

北京大学
PEKING UNIVERSITY

Memorization or Generalization: Some Empirical Results

Faith and Fate: Limits of Transformers on Compositionality (NeurIPS, 2023)

Case-Based or Rule-Based: How Do Transformers Do the Math? (ICML, 2024)

Number Cookbook: Number Understanding of Language Models and How to Improve It (under review)

Yi Hu, Haotong Yang
Peking University

北京大学
PEKING UNIVERSITY

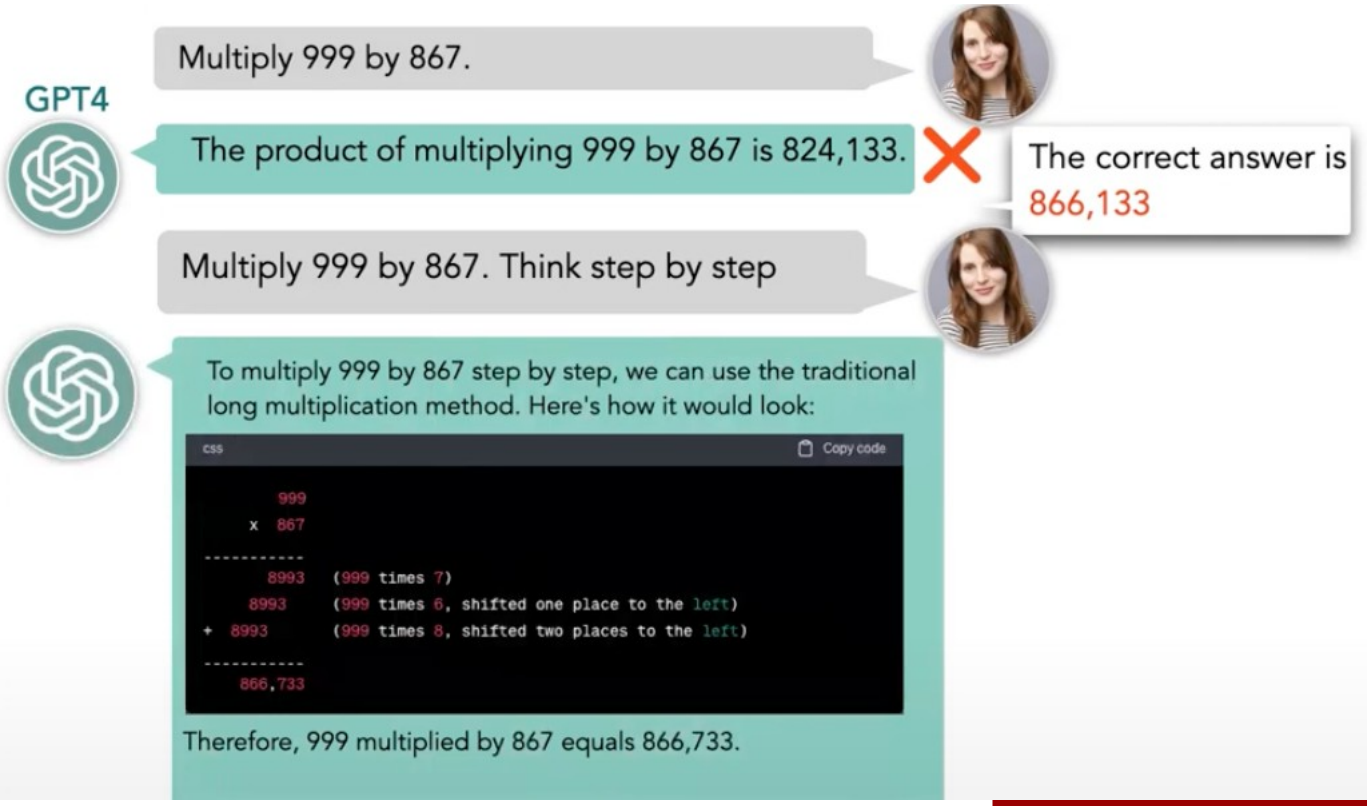
Faith and Fate: Limits of Transformers on Compositionality

NeurIPS 2023 (spotlight)

Does “sparks of AGI” solve multiplication?

北京大学 PEKING UNIVERSITY

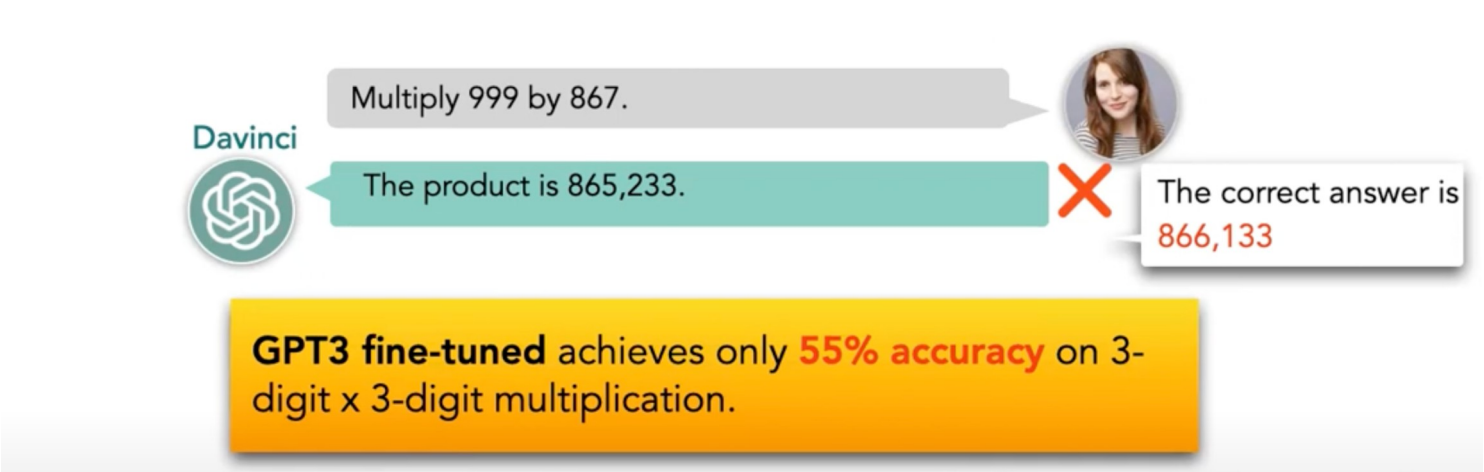
- LLMs have shown exceptional performance on multi-step reasoning tasks, but they simultaneously show failures on surprisingly trivial problems.
- for example, GPT-4 fail to solve multi-digit multiplication
  - gives an answer looks plausible, however wrong (in both direct answer and CoT settings)
  - GPT-4 achieves only **59% accuracy** on 3-digit × 3-digit multiplication, **4%** for 4×4, **0%** for 5×5



What about supervised training? (w/ GPT-3)

北京大学 PEKING UNIVERSITY

- fine-tune GPT-3 exhaustively does not solve multiplication
  - 1.8M examples include 1×1, 2×1, 2×2. 3×1, 3×2, 3×3, 4×1, 4×2 multiplication problem sizes, 12 epoch, 8/1/1 train/dev/test
- maybe tokenization is the issue? → Let's train GPT-2 from scratch



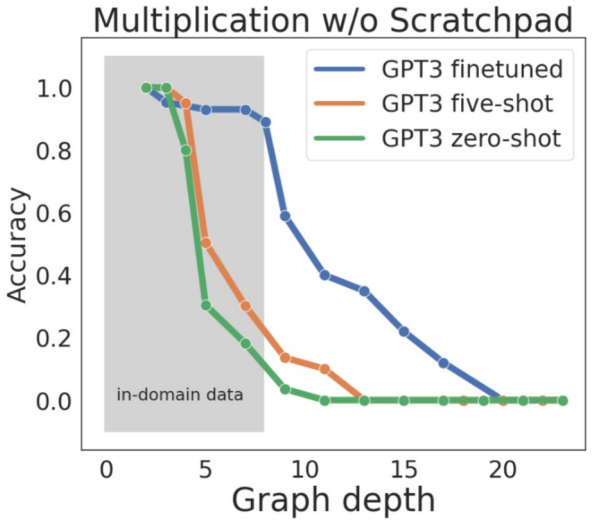
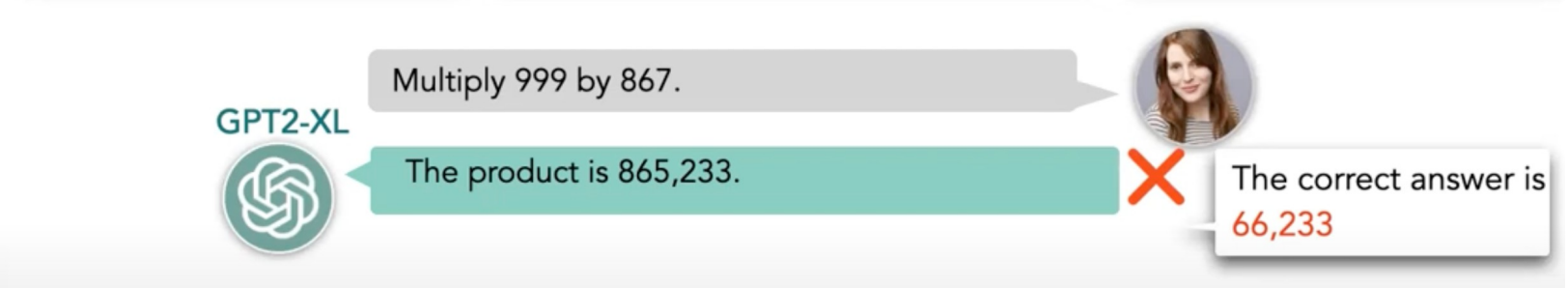


Figure 19: GPT3 finetuned on the multiplication task using **question-answer** pairs

Supervised training with correct tokenization? (GPT-2 XL from scratch)

北京大学 PEKING UNIVERSITY

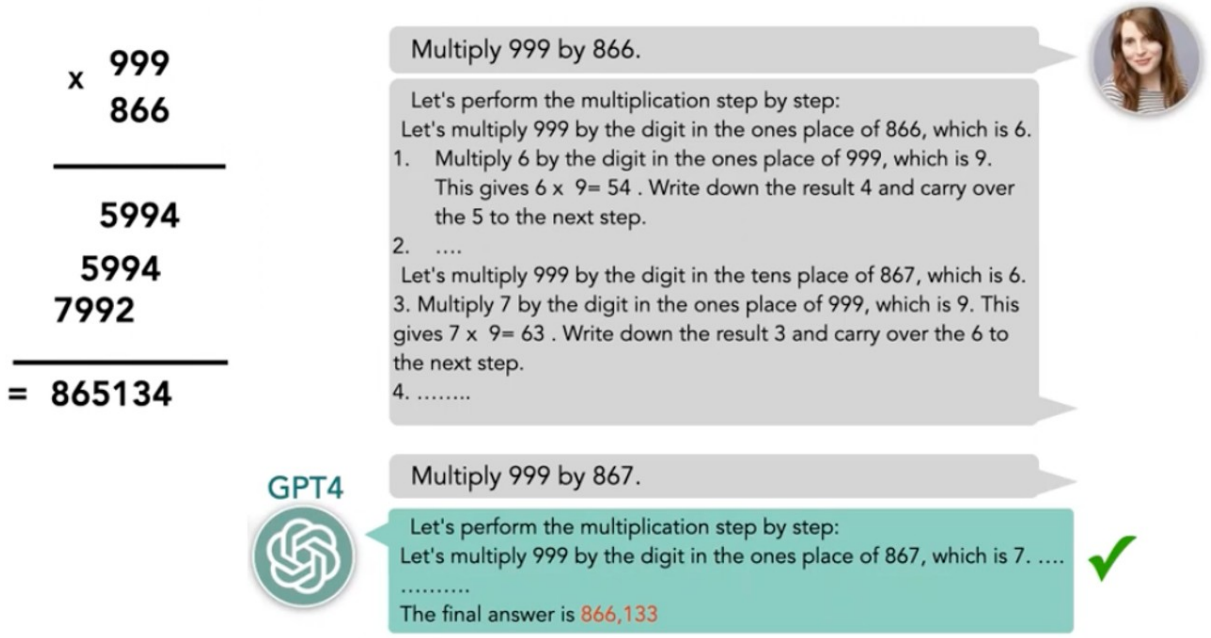
- train GPT-2 XL from scratch
  - each digit is assigned to one token
  - each math symbol (e.g. "=", "×") is assigned to one token
  - 90M examples include 1×1, 2×1, 2×2. 3×1, 3×2, 3×3, 4×1, 4×2, 4×3, 4×4 multiplication problem sizes, 14 epoch, 8/1/1 train/dev/test
- **tokenization does not seem the issue** for the low performance
  - GPT2-XL achieves only **10% accuracy** on 3-digit × 3-digit multiplication
- → maybe step-by-step? GPT-4 + scratchpad

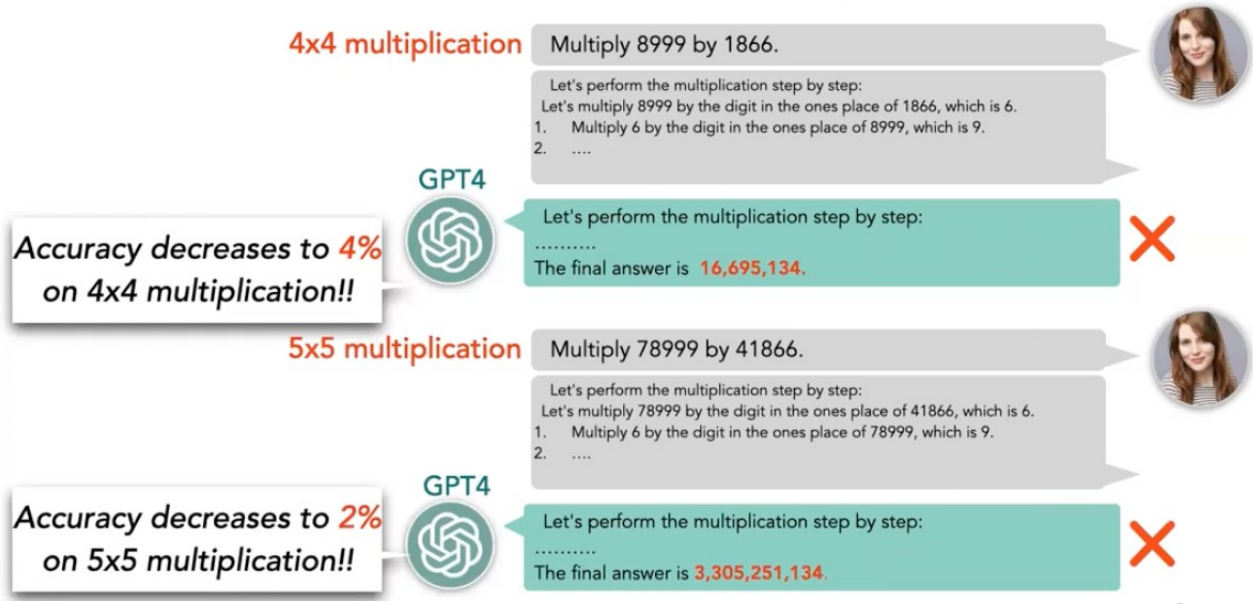


Just a matter of step-by-step? (w/ GPT-4+scratchpad)

北京大学 PEKING UNIVERSITY

- few-shot scratchpad with GPT-4
  - 5-shot 3-digit × 3-digit exemplars in each prompt
  - accuracy increases to **92%** on 3×3 multiplication
- How about more complex compositions?
  - **4%** for 4×4 multiplication
  - **2%** for 5×5 multiplication

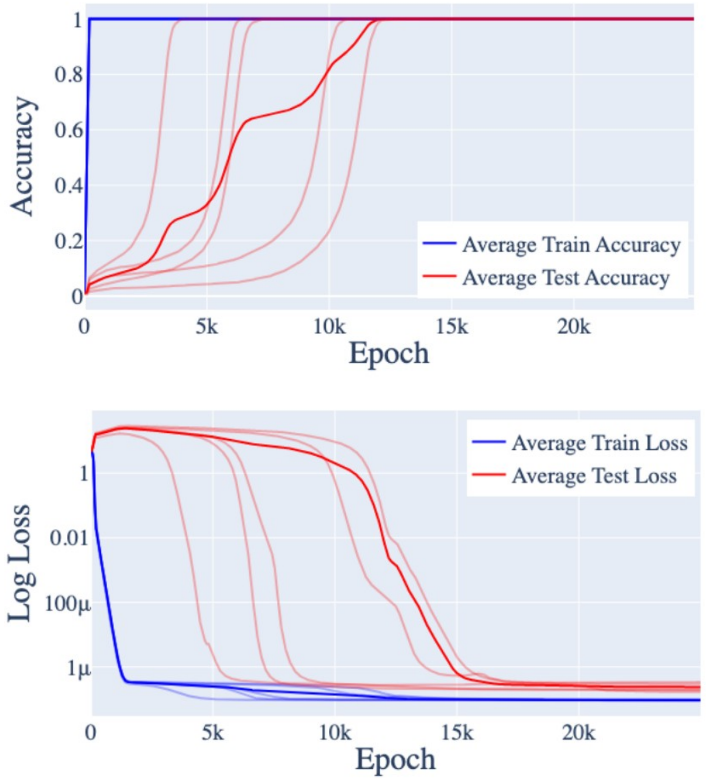


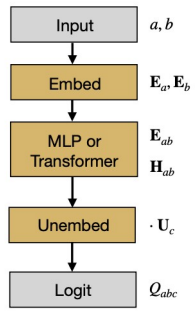


What about grokking?

