# A Compositional Tasks

## A.1 Multiplication

**Data Construction** We exhaustively generate multiplication problems as question-answer pairs (e.g., Q: “What is 4 times 32?” A: “128”). We focus on multiplications of two numbers $x = (x_1, x_2, \ldots, x_k)$ and $y = (y_1, y_2, \ldots, y_k)$ where each number can have up to $k$ digits, amounting to $9 \times 10^{(k-1)}$ combinations per each number. We set $k$ to 5 in our experiments. Figure 7 showcases an example prompt for performing few-shot learning without the inclusion of a scratchpad, while Figure 8 demonstrates an example prompt using a scratchpad. Throughout our experimentation, we explored various versions of the scratchpad, ranging from verbose and detailed to more concise alternatives. Among these variations, the scratchpad version depicted in Figure 8 ultimately produced the most favorable outcomes. Listing 1 shows the Python code for solving the task.

```
To multiply two numbers, start by multiplying the rightmost digit of the
multiplicand by each digit of the multiplier, writing down the products and
carrying over any remainders.  Repeat this process for each digit of the
multiplicand, and then add up all the partial products to obtain the final
result.

Questions: what's 22 times 2? Answer 44.
```

Figure 7: Example prompt for the multiplication task used for the few-shot setting.

```
Question: What is 35 times 90?


Scratchpad: Let's perform the multiplication step by step:

Let's multiply 35 by the digit in the ones place of 90, which is 0.

1. Multiply 0 by the digit in the ones place of 35, which is 5. This gives 5 x 0
= 0. Write down the result 0.
2. Multiply 0 by the digit in the tens place of 35, which is 3. This gives 3 x 0
= 0. Write down the result 0.
3. The partial product for this step is A=0 which is the concatenation of the
digits we found in each step.

Now, let's multiply 35 by the digit in the tens place of 90, which is 9.\n\n4.
Multiply 9 by the digit in the ones place of 35, which is 5. This gives 5 x 9 =
45. Write down the result 5 and carry over the 4 to the next step.

5. Multiply 9 by the digit in the tens place of 35, which is 3. Add the carryover
from the previous step to account for this. This gives (3 x 9) + 4 = 31. Write
down the result 31.
6. The partial product for this step is B=315 which is the concatenation of the
digits we found in each step.

Now, let's sum the 2 partial products A and B, and take into account the position
of each digit: A=0 (from multiplication by 0) and B=315 (from multiplication by 9
but shifted one place to the left, so it becomes 3150). The final answer is 0 x 1
+ 315 x 10 = 0 + 3150 = 3150.
```

Figure 8: A sample scratchpad for the multiplication task.

```python
def multiply(x, y):
    summands = [0] * len(y)
    for i in range(len(y) - 1, -1, -1):
```

```python
        digits = [0] * len(x)
        carry = 0
        for j in range(len(x) - 1, -1, -1):
            t = x[j] * y[i]
            t += carry
            carry = t // 10
            digits[j] = t % 10
        digits.insert(0, carry)
        summands[i] = sum(digits[-k] * (10 ** (k - 1)) for k in range(1, len(digits) + 1))

    product = sum(summands[-i] * (10 ** (i - 1)) for i in range(1, len(y) + 1))
    return product
```

Listing 1: Example Python code for solving the multiplication task.

### A.2 Einstein's Puzzle

**Data Construction** In our experiments, we initially establish a set of properties, such as Color, PhoneModel, Pet, and so forth, along with their corresponding values expressed in natural language templates (e.g., "The house has a red color."). We then devise a fundamental and straightforward set of clue types: 1) 'found_at', e.g., "Alice lives in House 2", 2) 'same_house', e.g., "The person who is a cat lover lives in the house that has a red color.", 3) 'direct_left', e.g., "The person who has a dog as a pet lives to the left of the person who lives in a red house.", and 4) 'besides', e.g., "The person who has a dog as a pet and the person who has a red house live next to each other." In addition, we also set up harder clue types such as 'not_at', 'left_of' (not necessarily directly left of), 'two_house_between', etc. which are only used in auxiliary experiments.

The solution to the puzzle is a matrix of size $K \times M$, where $K$ represents the number of houses and $M$ the number of attributes. During the puzzle generation, the $M$ properties are randomly selected from the candidate pool, followed by the random sampling of $K$ values for each property. The sampled values are then randomly permuted and assigned within the table to create the solution. It is important to note that we ensure one of the sampled properties is 'Name' to enhance the readability and comprehensibility of the puzzles. To construct the clues, we initially over-generate all valid clues based on the solution and subsequently remove redundant clues at random until we obtain a set with a

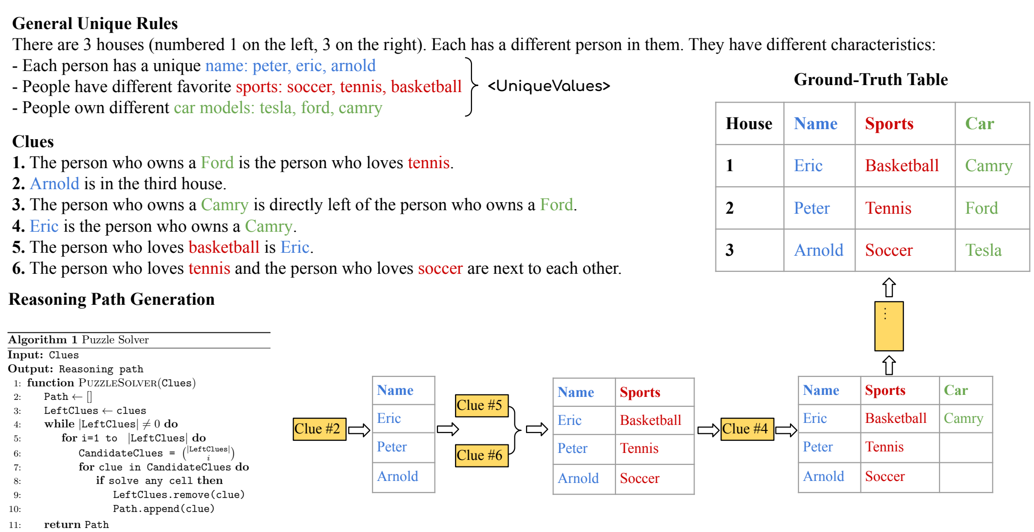


Figure 9: A sample of the puzzle task and the reasoning path to reach a solution.

```
This is a logic puzzle. There are 3 houses (numbered 1 on the left, 3 on the
right). Each has a different person in them. They have different characteristics:
- Each person has a unique name: peter, eric, arnold
- People have different favorite sports: soccer, tennis, basketball
- People own different car models: tesla model 3, ford f150, toyota camry

1. The person who owns a Ford F-150 is the person who loves tennis.
2. Arnold is in the third house.
3. The person who owns a Toyota Camry is directly left of the person who owns a
Ford F-150.
4. Eric is the person who owns a Toyota Camry.
5. The person who loves basketball is Eric.
6. The person who loves tennis and the person who loves soccer are next to each
other.

Let's think step by step. Please first briefly talk about your reasoning and show
your final solution by filling the blanks in the below table.

$ House: ___ $ Name: ___ $ Sports: ___ $ Car: ___
$ House: ___ $ Name: ___ $ Sports: ___ $ Car: ___
$ House: ___ $ Name: ___ $ Sports: ___ $ Car: ___

Reasoning:
Step 1: First apply clue <Arnold is in the third house.> We know that The Name in
house 3 is arnold.
Step 2: Then combine clues: <The person who loves tennis and the person who loves
soccer are next to each other.> <The person who loves basketball is Eric.>
Unique Values Rules and the fixed table structure. We know that The Name in house
1 is eric. The FavoriteSport in house 1 is basketball. The Name in house 2 is
peter.
Step 3: Then apply clue <Eric is the person who owns a Toyota Camry.> We know
that The CarModel in house 1 is toyota camry.
Step 4: Then apply clue <The person who owns a Toyota Camry is directly left of
the person who owns a Ford F-150.> and Unique Values We know that The CarModel in
house 2 is ford f150. The CarModel in house 3 is tesla model 3.
Step 5: Then apply clue <The person who owns a Ford F-150 is the person who loves
tennis.> and Unique Values We know that The FavoriteSport in house 2 is tennis.
The FavoriteSport in house 3 is soccer.
The puzzle is solved.

Final solution:
$ House: 1 $ Name: Eric   $ Sports: Basketball $ Car: Camry
$ House: 2 $ Name: Peter  $ Sports: Tennis     $ Car: Ford
$ House: 3 $ Name: Arnold $ Sports: Soccer     $ Car: Tesla
```

Figure 10: A sample scratchpad for the puzzle task.

unique solution, as previously sampled. This process ensures a coherent and engaging puzzle-solving experience. Refer to Figure 9 for an example.

**Graph Construction Algorithm** To solve the complex compositional reasoning process for a logical grid puzzle, we use existing puzzle solvers [18] to generate the computation graph. It follows the basic greedy principle of applying the minimum number of rules to solve any cell, i.e., if using only one rule to solve any given cell, then apply this rule. This algorithm iterates through all clues in the clue set until one or a set of clue combinations can solve any cell in the table. While it may not be the most efficient way to solve the puzzle, it provides models with explicit scratchpad verbalization through an intuitive computation graph. Refer to Figure 9 for the pseudo-code of the process, and Figure 10 for a scratchpad example.

## A.3 Dynamic Programming Problem

### A.3.1 Solution to this problem

Let $a = [a_1, \ldots, a_n]$ be an input. Let $dp_i$ be the maximum sum of a subsequence that does not include adjacent elements, when considering only the elements of the input from the $i$-th position onwards.

Trivially, $dp_n = \max(a_n, 0)$ since we only want to choose a number if it is non-negative. Moreover, $dp_{n-1} = \max(a_n, a_{n-1}, 0)$ since we cannot choose adjacent numbers.

