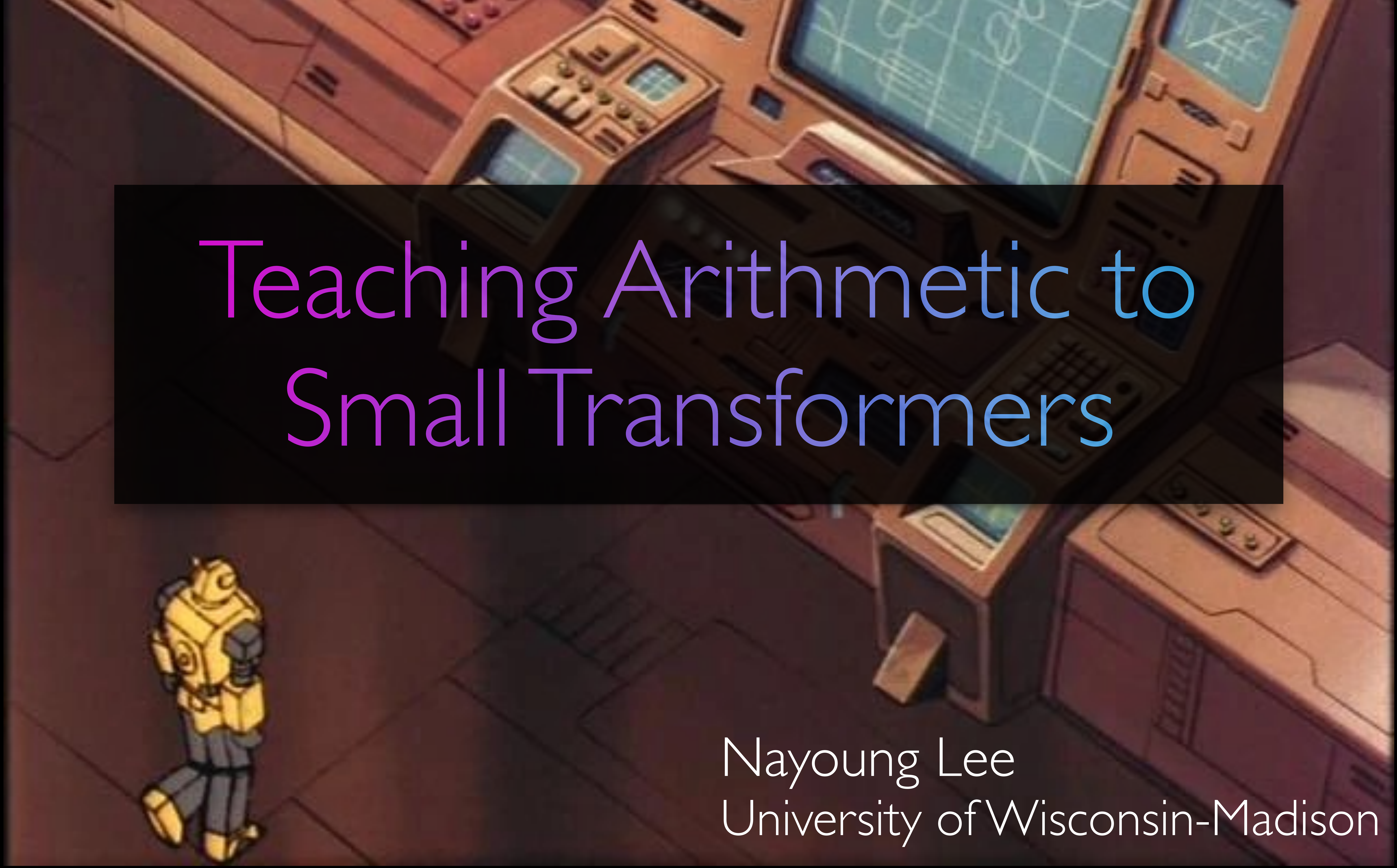


Teaching Arithmetic to Small Transformers

Nayoung Lee
University of Wisconsin-Madison

Joint work with




Kartik Sreenivasan

Kangwook Lee

Jason Lee

Dimitris Papailiopoulos

Emergent Properties

- LLMs when trained on vast amounts of data, eventually acquire impressive skills, including basic arithmetic

Which is *surprising*

- These tasks are not *explicitly encoded* in the *next-token prediction* objective

2023. Feb.

4809234 + 3509825

4809234 + 3509825 = 8319069

Looks like your addition is off by 10

I apologize for the error in my previous response. You are correct that my addition was off by 10.