北京大学 PEKING UNIVERSITY

- grokking
  - train long enough after overfitting may solve the problem
  - may develop systematic generalization after prolonged training
  - small models on modular addition in original work





Step 1: Embed token a and b to a circle where $w_k = 2\pi k/p$ for some $k \in [1,2\cdots,p-1]$

$a \to \mathbf{E}_a \equiv (\mathbf{E}_{a,x}, \mathbf{E}_{a,y}) = (\cos(w_k a), \sin(w_k a)), b \to \mathbf{E}_b \equiv (\mathbf{E}_{b,x}, \mathbf{E}_{b,y}) = (\cos(w_k b), \sin(w_k b))$

Clock Algorithm

Step 2: compute the angle sum using multiplication.

$$\mathbf{E}_{ab} \equiv \begin{pmatrix} \mathbf{E}_{ab,x} \\ \mathbf{E}_{ab,y} \end{pmatrix} = \begin{pmatrix} \mathbf{E}_{a,x}\mathbf{E}_{b,x} - \mathbf{E}_{a,y}\mathbf{E}_{b,y} \\ \mathbf{E}_{a,x}\mathbf{E}_{b,y} + \mathbf{E}_{a,y}\mathbf{E}_{b,x} \end{pmatrix} = \begin{pmatrix} \cos(w_k(a+b)) \\ \sin(w_k(a+b)) \end{pmatrix}$$

$\mathbf{H}_{ab} = \mathbf{E}_{ab}$

Pizza Algorithm

Step 2.1: compute the vector mean.

$\mathbf{E}_{ab} = (\mathbf{E}_a + \mathbf{E}_b)/2 = (\cos(w_k a) + \cos(w_k b), \sin(w_k a) + \sin(w_k b))/2$

Step 2.2: using $\mathbf{E}_{ab}$ and nonlinearities to compute $\mathbf{H}_{ab}$

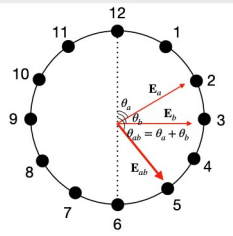
$\mathbf{H}_{ab} = |\cos(w_k(a-b)/2)|(\cos(w_k(a+b)), \sin(w_k(a+b)))$

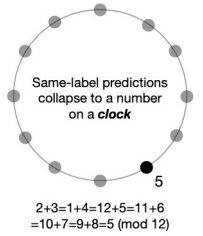
Step 3: score possible outputs c using a dot product.

$Q_{abc} = \mathbf{U}_c \cdot \mathbf{H}_{ab}$, $\mathbf{U}_c \equiv (\mathbf{E}_{c,x}, \mathbf{E}_{c,y}) = (\cos(w_k c), \sin(w_k c))$

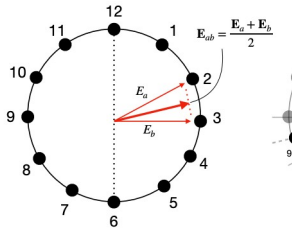
$Q_{abc}(\text{Clock}) = \cos(w_k(a+b-c))$

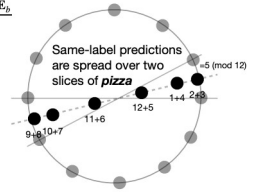
$Q_{abc}(\text{Pizza}) = |\cos(w_k(a-b)/2)| \cos(w_k(a+b-c))$





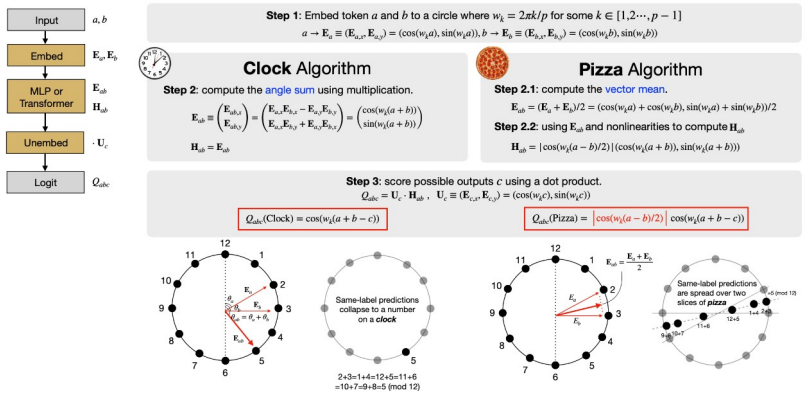
2+3=1+4=12+5=11+6
=10+7=9+8=5 (mod 12)

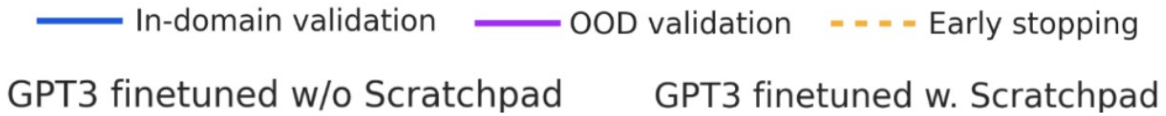




What about grokking?

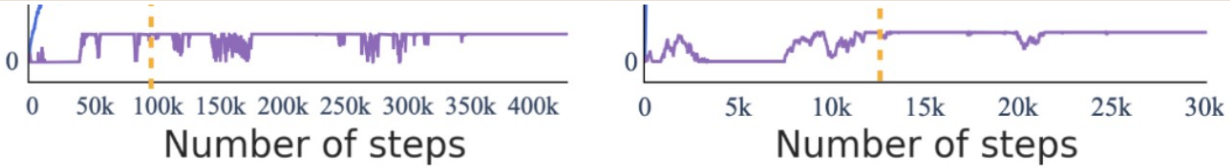
- grokking
  - train long enough after overfitting may solve multiplication
  - may develop systematic generalization after prolonged training
  - small models on modular addition in original work
- finetune GPT-3 both w/ and w/o scratchpad for a prolonged period of time





GPT-3 exhibits **poor performance** on OOD data even after prolonged training period.

→ The extent to which generalization improves beyond overfitting is **intricately linked to the inherent complexity of the task**.

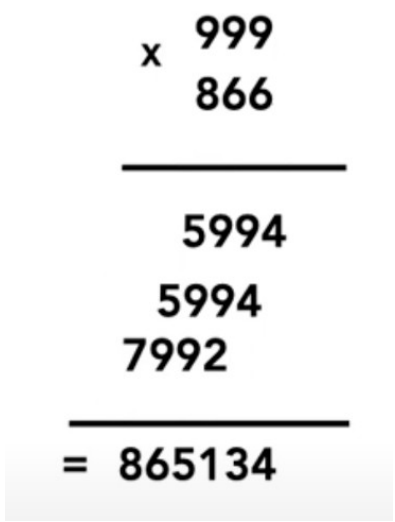


Understanding transformers' behavior via computation graphs

- tasks: compositional multi-hop reasoning
  - **multiplication**
  - **dynamic programming**: maximum sum of a subsequence that does not include adjacent elements
  - **Einstein's puzzle**

```
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].
```



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

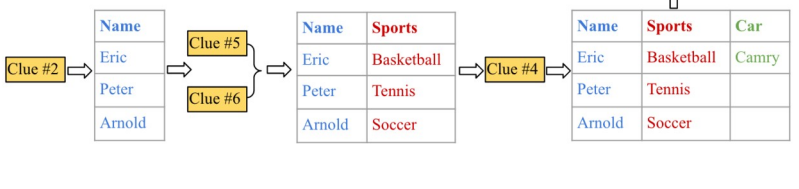
House	Name	Sports	Car
1	Eric	Basketball	Cami
2	Peter	Tennis	Ford
3	Arnold	Soccer	Tesla

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
```




```
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].
```

multiplication

Einstein's puzzle

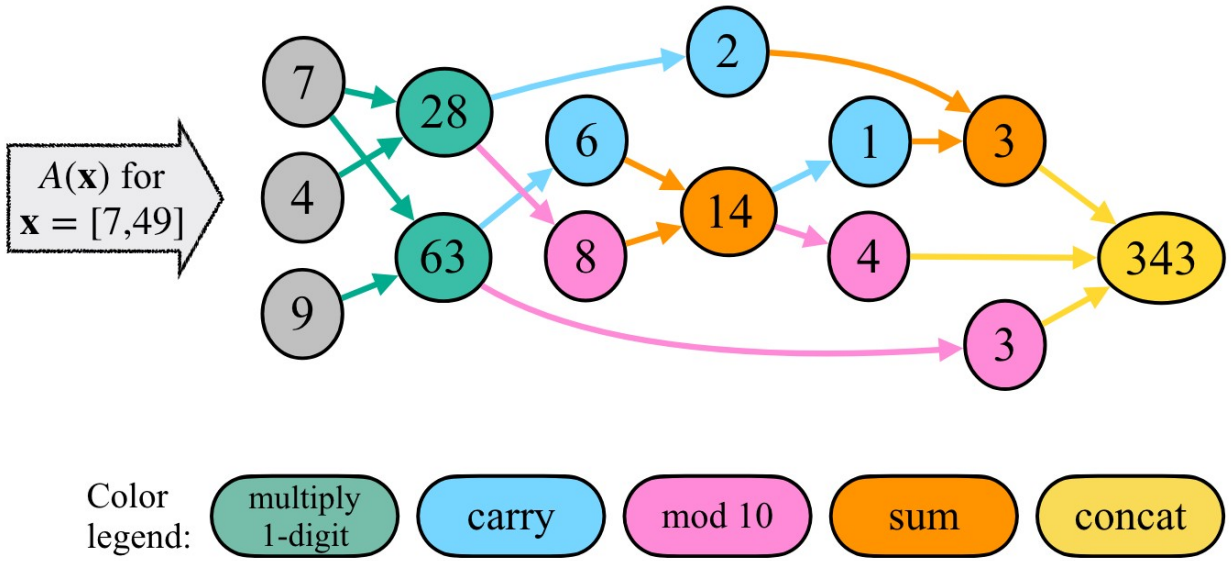
dynamic programming

Computation graph for 7×49

```
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

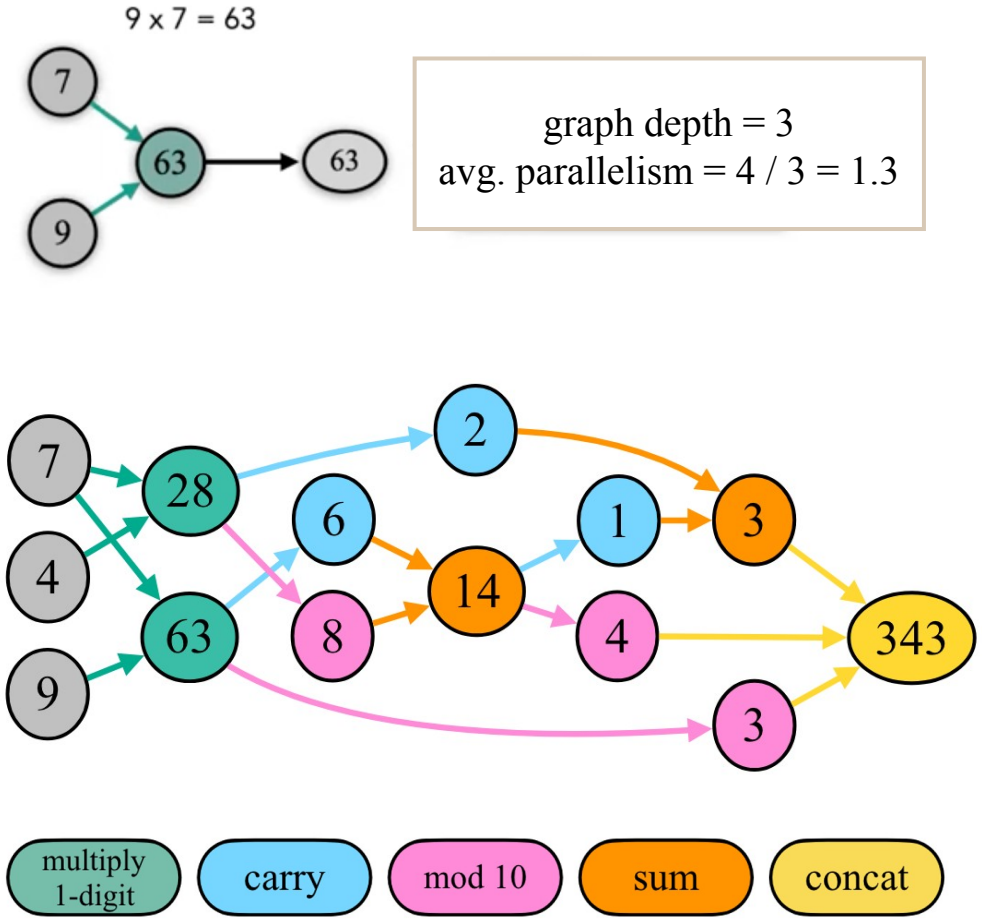
  // add partial results (computation not shown)
  product = ∑_{i=1}^{q} summands[q+1-i] · 10^{i-1}
  return product
```

$A(\mathbf{x})$