For any given $dp_i$ with $i \leq n-2$, we can express it in terms of $dp_{i+1}$ and $dp_{i+2}$. Concretely, the maximum sum of a subsequence starting at position $i$ may or may not include the element in the $i$-th position, $a_i$. If the subsequence includes $a_i$, then the maximum sum is $a_i + dp_{i+2}$, since using $a_i$ blocks us from using the next element. If the subsequence does not include $a_i$, then its sum is $dp_{i+1}$. Moreover, the answer may never be less than zero, because otherwise we would select the empty sequence[3]. In summary,

$$dp_i = \max(dp_{i+1},\ a_i + dp_{i+2},\ 0)$$

We now have a recursion with its base cases $dp_n = \max(a_n, 0)$ and $dp_{n-1} = \max(a_n, a_{n-1}, 0)$, and we can therefore compute all values in $O(n)$. It now only rests to reconstruct the lexicographically smallest subsequence that maximizes the desired sum, based solely on the computed $dp$ values.

Starting from $dp_1$ and iterating sequentially through $dp_{n-2}$, we choose an item if and only if $dp_i = a_i + dp_{i+2}$ (that is, the maximum sum comes from choosing the current element) and we have not chosen the previous element. This helps disambiguate cases where choosing or not choosing $a_i$ yields the same sum, but possibly only one of those will not incur in choosing adjacent numbers. Similarly, for positions $i = n-1$ and $i = n$ we choose the element if $dp_i = a_i$ (that is, choosing the element yields the maximum sum) and we have not chosen the immediately previous element. See an example Python solution in 2.

```
Given a sequence of integers, find a subsequence with the highest sum, such that
no two numbers in the subsequence are adjacent in the original sequence.

Output a list with "1" for chosen numbers and "2" for unchosen ones. If multiple
solutions exist, select the lexicographically smallest. input = [3, 2, 1, 5, 2].
```

Figure 11: Example prompt for the DP task, used for zero-shot and few-shot settings.

```python
def maximum_sum_nonadjacent_subsequence(arr):

    N = len(arr)
    dp = [0 for _ in range(N)]

    dp[N - 1] = max(arr[N - 1], 0)
    dp[N - 2] = max(max(arr[N - 1], arr[N - 2]), 0)

    for i in range(N - 3, -1, -1):
        dp[i] = max(max(dp[i + 1], arr[i] + dp[i + 2]), 0)

    # reconstruct the answer with a fixed-size graph
    result = []
    can_use_next_item = True

    for i in range(N - 2):
        if dp[i] == arr[i] + dp[i + 2] and can_use_next_item:
            result.append(1)
            can_use_next_item = False
        else:
```

[3] We don't need to explicitly check for this since $dp_n \geq 0$. However, we include the condition to ease the scratchpad logic.

```python
            result.append(2)
            can_use_next_item = True

    if dp[N - 2] == arr[N - 2] and can_use_next_item:
        result.append(1)
        can_use_next_item = False
    else:
        result.append(2)
        can_use_next_item = True

    if dp[N - 1] == arr[N - 1] and can_use_next_item:
        result.append(1)
    else:
        result.append(2)

    return result
```

Listing 2: Example Python code for solving the DP task. We chose this implementation because the computation graph has always the same topology for any given input length.

```
Question: Let's solve input = [3, 2, 1, 5, 2].


Scratchpad: dp[4] = max(input[4], 0) = max(2, 0) = 2
dp[3] = max(input[3], input[4], 0) = max(5, 2, 0) = 5
dp[2] = max(dp[3], input[2] + dp[4], 0) = max(5, 1 + 2, 0) = 5
dp[1] = max(dp[2], input[1] + dp[3], 0) = max(5, 2 + 5, 0) = 7
dp[0] = max(dp[1], input[0] + dp[2], 0) = max(7, 3 + 5, 0) = 8

Finally, we reconstruct the lexicographically smallest subsequence that fulfills
the task objective by selecting numbers as follows. We store the result on a list
named "output".

Let can_use_next_item = True.
Since dp[0] == input[0] + dp[2] (8 == 3 + 5) and can_use_next_item == True, we
store output[0] = 1. We update can_use_next_item = False.
Since dp[1] != input[1] + dp[3] (7 != 2 + 5) or can_use_next_item == False, we
store output[1] = 2. We update can_use_next_item = True.
Since dp[2] != input[2] + dp[4] (5 != 1 + 2) or can_use_next_item == False, we
store output[2] = 2. We update can_use_next_item = True.
Since dp[3] == input[3] (5 == 5) and can_use_next_item == True, we store
output[3] = 1. We update can_use_next_item = False.
Since dp[4] != input[4] (2 != 2) or can_use_next_item == False, we store
output[4] = 2.

Reconstructing all together, output=[1, 2, 2, 1, 2].
```

Figure 12: A sample scratchpad for the DP task used for fine-tuning with few-shot settings.

**Data Construction** We exhaustively generate data for this DP task. For question-answer setting, we include a thorough explanation of the task before asking to generate a solution (see Figure 11). We use all lists up to 5 elements as training, and we consider only lists where elements are in the range $[-5, 5]$ (giving a total of $11^n$ lists for an input list of size $n$). For out-of-domain evaluation, we use lists of sizes 6 to 10 inclusive. Example scratchpads and zero-shot prompts are shown in Figure 12 and 11 respectively. The scratchpad is generated automatically through templates. We considered five exemplars for the few-shot setup.

# B Experimental Setups & Empirical Results

## B.1 Models

For our experiments, we evaluate the performance of 6 LLMs: GPT4 (`gpt-4`) [42], ChatGPT (`GPT3.5-turbo`) [41], GPT3 (`text-davinci-003`) [8], FlanT5 [13] and LLaMa [58]. The evaluations were conducted from January 2023 to May 2023 using the OpenAI API. We perform fine-tuning on GPT3 (`text-davinci-003`) for the three tasks, observing faster convergence when training on question-scratchpad pairs rather than question-answer pairs. For question-answer pairs fine-tuning, we train separately the model for $\{12, 12, 4\}$ epochs for multiplication, puzzle, and DP respectively, saving the best model based on the validation set. Regarding training on question-scratchpad pairs, we train the model for $\{4, 8, 2\}$ epochs for multiplication, puzzle, and DP. The batch size is set to approximately 0.2% of the number of examples in the training set. Generally, we observe that larger batch sizes tend to yield better results for larger datasets. For the learning rate multiplier, we experiment with values ranging from 0.02 to 0.2 to determine the optimal setting for achieving the best results and chose 0.2. During inference, we set nucleus sampling $p$ to 0.7 and temperature to 1. For each task, we evaluate the performance of each model on 500 test examples.

## B.2 Limits of Transformers in Zero- and Few-shot Settings

Figure 14, Figure 16 and Figure 18 show the zero-shot performance of GPT4, ChatGPT, LLaMA and FlanT5 on the three tasks. Overall, there is a notable decline in performance as the task complexity increases (measured by graph parallelism for multiplication and DP, and propagation steps for puzzles as shown in Figure 13). The few-shot performance with question-answer pairs results in minimal improvement over the zero-shot setting as depicted in Figure 15 and Figure 18 for the multiplication and DP tasks. In contrast, the few-shot setting did not lead to any improvement in the puzzle task.

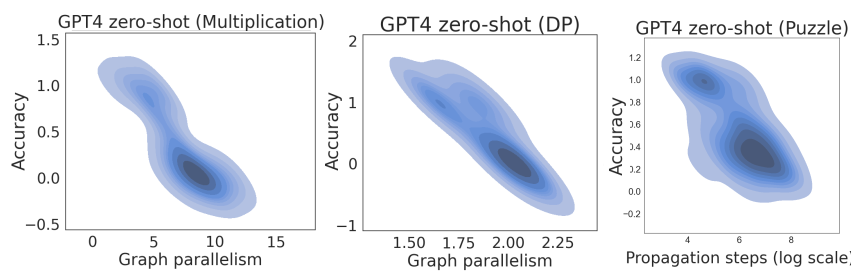


Figure 13: Graph parallelism vs accuracy. The accuracy decreases as the complexity increases.

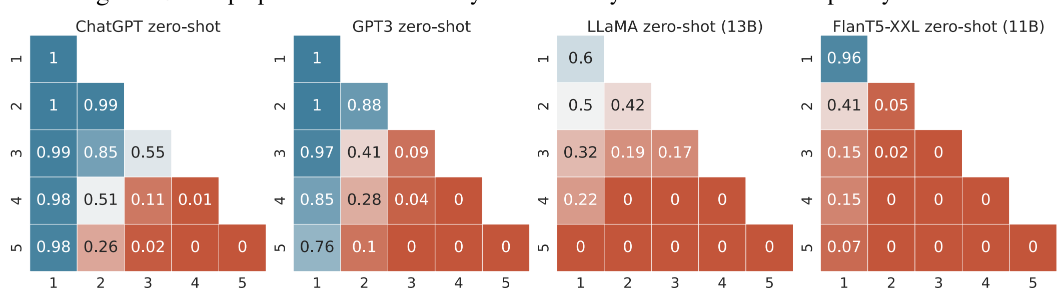


Figure 14: **Zero-shot accuracy**. Performance of ChatGPT, GPT3, LLaMA and FlanT5 on the **multiplication** task.

## B.3 Limits of Transformers with question-answer Training

Figure 17 and Figure 19 show the performance of GPT3 finetuned on question-answer pairs. The model was trained on various splits, considering the problem size, depth, and width of the computation graph. Specifically, for the multiplication task, the model was fine-tuned on a range of multiplication problems, spanning from 1-digit by 1-digit multiplication to 4-digit by 2-digit multiplication amounting to 1.8M pairs. As for the puzzle task, the model was fine-tuned on puzzles of sizes ranging from 2x2 to 4x4 resulting in a total of 142k pairs. Additionally, for the DP task, the model was fine-tuned on problems with a sequence length of 5 resulting in 41K pairs. In an additional setup, we divided

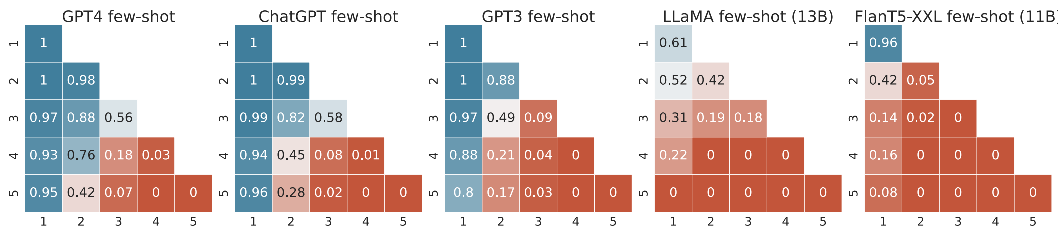


Figure 15: **Few-shot accuracy** with **question-answer** pairs. Performance of GPT4, ChatGPT, GPT3, LLaMA and FlanT5 on the **multiplication** task.


