Faith and Fate: Limits of Transformers on Compositionality

Maurycy Borkowski

Faith and Fate:
Limits of Transformers on Compositionality

Nouha Dziri[1*], **Ximing Lu**[1,2*], **Melanie Sclar**[2*],
Xiang Lorraine Li[1†], **Liwei Jiang**[1,2†], **Bill Yuchen Lin**[1†],
Peter West[1,2], **Chandra Bhagavatula**[1], **Ronan Le Bras**[1], **Jena D. Hwang**[1], **Soumya Sanyal**[3],
Sean Welleck[1,2], **Xiang Ren**[1,3], **Allyson Ettinger**[1,4], **Zaid Harchaoui**[1,2], **Yejin Choi**[1,2]

[1]Allen Institute for Artificial Intelligence [2]University of Washington
[3]University of Southern California [4]University of Chicago

nouhad@allenai.org, ximinglu@allenai.org, msclar@cs.washington.edu

Faith

The striking discrepancy between the impressive successes of transformer LLMs on *seemingly complex* tasks and the astonishing failures on *seemingly trivial* tasks spark critical open questions about how to faithfully interpret their mixed capabilities. Under what conditions do transformers succeed, fail, and why? What types of errors do they make? Can transformers uncover implicit problem-solving rules or be taught to follow reasoning paths?

It was the epoch of belief, it was the epoch of incredulity. – Charles Dickens, A Tale of Two Cities

Model – Computation Graph

A(**x**) <- deterministic algorithm
V <- all variables values during A's execution
F_A <- all primitives (functions), A uses

G_A(**x**) = (V, E, s, op)
s(**v**) = f(u_1, u_2, …, u_n) for some f (unique definition)
Source nodes, sole output node

Layer number - length of longest path from source to **v**
Reasoning depth – max. layer number - *maximum level of multi-hop reasoning required to solve the task*
Reasoning width – mode of lenghths shortest paths from src. - *max #variables required to maintain in parallel during the computation*
Average parallelism – |V| / reasoning depth – *the average width in computation through the graph*

To be able to train and evaluate a language model's ability to follow algorithm A we must linearize $G_{A(\mathbf{x})}$. Since we only consider autoregressive models, this linearization must also be a topological ordering.

Computation Graph - example

```
function multiply (x[1..p], y[1..q]):
  // multiply x for each y[i]
  for i = q to 1
    carry = 0
    for j = p to 1
      t = x[j] * y[i]
      t += carry
      carry = t // 10
      digits[j] = t mod 10
    summands[i] = digits

  // add partial results (computation not shown)
  product = ∑_{i=1}^{q} summands[q+1-i] · 10^{i-1}
  return product
```