```
   7
x 49
-----
  63
 28
-----
 343
```

Model performance decreases as graph complexity increases



Graph Complexity

- **layer number:** the length of the longest path from a source node to v
- **graph depth:** the largest layer number in the graph
- **average parallelism:** ratio between $|V|$ and reasoning depth

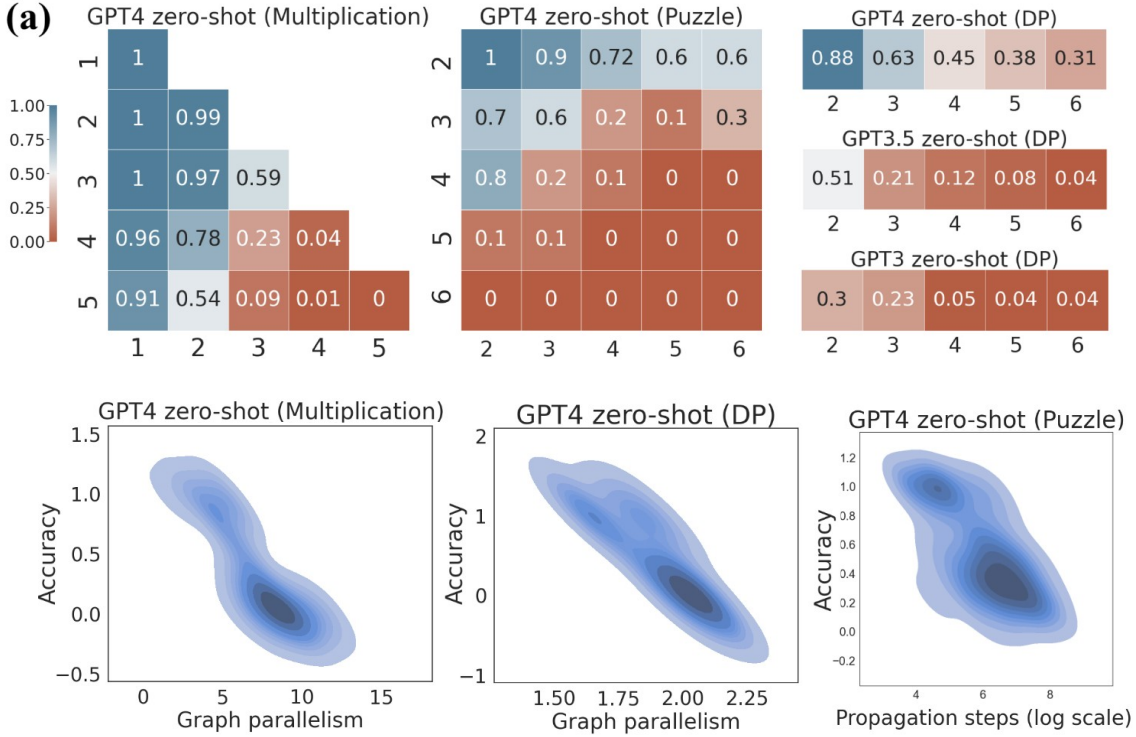


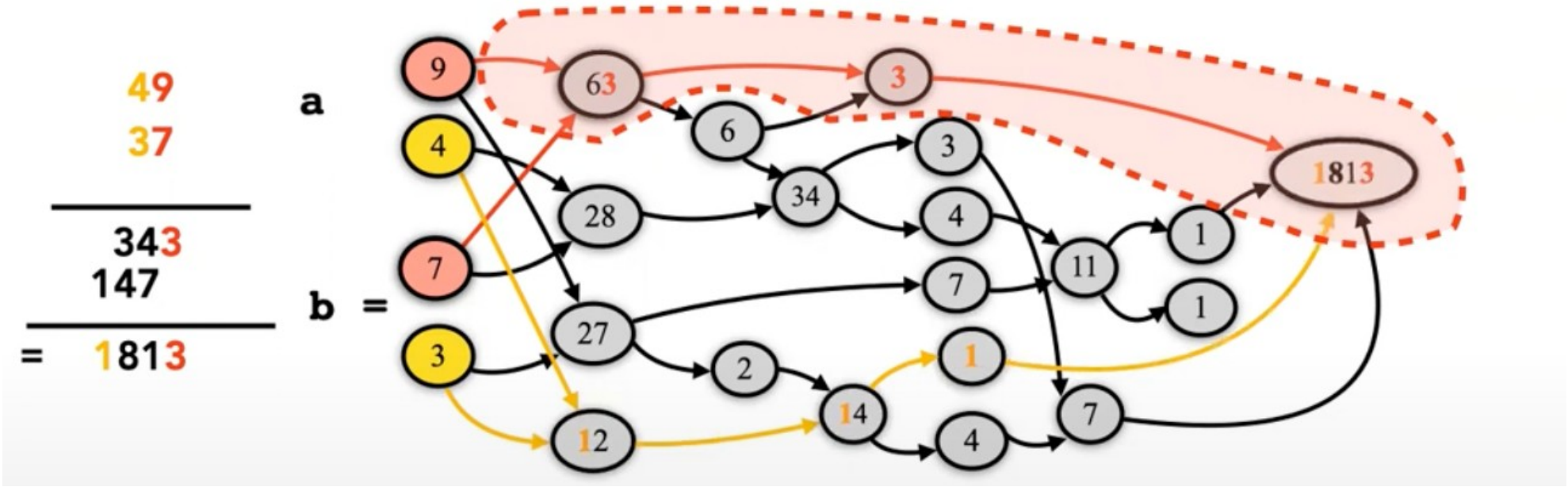
Figure 2: (a) **Zero-shot accuracy**. Axes refer to problem sizes (number of digits in multiplication, number of houses and attributes in puzzle, and sequence length in the DP task). Transformers' accuracy decreases to near zero as task complexity increases, measuring task complexity by the problem size. (b) **Average parallelism** negatively correlates with accuracy.

Information gain explains where transformers partially excel

北京大学 PEKING UNIVERSITY

- transformers predict partially correct answer even when the overall response is wrong
- measure the amount of information gained about an output element Y_j by observing a subset of input random variable $X \subset \{X_1, \cdots, X_n\}$

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



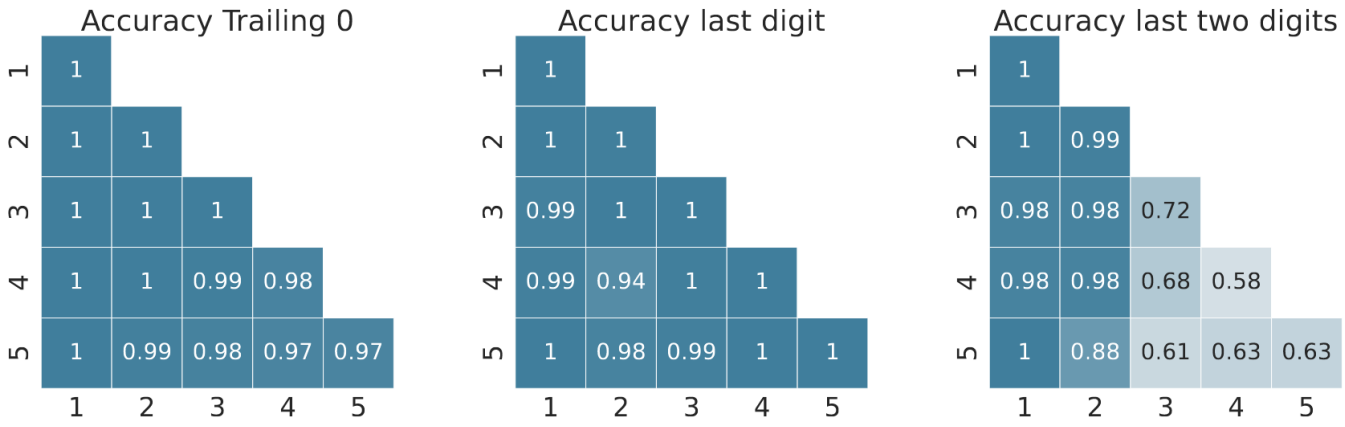
Information gain explains where transformers partially excel

PEKING UNIVERSITY

- transformers predict partially correct answer even when the overall response is wrong
- measure the amount of information gained about an output element Y_j by observing a subset of input random variable $X \subset \{X_1, \cdots, X_n\}$

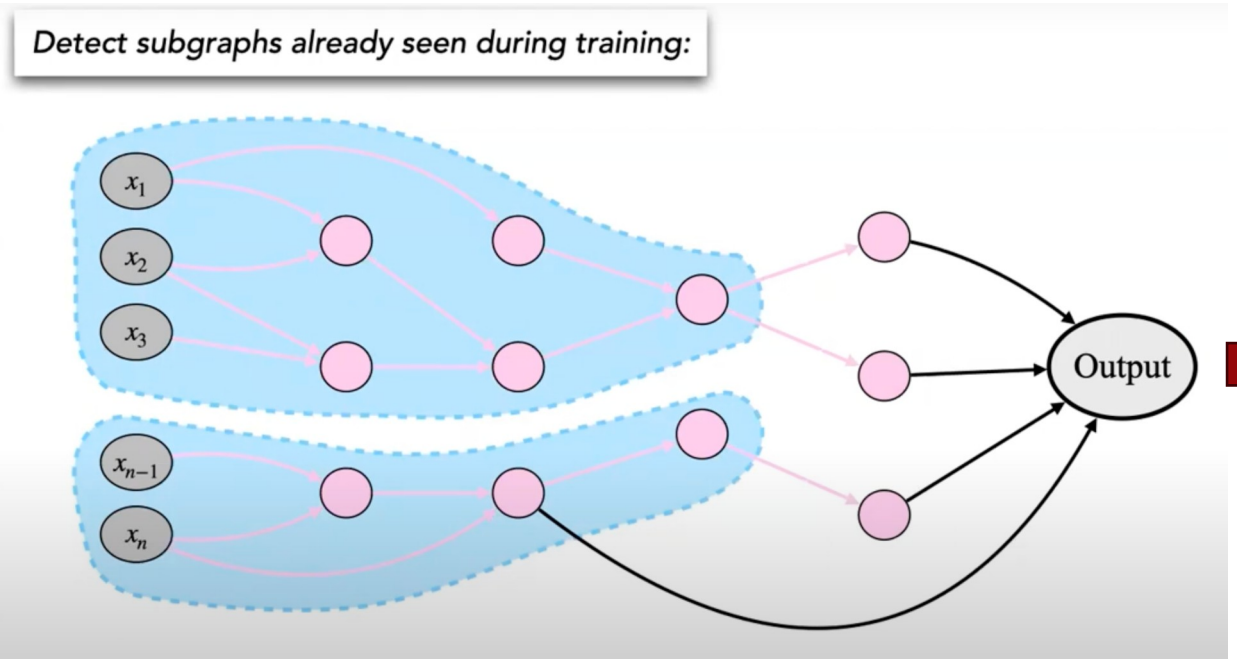
		Relative Information Gain			
Input variable	Output variable	2x2	3x3	4x4	5x5
x_n	z_{2n}	0.223	0.223	0.223	0.223
y_n	z_{2n}	0.223	0.223	0.223	0.223
x_1	z_1	0.198	0.199	0.199	0.199
y_1	z_1	0.198	0.199	0.199	0.199
$x_n\ y_n$	z_{2n}	1.000	1.000	1.000	1.000
$x_{n-1}\ x_n$	z_{2n}	0.223	0.223	0.223	0.223
$y_{n-1}\ y_n$	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

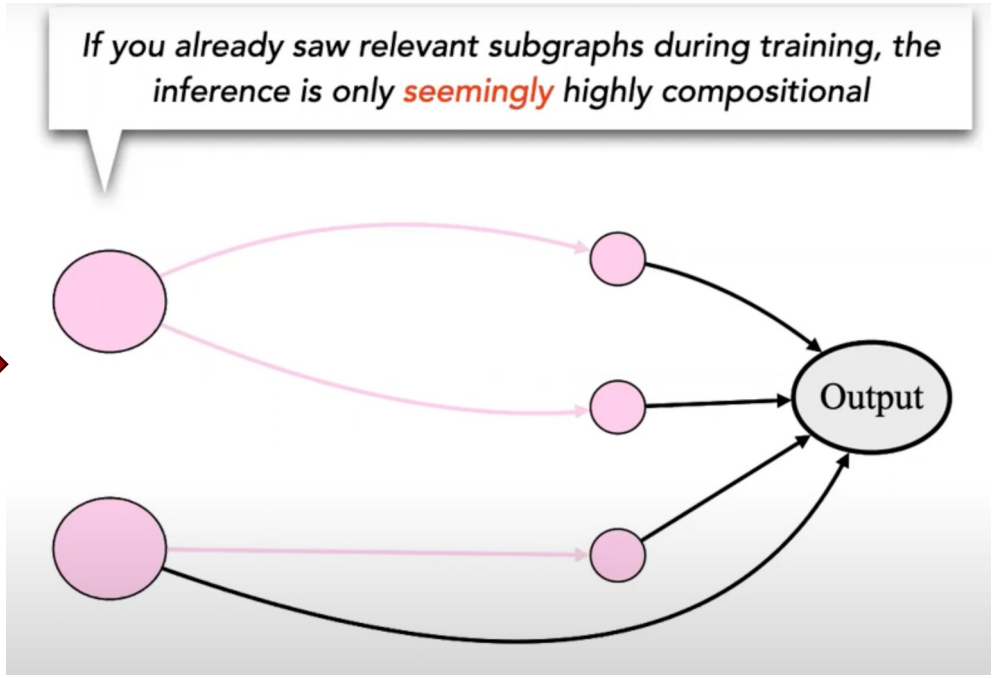
Table 2: **Highest Relative Information Gain Elements and Pairs of Elements**, for multiplications between $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, with $2 \leq n \leq 5$. We define $z := x \cdot y$, which will always have size $2n$ (with possibly a leading zero). z_{2n} denotes the least-significant 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.



GPT-4 zero shot

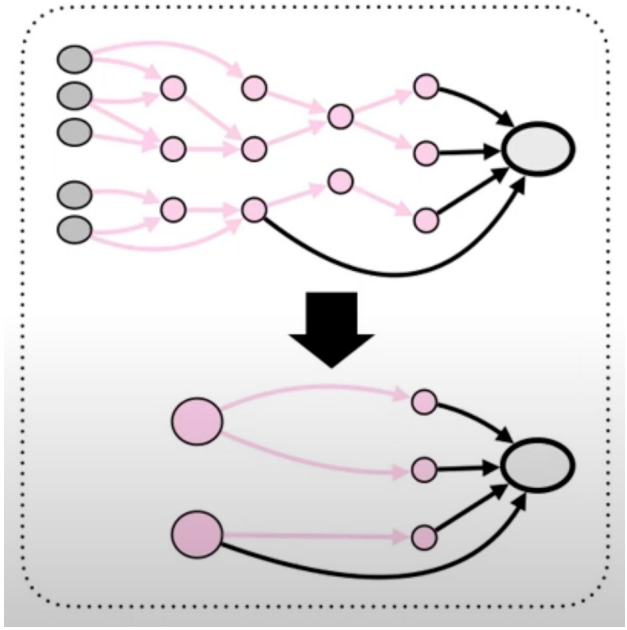
What is the correlation between a model generating a correct output and having seen relevant subgraphs during training?

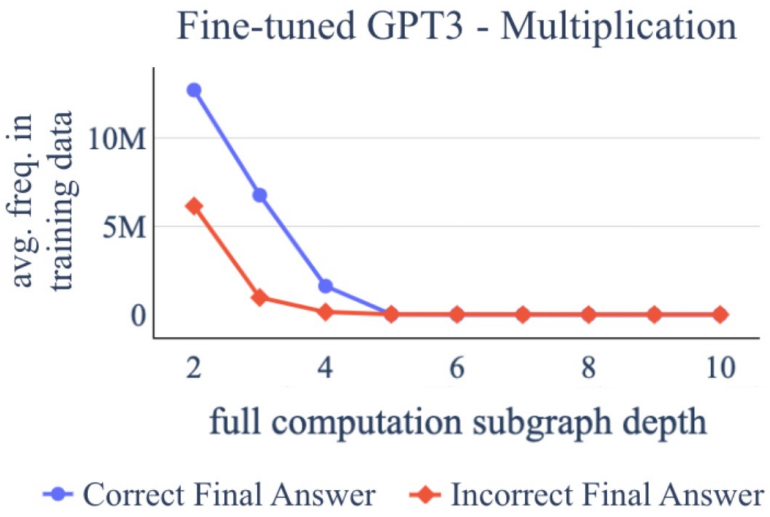


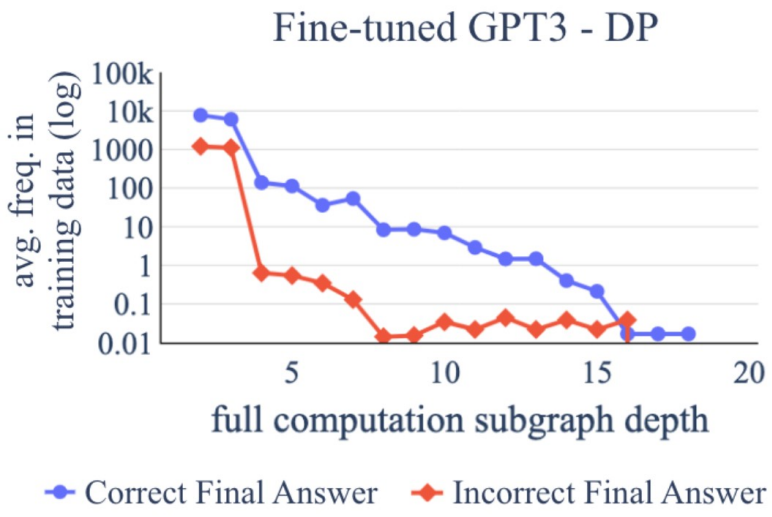


Transformers' reasoning mechanism

- Transformers' **successes are heavily linked to having seen** significant portions of the requires computation **graphs during training**





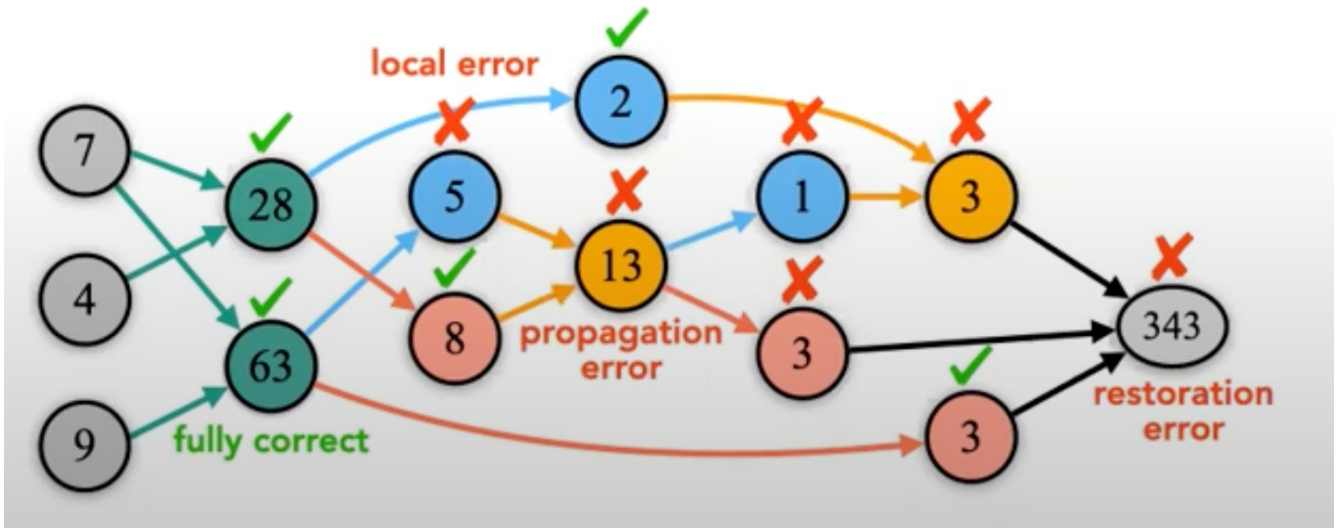


What types of errors do transformers make?

北京大学 PEKING UNIVERSITY

Error Type

- **Fully Correct:** v and its ancestors have correct values and are derived from correct computations.
- **Local Error:** v is derived from an incorrect computation but its parent nodes have correct values
- **Propagation Error:** v is derived from a correct computation but some of its parent nodes have incorrect values
- **Restoration Error:** v has a correct value but is derived from an incorrect computation

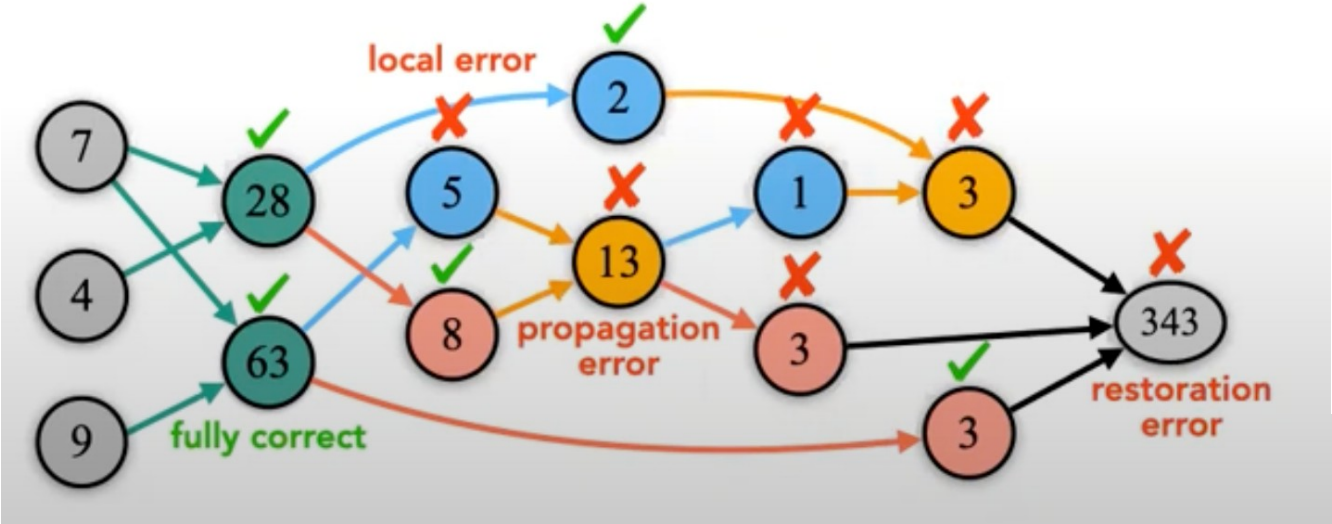


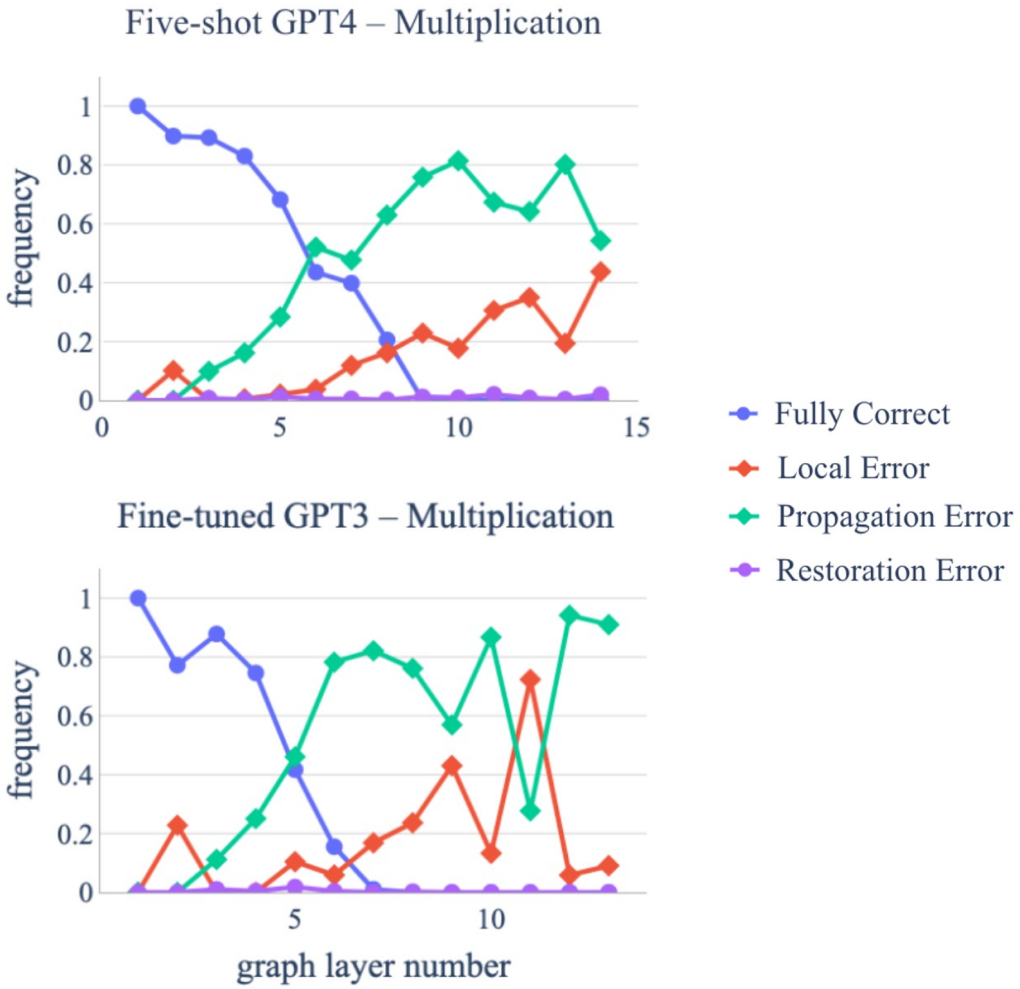
What types of errors do transformers make?

北京大学 PEKING UNIVERSITY

Error Type

- **Fully Correct:** v and its ancestors have correct values and are derived from correct computations.
- **Local Error:** v is derived from an incorrect computation but its parent nodes have correct values
- **Propagation Error:** v is derived from a correct computation but some of its parent nodes have incorrect values
- **Restoration Error:** v has a correct value but is derived from an incorrect computation





北京大学
PEKING UNIVERSITY

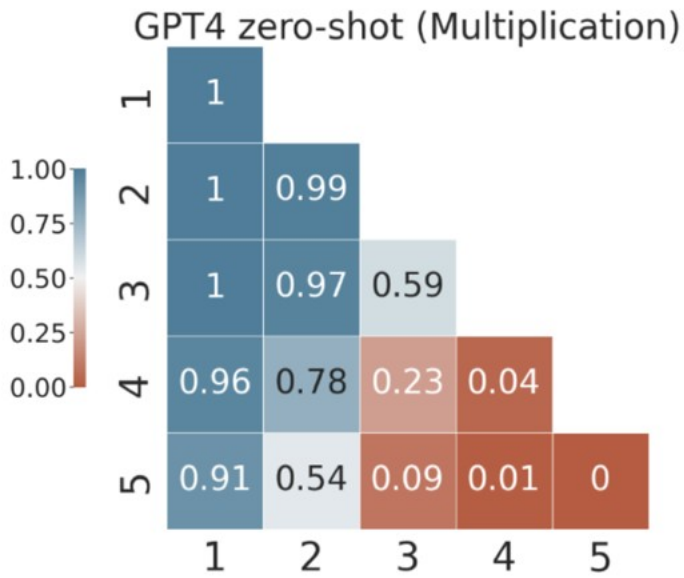
Case-Based or Rule-Based: How Do Transformers Do the Math?

ICML 2024

LLMs Towards AGI?

北京大学 PEKING UNIVERSITY

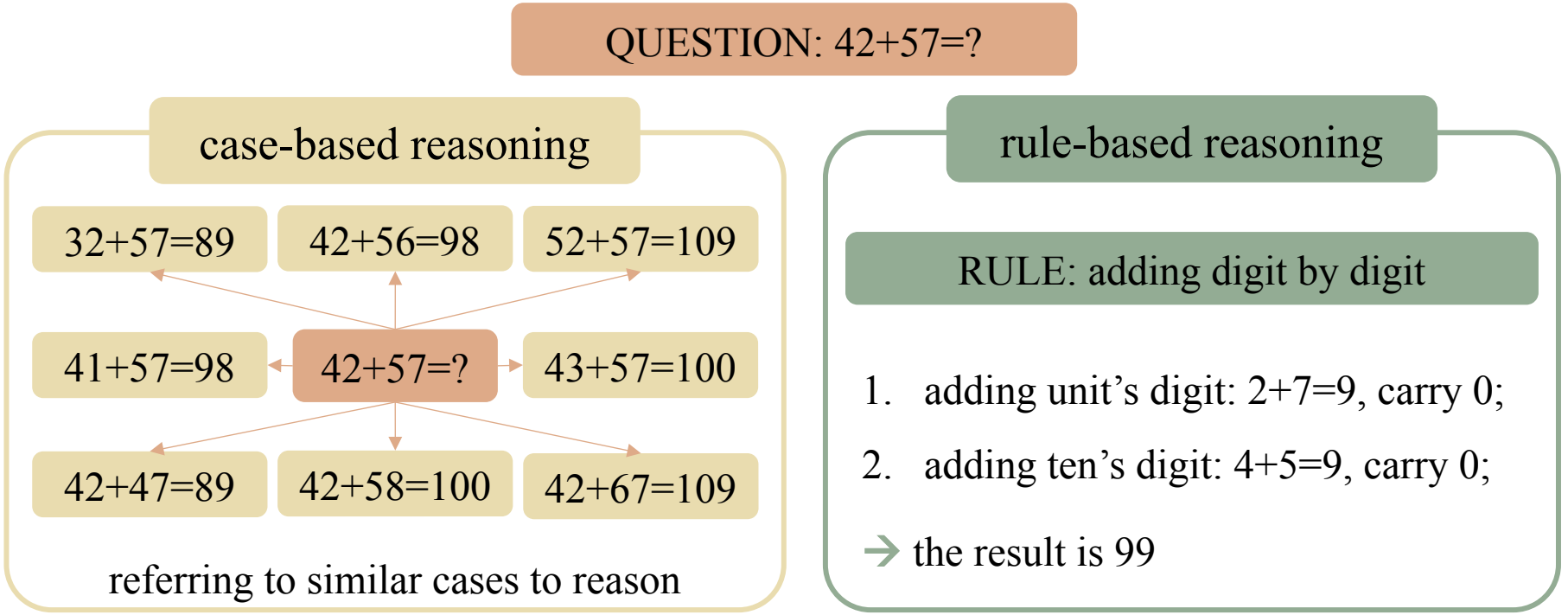
- Remarkable capabilities on a wide range of classical NLP tasks
- However still **puzzled by basic calculation tasks**, e.g. addition, multiplication
- **Length generalization**



accuracy of zero-shot GPT-4 on multiplication of numbers of multiple digits

