analogies

This repository has been assigned the following DOI/accession number:

https://zenodo.org/records/14047041

S3 Problem Set

Letter string analogy problems were created using the following permuted alphabet, also used by Hodel & West [7]:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Each letter string analogy problem involved one of six transformation types: sequence extension, successor, predecessor, removing a redundant letter, fixing a sequence, and sorting. These are the same transformation types used in our original study [8]. In the sequence extension transformation, the source involved an ordered sequence of four letters (based on the ordering in the permuted alphabet) followed by an extension of this sequence involving five letters (including the next letter in the permuted alphabet), as in the following example:

[x y l k] [x y l k w]

In the successor transformation, the source involved an ordered sequence of four letters, followed by that same sequence, but with the final letter replaced by its successor in the permuted alphabet, as in the following example:

[x y l k] [x y l w]

In the predecessor transformation, the source involved an ordered sequence of four letters, followed by that same sequence, but with the first letter replaced by its predecessor in the permuted alphabet, as in the following example:

[y l k w] [x l k w]

In the transformation involving removal of a redundant letter, the source involved an ordered sequence of five letters with one letter repeated, followed by that same sequence with the redundant letter removed, as in the following example:

[x y y l k w] [x y l k w]

In the transformation involving fixing a sequence, the source involved an ordered sequence of five letters with one out-of-place letter (violating the ordering in the permuted alphabet), followed by that same sequence with the out-of-place letter replaced, as in the following example:

[x y l g w] [x y l k w]

In the sorting transformation, the source involved an ordered sequence of five letters with the position of two letters swapped, followed by a sorted version of the same sequence (sorted based on the ordering in the permuted alphabet), as in the following example:

[x k l y w] [x y l k w]

Similar to Hodel & West [7], we only investigated problems referred to as 'zero-generalization' in our original study, meaning that the only difference between source and target sequences was the use of different letters. An example problem (involving a successor relation) is presented below:

[x y l k] [x y l w]
[j r q a] [j r q h]

We investigated problems involving an interval size of either 1 or 2. The problems with an interval size of 1 are exactly as described above. The interval size of 2 was employed in different ways, depending on the transformation type. In the sequence extension transformation, the first sequence was presented as usual, but the extension of that sequence involved the addition of a letter separated by a successor interval of 2 in the permuted alphabet, as in the following example:

[x y l k] [x y l k b]

Similarly, in the successor transformation, the final letter in the first sequence was replaced by a letter separated by a successor interval of 2, as in the following example:

[x y l k] [x y l b]

In the predecessor transformation, the first letter in the sequence was replaced by a letter that preceded it by an interval of 2, as in the following example:

[l k w b] [x k w b]

In the transformation involving removal of a redundant letter, fixing a sequence, and sorting, the source involved an ordered sequence of five letters, in which each letter was separated by an interval of 2 in the permuted alphabet. The following example illustrates this for the transformation involving removal of a redundant letter:

[x l l w f t] [x l w f t]

The following example illustrates this for the transformation involving fixing a sequence:

[x l w g t] [x l w f t]

The following example illustrates this for the sorting transformation:

[x f w l t] [x l w f t]

Problems with an interval of size 2 were the same as those used by Hodel & West [7], and problems with an interval size of 1 were generated using code from HW, both downloaded from the following repository:

https://github.com/hodeld/emergent_analogies_LLM_fork

There were 600 problems with an interval size of 1, and 600 problems with an interval size of 2 (each involving 100 problems with each transformation type). The same problem sets were used for testing GPT-4 and human participants.

S4 Evaluating GPT-4

We queried GPT-4 in an automated fashion using the OpenAI Chat Completions API. Experiments were performed using the 'gpt-4-0125-preview' engine, with temperature and top-p set both set to 0. Prompts and analogy problems were presented using the 'User' role, with the default message from the 'System' role ('You are a helpful assistant.'). The prompt format that we used is illustrated in the following example:

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[x y l k] [x y l w]
[j r q a] [?]

Please only provide the answer. Do not provide any additional explanation.