$A(\mathbf{x})$

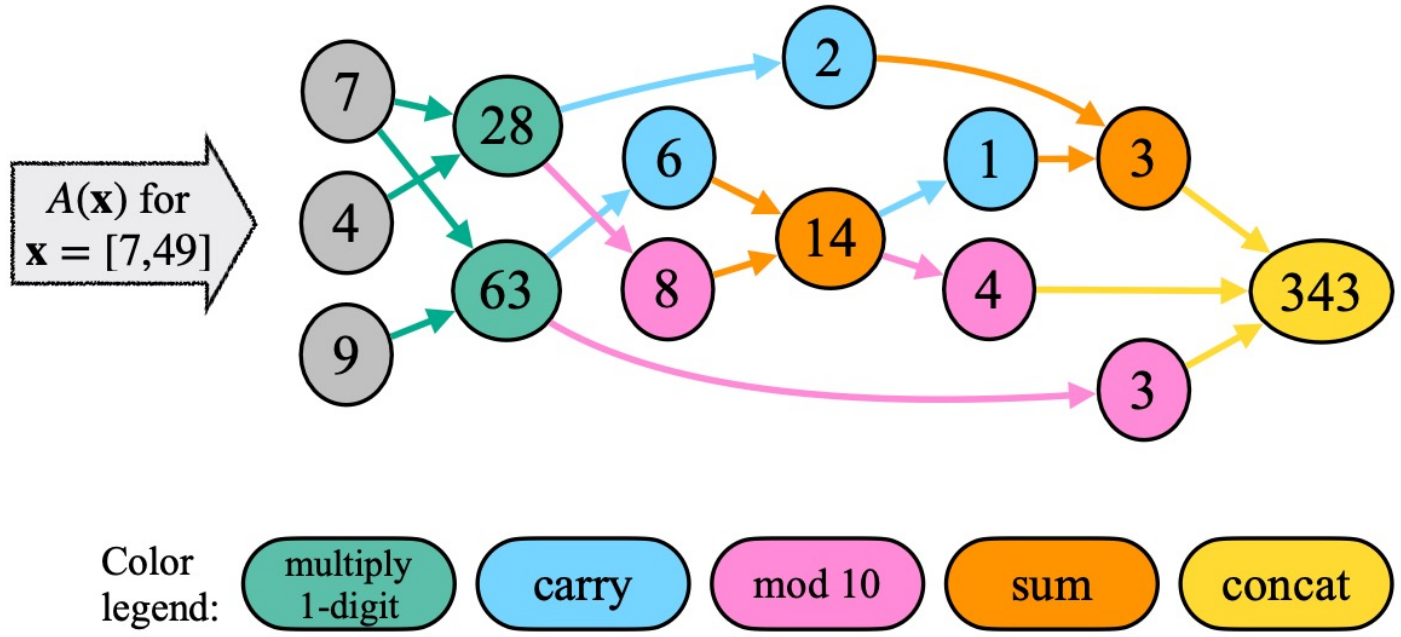


Figure 1: Transformation of an algorithm A to its computational graph $G_{A(\mathbf{x})}$. The depicted example is of long-form multiplication algorithm A, for inputs $\mathbf{x} = [7, 49]$ (i.e. computing 7×49).

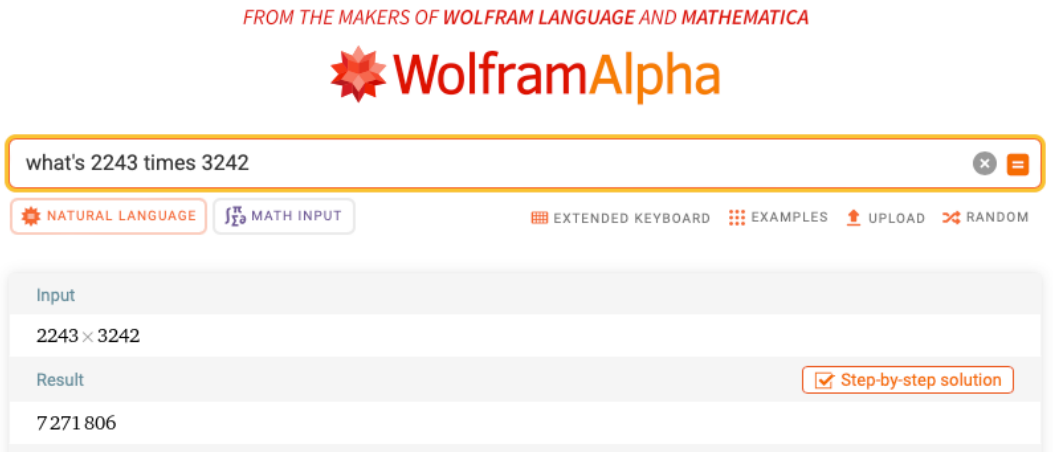
Multiplication

```
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 8: 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.

4. 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 9: A sample scratchpad for the multiplication task.



Einstein Puzzle

General Unique Rules

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, ford, camry

<UniqueValues>

Clues

1. The person who owns a Ford is the person who loves tennis.
2. Arnold is in the third house.
3. The person who owns a Camry is directly left of the person who owns a Ford.
4. Eric is the person who owns a 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.

Ground-Truth Table

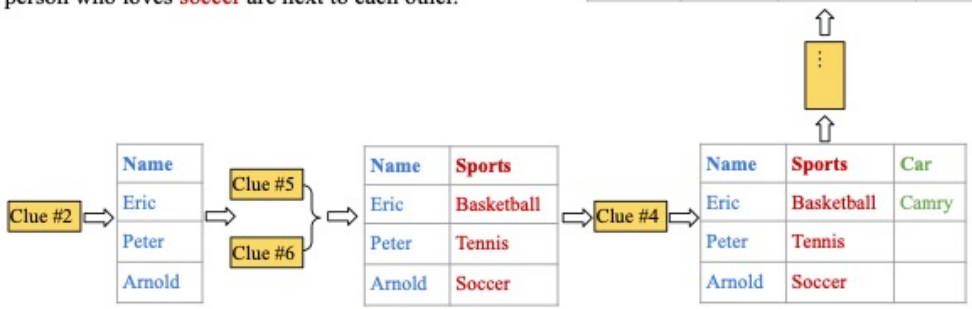
House	Name	Sports	Car
1	Eric	Basketball	Camry
2	Peter	Tennis	Ford
3	Arnold	Soccer	Tesla

Reasoning Path Generation

Algorithm 1 Puzzle Solver

Input: Clues
Output: Reasoning path