figure from Dziri et al., Faith and Fate: Limits of Transformers on Compositionality, 2023

Two Reasoning Paradigms (1)



	case-based reasoning	rule-based reasoning
pros	may provide some shortcuts for fast inference	essential to achieve systematic generalization
cons	sensitive to the set of seen cases; set limitations to generalization	do not rely on shortcuts

Two Reasoning Paradigms (2)

北京大学 PEKING UNIVERSITY

- **Intuition**
  - Case-based Reasoning is **sensitive to the data split**; once a test example has no reference training case, it cannot be answered correctly.
  - Rule-based Reasoning is **insensitive to the data split** as the model learns underlying rules.
- **Distinguish two reasoning paradigms**
  - we artificially remove certain regions of the training data to see its effect on test performance.
  - For example, in addition, we hypothesize the model **is relying on some surrounding cases** of the test question. We can remove a small set of surrounding cases from the training set and see whether the model can still answer the question.

Two Reasoning Paradigms (3)

北京大学 PEKING UNIVERSITY

- **Rule-based reasoning also involves a degree of memorization.**
  - Unit rules, e.g., to solve 42+57, we need to know 2+7=9 & 4+5=9
  - The set of unit rules should be **finite** and can be easily covered by a training set of a reasonable size.
  - Otherwise, if it is hard or even impossible for a training set to cover all the unit rules, we consider the model performing case-based reasoning.

Datasets (1)

- We focus on binary operations, which take two numbers a, b as inputs. Denoting c as the target label, we construct datasets like D = {((ai, bi), ci)} for 5 math tasks:
  - **Addition.**
    - input: “a + b”;
    - output: “c”, where c = a + b;
    - a, b range from 0 to 99.
  - **Modular addition.**
    - input: “a + b”;
    - output: “c”, where c = a + b mod P;
    - a, b range from 0 to 112. We set P = 113 as a constant.
  - **Base addition.**
    - same as addition, except that all numbers a, b, c are expressed in the base-n numerical system, n = 9.

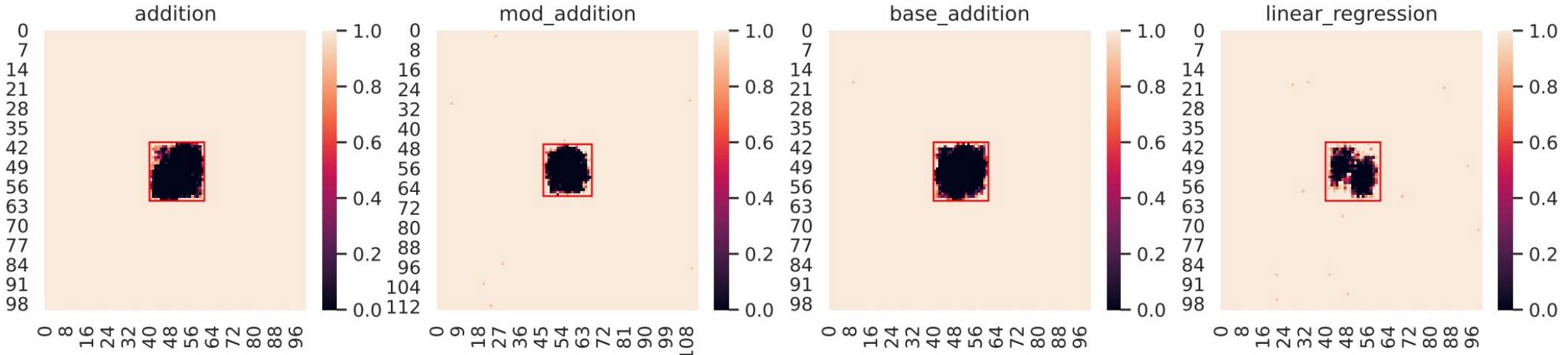
Datasets (2)

北京大学 PEKING UNIVERSITY

- **Linear regression.**
  - input: “(a, b) =”;
  - output is “c”, where $c = m \cdot a + n \cdot b + p$;
  - a, b range from 0 to 99. We set $m = 1$, $n = 2$, $p = 3$ as constants.
- **Chicken & rabbit problem.**
  - input: “Q: Rabbits have 4 legs and 1 head. Chickens have 2 legs and 1 head. There are a legs and b heads on the farm. How many rabbits and chickens are there?”;
  - output: “A: There are c rabbits and d chickens.”, where $c=(a-2b)/2$, $d=(4b-a)/2$;
  - b ranges from 0 to 99. For each b, a ranges from 2b to 4b with a step of 2.

Method: Leave Square Out

- Hypothesis: the model is relying on **surrounding cases** to perform reasoning
- Setting 1: **random split**, 70% training set and 30% test set
- Setting 2: **leave square out**: hold out a continuous square of examples as test set; > 95% training set, < 5% test set
- Transformers achieve 100% test accuracy in random-split setting, but fail almost completely in leave square out experiment! Indicated by the **holes within the square**.

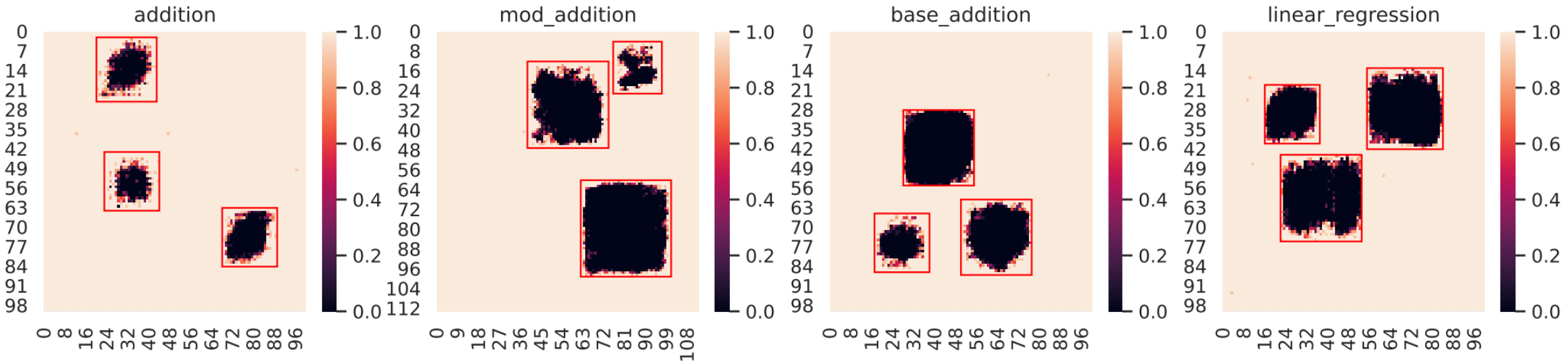


accuracy of fine-tuned GPT-2 on the whole dataset; test square is marked as red box

Ablations: Test Square Locations

北京大学 PEKING UNIVERSITY

- Do locations of test squares matter? **No, still holes!**

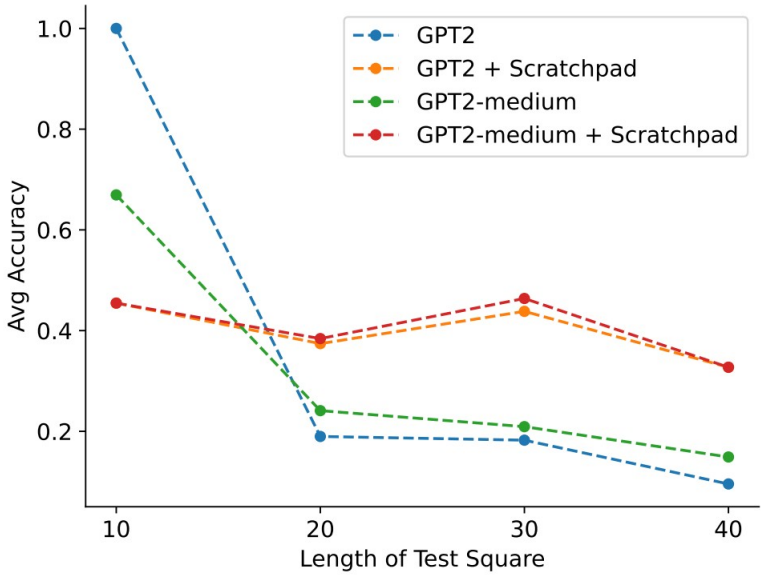


accuracy of fine-tuned GPT-2 on the whole dataset; test square is marked as red box

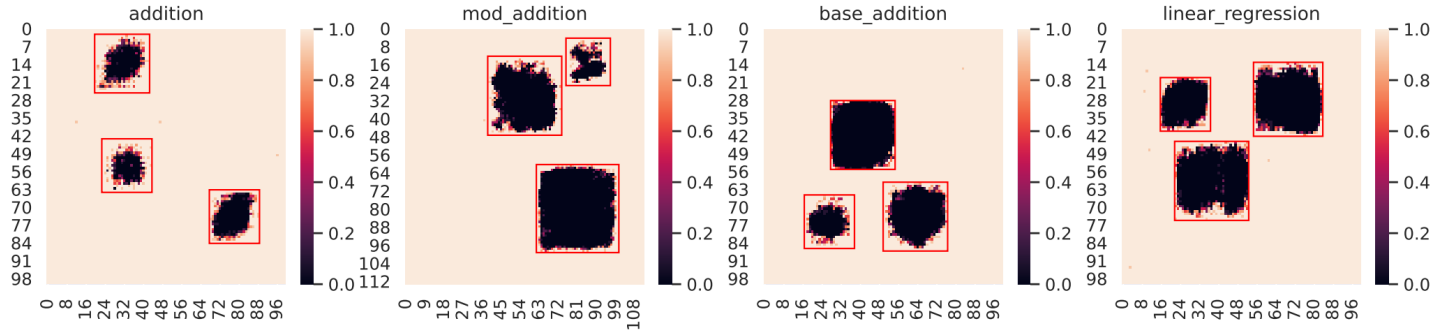
Ablations: Test Square Size

北京大学 PEKING UNIVERSITY

- Does the size of test square matter? **Yes,** test accuracy drops with test square length increasing, and holes may disappear if the test square is too small.



In the task of addition, we show the average accuracy over all test samples (samples within the square) with side length $l_k = 10, 20, 30, 40$. We test four models: GPT-2, GPT-2 with scratchpad, GPT-2-medium and GPT-2-medium with scratchpad.

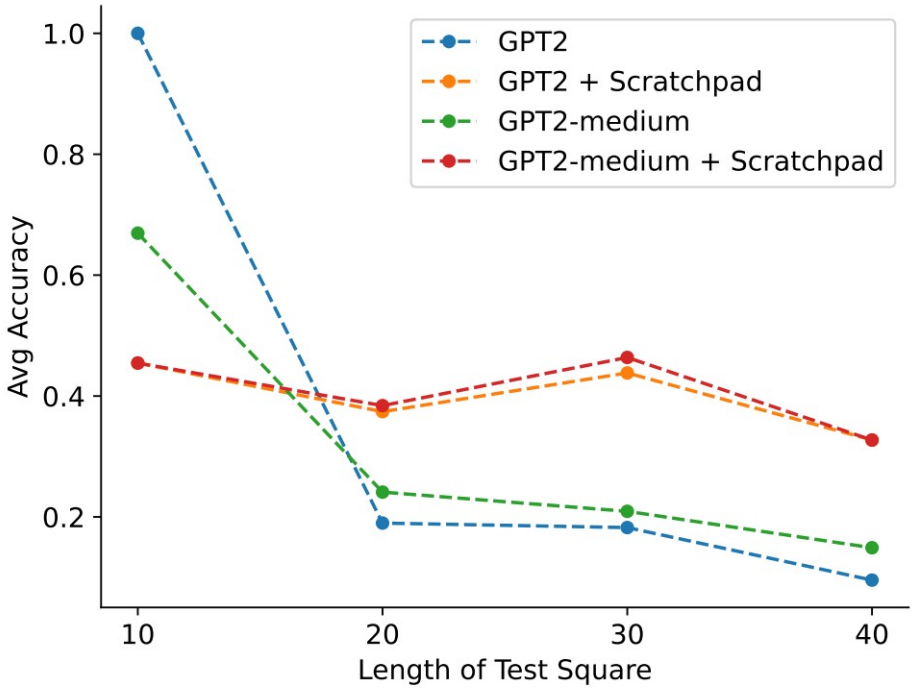


accuracy of fine-tuned GPT-2 on the whole dataset; test square is marked as red box

Ablations: Scratchpad (1)

北京大学 PEKING UNIVERSITY

- Does adding scratchpad (CoT for math) help?



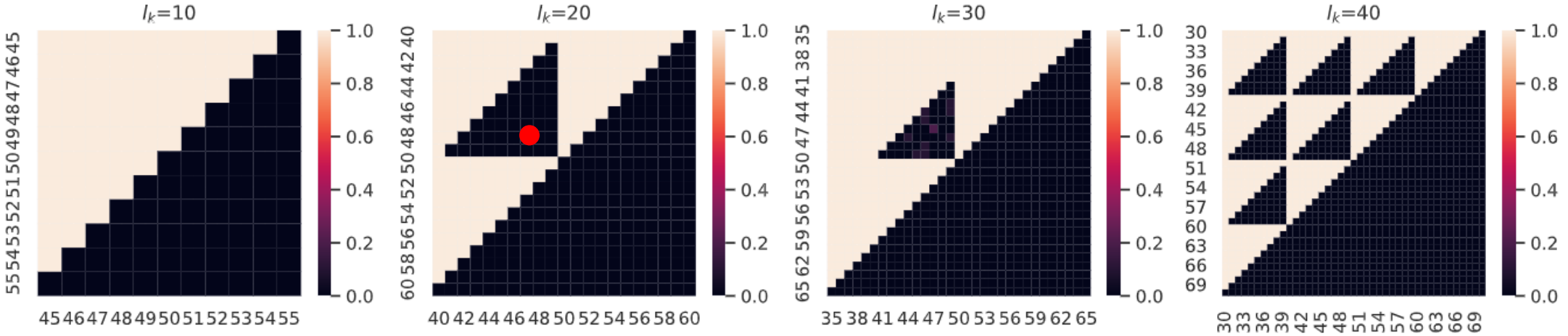