Answer:

This prompt was employed to ensure that responses adhered to a regular format, thus avoiding the difficulty involved in parsing the verbose responses that GPT-4 typically provides. Line breaks were indicated using the '\n' character. We evaluated GPT-4 on 300 problems with an interval size of 1, and 300 problems with an interval size of 2 (each involving 50 problems with each transformation type).

S5 Evaluating GPT-4 with code execution

We also evaluated a variant of GPT-4 that was augmented with a capacity to write and execute code. These experiments were performed with the OpenAI Assistants API, using the same engine ('gpt-4-0125-preview') as the experiment without code execution. Temperature and top-p were both set to 0. The prompt format that we used is illustrated in the following example:

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[x y l k] [x y l w]
[j r q a] [?]

This is the only prompt that was provided to GPT-4. We did not provide any specific instructions to use code execution to solve the problem, nor did we indicate the importance of the position or interval size between letters in the permuted alphabet. GPT-4 typically responded with an extended 'chain-of-thought' response [9], often invoking the code execution function. Code execution was typically invoked to identify the corresponding position of the letters in the permuted alphabet.

GPT-4 sometimes refused to provide an answer, insisting that there was too much uncertainty about the pattern underlying the analogy problem. In these cases, GPT-4's response was re-generated until it provided an answer. When GPT-4 did provide an answer, this was counted as final, even if GPT-4 expressed uncertainty.

GPT-4's chain-of-thought responses typically had a highly variable format, making it difficult to automate evaluation. We therefore manually inspected the responses to determine accuracy. Due to the time involved in manually assessing these responses, these experiments involved a smaller subset of 60 problems with an interval size of 1, and 60 problems with an interval size of 2 (each involving 10 problems with each transformation type). GPT-4's complete response, including any code generated, was recorded.

S6 Evaluating GPT-4 with zero-shot chain-of-thought

We also evaluated GPT-4 while allowing it to perform zero-shot chain-of-thought. Chain-of-thought refers to the use of a language model's autoregressive outputs to perform intermediate reasoning steps [9]. Although this approach originally required prompting with carefully crafted examples, more recent approaches involve prompting to elicit zero-shot chain-of-thought [10], and language models are now trained to employ this technique by default. Thus, in order to enable zero-shot chain-of-thought reasoning for our task, we simply removed the part of the prompt that instructed the model to provide only the answer (i.e., the prompt was the same as it was for the code execution model).

These experiments used the same GPT-4 engine ('gpt-4-0125-preview') and the same hyperparameters (temperature and top-p set to 0). Because the chain-of-thought responses were highly variable, we used the same evaluation metholodology as the one used for the code execution model, manually evaluating each response. This evaluation also involved 60 problems with an interval size of 1, and 60 problems with an interval size of 2.

S7 Code execution control model

There is some possibility that the code execution model employs auxiliary prompts (hidden from the user) intended to trigger code execution. To determine whether these prompts themselves impact the performance of the model, we performed a test in which the model was explicitly instructed not to invoke code execution. This allows us to test the effects of the auxiliary prompts in the absence of code execution. The prompt used is shown below:

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[x y l k] [x y l w]
[j r q a] [?]

Please only provide the answer. Do not provide any additional explanation, and do not use the code interpreter.

Answer:

The model used the same GPT-4 engine and hyperparameter settings as the other experiments. When prompted in this manner, the model did not evoke code execution on any trials. The model was evaluated on 300 problems with an interval size of 1, and 300 problems with an interval size of 2.

S8 Counterfactual comprehension check

We performed an experiment to test the ability of GPT-4 to perform indexing. We presented the synthetic alphabet, followed by two letters with an interval of -2, -1, $+1$, or $+2$, and asked GPT-4 to indicate the interval between the two letters. The model was evaluated on 100 problems for each interval size. An example prompt is shown below:

Consider the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

In this alphabet, what is the interval (distance and direction) from f to t?

Please respond only with +/- followed by an integer (e.g., +1, -1, +2, -2).