```
function PuzzleSolver(Clues)
    Path ← []
    LeftClues ← clues
    while |LeftClues| ≠ 0 do
        for i=1 to |LeftClues| do
            CandidateClues = (|LeftClues| choose i)
            for clue in CandidateClues do
                if solve any cell then
                    LeftClues.remove(clue)
                    Path.append(clue)
    return Path
```




```
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 11: A sample scratchpad for the puzzle task.

Dynamic Programming

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

|a_i|<=5

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

$$dp_{n-1} = \max(a_n, a_{n-1}, 0)$$

$$dp_n = \max(a_n, 0)$$

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

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

Problems – Computation Graphs

Multiplication

- F_a = {one-digit-mult, sum, mod10, carry over, concat.}
- S = {input_numbers}
- O = {mult result}

Einstein Puzzle

- F_a = {elimination function*}
- S = {input clues}
- O = {solution matrix}

*deterministically fills the cell(s) that requires the minimum number of clues among all current unfilled cells

Dynamic Programming

- F_a = {equals, and, not, inficator function, sum, max}
- S = {input item list}
- O = {items selection}

Experiment setup

- Models: GPT3, GPT3.5 (ChatGPT), GPT4, FlanT5, Llama
- Evaluation: jan 2023 - may 2023 (OpenAI API)
- Methods: zero-shot, few-shot, finetuning:
  - GPT3 (text-davinci-003)
    - epochs: {14,12,4} / {16,8,2}
    - temp: 1 (inference)
    - learning rate: 0.2
    - batch size: 0.2%
    - size(test): 500

task	size(training)	range
multiplication	1.8M	1x1 to 4x2
puzzle	142k	2x2 to 4x4
DP	41k	5