Figure 16: **Zero-shot accuracy**. Performance of ChatGPT, GPT3, LLaMA and FlanT5 on the **puzzle** task. Few-shot performance led to worse performance.

those datasets based on the depth and width of the computation graph for all the tasks and finetuned on different splits. The results indicate a lack of generalization for out-of-domain (OOD) examples while showcasing near-perfect performance for in-domain examples.

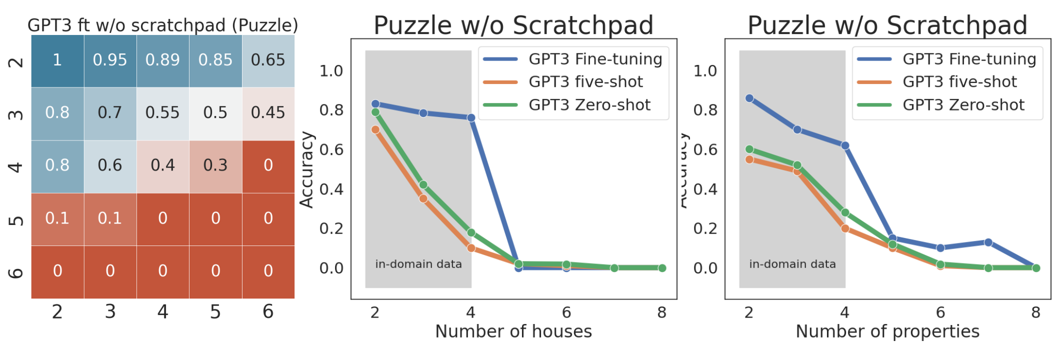


Figure 17: GPT3 finetuned on the puzzle task using **question-answer** pairs. The training data consisted of puzzles of size 4x4, and the model was subsequently evaluated on larger puzzle sizes for OOD testing.

**GPT3 finetuning cost** We will discuss here the approximate cost of fine-tuning GPT3 for the multiplication task. When fine-tuning with question-answer pairs, each example typically consists of around 20 tokens, and 250 tokens for question-scratchpad pairs. The cost for utilizing the `text-davinci-003` model amounts to \$0.02 (USD) per 1,000 tokens. With this particular setup, the total number of training examples required for multiplication up to 5 digits by 5 digits reaches an astonishing figure of approximately 9.1 billion examples. Should we choose to fine-tune GPT3 for 4 epochs on question-answer pairs, the cost would amount to \$12 million and \$700 million for question-scratchpad training. For a more comprehensive breakdown of the cost per problem size, please refer to Table 1.

### B.4 Limits of Transformers with Explicit Scratchpad Training

Figure 21, 22, 20 show the performance of GPT3 finetuned on different splits of the tasks using question-scratchpad pairs. Specifically, for the multiplica-

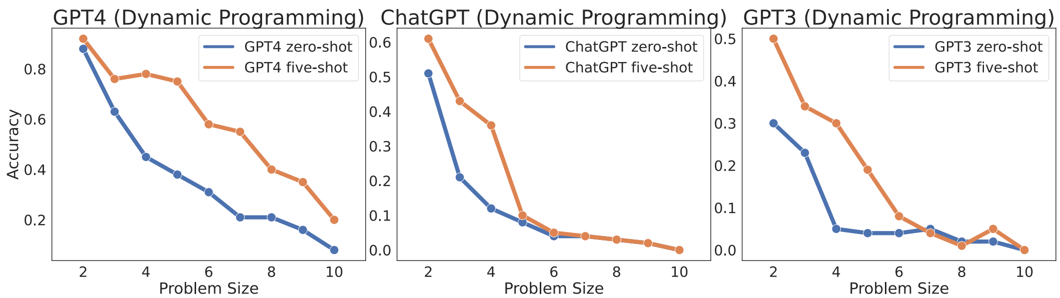


Figure 18: **Zero-shot** and **Few-shot accuracy** using **question-answer** pairs. Performance of GPT4, ChatGPT, and GPT3 on the **dynamic programming** task. LLaMA and FlanT5 results are near zero for all problem sizes.

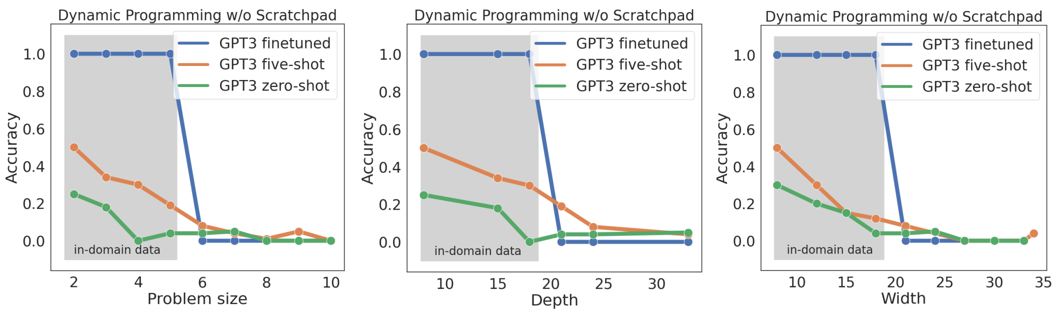


Figure 19: GPT3 finetuned on the **dynamic programming** task using **question-answer** pairs. We consider different data splits: problem size, depth, and width of the graph. Specifically, the model was trained with a problem size of 5, and the graph's depth and width were set to 18.

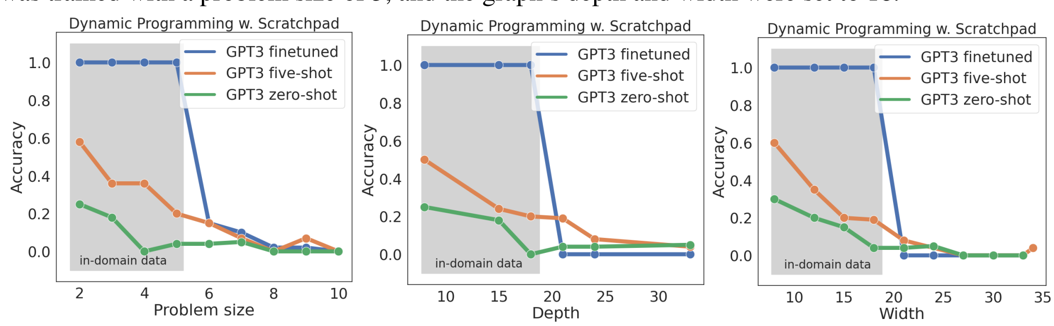


Figure 20: GPT3 finetuned on the **dynamic programming** task using **question-scratchpad** pairs. We consider different data splits: problem size, depth, and width of the graph. Specifically, the model was trained with a problem size of 5, and the graph's depth and width were set to 18.

tion task, the model was fine-tuned on a range of multiplication problems, spanning from 1-digit by 1-digit multiplication to 3-digit by 2-digit multiplication.

As for the puzzle task, the model was fine-tuned on puzzles of sizes ranging from 2x2 to 4x4. Additionally, for the DP task, the model was fine-tuned on problems with a sequence length of 5. Furthermore, different data splits were considered, including variations based on the number of hours, number of properties, depth and width of the graph, and the number of digits in the multiplication output. On all tasks, we can see that the model fails to generalize to OOD data while achieving perfect accuracy on in-domain data, indicating that it cannot learn the underlying computational rules.

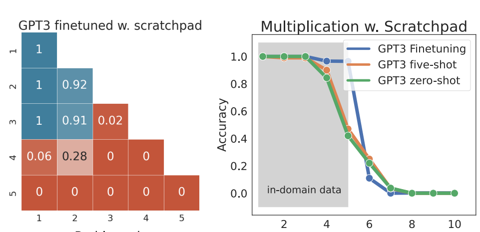


Figure 21: GPT3 finetuned exhaustively on task-specific data up to a certain problem size. In particular, we train on examples up to 3-digit by 2-digit multiplication (left) and on examples that have up to 5 digits in the output response (right). The **blue** region represents the in-distribution examples and the **red** region refers to OOD examples.

| Problem size | # examples | GPT3 Cost without scratchpad | GPT3 Cost with scratchpad |
|---|---|---|---|
| 1 x 1 | 81 | $0.12 | $7.44 |
| 2 x 1 | 810 | $1.28 | $74.4 |
| 2 x 2 | 8100 | $12.96 | $744 |
| 3 x 1 | 8100 | $12.96 | $744 |
| 3 x 2 | 81000 | $129.6 | $7440 |
| 3 x 3 | 810000 | $1296 | $74,404 |
| 4 x 1 | 81000 | $129.6 | $7440 |
| 4 x 2 | 810000 | $1296 | $74,404 |
| 4 x 3 | 8100000 | $12,960 | $744,040 |
| 4 x 4 | 81000000 | $129,600 | $7,440,400 |
| 5 x 1 | 810000 | $1296 | $74,404 |
| 5 x 2 | 8100000 | $12,960 | $744,040 |
| 5 x 3 | 81000000 | $129,600 | $7,440,400 |
| 5 x 4 | 810000000 | $1,296,000 | $70,440,400 |
| 5 x 5 | 8100000000 | $12,960,000 | $700,440,400 |

Table 1: Finetuning cost of GPT3 model on the multiplication data.