```
Input:
2 9 + 5 7

Target:
<scratch>
2 9 + 5 7 ,  C: 0
2 + 5 , 6 C: 1  # added 9 + 7 = 6 carry 1
, 8 6 C: 0  # added 2 + 5 + 1 = 8 carry 0
0 8 6
</scratch>
8 6
```

figure from Nye et al., Show Your Work: Scratchpads for Intermediate Computation with Language Models, 2021

Ablations: Scratchpad (2)

- Does adding scratchpad help? **No!** There are still holes.

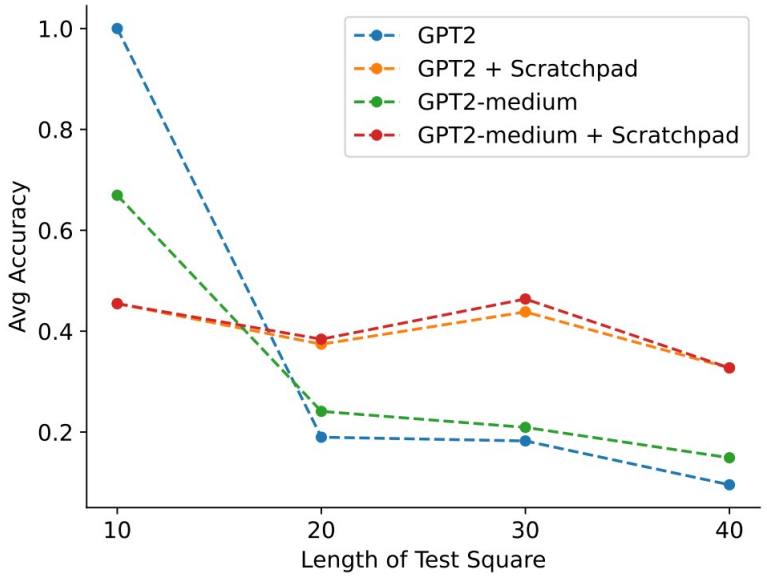


  - For the small triangle of $l = 20$,
    - example: 47+48
    - 2 steps: unit's: 7+8=5, carry 1; **ten's: 4+4+1=9**
    - **missing case**: 4+4+1 in the ten's digit
  - The models succeed in test cases where **every step of the corresponding scratchpad has appeared in the training set** and fail otherwise.

Ablations: Model Size

北京大学 PEKING UNIVERSITY

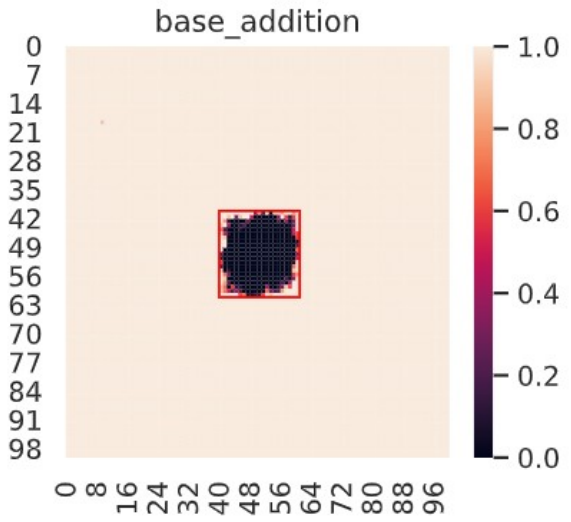
- Does the model size matter?



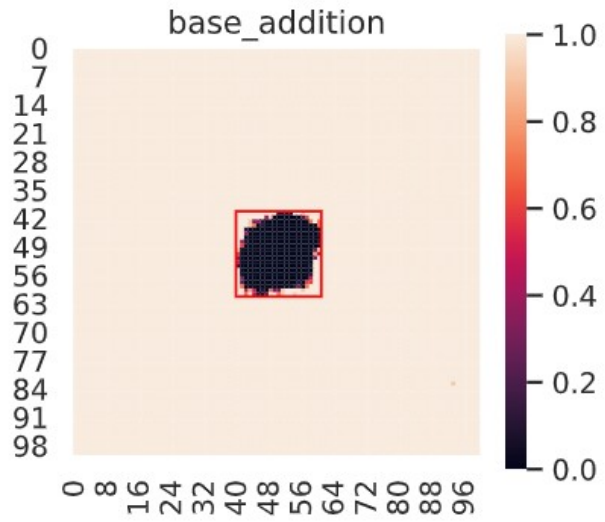
In the task of addition, we show the average accuracy over all test samples (samples within the square) with side length $l_k = 10, 20, 30, 40$. We test four models: GPT-2, GPT-2 with scratchpad, GPT-2-medium and GPT-2-medium with scratchpad.

Ablations: Larger Model Size (1)

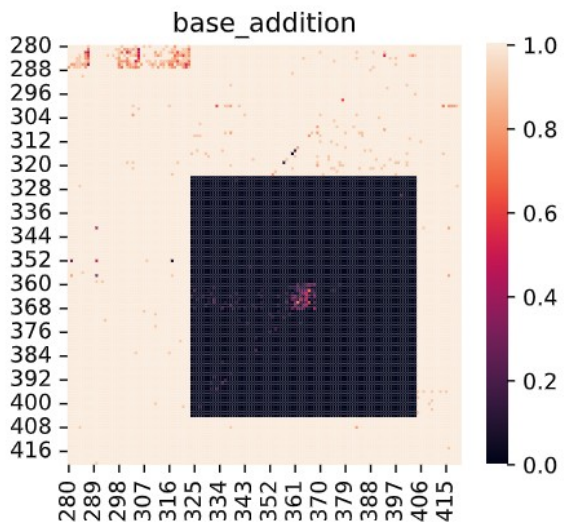
- Does the model size matter? (for larger models including llama-2-7b & GPT-3.5)



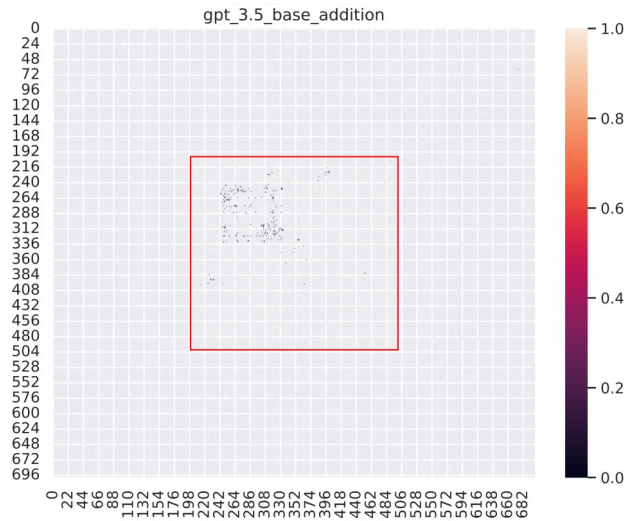
GPT-2 (fine-tuned)
whole dataset accuracy



GPT-2 Medium (fine-tuned)
whole dataset accuracy



llama-2-7B (from scratch)
test accuracy

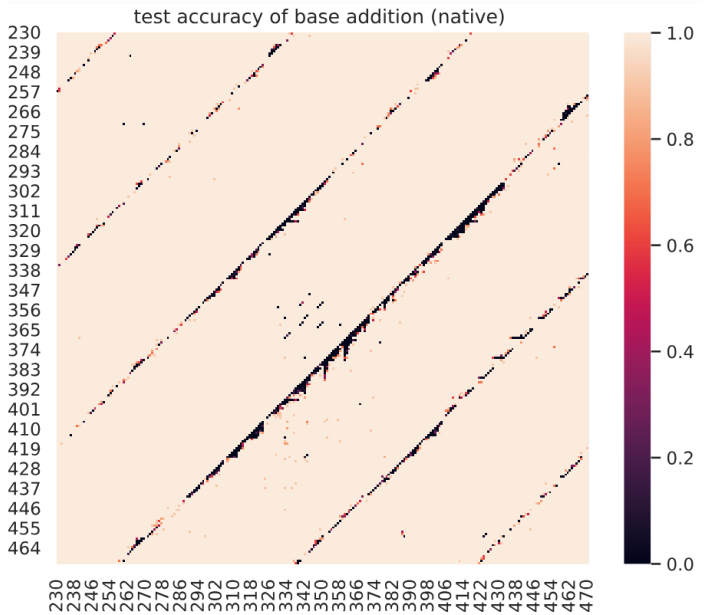


GPT-3.5 (fine-tuned)
whole dataset accuracy
(sample 20% test set, 10% train set)

- The behavior of case-based reasoning **exists in both small and large models**.

Ablations: Larger Model Size (2)

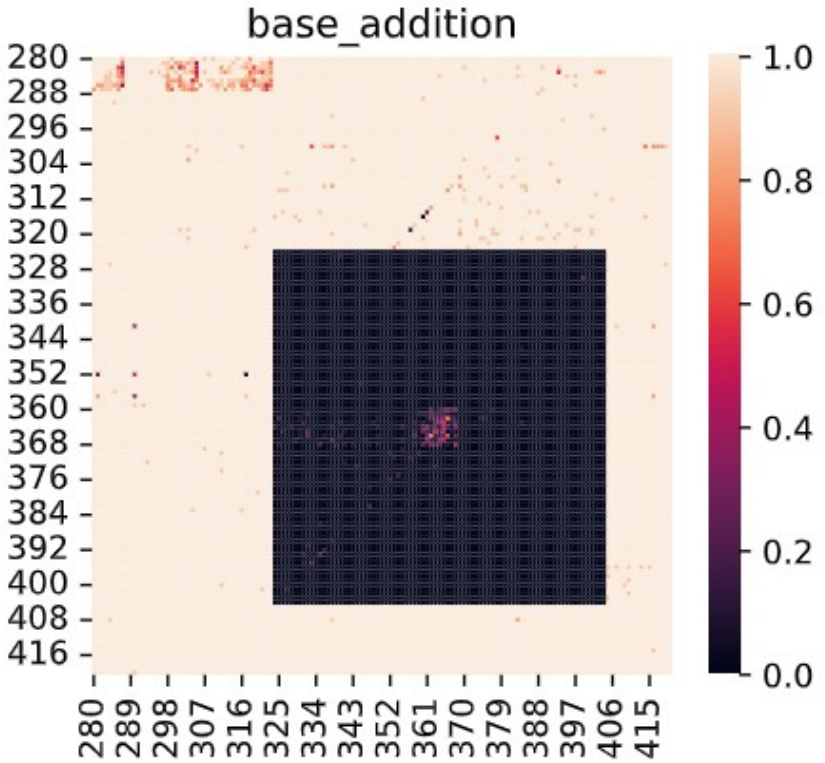
- Discussion about behavior of Llama-2-7b on base addition
  - **fine-tune: the hole disappears**
    - **reason 1:** It is possible that the left-out square are **not really the dependent cases** for pretrained Llama-2-7B on this task.
    - **reason 2:** There might be **data leakage** during pretraining. Besides, even without considering data leakage, pretraining may still have introduced **strong biases** that happen to suit base addition well, making the model generalize to most parts of the test square.



llama-2-7B (fine-tune on base-9 addition)

Ablations: Larger Model Size (3)

- Discussion about behavior of Llama-2-7b on base addition
  - **fine-tune:** the hole disappears
  - **train from scratch: the hole still exists**



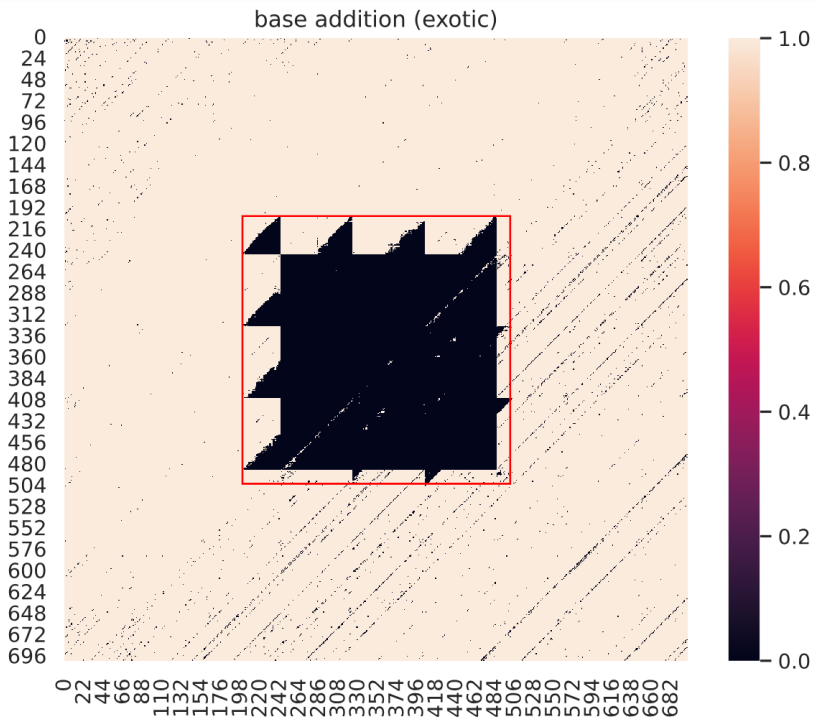
llama-2-7B (from scratch)

Ablations: Larger Model Size (4)

- Discussion about behavior of Llama-2-7b on base addition
  - **fine-tune on base addition with native digits :** the hole disappears
  - **train from scratch:** the hole still exists
  - **fine-tune on base addition with exotic digits: the hole still exists**