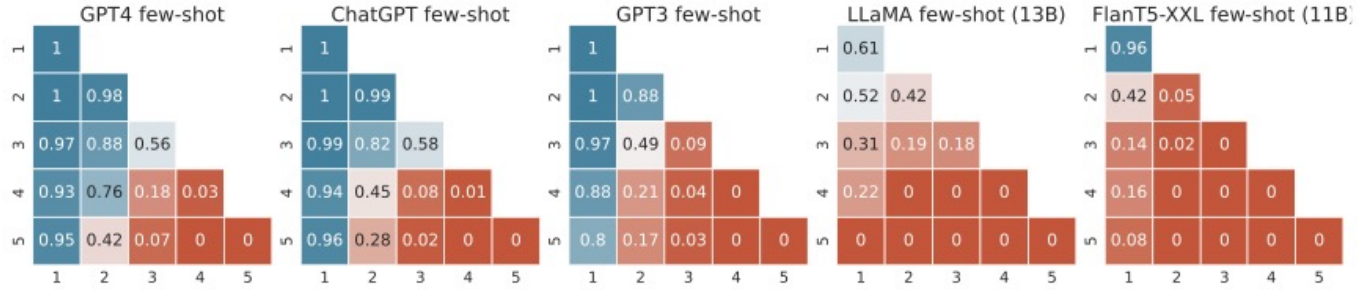
Problem size	# examples	GPT3 Cost	
		without scratchpad	with scratchpad
1 x 1	81	$0.12	$7.44
2 x 1	810	$1.28	$74.4
2 x 2	8100	$12.96	$744
3 x 1	8100	$12.96	$744
3 x 2	81000	$129.6	$7440
3 x 3	810000	$1296	$74,404
4 x 1	81000	$129.6	$7440
4 x 2	810000	$1296	$74,404
4 x 3	8100000	$12,960	$744,040
4 x 4	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.

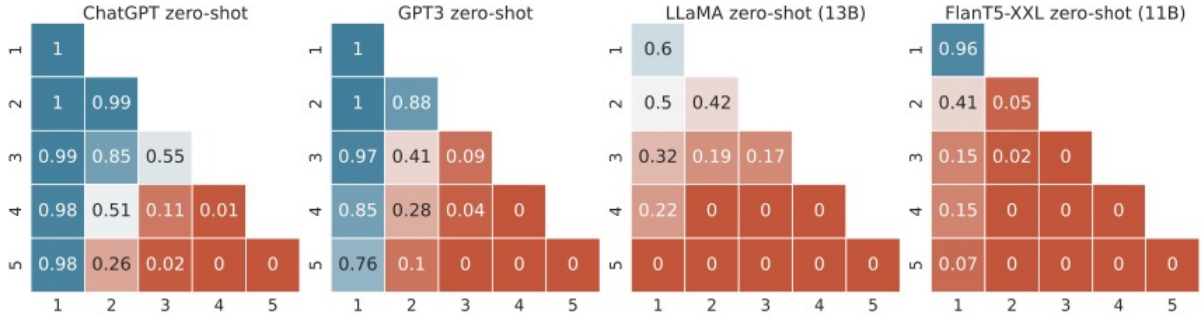
- Scratchpads *are* a verbalization of the computation graphs
- Trained with: question-answer and question-scratchpads pairs

[5]The training duration for question-answer pairs is equivalent to 60 epochs and costs 50,000 USD. Training on question-scratchpad pairs was conducted for 40 epochs and costs 40,000 USD.