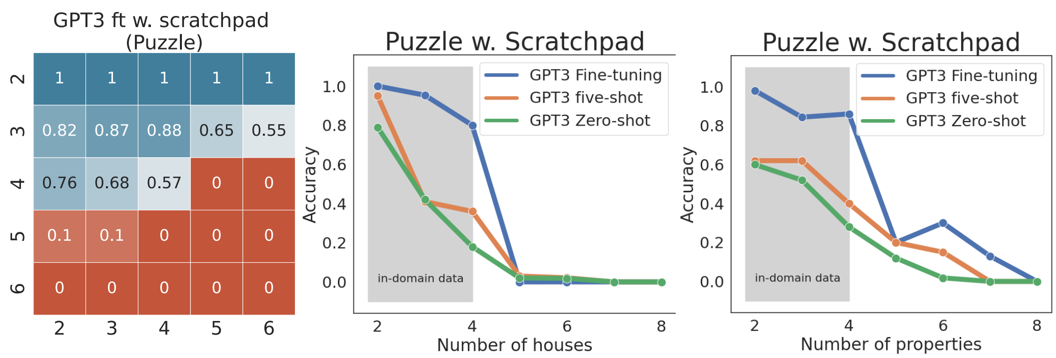


Figure 22: GPT3 finetuned on the puzzle task using **question-scratchpad** pairs. The training data consisted of puzzles of size 4x4, and the model was subsequently evaluated on larger puzzle sizes for OOD testing.

790

# C Surface Patterns

## C.1 Relative Information Gain Predictions for Multiplication

| | | Relative Information Gain | | | |
|---|---|---|---|---|---|
| Input variable | Output variable | 2x2 | 3x3 | 4x4 | 5x5 |
| $x_n$ | $z_{2n}$ | 0.223 | 0.223 | 0.223 | 0.223 |
| $y_n$ | $z_{2n}$ | 0.223 | 0.223 | 0.223 | 0.223 |
| $x_1$ | $z_1$ | 0.198 | 0.199 | 0.199 | 0.199 |
| $y_1$ | $z_1$ | 0.198 | 0.199 | 0.199 | 0.199 |
| $x_n\ y_n$ | $z_{2n}$ | 1.000 | 1.000 | 1.000 | 1.000 |
| $x_{n-1}\ x_n$ | $z_{2n}$ | 0.223 | 0.223 | 0.223 | 0.223 |
| $y_{n-1}\ y_n$ | $z_{2n}$ | 0.223 | 0.223 | 0.223 | 0.223 |
| $x_n\ y_n$ | $z_{2n-1}$ | 0.110 | 0.101 | 0.101 | 0.101 |
| $y_{n-1}\ y_n$ | $z_{2n-1}$ | 0.032 | 0.036 | 0.036 | 0.036 |
| $x_{n-1}\ x_n$ | $z_{2n-1}$ | 0.032 | 0.036 | 0.036 | 0.036 |
| $x_{n-1}\ y_{n-1}$ | $z_{2n-1}$ | 0.018 | 0.025 | 0.025 | 0.025 |
| $x_1\ y_1$ | $z_2$ | 0.099 | 0.088 | 0.088 | 0.088 |
| $x_2\ y_2$ | $z_2$ | 0.025 | 0.016 | 0.016 | 0.016 |
| $x_1\ y_1$ | $z_1$ | 0.788 | 0.792 | 0.793 | 0.793 |
| $y_1\ y_2$ | $z_1$ | 0.213 | 0.211 | 0.211 | 0.211 |
| $x_1\ x_2$ | $z_1$ | 0.213 | 0.211 | 0.211 | 0.211 |

Table 2: **Highest Relative Information Gain Elements and Pairs of Elements**, for multiplications between $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, with $2 \leq n \leq 5$. We define $z := x \cdot y$, which will always have size $2n$ (with possibly a leading zero). $z_{2n}$ denotes the least-significant digit of $z$, and $z_1$ denotes the left-most digit. Only (input, output) pairs above 0.01 are shown. Note that since multiplication is commutative, several pairs of input variables (e.g. $a_0$ and $b_0$) exhibit the same relative information gain.

### C.2 Empirical Surface Pattern Analysis for Multiplication with GPT4, ChatGPT and GPT3

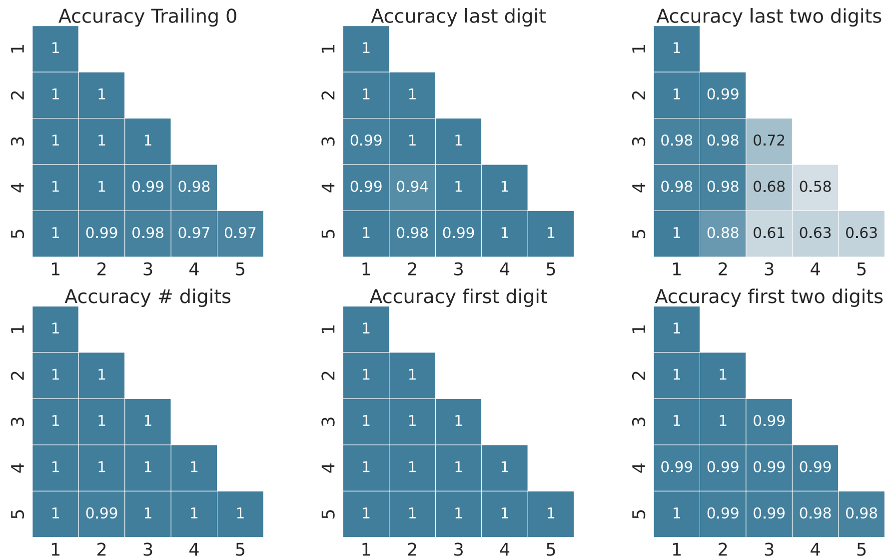


Figure 23: GPT4 zero-shot accuracy in predicting partially correct responses. This evidences surface pattern learning, since the accuracy of full answer prediction is significantly lower–and often near zero (see Figure 2). Specifically, 'accuracy trailing zeros' pertains to accurately predicting the number of zeros in the output number, which is known to be relatively easy to predict based on arithmetic calculations.

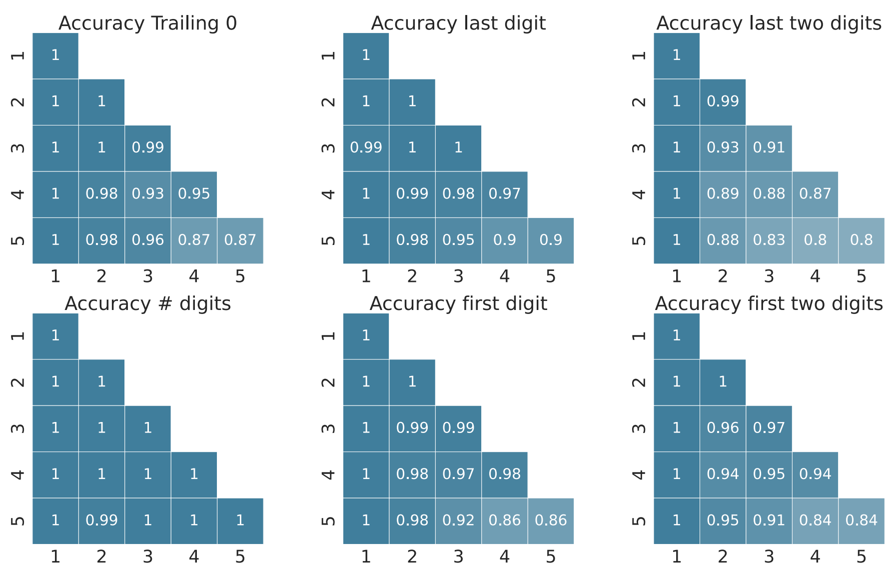


Figure 24: ChatGPT zero-shot accuracy in predicting partially correct responses. We observe the same trend for GPT3 predictions.

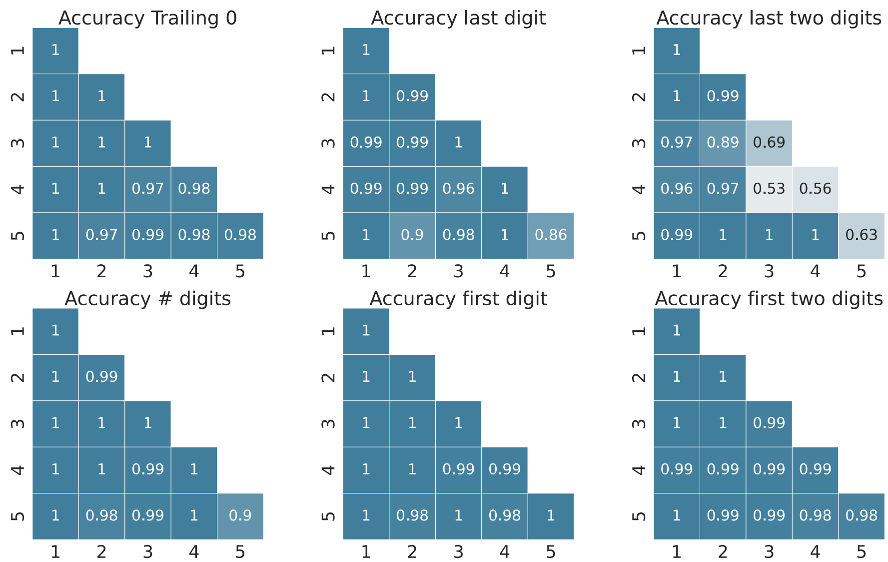


Figure 25: GPT4 five-shot accuracy in predicting partially correct responses. We observe the same trend for ChatGPT, GPT3 few-shot predictions.