The correct answer to 4809234 + 3509825 is 8319069. Thank you for bringing this to my attention.

Addition is Hard…

Q. How do decoder models learn addition?

How do we elicit emergence fast??

- Prior research delved into emergence wrt scale

- Untangling the factors that contribute to emergence is hard:
  - Data: Too much data
  - Models: Too many models
  - Scale: Too many parameters

- Our solution:
  1. Choose the simplest setting: Addition
  2. Albate, ablate, ablate

ABLATE!

Let's teach addition to nanoGPT

Let's focus on a simple setting

- NanoGPT: small decoder-only TF architecture
  - # param: ~10M
  - 6 layers
  - 6 heads/layer
  - 384 embedding dimension

- Character level tokenizer, i.e., {0,1,2..,9, +, \n}

- Task: Primarily addition $(+)$, extended to $(-, \times, \sqrt{\ }, \sin)$

- Goal: Evaluate the importance of sampling, *formatting*, and prompting

How does training happen?

0+1 = 1
1+2 =3
10+5 = 15
10+20 = 30
...

the loss is cross-entropy on

$$\Pr(c \,|\, '1', '+', '2', '=')$$

against the one hot vector that is
1 at $c = 3$ and 0 elsewhere

But then…

next-word prediction
so weird for arithmetic!

P(digit| "43+99= ") = ?

Format of training examples matters!