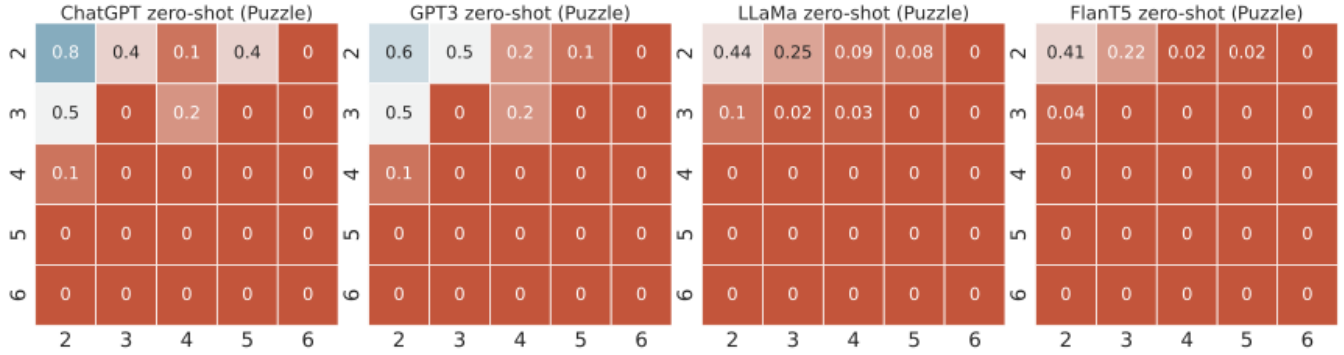
Zero- and Few-shot



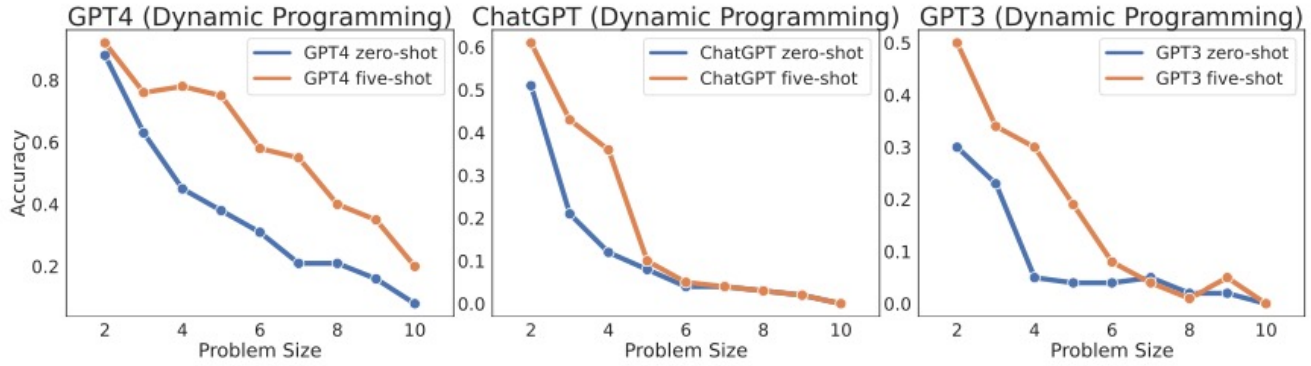
- Multiplication



- Puzzle

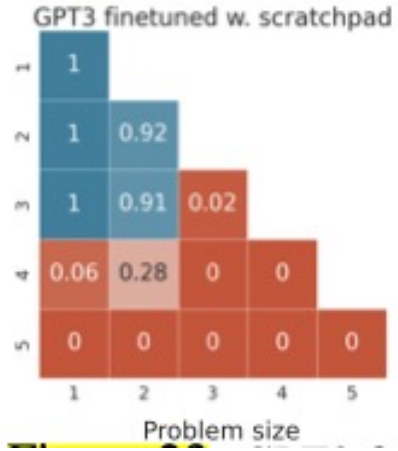


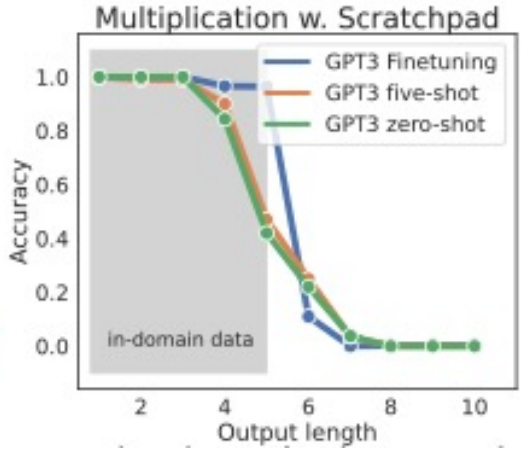
- DP

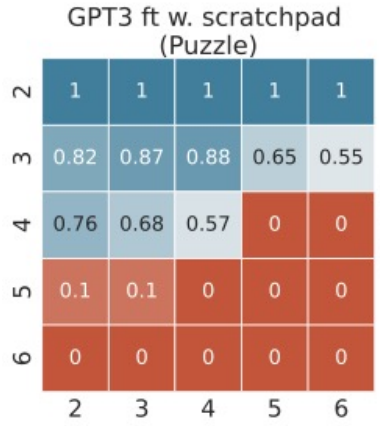


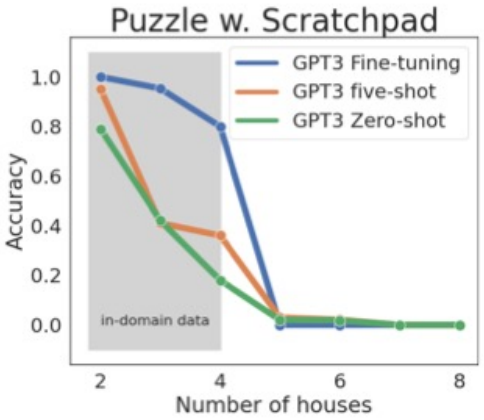
Finetuning

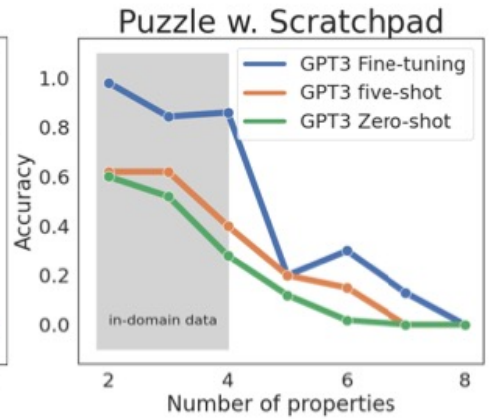
- Multiplication
- Puzzle
- DP

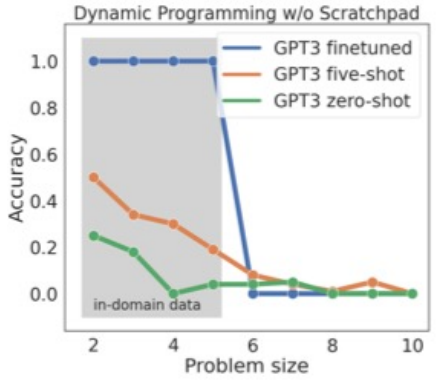


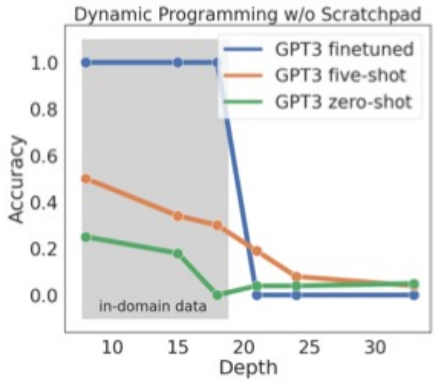


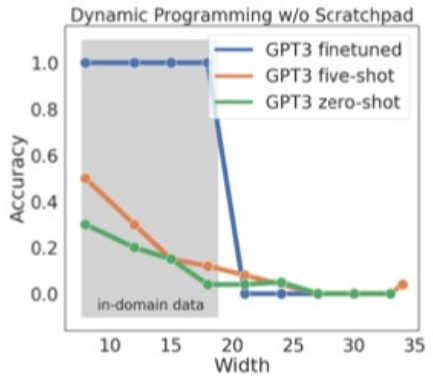












Results

- *GPT3 faster convergence when training on question-scratchpad pairs rather than question-answer pairs*
- *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.*
- *Even if grokking were to emerge through more prolonged training, such an approach would prove inefficient and unscalable. Future work is required to accurately explain when and how grokking occurs.*
- *Autoregressive characteristic of transformers, which forces them to tackle problems sequentially, presents a fundamental challenge that cannot be resolved by instructing the model to generate a step-by-step solution. Instead, models depend on a greedy process of producing the next word to make predictions without a rigorous global understanding of the task*

Surface Patterns

$$\textbf{RelativeIG}(Y_j, X) = \frac{H(Y_j) - H(Y_j|X)}{H(Y_j)} \in [0, 1]$$

These experiments suggest that if an output element heavily relies on a single or a small set of input features, transformers are likely to recognize such correlation during training and directly map these input features to predict the output element in testing, without going through the rigorous multi-hop reasoning and giving a false illusion of performing compositional reasoning.