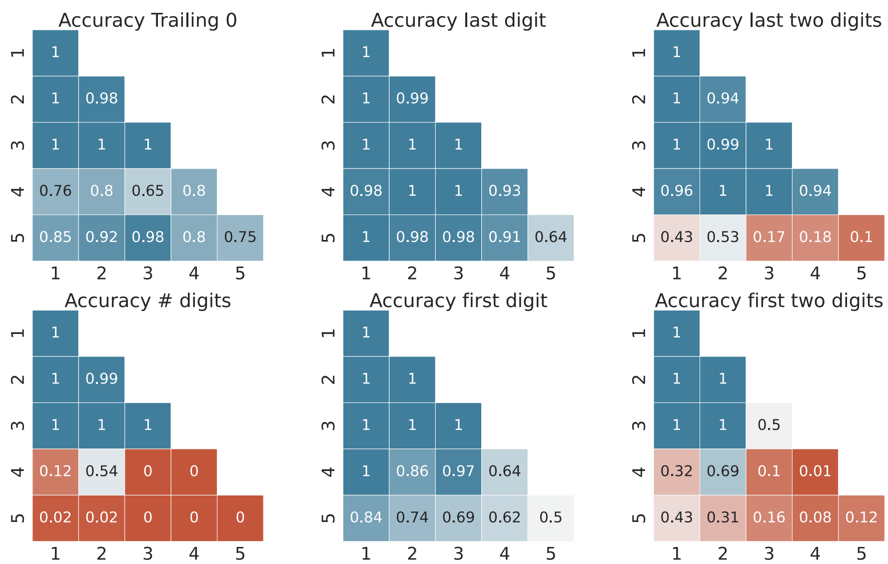


Figure 26: GPT3 finetuned on question-scratchpad pairs. Accuracy of predicting partially correct responses.

### C.3 Relative Information Gain Predictions for Dynamic Programming Task

Let $a_i$ be the $i$-th element of the input sequence, and let $o_i$ be the $i$-th element of the output sequence. As shown in Table 3, $a_i$ is a good predictor of $o_i$, and this is especially true for $a_1$ and $a_{n-1}$, the first and last elements of the sequence. This matches the task intuition, since one would never pick an element $a_i < 0$ and decrease the final sum (one may pick $a_i = 0$ if it makes a lexicographically smaller output sequence).

$a_i$ weakly helps to predict its neighbors. The only case of this behavior with RelativeIG>0.1 is at the start of the sequence, where the first element helps predict the value of the second. This again matches intuition, since a very high $a_1$ indicates that with high probability $o_2$ will not be selected for the final subsequence.

| | | Relative Information Gain for each problem size | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input variable | Output variable | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $a_1$ | $o_2$ | 0.15 | 0.13 | 0.14 | 0.14 | 0.14 | 0.14 | 0.14 | 0.14 | 0.14 |
| $a_1$ | $o_1$ | 0.64 | 0.71 | 0.69 | 0.69 | 0.69 | 0.69 | 0.69 | 0.69 | 0.69 |
| $a_2$ | $o_2$ | 0.53 | 0.42 | 0.45 | 0.44 | 0.45 | 0.44 | 0.44 | 0.45 | 0.44 |
| $a_3$ | $o_3$ | | 0.64 | 0.49 | 0.53 | 0.52 | 0.52 | 0.52 | 0.52 | 0.52 |
| $a_4$ | $o_4$ | | | 0.60 | 0.46 | 0.50 | 0.49 | 0.49 | 0.49 | 0.49 |
| $a_5$ | $o_5$ | | | | 0.62 | 0.47 | 0.51 | 0.50 | 0.50 | 0.50 |
| $a_6$ | $o_6$ | | | | | 0.61 | 0.47 | 0.51 | 0.49 | 0.50 |
| $a_7$ | $o_7$ | | | | | | 0.61 | 0.47 | 0.51 | 0.50 |
| $a_8$ | $o_8$ | | | | | | | 0.61 | 0.47 | 0.51 |
| $a_9$ | $o_9$ | | | | | | | | 0.61 | 0.47 |
| $a_{10}$ | $o_{10}$ | | | | | | | | | 0.61 |
| $a_{n-1}$ | $o_{n-1}$ | | 0.64 | 0.60 | 0.62 | 0.61 | 0.61 | 0.61 | 0.61 | 0.61 |
| $a_{n-2}$ | $o_{n-2}$ | | | | 0.46 | 0.47 | 0.47 | 0.47 | 0.47 | 0.47 |
| $a_{n-3}$ | $o_{n-3}$ | | | | | | 0.51 | 0.51 | 0.51 | 0.51 |
| $a_{n-4}$ | $o_{n-4}$ | | | | | | | | 0.49 | 0.50 |

Table 3: **Highest Relative Information Gain Elements**, for DP problems of size $2 \leq n \leq 10$. We only show the (input, output) pairs where at least three problem sizes have RelativeIG>0, and at least one with RelativeIG>0.1. $a_{n-1}$ refers to the last element of the sequence, regardless of its actual id in the sequence.

Similar behaviors, but with higher relative information gains overall, are observed when analyzing triples of consecutive elements in the list. Table 4 shows that $o_i$ is highly predicted by $(a_{i-1}, a_i, a_{i+1})$. Moreover, $o_i$ is highly predicted by both $(a_{i-2}, a_{i-1}, a_i)$ and $(a_i, a_{i+1}, a_{i+2})$, with the former generally having higher scores than the latter. This again matches the task intuitions, since the value of the neighbors helps determine whether to select a number for the subsequence; and asking for the lexicographically smallest sequence biases the output subsequence to care more about the previous numbers rather than the following ones. We believe that this last point is the cause of the weakly predictive power of $(a_{i-3}, a_{i-2}, a_{i-1})$ to predict $o_i$; whereas $(a_{i+1}, a_{i+2}, a_{i+3})$ is not shown, since all the relative information gain values were below 0.1.

| Input variable | Output variable | Relative Information Gain for each problem size | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $a_{n-3}$ $a_{n-2}$ $a_{n-1}$ | $o_{n-1}$ | | | | | 0.95 | 0.95 | 0.95 | 0.95 |
| $a_{n-3}$ $a_{n-2}$ $a_{n-1}$ | $o_{n-2}$ | | | | | 0.87 | 0.87 | 0.87 | 0.87 |
| $a_{n-3}$ $a_{n-2}$ $a_{n-1}$ | $o_{n-3}$ | | | | | 0.64 | 0.64 | 0.64 | 0.64 |
| $a_1$ $a_2$ $a_3$ | $o_1$ | 1.00 | 0.96 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 |
| $a_1$ $a_2$ $a_3$ | $o_2$ | 1.00 | 0.91 | 0.92 | 0.91 | 0.92 | 0.91 | 0.92 | 0.91 |
| $a_2$ $a_3$ $a_4$ | $o_2$ | | 0.56 | 0.55 | 0.55 | 0.55 | 0.55 | 0.55 | 0.56 |
| $a_1$ $a_2$ $a_3$ | $o_3$ | 1.00 | 0.66 | 0.73 | 0.71 | 0.72 | 0.72 | 0.72 | 0.72 |
| $a_2$ $a_3$ $a_4$ | $o_3$ | | 0.86 | 0.77 | 0.78 | 0.78 | 0.78 | 0.78 | 0.78 |
| $a_3$ $a_4$ $a_5$ | $o_3$ | | | 0.67 | 0.66 | 0.66 | 0.66 | 0.66 | 0.66 |
| $a_2$ $a_3$ $a_4$ | $o_4$ | | 0.94 | 0.64 | 0.7 | 0.68 | 0.69 | 0.69 | 0.69 |
| $a_3$ $a_4$ $a_5$ | $o_4$ | | | 0.88 | 0.79 | 0.81 | 0.8 | 0.8 | 0.8 |
| $a_4$ $a_5$ $a_6$ | $o_4$ | | | | 0.63 | 0.62 | 0.62 | 0.62 | 0.62 |
| $a_3$ $a_4$ $a_5$ | $o_5$ | | | 0.95 | 0.65 | 0.71 | 0.69 | 0.7 | 0.7 |
| $a_4$ $a_5$ $a_6$ | $o_5$ | | | | 0.87 | 0.78 | 0.79 | 0.79 | 0.79 |
| $a_5$ $a_6$ $a_7$ | $o_5$ | | | | | 0.64 | 0.63 | 0.63 | 0.64 |
| $a_4$ $a_5$ $a_6$ | $o_6$ | | | | 0.94 | 0.64 | 0.71 | 0.69 | 0.7 |
| $a_5$ $a_6$ $a_7$ | $o_6$ | | | | | 0.87 | 0.78 | 0.8 | 0.8 |
| $a_6$ $a_7$ $a_8$ | $o_6$ | | | | | | 0.64 | 0.62 | 0.63 |
| $a_5$ $a_6$ $a_7$ | $o_7$ | | | | | 0.95 | 0.64 | 0.71 | 0.69 |
| $a_6$ $a_7$ $a_8$ | $o_7$ | | | | | | 0.87 | 0.78 | 0.8 |
| $a_6$ $a_7$ $a_8$ | $o_8$ | | | | | | 0.95 | 0.64 | 0.71 |
| $a_1$ $a_2$ $a_3$ | $o_4$ | | 0.12 | 0.1 | 0.11 | 0.11 | 0.11 | 0.11 | 0.11 |
| $a_2$ $a_3$ $a_4$ | $o_5$ | | | 0.1 | 0.09 | 0.1 | 0.09 | 0.1 | 0.1 |
| $a_3$ $a_4$ $a_5$ | $o_6$ | | | | 0.11 | 0.1 | 0.1 | 0.1 | 0.11 |
| $a_4$ $a_5$ $a_6$ | $o_7$ | | | | | 0.11 | 0.09 | 0.1 | 0.11 |
| $a_5$ $a_6$ $a_7$ | $o_8$ | | | | | | 0.11 | 0.09 | 0.11 |

Table 4: **Highest Relative Information Gain Contiguous Triples**, for DP problems of size $3 \leq n \leq 10$. We only show the (input, output) pairs where at least three problem sizes have RelativeIG>0, and at least one with RelativeIG>0.1. $a_{n-1}$ refers to the last element of the sequence, regardless of its actual id in the sequence.

## C.4 Empirical Surface Pattern Results for Dynamic Programming Task

We observe that all analyzed models match the Relative Information Gain prediction that $o_1$ (whether the first element goes into the output sequence or not) should be the easiest value to predict (see Figures 27, 28, and 29). However, since GPT3 often predicts shorter output sequences than the required size, the analysis of the predictive power of $o_{n-1}$ is only done for GPT4. In GPT4, we observe that $o_{n-1}$ is among the easiest values to predict as expected by Relative Information Gain.