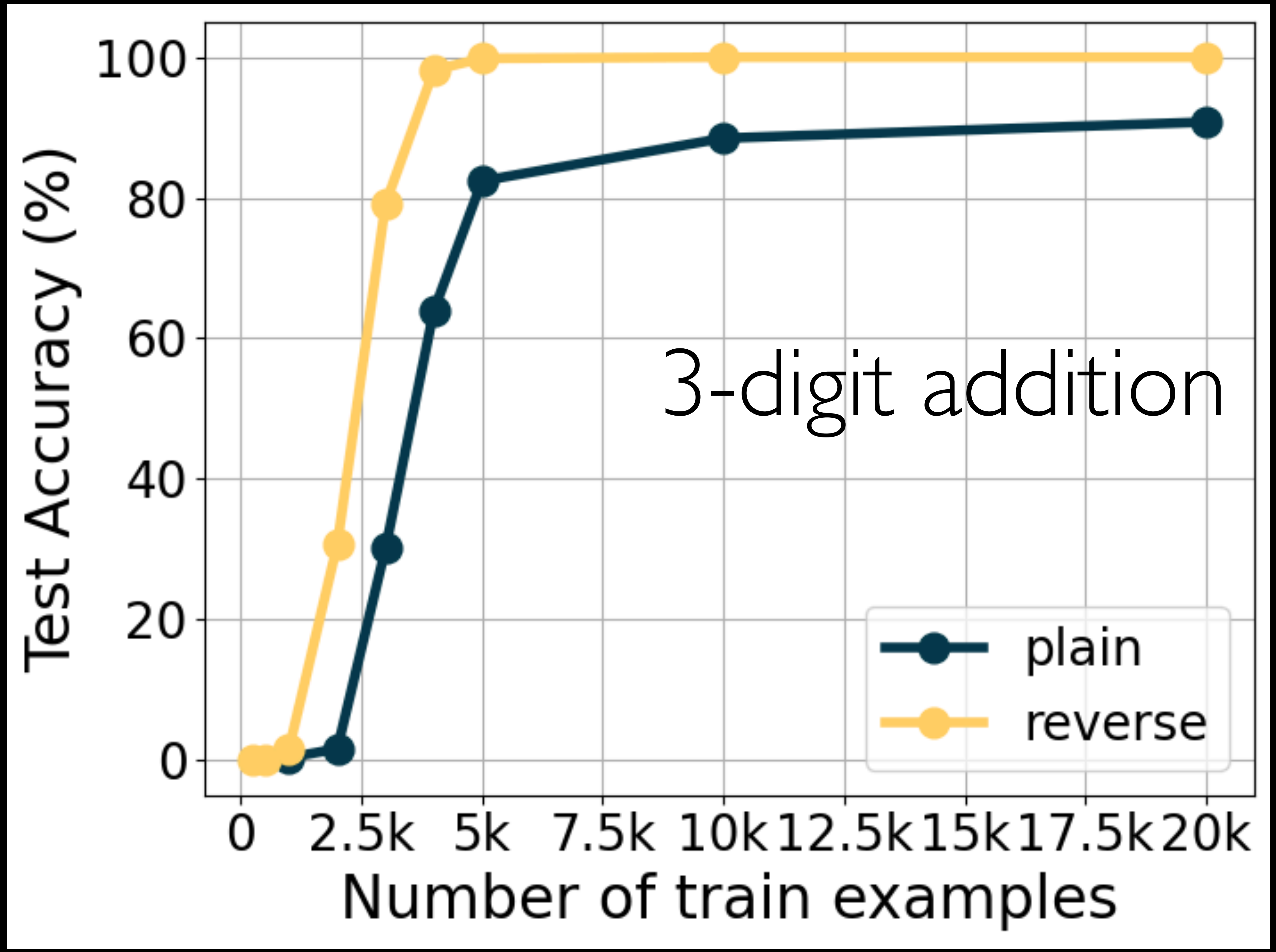
n-digit addition

Addition	Reversed output
128+367=495	128+367=594
MSB first: one needs to know all 2n digits	LSB first: one needs to know 2 digits + carry

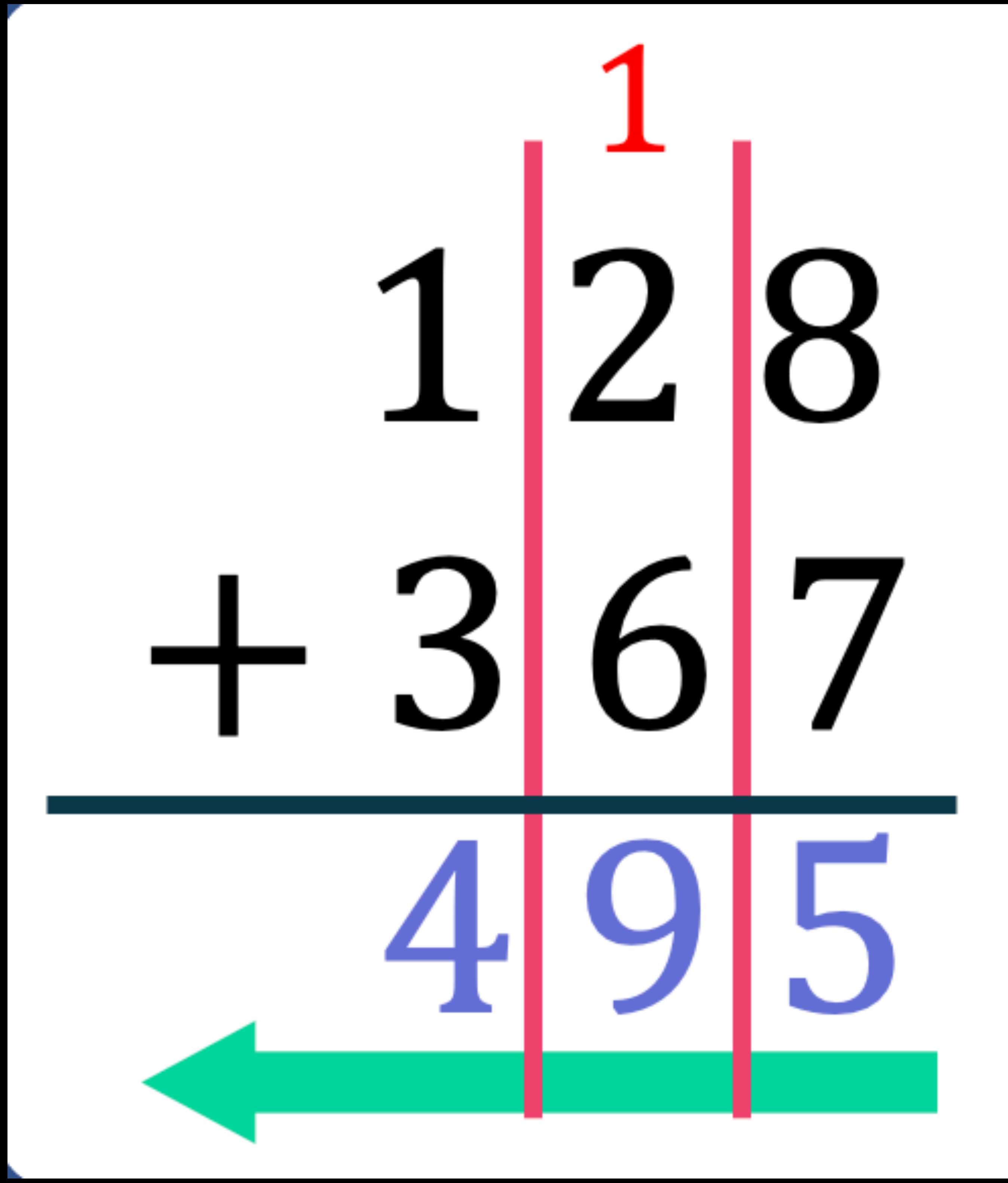
Format of training examples matters!