		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.

Computaion Graph - results

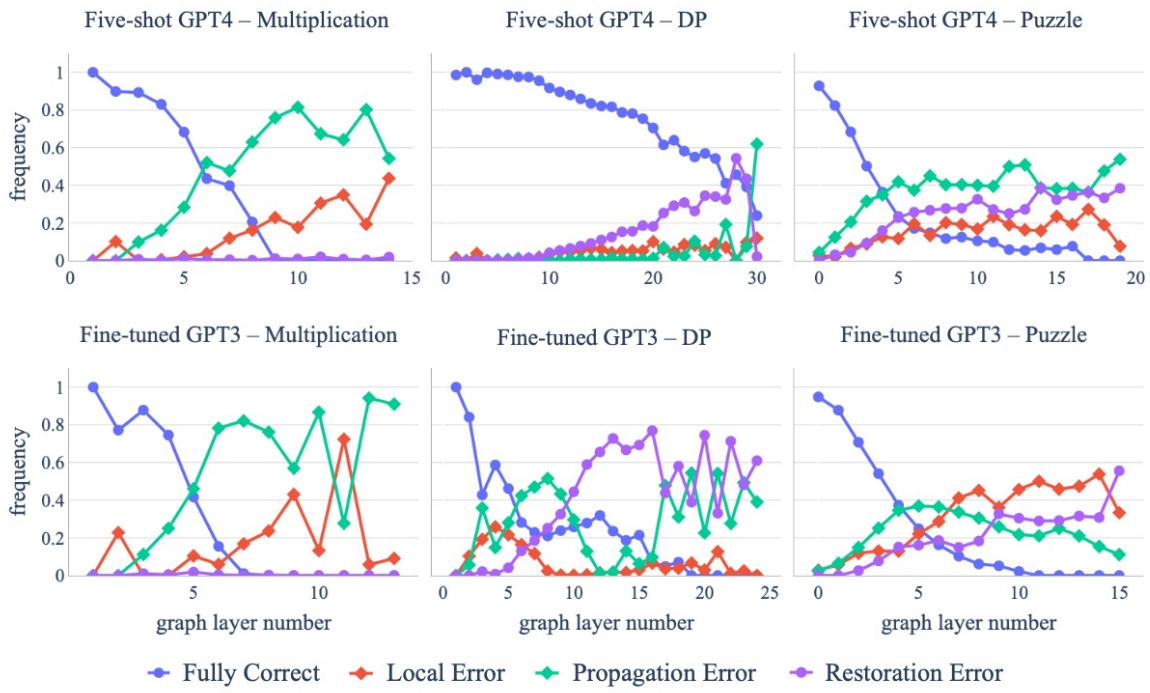
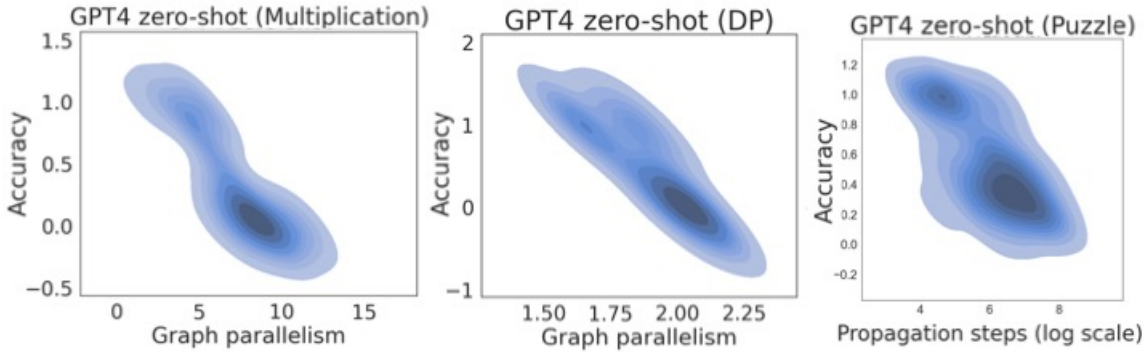