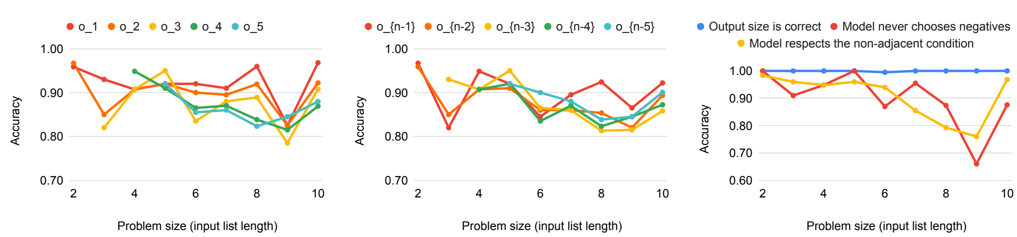


Figure 27: GPT4 five-shot with scratchpad accuracy in predicting output elements $o_i$ in the DP task. All $o_i$ are predicted with high accuracy with $o_1$ and $o_{n-1}$ being consistently among the highest. These observations go in line with the Relative Information Gain prediction.

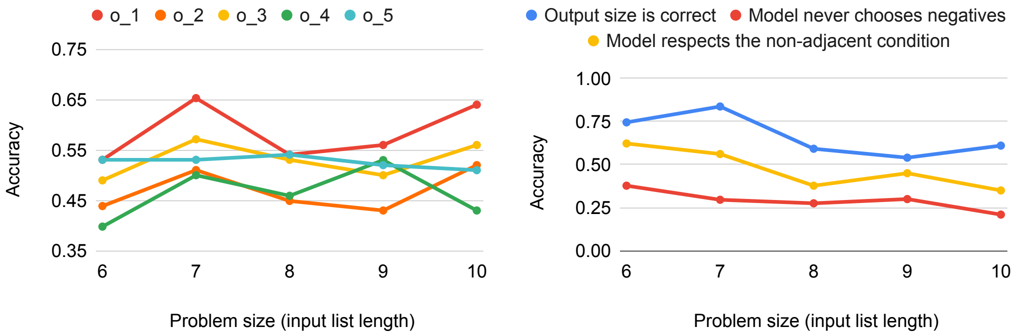


Figure 28: GPT3 few-shot without scratchpad accuracy in predicting output elements $o_i$ in the DP task. As predicted by Relative Information Gain, the model predicts $o_1$ correctly with the highest probability. However, because GPT3 often does not produce the correct output size, it hinders us from analyzing $o_{n-1}$.

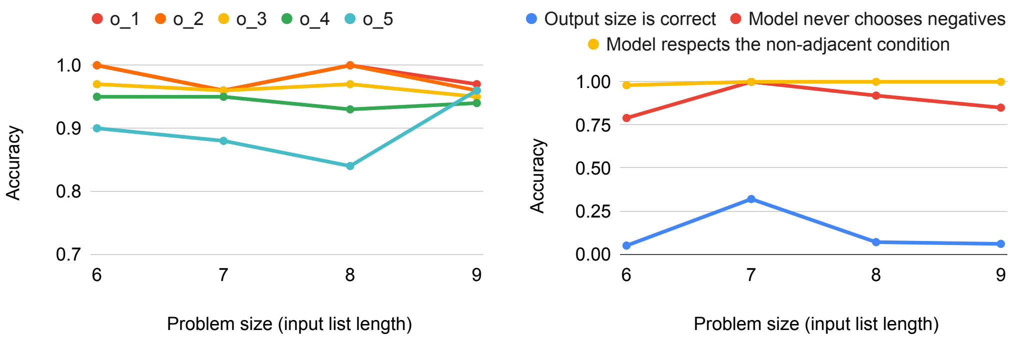


Figure 29: GPT3 fine-tuned without scratchpad accuracy in predicting output elements $o_i$ in the DP task. As predicted by Relative Information Gain, the model predicts $o_1$ correctly with the highest probability. However, because GPT3 often does not produce the correct output size, it hinders us from analyzing $o_{n-1}$.

# D Derivations

## D.1 Transformers struggle with problems with increasingly larger parallelism (*width*)

**Proposition D.1.** *Let $f_n(\mathbf{x}) = h_n(g(\mathbf{x},1), g(\mathbf{x},2)), \ldots, g(\mathbf{x},n))$. Let $\widehat{h}_n, \widehat{g}, \widehat{f}_n$ be estimators of $h_n, g, f_n$ respectively. Assume $\mathbb{P}(h_n = \widehat{h}_n) = 1$ and $\mathbb{P}(h_n(X) = h_n(Y) \mid X \neq Y) < \beta\alpha^n$ for some $\alpha \in (0,1)$ and $\beta > 0$ (i.e. $\widehat{h}_n$ perfectly estimates $h_n$, and $h_n$ is almost injective). If $\mathbb{P}(g \neq \widehat{g}) = \epsilon > 0$ and errors in $\widehat{g}$ are independent, then $\lim_{n\to+\infty} \mathbb{P}(f_n \neq \widehat{f}_n) = 1$.*

*Proof.* For ease of writing, let $X_i = g(X,i)$ and $Y_i = \widehat{g}(X,i)$, and let $\boldsymbol{X} = (X_1, \ldots, X_n)$ and $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$. We will compute some auxiliary probabilities, and then upper bound $\mathbb{P}(f = \widehat{f})$, to finally compute its limit.

$$\begin{aligned}\mathbb{P}(\boldsymbol{X} = \boldsymbol{Y}) &= \mathbb{P}(X_1 = Y_1, X_2 = Y_2, \ldots, X_n = Y_n)\\ &= \mathbb{P}(X_1 = Y_1) \cdot \mathbb{P}(X_2 = Y_2) \ldots \cdot \mathbb{P}(X_n = Y_n) = \mathbb{P}(g = \widehat{g})^n = (1-\epsilon)^n \end{aligned} \tag{2}$$

Since by hypothesis we know $\mathbb{P}(h_n(\boldsymbol{Y}) = \widehat{h}_n(\boldsymbol{Y})) = 1$, we have that:

$$\begin{aligned}\mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y}) &= \mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{Y}) \cap h_n(\boldsymbol{Y}) = \widehat{h}_n(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y})\\ &= \mathbb{P}(h_n(\boldsymbol{X}) = h_n(\boldsymbol{Y}) = \widehat{h}_n(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y})\\ &\leq \mathbb{P}(h_n(\boldsymbol{X}) = h_n(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y})\\ &< \beta\alpha^n \end{aligned} \tag{3}$$

We will now estimate $\mathbb{P}(f_n = \widehat{f}_n)$ using the law of total probability w.r.t. the event $\boldsymbol{X} = \boldsymbol{Y}$.

$$\begin{aligned}\mathbb{P}(f_n = \widehat{f}_n) &= \mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{Y}))\\ &= \mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{Y}) \mid \boldsymbol{X} = \boldsymbol{Y}) \cdot \mathbb{P}(\boldsymbol{X} = \boldsymbol{Y}) + \mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y}) \cdot \mathbb{P}(\boldsymbol{X} \neq \boldsymbol{Y})\\ &= \mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{X})) \cdot \mathbb{P}(\boldsymbol{X} = \boldsymbol{Y}) + \mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y}) \cdot (1 - \mathbb{P}(\boldsymbol{X} = \boldsymbol{Y}))\\ &= 1 \cdot (1-\epsilon)^n + \mathbb{P}(h_n(\boldsymbol{X}) = \widehat{h}_n(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y}) \cdot (1 - (1-\epsilon)^n) \qquad \text{(using 2 and hypothesis)}\\ &< (1-\epsilon)^n + \beta\alpha^n \cdot (1 - (1-\epsilon)^n) \qquad \text{(using 3)}\\ &< \beta\alpha^n + (1-\epsilon)^n \cdot (1 - \beta\alpha^n) \end{aligned}$$

To conclude our proof, we will show that $\lim_{n\to+\infty} \mathbb{P}(f_n = \widehat{f}_n)$ exists and compute its value. Note that since $1 - \epsilon \in [0,1)$ and $\alpha \in (0,1)$, trivially $\lim_{n\to+\infty} \beta\alpha^n + (1-\epsilon)^n \cdot (1 - \beta\alpha^n) = 0$.

$$0 \leq \liminf_{n\to+\infty} \mathbb{P}(f_n = \widehat{f}_n) \leq \limsup_{n\to+\infty} \mathbb{P}(f_n = \widehat{f}_n) \leq \limsup_{n\to+\infty} \beta\alpha^n + (1-\epsilon)^n \cdot (1 - \beta\alpha^n) = 0$$

Then, $\lim_{n\to+\infty} \mathbb{P}(f_n = \widehat{f}_n) = 0$ and we conclude $\lim_{n\to+\infty} \mathbb{P}(f_n \neq \widehat{f}_n) = 0$. □

**Corollary D.1.** *Assume that a model $\mathcal{M}$ solves shifted addition perfectly, but it incorrectly solves **at least one** $m$ digit by 1 digit multiplication for some fixed $m$. Then, the probability that $\mathcal{M}$ will solve **any** $m$ digit by $n$ digit multiplication using the long-form multiplication algorithm tends to 0.*

*Proof.* We define $s : \mathbb{Z}_{10}^{m+n} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$, $d : \mathbb{N} \times \mathbb{Z}_{10} \to \mathbb{N}$, $h_n : \mathbb{N}^n \to \mathbb{N}$, and $f_n : \mathbb{Z}_{10}^{m+n} \to \mathbb{N}$ as follows.

$$\begin{aligned} s([x_1, \ldots, x_m, x_{m+1}, \ldots, x_{m+n}], j) &:= (x_1 ^\frown x_2 ^\frown \ldots ^\frown x_m,\ x_{m+j})\\ &\qquad \text{where } x_1 ^\frown x_2 ^\frown \ldots ^\frown x_m \text{ denotes concatenating digits } x_i\\ d(x,y) &:= x \cdot y\\ g &:= d \circ s\\ h_n(x_1, \ldots, x_n) &:= \sum_{i=1}^{n} x_i 10^{n-i}\\ f_n(\mathbf{x}) &:= h_n(g(\mathbf{x},1), g(\mathbf{x},2)), \ldots, g(\mathbf{x},n)) \end{aligned}$$

Note that $g$ defines the base-10 multiplication between $m$-digit numbers $(x_1 x_2 \dots x_m)$ and 1-digit numbers $(x_{m+j})$, where $s$ denotes the selection of the numbers to multiply and $d$ denotes the actual multiplication. Note that $h_n$ describes the shifted addition used at the end of long-form multiplication to combine $n$ $m$-digit by 1-digit multiplications. Therefore, $f_n$ describes the long-form multiplication of $m$-digit by $n$-digit numbers.