```
example of base-9 addition with native
digits:
1+8=10

example of base-9 addition with exotic
digits:
B+I=BA
```



llama-2-7B (fine-tune on exotic base-9 addition)

In-Context Learning (1)

北京大学 PEKING UNIVERSITY

- Model: GPT-3.5-turbo-0125
- Method
  - We randomly collected pairs of base-9 integers whose zero-shot addition accuracy is less than 20%.
  - We provide 10 few-shot examples.
    - 5 from random group: randomly selected
    - 5 from similar group: obtained by simultaneously replacing only one digit of the pair
  - Scratchpad is used in each few-shot example.
  - We choose the 14 test samples where the improvement with few-shot examples is more than 80%.
  - We mask some in-context examples from either the similar group or the random group and compare the accuracy drop:

$$c_i = \left(\sum_{m \in \mathcal{M}} \mathbb{1}\{i \in m\} \cdot \frac{accu_{icl} - accu_m}{accu_{icl} - accu_{orig}} \right) \Big/ N_i$$

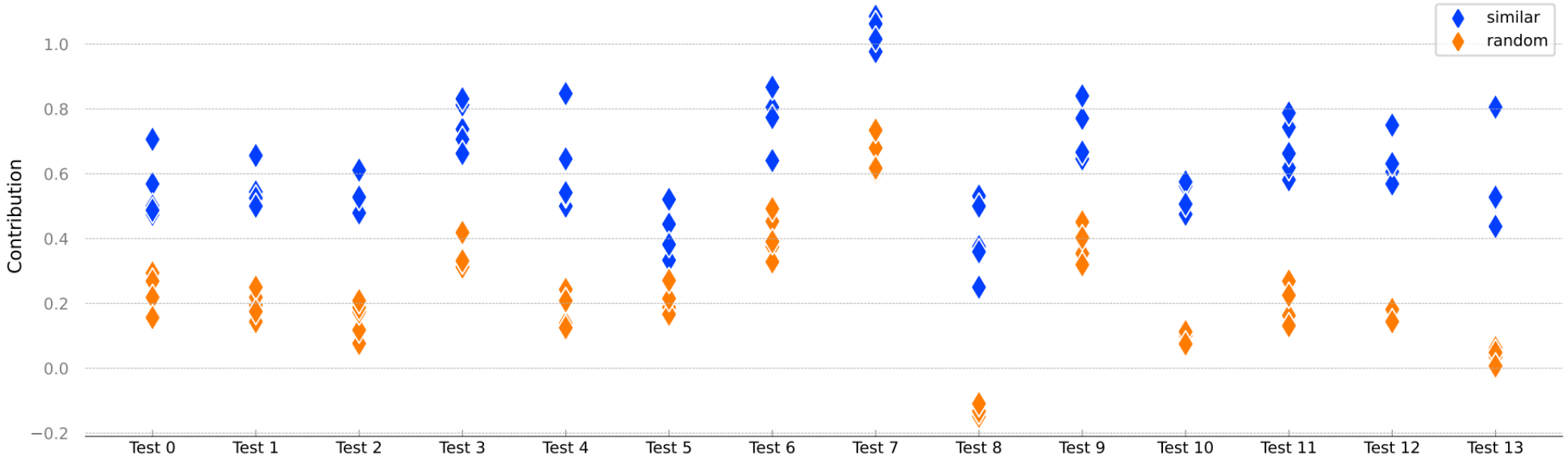
where $accu_{orig}, accu_{icl}$ and $accu_m$ represent the accuracy without few-shot examples, with all examples and with non-masked examples, respectively. $\mathcal{M}$ is the mask set that contains all possible combinations in the random and similar group. N_i is the number of masks that contain i.

In-Context Learning (2)

北京大学 PEKING UNIVERSITY

- Results

$$c_i = \left(\sum_{m \in \mathcal{M}} \mathbb{1}\{i \in m\} \cdot \frac{accu_{icl} - accu_m}{accu_{icl} - accu_{orig}} \right) \Big/ N_i$$



The contributions of similar and random examples in 14 tests

Rule-Following Fine-Tuning (RFFT)

北京大学 PEKING UNIVERSITY

- To enforce LLMs to do rule-based reasoning, we propose **Rule-Following Fine-Tuning (RFFT)**

direct answer

```
input:
59+13=

output:
72
```

scratchpad

```
input:
59+13=

output:
59+13, , C:0
# added 9+3+0=2
5+1, 2, C:1
# added 5+1+1=7
+, 72, C:0

72
```

Explicitly list the rules

```
input:
Follow the code step by step to
answer the question:
[5, 9]+[1, 3]=
def sum_digit_by_digit(num1, num2):
    result=[]
    carry=0
    while num1 or num2:
        digit1=num1.pop() if num1 else 0
        digit2=num2.pop() if num2 else 0
        total=digit1+digit2+carry
        result.insert(0,total%10)
        carry=total//10
    if carry:
        result.insert(0,carry)
    return result

output:
num1 = [5,9]
num2 = [1,3]
1. Initialize Result and Carry
result = []
carry = 0
```

Recite the rule used in each step

rule following

```
2. Loop Through Each Digit
```while num1 or num2:```
2.1 check the stop criterion
num1 = [5,9]
num2 = [1,3]
bool(num1) = True
bool(num2) = True
num1 or num2 = True
enter the loop
2.2 one iteration
```digit1=num1.pop() if num1 else 0```
num1 = [5,9]
bool(num1) = True
num1.pop()
num1 = [5]
digit1 = 9
```digit2=num2.pop() if num2 else 0```
num2 = [1,3]
bool(num2) = True
num2.pop()
num2 = [1]
digit2 = 3
```total=digit1+digit2+carry```
total = digit1+digit2+carry =
9+3+0 = 12
```result.insert(0,total%10)```
result = []
total%10 = 12%10 = 2
result = [2]
```carry=total//10```
carry = 12//10 = 1
2.3 back to the start of the loop
...
3. Check Remaining Carry
```if carry: result.insert(0,carry)```
result = [7,2]
carry = 0
bool(carry) = False
pass
result = [7,2]
4. Return Result
```return result```
result = [7,2]
```

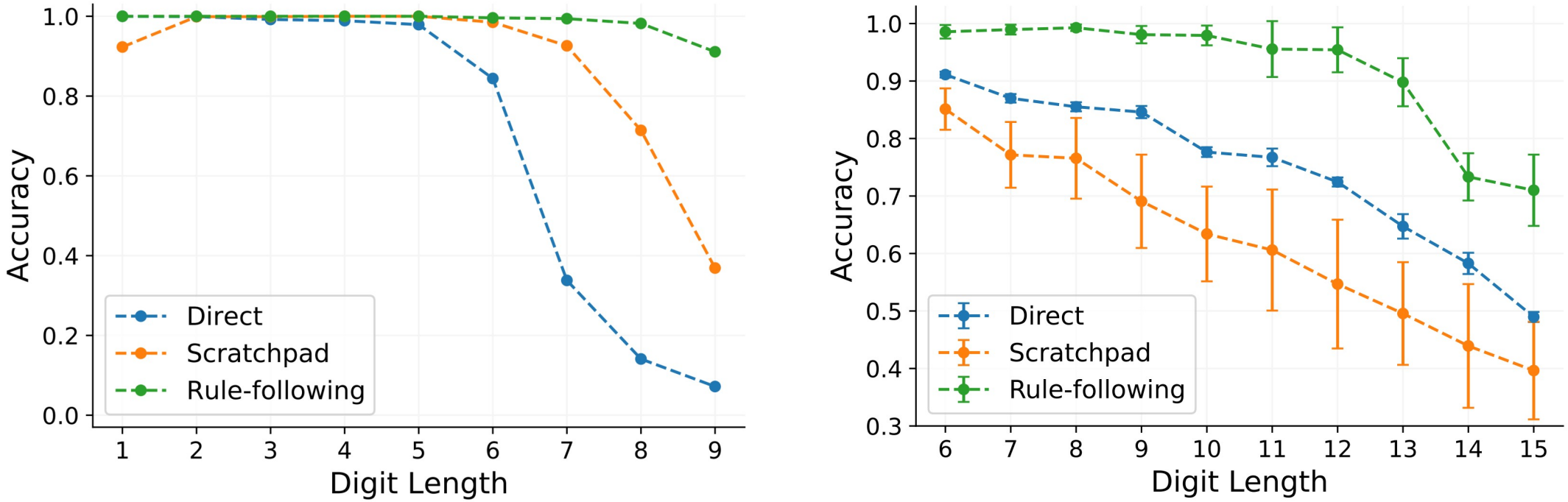
Step by step execution

Every step reminds what I am doing

RFFT: Performance (1)

北京大学 PEKING UNIVERSITY

- Finetune models on 1-5 digit addition
- Llama-2-7B: 500k samples; GPT-3.5: **100 samples**
- Significantly higher accuracy on long digits



accuracy of fine-tuned llama-2-7b (left) and GPT-3.5-turbo (right) on addition

RFFT: Natural Language (1)

北京大学 PEKING UNIVERSITY

- RFFT with natural language rule (input)

Follow the rules step by step to answer the question: ‘6’+‘7’=
Add two numbers in order from the lowest digit to the highest digit. The operation rules are as follows:
1. In the initial state, the carry from the previous digit is 0 and the result is <empty>.
2. Begin the loop through each digit:
2.1. Determine whether the two numbers are <empty>. If both are <empty>, the loop is over. If one of them is not <empty>, go to the iteration.
2.2. One iteration:
2.2.1. Find the rightmost digit of each number and remove the digit from the number. If one of them is <empty>, the digit is 0.
2.2.2. Calculate the sum of the current digits and the carry.
2.2.3. Insert the last digit of the summation to the beginning of the result and update carry. If the sum is larger than 9, carry is 1. Otherwise, carry is 0.
2.3. Back to the start of the loop.
3. When the loop through each digit is over, if there’s a remaining carry, insert it to the beginning of the result.
4. Return the result.

RFFT: Natural Language (2)

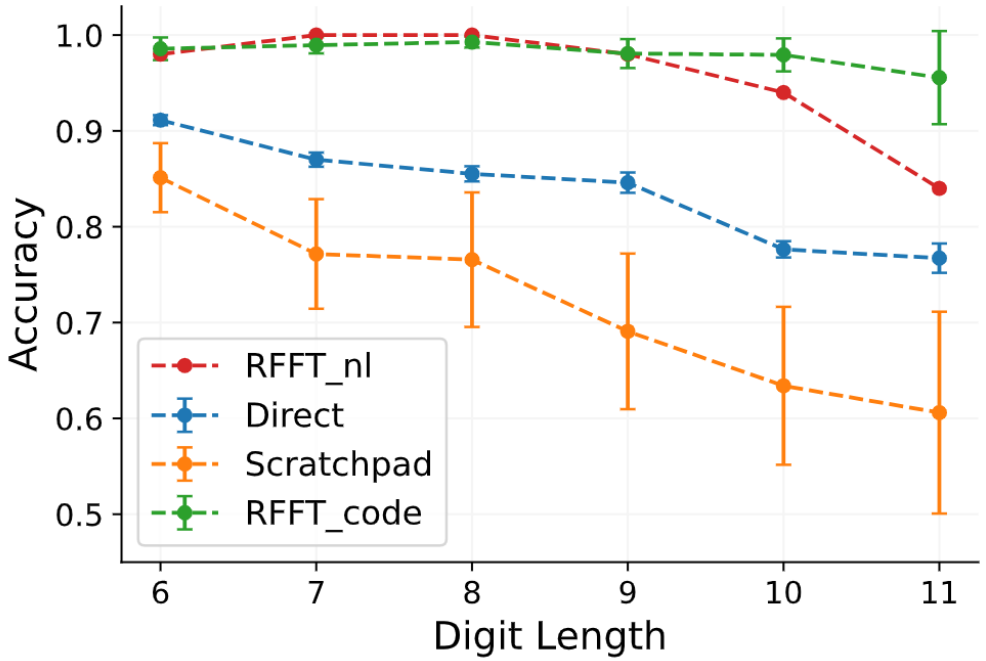
北京大学 PEKING UNIVERSITY

- RFFT with natural language rule (output)

num1='6'; num2='7'
```
1. In the initial state, the carry from the previous digit is 0 and the result is <empty>.
```
Initialize the carry as 0 and the result is '<empty>'.
```
2. Begin the loop through each digit:
2.1. Determine whether the two numbers are <empty>. If both are <empty>, the loop is over. If one of them
is not <empty>, go to the iteration.
```
num1 '6' is not <empty>. num2 '7' is not <empty>.
Because both are not <empty>, go to the iteration.
```
2.2. One iteration:
2.2.1. Find the rightmost digit of each number and remove the digit from the number. If one of them is
<empty>, the digit is 0.
```
The rightmost digit of num1 '6' is '6'. The rightmost digit of num2 '7' is '7'. Then the remaining num1
'6' is '<empty>' and the remaining num2 '7' is '<empty>'.
```
2.2.2. Calculate the sum of the current digits and the carry.
```
The sum of '6', '7' and carry '0' is 6 + 7 + 0 = 13.
```
2.2.3. Insert the last digit of the summation to the beginning of the result and update carry.
```
The last digit of the summation is 13 % 10 = 3. The previous result is '<empty>', now the result is: '3'.
The carry is 13 // 10 = 1.
```
2.3. Back to the start of the loop.
2.1. Determine whether the two numbers are <empty>. If both are <empty>, the loop is over. If one of them
is not <empty>, go to the iteration.
```
num1 is '<empty>'. num2 is '<empty>'.
Because both are <empty>, the loop is over.
```
3. When the loop through each digit is over, if there's a remaining carry, insert it to the beginning of the result.
```
The carry is 1, so add an 1 to the beginning of the result '3'. So the final result is '1,3'.
```
4. Return the result.
```
The final result is '1,3'.

RFFT: Natural Language (3)

- Performance

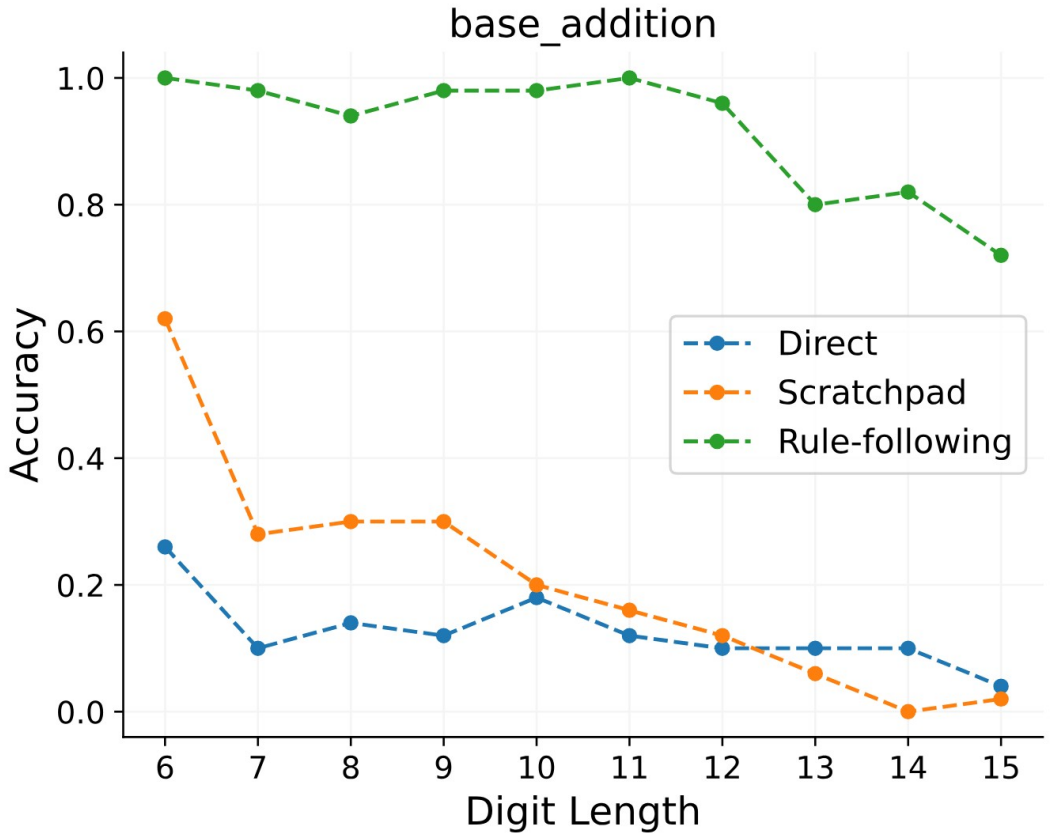


accuracy of GPT-3.5-turbo fine-tuned on 1-5 digit addition

RFFT: Other Tasks

北京大学 PEKING UNIVERSITY

- RFFT on other tasks
  - **base addition**： finetune on 1-5 digit base addition, test on 6-15 digit base addition

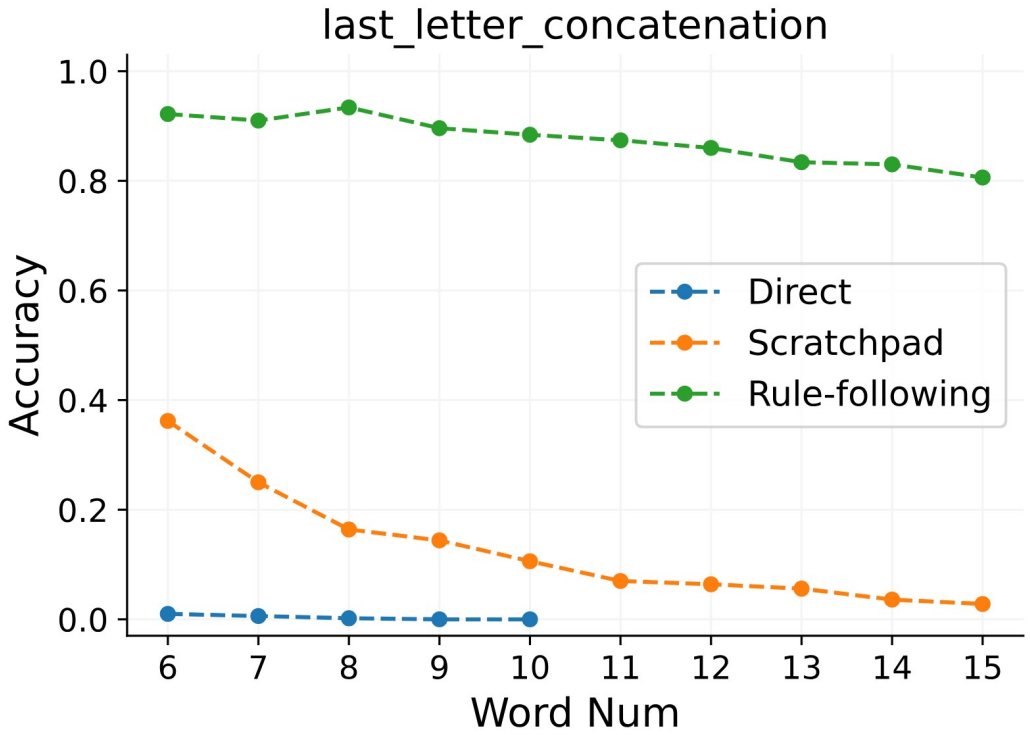


RFFT: Other Tasks

北京大学 PEKING UNIVERSITY

- RFFT on other tasks
  - base addition
  - **last letter concatenation**： fine-tune on 1-5 words concatenation, test on 6-15 words.