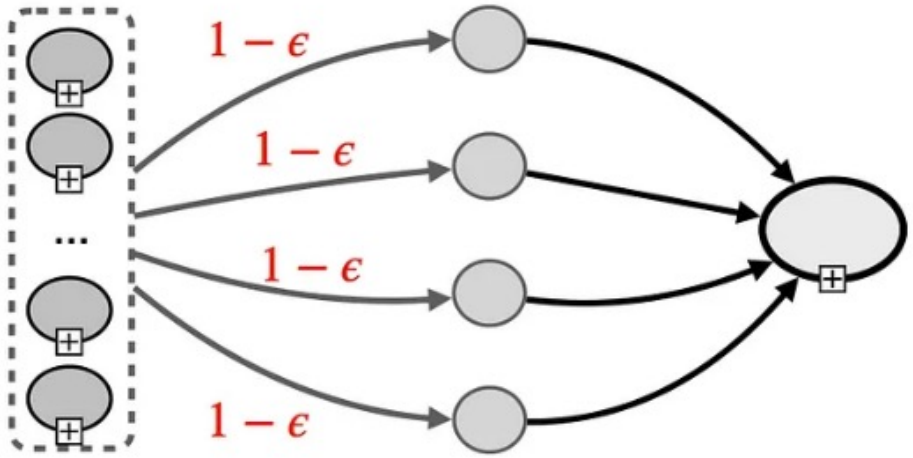
Figure 7: Ratio of nodes in each of the four correct/error categories for each layer in computation graph. Results shown are for few-shot prompting and fine-tuning with scratchpad.



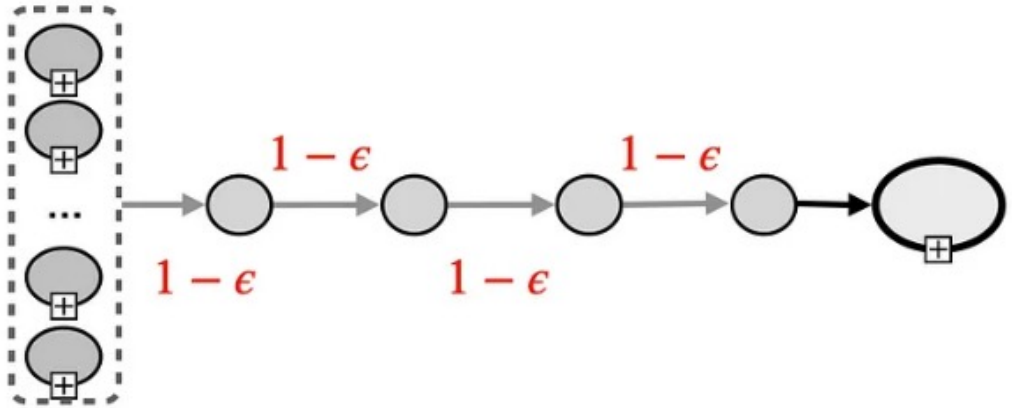
Both phenomena suggest that models are able to correctly perform single-step reasoning, potentially due to memorizing such single-step operations during training, but fail to plan and compose several of these steps for an overall correct reasoning.

Dziękuję za uwagę!

If the probability of making an error in a single reasoning step is ϵ, probability of success is...



$$\approx (1-\epsilon)^n$$



$$\approx (1-\epsilon)^n$$