Answer:

S9 Human Behavioral Experiment

Human behavioral data was collected in an online experiment. The experiment was approved by the UCLA Institutional Review Board (IRB protocol #22-000841, approved May 17, 2022), and all participants provided informed consent. Participants were recruited through the Prolific platform. Ninety-nine participants (32 female, 19-60 years old, average age = 29 years old) completed the experiment in the interval-size-1 condition, and ninety-seven participants (40 female, 18-54 years old, average age = 29 years old) completed the experiment in the interval-size-2 condition. All participants spoke fluent English and had normal or corrected-to-normal vision. No statistical methods were used to pre-determine sample sizes. Participants were paid $3.00 to participate, and the experiment took a maximum of 10 minutes to complete.

Participants were first presented with a set of instructions, and the following example problem (not involving the permuted alphabet or any of the transformations employed in the actual experiment):

[a a a] [b b b]
[c c c] [?]

Each participant completed 6 problems, including 1 problem for each transformation type. The specific problem instances were randomly sampled for each participant, and participants received these problems in a random order. Participants generated a free response for each problem. Participants performed the experiment in full-screen mode, and were instructed to turn off any music or other potential sources of distraction. After completing the problems, participants completed an attention check involving a simple categorical judgment (they were presented with a list of items and asked to identify which one was categorized as a vegetable). All participants successfully completed the attention check.

S10 Statistical analyses

Logistic regression analyses were carried out at the individual trial level, with each data point corresponding to a particular trial from a particular participant (or GPT-4). The dependent variable in all regression analyses was a binary variable coding for whether a particular response was correct or incorrect.

One regression analysis compared human participants with GPT-4. This analysis included three predictors: a binary predictor coding for interval size (1 vs. 2), a binary predictor coding for GPT-4 vs. human participants, and an interaction term. A second regression analysis compared human participants with GPT-4 + code execution, using the same predictors.

S11 Example GPT-4 response involving code execution

The following example illustrates the behavior of GPT-4 when augmented with the ability to execute code. This example is in response to the following analogy problem:

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[q a h v] [q a h m]
[k w b f] [?]

This problem involves a successor relation with an interval size of 2. The correct answer is [k w b t]. GPT-4's response is as follows (typewriter font indicates code generated and executed by GPT-4):

> This puzzle appears to revolve around matching sequences of characters from the provided fictional alphabet. To solve the puzzle, we'll need to determine the relationship between the given sequences and then apply that relationship to find the missing sequence. Let's

identify the pattern or rule that connects the first pair of sequences ([q a h v] and [q a h m]) and then use this pattern to solve for the missing sequence corresponding to [k w b f].

Let's go through the fictional alphabet and the given sequences to identify the pattern.

Looking at the sequences:

1. ([q a h v]) changes to ([q a h m]).

By examining the given fictional alphabet, let's find the positioning of each character and identify the transformation applied from the first sequence to the second:

Fictional Alphabet: ([x y l k w b f z t n j r q a h v g m u o p d i c s e])

Let's start by determining the index of each character in the first sequence and then see how it transforms into the second sequence.

```
# Fictional alphabet as a list of characters for indexing purposes
fictional_alphabet = ['x', 'y', 'l', 'k', 'w', 'b', 'f', 'z', 't', 'n', 'j',
'r', 'q', 'a', 'h', 'v', 'g', 'm', 'u', 'o', 'p', 'd', 'i', 'c', 's', 'e']

# Given sequences
sequence1 = ['q', 'a', 'h', 'v']
sequence2 = ['q', 'a', 'h', 'm']

# Let's find the indexes of each character in both sequences
sequence1_indexes = [fictional_alphabet.index(char) for char in sequence1]
sequence2_indexes = [fictional_alphabet.index(char) for char in sequence2]

sequence1_indexes, sequence2_indexes
```

The indexes of the characters in the first sequence ([q a h v]) are ([12, 13, 14, 15]) and in the second sequence ([q a h m]) are ([12, 13, 14, 17]).

Observing the transformation from the first sequence to the second, it looks like all but the last item remain the same. The last character changes from having an index of (15) to an index of (17).