```
Q: Concatenate the last letter of the following names: Alice, Bob,
Cathy.
A: eby
```



RFFT as a Meta Learning Ability

北京大学 PEKING UNIVERSITY

- **Phenomenon:** finetune models on 1-5 digit addition:
  - Llama-2-7B: **150k samples**; GPT-3.5: **100 samples**
- **Hypothesis:** rule-following may be a meta learning ability
  - it might be "learned" through pre-training on diverse rule-following data and transfer to new unseen domains
  - the stronger the foundation model is, the easier it can understand and learn the rules
- **Experiments:**
  - Llama-2-70b achieves comparable performance to Llama-2-7b after seeing much less samples (**20k samples**)

Model	6 digit	7 digit	8 digit	9-digit
Llama-2-7b	99.6	99.4	96.2	90.1
Llama-2-70b	99.3	97.8	97.0	89.8

fine-tuned Llama-2-7b and Llama-2-70b on 6-9 digit addition task

Conclusion

北京大学 PEKING UNIVERSITY

- For the first step differentiate the two reasoning paradigms of LLMs
- Verifies that LLMs do math mainly in a **case-based reasoning** way, which prevents from systematic and length generalization
- Proposes **Rule-Following Fine-Tuning** (**RFFT**), explicitly teaches LLMs to reason by rules; significantly better length generalization ability

- Paper arxiv: https://arxiv.org/pdf/2402.17709.pdf
- Github link: https://github.com/GraphPKU/Case_or_Rule
- SR code:

Number Cookbook: Number Understanding of Language Models and How to Improve It

Preliminary work, under review

Background

北京大学 PEKING UNIVERSITY

- **Number:** most common object in complex reasoning tasks
  - Arithmetic, math, physics, financial, …
  - Involved in most topics in LLM research: *reasoning ability*, *length generalization*, *foundation model* …
- In current research, **number understanding and processing** is …
  - Not a separate task, but mixed in with other tasks. (Like arithmetic)
  - Too simple and impractical (only short integer)

Current datasets

```
Ann's favorite store was having a summer clearance. For $75 she bought 5 pairs of shorts for $7 each and 2 pairs of shoes for $10 each. She also bought 4 tops, all at the same price. How much did each top cost?
```