By hypothesis, $\mathbb{P}(g \neq \widehat{g}) = \epsilon > 0$ and $\mathbb{P}(h_n = \widehat{h}_n) = 1$, where $\widehat{g}$ and $\widehat{h}_n$ denote estimators using model $\mathcal{M}$. It can be shown that $\mathbb{P}(h_n(X) = h_n(Y) \mid X \neq Y) < \beta\alpha^n$ for $\alpha = 0.1$ and $\beta = 10^m$. Using Lemma D.1, $\lim_{n\to+\infty} \mathbb{P}(f_n \neq \widehat{f}_n) = 1$, which concludes our proof.

$\square$

Note that Lemma D.1's proofs gives us empirical bounds once $\epsilon$ and $\alpha$ are approximated. Also **note that our definition of $g$ in the proof of Corollary D.1 highlights two possible sources of exponentially-accumulating error**: errors in the selection of the numbers to multiply $s$, and errors in the actual $m$-digit by 1-digit multiplication $d$.

### D.2 Transformers struggle with problems that require increasingly larger iterative applications of a function (*depth*)

**Proposition D.2.** *Let $f_n(\mathbf{x}) = g^n(\mathbf{x})$. Assume $\mathbb{P}(g(X) = \widehat{g}(Y) \mid X \neq Y) \leq c$ (i.e. recovering from a mistake due to the randomness of applying the estimator on an incorrect input has probability at most $c$). If $\mathbb{P}(g \neq \widehat{g}) = \epsilon > 0$ with $c + \epsilon < 1$, then $\liminf_{n\to+\infty} \mathbb{P}(f_n \neq \widehat{f}_n) = 1 - \dfrac{c}{c+\epsilon}$.*

*Proof.* We first derive a recursive upper bound using the law of total probability, and then prove a non-recursive upper bound by induction.

$$\begin{aligned}
s_n := \mathbb{P}(f_n = \widehat{f}_n) &= \mathbb{P}(g(g^{n-1}(Z)) = \widehat{g}(\widehat{g}^{n-1}(Z))) \\
&= \mathbb{P}(g(\boldsymbol{X}) = \widehat{g}(\boldsymbol{Y})) \quad \text{ where } \boldsymbol{X} := g^{n-1}(Z) \text{ and } \boldsymbol{Y} := \widehat{g}^{n-1}(Z) \\
&= \mathbb{P}(g(\boldsymbol{X}) = \widehat{g}(\boldsymbol{Y}) \mid \boldsymbol{X} = \boldsymbol{Y}) \cdot \mathbb{P}(\boldsymbol{X} = \boldsymbol{Y}) + \mathbb{P}(g(\boldsymbol{X}) = \widehat{g}(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y}) \cdot \mathbb{P}(\boldsymbol{X} \neq \boldsymbol{Y}) \\
&= \mathbb{P}(g(\boldsymbol{X}) = \widehat{g}(\boldsymbol{X})) \cdot \mathbb{P}(\boldsymbol{X} = \boldsymbol{Y}) + \mathbb{P}(g(\boldsymbol{X}) = \widehat{g}(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y}) \cdot (1 - \mathbb{P}(\boldsymbol{X} = \boldsymbol{Y})) \\
&= \mathbb{P}(g(\boldsymbol{X}) = \widehat{g}(\boldsymbol{X})) \cdot s_{n-1} + \mathbb{P}(g(\boldsymbol{X}) = \widehat{g}(\boldsymbol{Y}) \mid \boldsymbol{X} \neq \boldsymbol{Y}) \cdot (1 - s_{n-1}) \\
&\leq (1-\epsilon) \cdot s_{n-1} + c \cdot (1 - s_{n-1}) \\
&\leq (1 - \epsilon - c) \cdot s_{n-1} + c
\end{aligned}$$

We know $s_1 = (1-\epsilon)$ since $s_1 = \mathbb{P}(f_1 = \widehat{f}_1) = \mathbb{P}(g = \widehat{g})$. Let $b := 1 - \epsilon - c$ for ease of writing. Then, we have

$$s_n \leq b \cdot s_{n-1} + c \tag{4}$$

It can be easily shown by induction that $s_n \leq b^{n-1}(1-\epsilon) + c\sum_{i=0}^{n-2} b^i$:

- The **base case** $n = 2$ is true since we know $s_2 \leq b \cdot s_1 + c$, and $b \cdot s_1 + c = b(1-\epsilon) + c = b^{2-1}(1-\epsilon) + c\sum_{i=0}^{2-2} b^i$, thus showing $s_2 \leq b^{2-1}(1-\epsilon) + c\sum_{i=0}^{2-2} b^i$
- The **inductive step** yields directly using Equation 4,

$$\begin{aligned}
&s_n \leq b \cdot s_{n-1} + c \\
&\quad \leq b \cdot \left(b^{n-2}(1-\epsilon) + c\sum_{i=0}^{n-3} b^i\right) + c \leq b^{n-1}(1-\epsilon) + c\sum_{i=1}^{n-2} b^i + c \leq b^{n-1}(1-\epsilon) + c\sum_{i=0}^{n-2} b^i
\end{aligned}$$

We can rewrite the geometric series $\sum_{i=0}^{n-2} b^i$ in its closed form $\frac{1-b^{n-1}}{1-b}$, and recalling $b := 1 - \epsilon - c$,

$$\begin{aligned}
s_n \leq b^{n-1}(1-\epsilon) + c\frac{1-b^{n-1}}{1-b} &= b^{n-1}(1-\epsilon) + c\frac{1-b^{n-1}}{c+\epsilon} \\
&= b^{n-1}(1-\epsilon) + \frac{c}{c+\epsilon} - b^{n-1}\frac{c}{c+\epsilon} \\
&= b^{n-1}\left(1 - \epsilon - \frac{c}{c+\epsilon}\right) + \frac{c}{c+\epsilon}
\end{aligned}$$

Recalling that $s_n = \mathbb{P}(f_n = \widehat{f}_n)$, we compute the limit inferior of $\mathbb{P}(f_n \neq \widehat{f}_n) = 1 - s_n \geq 1 - b^{n-1}(1 - \epsilon - \frac{c}{c+\epsilon}) - \frac{c}{c+\epsilon}$.

$$\liminf_{n\to+\infty} \mathbb{P}(f_n \neq \widehat{f}_n) \geq \lim_{n\to+\infty} 1 - b^{n-1}\left(1 - \epsilon - \frac{c}{c+\epsilon}\right) - \frac{c}{c+\epsilon} = 1 - \frac{c}{c+\epsilon}$$

that concludes our proof. □

We can generalize the proof in Lemma 4.2 to tasks where there are potentially many valid reasoning chains with the following alternative state-transition framing.

**Lemma D.2.** *Let $S$ denote the set of all possible states a language model can generate, and let $z : S \to \{0, 1\}$ defines if a state is valid (0 = invalid). Let $\widehat{g} : S \to \Pi(S)$ be a state-transition function representing a language model's probability distribution of generating each possible next state when attempting to perform a single reasoning step. Assume $\mathbb{P}(z(\widehat{g}(X)) = 1 \mid z(X) = 0) \leq c$ and $\mathbb{P}(z(\widehat{g}(X)) = 0 \mid z(X) = 1) = \epsilon > 0$ with $c + \epsilon < 1$. Then, $\liminf_{n\to+\infty} \mathbb{P}(z(\widehat{g}^n) = 0) = 1 - \dfrac{c}{c+\epsilon}$.*

If for task $T$ we know that all valid reasoning chains to arrive at a correct result have at least length $n$ (i.e., the equivalent of defining $f_n = g^n$ in Lemma D.1) then the probability of solving task $T$ correctly tends to at most $\frac{c}{c+\epsilon}$.

**Corollary D.3.** *The recursions for dynamic programming tasks, the $m$-by-1 digit multiplication, and the puzzle's elimination function are all tasks where there is a fixed reasoning step $g$ being repeatedly applied. Therefore, we can directly apply Proposition 4.2 to these tasks.*

*Proof.* Let's analyze the three tasks separately below.

**$m$-by-1 digit multiplication may be viewed as $f^m(\mathbf{x})$** Let $x = (x_1, \ldots, x_m)$ be the $m$-digit number that we multiply by the 1-digit number $y$ ($0 \leq y < 10$). Let $z = (z_1, \ldots, z_{m+1})$ denote $z = x \cdot y$, which is guaranteed to have exactly $m + 1$ digits (with possibly leading zeros). We define $f$ as:

$$f(x_1, \ldots, x_m, y, i, c) := (x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots x_m, y, i - 1, c')$$

where $x_i' := (x_i \cdot y + c) \mod 10$ and $c' := \lfloor (x_i \cdot y + c)/10 \rfloor$. Note that $x_i' = z_{i+1}$ since $f$ is performing one step of the long-form multiplication algorithm.

Let the initial input be $\mathbf{x} := (x_1, \ldots, x_m, y, m, 0)$. Then, it can be easily shown that $f^m(\mathbf{x}) = (z_2, \ldots, z_{m+1}, y, 0, c)$. Since $c$ is the left-most carry, it is the leading digit of $z$, i.e. $c = z_1$ (possibly zero) . Thus, the value of $z$ can be directly extracted from $f^m(\mathbf{x}) = (z_2, \ldots, z_{m+1}, y, 0, z_1)$.