Now let's apply this observed transformation rule to the sequence ([k w b f]) and find the corresponding missing sequence. We will maintain the first three characters and find the character two positions after the last character in the fictional alphabet.

```
# Sequence to transform
sequence_to_transform = ['k', 'w', 'b', 'f']

# Getting indexes of the characters in the sequence to transform
transform_indexes = [fictional_alphabet.index(char) for char in sequence_to_transform]

# Change the last character as per observed transformation:  2 positions after
the original
new_last_char_index = (transform_indexes[-1] + 2) % len(fictional_alphabet) #
Using modulo in case of overflow

# create the new sequence with the transformation applied
new_sequence = sequence_to_transform[:3] + [fictional_alphabet[new_last_char_index]]

new_sequence
```

The missing sequence corresponding to ([k w b f]) after applying the observed transformation rule is ([k w b t]).

So the solution to the puzzle is:

- ([q a h v]) transforms to ([q a h m])
- ([k w b f]) transforms to ([k w b t])

A number of observations can be made about this response. First, GPT-4 spontaneously uses chain-of-thought reasoning [9] to decompose the problem, developing a plan that involves first identifying the change between the first and second sequences, and then applying that change to the third sequence. GPT-4 then uses code execution to convert the letters in the source sequences to their indices in the fictional alphabet, and identifies the rule based on these indices (that the first three letters stay the same, while the last letter is incremented by an interval of 2). Finally, GPT-4 again uses code execution to apply this rule to the target sequence and generate an answer. GPT-4's responses typically involved a similar combination of chain-of-thought reasoning and code execution, and correct answers were always accompanied by an accurate explanation of the underlying rule. Incorrect answers were often accompanied by an analysis that identified a valid alternative rule. The complete set of responses is available at:

https://github.com/taylorwwebb/counterfactual_analogies/GPT4_analysis_responses

S12 Analysis of errors based on alternate rules

For some problem types, it is possible to define an alternative rule that validly describes the problems. These alternative rules are typically less abstract than the intended rule. For example, consider the following example problem, based on a sort rule:

Fictional alphabet: [x y l k w b f z t n j r q a h v g m u o p d i c s e].

[x k l y w] [x y l k w]
[h r q a j] [?]

In this problem, the second sequence ([x y l k w]) can be obtained by sorting the first sequence ([x k l y w]) according to the ordering in the permuted alphabet. The correct answer is [j r q a h], which is obtained by applying this rule to the third sequence ([h r q a j]). However, an alternative interpretation is that the rule involves swapping the second and fourth entries in a sequence. Applying this alternative rule yields the answer [h a q r j].

We quantified the number of errors made by GPT-4 + code execution that could be classified as involving a valid alternative rule. Errors based on alternative rule use were especially common for problems involving fixing an alphabetic sequence or sorting. Altogether, errors based on alternative rules constituted 38% (18/47) of GPT-4's errors, consistent with the rate observed in the human behavioral data from Lewis & Mitchell [11]. This result suggests that, in addition to improving overall performance, the ability to count using code execution also leads GPT-4 to make more human-like errors.

Supplementary References

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: `10.1038/s41586-020-2649-2`. URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[2] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `10.1038/s41592-019-0686-2`.

[3] Skipper Seabold and Josef Perktold. "statsmodels: Econometric and statistical modeling with python". In: *9th Python in Science Conference*. 2010.

[4] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/MCSE.2007.55`.

[5] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: `https://www.R-project.org/`.

[6] Joshua R De Leeuw. "jsPsych: A JavaScript library for creating behavioral experiments in a Web browser". In: *Behavior research methods* 47.1 (2015), pp. 1–12.

[7] Damian Hodel and Jevin West. "Response: Emergent analogical reasoning in large language models". In: *arXiv preprint arXiv:2308.16118* (2023).

[8] Taylor Webb, Keith J Holyoak, and Hongjing Lu. "Emergent analogical reasoning in large language models". In: *Nature Human Behaviour* 7.9 (2023), pp. 1526–1541.

[9] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[10] Takeshi Kojima et al. "Large language models are zero-shot reasoners". In: *Advances in neural information processing systems* 35 (2022), pp. 22199–22213.

[11] Martha Lewis and Melanie Mitchell. "Using counterfactual tasks to evaluate the generality of analogical reasoning in large language models". In: *arXiv preprint arXiv:2402.08955* (2024).