```
She bought 5 shorts at $7 each so 5*7=$<<5*7=35>>35 She bought 2 pair of shoes at $10 each so 2*10=$<<2*10=20>>20 The shorts and shoes cost her 35+20 = $<<35+20=55>>55 We know she spent 75 total and the shorts and shoes cost $55 which left a difference of 75-55 = $<<75-55=20>>20 She bought 4 tops for a total of $20 so 20/4 = $5 #### 5
```

Real-world application

项目	本报告期	本报告期比上年同期增减变动幅度(%)	年初至报告期末	年初至报告期末比上年同期增减变动幅度(%)
营业总收入	169,229,054.07	476.44	250,214,803.96	195.97
归属于上市公司股东的净利润	5,960,520.43	-37.34	41,502,514.99	20.68
归属于上市公司股东的扣除非经常性损益的净利润	79,383,193.01	276.38	140,948,014.09	139.94

More representations

北京大学 PEKING UNIVERSITY

- We propose **NUPA** (number understanding and processing ability) test.
  - Scope: primary and second school education
  - Comprehensive and practical (not too simple or too hard)
- **4** number representations:
  - **Integer:** 123, 12, …
  - **Float:** 3.1415926 …
  - **Fraction:** 7/11, 4/3, …
  - **Scientific Notation:** 1e10, 3.1e2, …

More tasks

北京大学 PEKING UNIVERSITY

- **17 tasks** in **4 ability categories:**
  - **Elementary arithmetic:** $+, -, \times, /, //, \%$
  - **Compare:** $>, <$
  - **Digit understanding:**
    - Get digit: $digit(123454321,3) = 4$
    - Get length: $l(12345) = 5$
    - Count: $c(1122233,2) = 3$
    - Digit operator $(>,<,+)$
  - **Conversion between representations:**
    - (Frac, Scient)2Float, Float2Scient, significant figures

NUPA test

北京大学 PEKING UNIVERSITY

- **41 meaningful composition:**

Table 1: Task overview of NUPA Test. ✓: 41 tasks included in our test. ✗: Not included, too complex. ◯: Not directly included but can be easily transferred from an included task. –: Not applicable.

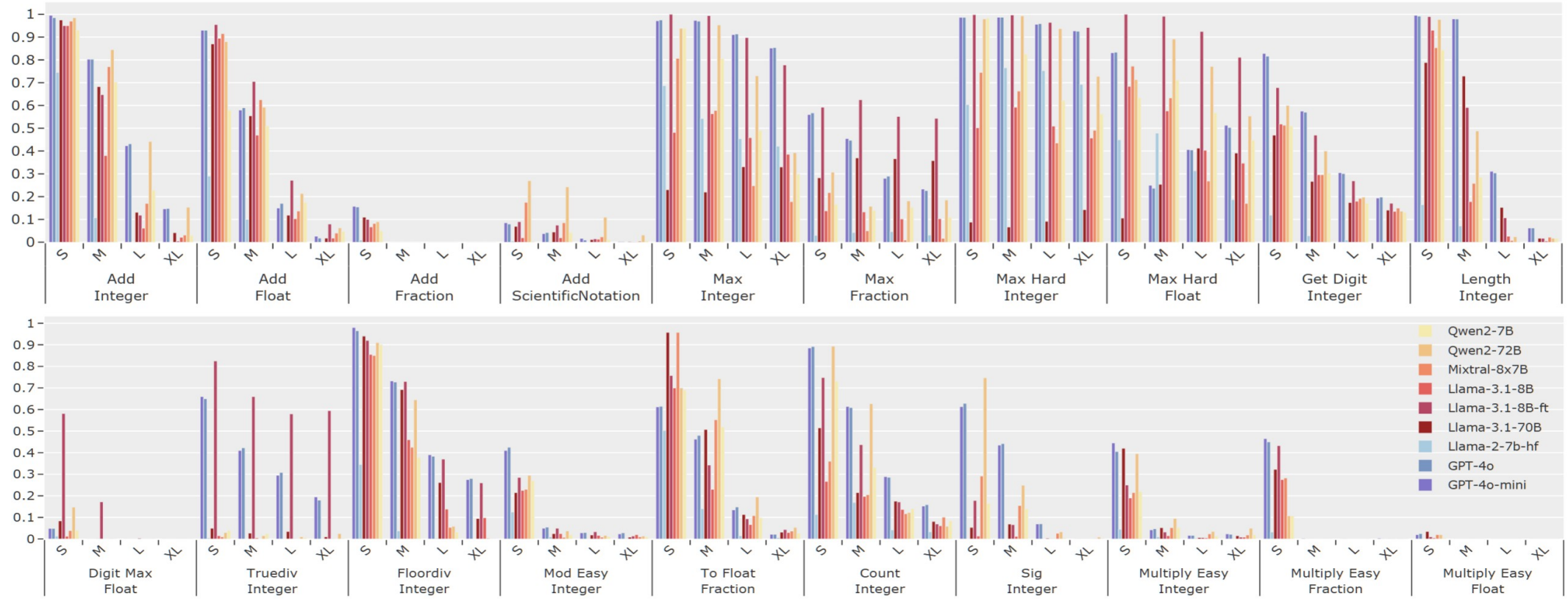
	Add	Sub	Multiply	Truediv	Floordiv	Mod	Max	Min	Digit Max	Digit Min	Digit Add	Get Digit	Length	Count	To Float	To Scientific	Sig. Fig.
Integer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓
Float	✓	✓	✓	✗	–	–	✓	✓	✓	✓	✓	✓	✓	◯	–	✓	✓
Fraction	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	◯	✓	◯	◯
Scientific	✓	✓	✓	✗	–	–	✓	✓	–	–	–	–	–	◯	✓	–	◯

- For each task, we generate training, valid and test set with different digits.
  - For hard task, training 1-8, valid and test: 1-20
  - For easy task, training 1-20, valid and test 1-100
- Some procession to make sure not too hard or too easy.
- **3 Metrics:** exact match, digit match, d(ifference of)length

Generation: 425.925535321
Groundtruth: 31415.92653582

Exact match: 0
Digit match: 8 / (8 + 5) =0.62
dlength: 3

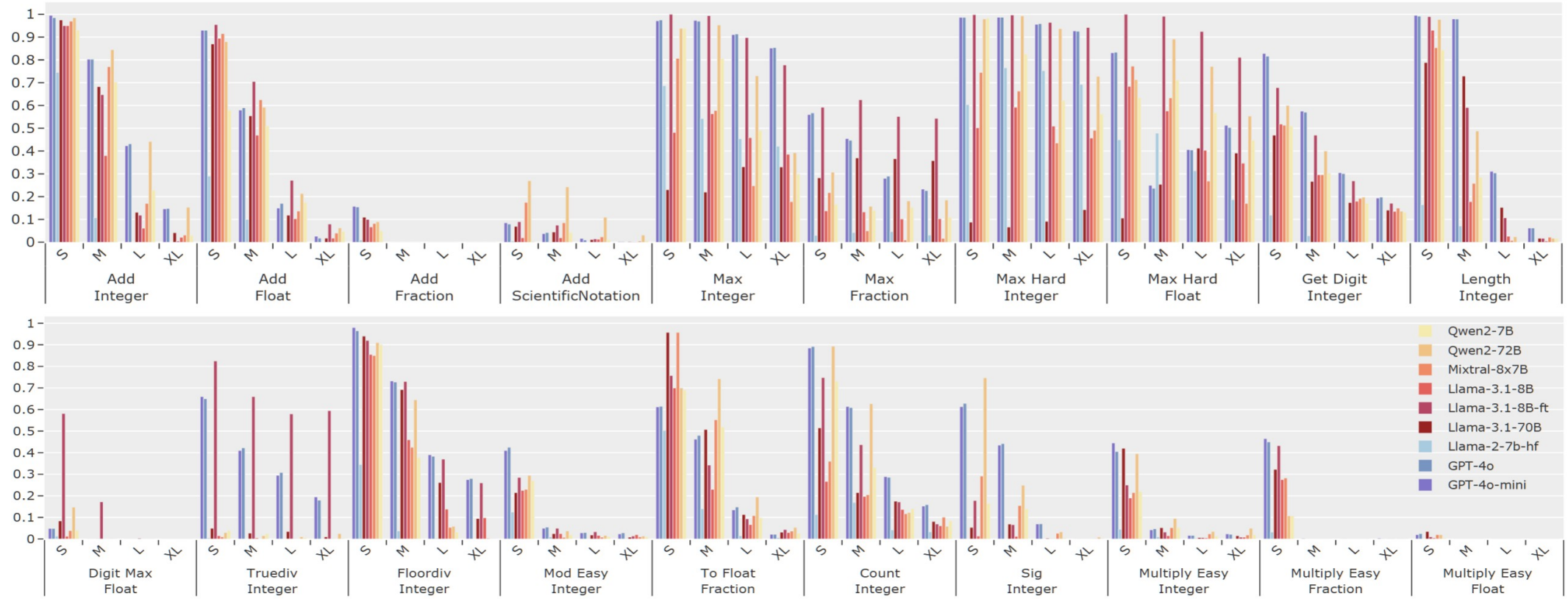
NUPA test



- Performs well on typical tasks, but declines on many less common tasks.
- Length still remains an obstacle for NUPA of LLMs.
- Concept “digit” is not as easy as we expected.

NUPA is not a solved problem for even the most powerful LLMs.

NUPA test



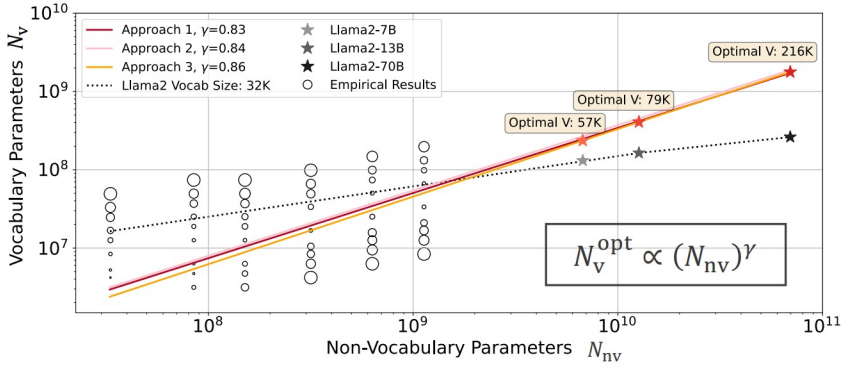
- Performs well on typical tasks, but declines on many less common tasks.
- Length still remains an obstacle for NUPA of LLMs.
- Concept “digit” is not as easy as we expected.

NUPA is not a solved problem for even the most powerful LLMs.

How to improve NUPA?

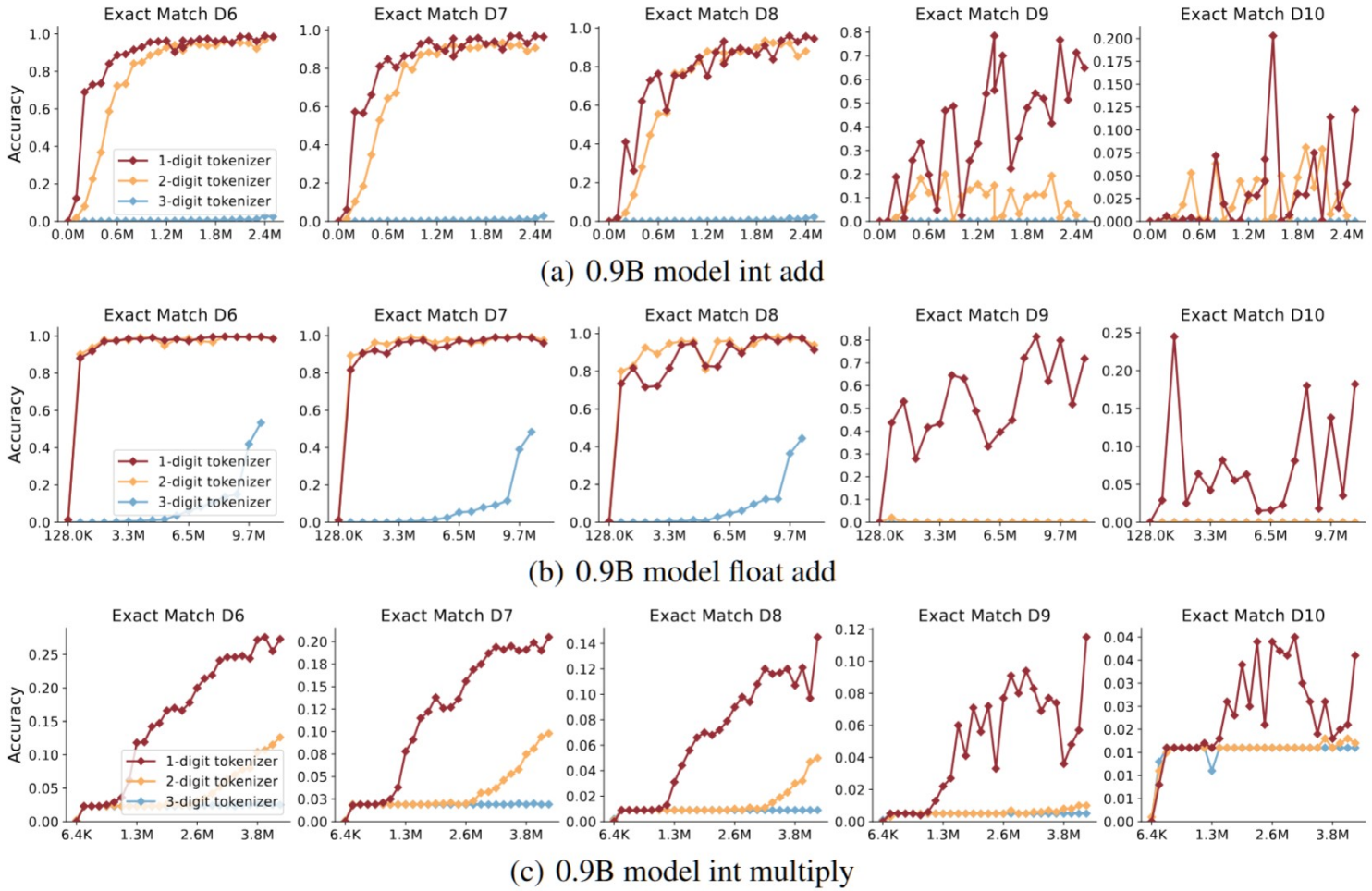
北京大学 PEKING UNIVERSITY

- **Three aspects of techniques:** tokenizers, PEs and data format
- Tokenizer:
  - Trend: larger number vocab 31415.926535897932 ->31415.926535897932
  - Optimal vocab ∝ model size for **word** tokens
  - Our conclusion: **one-digit tokenizer is best.**



- Our experiments:
  - 0.1B, 0.9B, 3B models
  - Int add, int multiply, float add
  - The one-digit token is always best for both **in-domain** and **out-of-domain**
  - Training is much **faster** than others

How to improve NUPA?



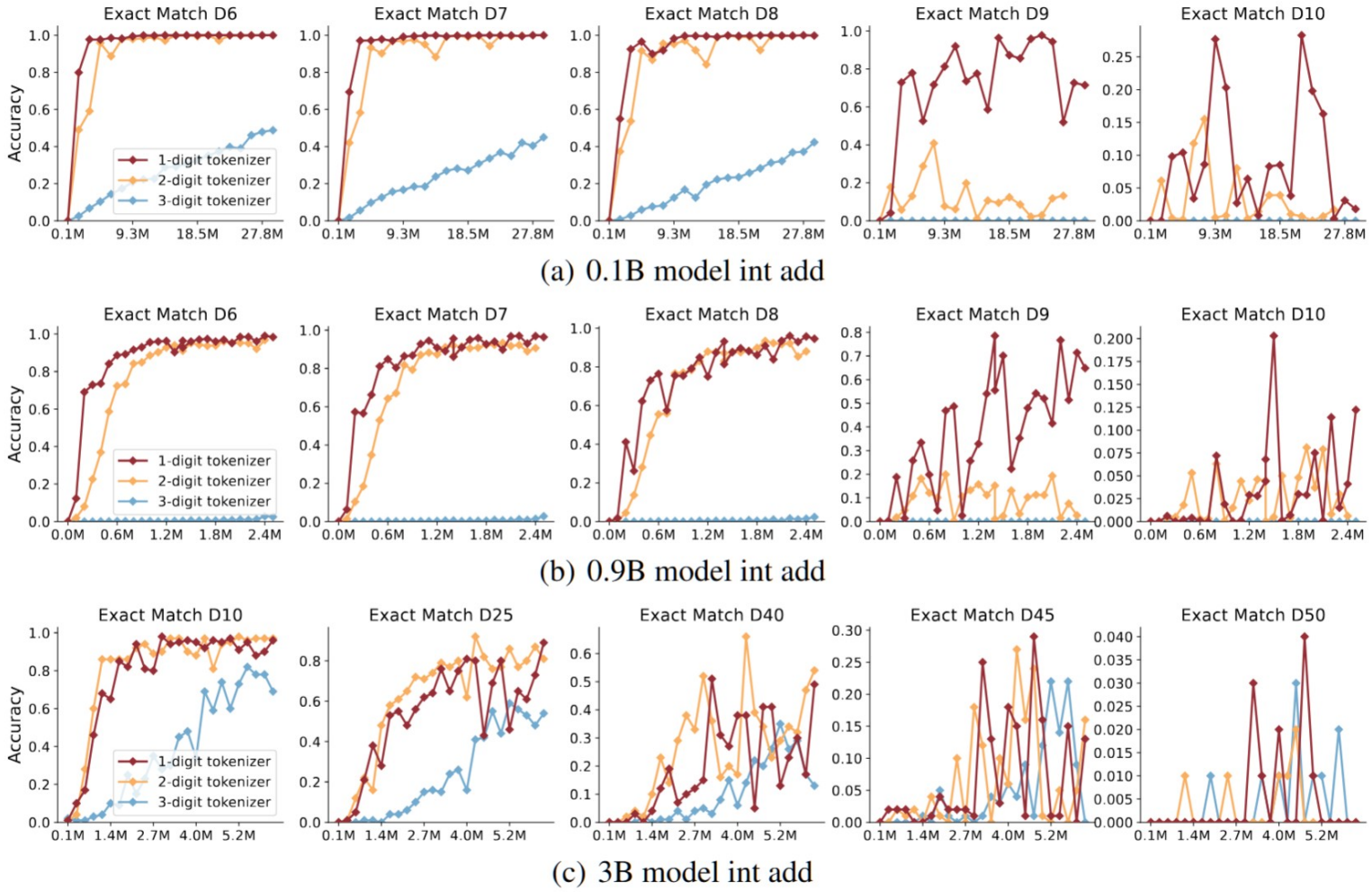
(a) 0.9B model int add

(b) 0.9B model float add

(c) 0.9B model int multiply

0.9B model with three different tasks. X-axis is number of seen training samples.

How to improve NUPA?



Different model size. X-axis is number of seen training samples.

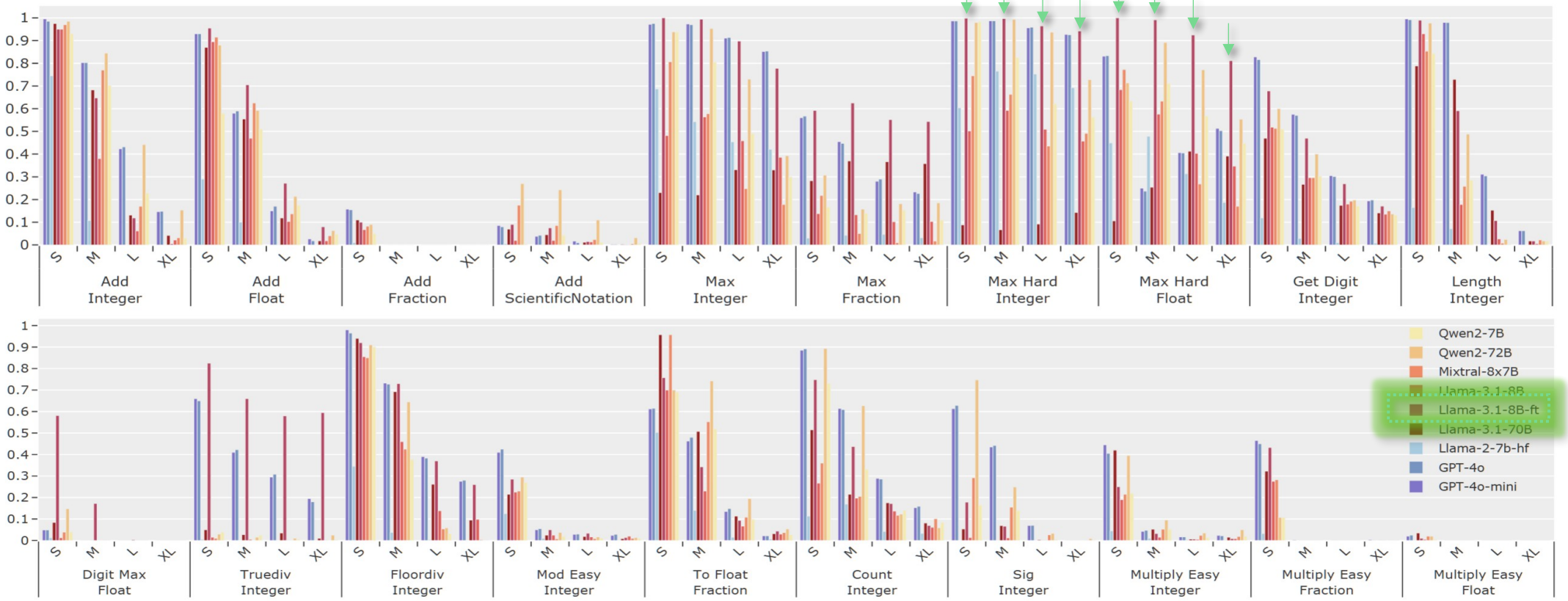
How to improve NUPA?

- **Three aspects of techniques: tokenizers, PEs** and **data format**
- **PEs**: We find NoPE and Alibi (as a representation of RPE) are always better on different number representation and tasks for length generalization.
- **Data format:**
  - Reverse (int only) is effective.
  - Zero-padding slightly improve further.
  - Index-hint does not help.

How to improve NUPA?

北京大学 PEKING UNIVERSITY

- These techniques are proposed for pre-training stage.
- How about finetuning? SFT
- A simple SFT with lora can improve performance significantly.



How to improve NUPA?

北京大学 PEKING UNIVERSITY

- These techniques are proposed for pre-training stage.
- How about **finetuning**? SFT
- These techniques **cannot** be added in SFT stage.

	Add Integer					
	S	M	L	XL	wl-digit	pp-digit
FT	**0.95**	**0.65**	**0.12**	0.01	**4**	**12**
without FT	**0.95**	0.38	0.06	**0.02**	**4**	9
NoPE +reverse	0.89	0.35	0.06	**0.02**	3	9
NoPE +reverse +pad	0.87	0.34	0.05	**0.02**	0	9
RoPE +reverse +pad	0.40	0.20	0.04	0.00	0	7

	Add Float					
	S	M	L	XL	wl-digit	pp-digit
FT	**0.96**	**0.71**	**0.27**	**0.08**	**5**	**17**
without FT	0.90	0.47	0.10	0.02	3	11
NoPE +reverse	0.81	0.38	0.09	0.01	0	11
NoPE +reverse +pad	0.74	0.38	0.06	0.01	0	9
RoPE +reverse +pad	0.35	0.30	0.09	0.02	0	11

How to improve NUPA?

北京大学 PEKING UNIVERSITY

- Chain-of-thought?
- We train **RFFT** on several tasks in NUPA tests.
- RFFT has **better performance …**
  - but it is restricted by **context window** and **inference time.**

Table 5: Average inference time.

	batchsize	sec / sample
RF CoT	128	5.625
Direct	128	0.371
Direct	256	0.336

Table 6: Performance of RF CoT. "-" means exceeding context window limitation (2k tokens).

Exact Match	Add Float			Multiply Fraction			Max Scientific			Mod Integer		
# Digit	5	6	7	2	3	4	38	39	40	6	7	8
RF CoT	**1.00**	**1.00**	-	**0.90**	**0.85**	-	1.00	0.85	0.90	**0.70**	**0.35**	-
GPT-4o	0.78	0.66	0.49	0.53	0.20	0.00	0.37	0.46	0.36	0.01	0.00	0.00
Qwen2-72B	0.62	0.50	0.70	0.05	0.00	0.00	0.96	**0.98**	**0.95**	0.03	0.00	0.00

Take-home message

北京大学 PEKING UNIVERSITY

- We propose **NUPA tests**, which is a comprehensive benchmark to test the **number understanding and processing ability** of LLMs.
- Current LLMs shows pool ability on **less common** tasks or **longer** numbers
- Some tricks seem to be **useful** in **pretraining** stage but **cannot** be directly used in **SFT.**
- **CoT** is a possible way to improve the NUPA but **inference time** and **context window** are bottleneck.



Thanks for listening!