**In the DP task, $dp$'s computation may be viewed as $f^{m-2}(x)$ for a list of size $m$** See §A.3.1 for details on the solution to this problem. We will use identical notation. Let $a_1, \ldots, a_m$ be an input list. Let $\mathbf{x} = (a_1, \ldots, a_{m-2}, a_{m-1}', a_m', m - 2)$, where $a_m' := \max(a_m, 0)$ and $a_{m-1}' := \max(a_{m-1}, a_m, 0)$. Intuitively, this means that we have applied the first two steps of the $dp$ computation, and stored the results in $a_{m-1}'$ and $a_m'$. Let $f$ be a function representing the recursive computation of $dp_i$:

$$f(a_1, \ldots, a_i, a_{i+1}', \ldots, a_m', i) = (a_1, \ldots, a_{i-1}, a_i', \ldots, a_m', i - 1)$$

where $a_i' := \max(a_{i+1}', a_i + a_{i+2}', 0)$.

Note that since $a_{i+1}'$ stores the value of $dp_{i+1}$ and $a_{i+2}'$ stores the value of $dp_{i+2}$, it can be easily shown that $f^{m-2}(\mathbf{x}) = (a_1', \ldots, a_m', 0) = (dp_1, \ldots, dp_m, 0)$. Therefore, $f^{m-2}$ computes all recursive values of $dp_i$ when given the base cases.

**In the DP task, the reconstruction of the desired subsequence given already computed $dp$ values may be viewed as $f^m(x)$ for an input list of size $m$.** This case is similar to the previous one. Let $r = (r_1, \ldots, r_m)$ be the result, where $r_i = 1$ if $a_i$ was selected for the desired subsequence, and $r_i = 2$ otherwise. Let $\mathbf{x} := (dp_1, \ldots, dp_m, 0, 0, a_1, \ldots, a_m, 1, 1)$. Let $f$ be defined as follows:

$$f(dp_1, \ldots, dp_m, 0, 0, a'_1, \ldots, a'_{i-1}, a_i, \ldots, a_m, i, u) = (dp_1, \ldots, dp_m, 0, 0, a'_1, \ldots, a'_i, a_{i+1}, \ldots, a_m, i+1, u')$$

where $a'_i := 2 - \mathbb{1}\{dp_i = a_i + dp_{i+2} \text{ and } u = 1\}$ and $u := 1 - \mathbb{1}\{dp_i = a_i + dp_{i+2} \text{ and } u = 1\}$. Intuitively, $a'_i$ stores whether the $i$-th element of the list should be selected for the final subsequence, assigning 1 if the element should be taken, and 2 otherwise (i.e., $a'_i = r_i$). Moreover, if the $i$-th element has been selected, we mark that the next item will not be available using $u'$. Therefore, $f$ performs one step of the final output reconstruction as defined in §A.3.1.

It can be easily shown that $f^m(\mathbf{x}) := (dp_1, \ldots, dp_m, 0, 0, a'_1, \ldots, a'_m, m+1, u') = (dp_1, \ldots, dp_m, 0, 0, r_1, \ldots, r_m, m+1, u')$. Note that the extra two elements in the input state allow lifting the special cases $m-1$ and $m$ in the solution shown in §A.3.1 without falling out of bounds.

**Solving the puzzle task may be seen as $f^m$ for some $m$, where $f$ is the elimination function** Let $c_1, \ldots, c_n$ be the list of clues, let $H$ be the number of houses, and let $A$ be a partially filled solution of size $K \times M$ as defined in §2.4. Each cell $A_{ij}$ can take $H+1$ values: the $H$ options for the cell and the value ø, implying this cell has not been filled. An elimination step $f$ may be defined as:

$$f(c_1, \ldots, c_n, A_{11}, \ldots A_{1M}, \ldots, A_{K1}, \ldots A_{KM}) = (c_1, \ldots, c_n, A'_{11}, \ldots A'_{1M}, \ldots, A'_{K1}, \ldots A'_{KM})$$

where $A'$ is also a partially filled matrix, with $A_{ij} = A'_{ij}$ for every $A_{ij} \neq$ø and where $A'$ has at least one more filled cell.

Let $\mathbf{x} = (c_1, \ldots, c_n, E)$ where $E$ is an empty matrix of size $K \times M$ (all cell values of $E$ are ø).

Then, a full solution is computed as $f^m(\mathbf{x})$ for some value of $m$ that increases with the problem size. In contrast to other tasks, the value of $m$ is not fixed, and depends on the task instance, but using solvers we know that $m$ increases with problem size. □

### D.3 Discussing $c \ll \epsilon$ in the context of Proposition 4.2

Note that in Proposition 4.2, if $c \ll \epsilon$ then $\liminf_{n \to +\infty} \mathbb{P}(f_n \neq \widehat{f}_n) \approx 1$. This is because assuming $\epsilon = m \cdot c$ for some $m > 0$, we have $1 - \frac{c}{c+\epsilon} = 1 - \frac{c}{c + m \cdot c} = 1 - \frac{1}{m+1} = \frac{m}{m+1}$, and $\frac{m}{m+1}$ is a monotonically increasing function for all $m > 0$ that tends to 1 when $m$ goes to infinity. Therefore, large $m$'s (or alternatively, $c \ll \epsilon$) imply $\frac{m}{m+1}$ will be close to 1.

It is reasonable to assume $c \ll \epsilon$ when $g$ has low collision, since $c$ represents the probability of the estimator $\widehat{g}(y)$ arriving at the correct output $g(x)$ by chance when given the wrong input $y \neq x$.

If $g$ is discrete, it can take $|\mathrm{Im}(g)|$ values, where $|\mathrm{Im}(g)|$ denotes the cardinal of the image space of $g$.Assuming approximately uniform errors, $c \approx \epsilon / |\mathrm{Im}(g)|$, which in turn implies $c \ll \epsilon$ since $g$ being low collision implies $|\mathrm{Im}(g)|$ is large.

If $g$ is continuous, then assuming approximately uniform errors we have $c \approx 0$.

Summarizing both cases, if errors are approximately evenly distributed we obtain that $\liminf_{n \to +\infty} \mathbb{P}(f_n \neq \widehat{f}_n) \approx 1$.

## D.4 Error rates in repeated applications of a function may be unbounded

Time series analysis studies series $(y_t)_t$ where each $y_t$ linearly depends on the immediately previous $p \geq 1$ time steps, and potentially including an error component. We will focus on the vectorial case, defined as follows.

**Definition D.1** (Hamilton 1994, $p$-th order vector autorregressions, VAR(p))**.** *Let*

$$y_t = c + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \ldots + \Phi_p y_{t-p} + \epsilon_t$$

*where $c$ denotes a $n \times 1$ vector of constants and $\Phi_i$ denotes an $n \times n$ matrix of autoregressive coefficients. The $n \times 1$ $\epsilon_t$ vector is a generalization of white noise: $E(\epsilon_t) = 0$, $E(\epsilon_t, \epsilon'_\Gamma) = 0$ for $t \neq \Gamma$, and $E(\epsilon_t, \epsilon_t) = \Omega$ with $\Omega$ symmetric positive definite matrix.*

We say a process is covariance-stationary if its first and second moments ($E[y_t]$ and $E[y'_{t-j}]$) are independent of the time $t$. Intuitively, this implies that the consequences of any $\epsilon_t$ must eventually die out. Such a process may also be referred to as a *stable process* (e.g., in Lütkepohl 2005). The following necessary and sufficient condition for stableness can be derived:

**Proposition D.3** (Hamilton 1994, Proposition 10.1)**.** *Let $F$ be an $np \times np$ matrix defined as follows.*

$$F = \begin{bmatrix} \Phi_1 & \Phi_2 & \Phi_3 & \ldots & \Phi_{p-1} & \Phi_p \\ I_n & 0 & 0 & \ldots & 0 & 0 \\ 0 & I_n & 0 & \ldots & 0 & 0 \\ 0 & 0 & I_n & \ldots & 0 & 0 \\ 0 & 0 & 0 & \ldots & I_n & 0 \end{bmatrix}$$

*The eigenvalues of matrix $F$ satisfy*

$$|I_n \lambda^p - \Phi_1 \lambda^{p-1} - \Phi_2 \lambda^{p-2} - \Phi_p| = 0$$

*Hence, a VAR(p) is covariance-stationary as long as $|\lambda| < 1$ for all values of $\lambda$ satisfying this equation.*

In our case, repeated iterations only involve considering the immediately previous step, i.e. $p = 1$. Then, a $VAR(1)$ process $y_t = c + \Phi_1 y_{t-1} + \epsilon_t$ is covariance-stationary (or stable) if and only if the eigenvalues of $F = \Phi_1$ lie inside the unit circle. $F$ will be unstable if at least one eigenvalue lies outside the unit circle, which in turn usually means an explosive system.

**Intuition** For VAR(1) (i.e., $y_t = c + \Phi_1 y_{t-1} + \epsilon_t$), we can intuitively see why large eigenvalues are problematic. If $y_t$ is VAR(1), then it can be rewritten as

$$y_t = \Phi_1^t y_0 + \sum_{i=1}^{t-1} \Phi_1^i \epsilon_{t-i} + \Big(I_n + \sum_{i=0}^{t-1} \Phi_1^i\Big) c$$

Intuitively, large eigenvalues are problematic because if we diagonalize $\Phi_1 = PDP^{-1}$, then $\Phi_1^t = PD^t P^{-1}$, with $D_{ii} = \lambda_i^t$. Thus, a component of $\Phi_1^t$ will diverge if $|\lambda_i| > 1$. If $\Phi_1$ is not diagonalizable, a similar argument holds for its Jordan decomposition. See Lütkepohl 2005, Section 2.1.1 for details.

# E Societal impact

Our work on analyzing the limitations of current Transformers in compositional tasks can have a positive societal impact in several ways. By shedding light on these limitations, we contribute to a deeper understanding of the capabilities and constraints of these models. This knowledge is essential for researchers, developers, and policymakers in making informed decisions regarding the application of Transformers in various domains.

Understanding the limitations of Transformers in compositional reasoning is crucial for developing more reliable and robust AI systems. By identifying these shortcomings, we can direct future research efforts toward addressing these limitations and developing models that exhibit improved performance in handling complex tasks requiring compositional reasoning.

We do not foresee any negative societal impacts, as our analysis aims to understand the reasons behind transformers' failures and successes, but does not introduce any new model or dataset that future work may leverage.