Addition	Reversed output
128+367=495	128+367=594
MSB first: one needs to know all 2n digits	LSB first: one needs to know 2 digits + carry

Model can learn a *simpler function* with reversed output!



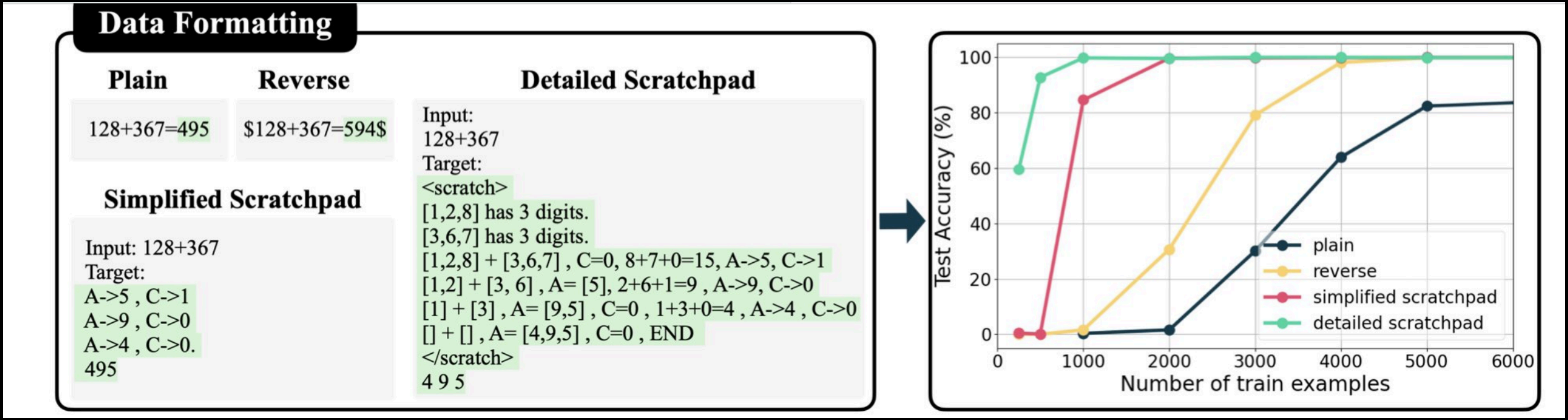
Also.. How do we add in practice?



We add by

- 1) going in reverse significance order
- 2) producing intermediate carries
- 3) taking it STEP BY STEP

Varying training data formats

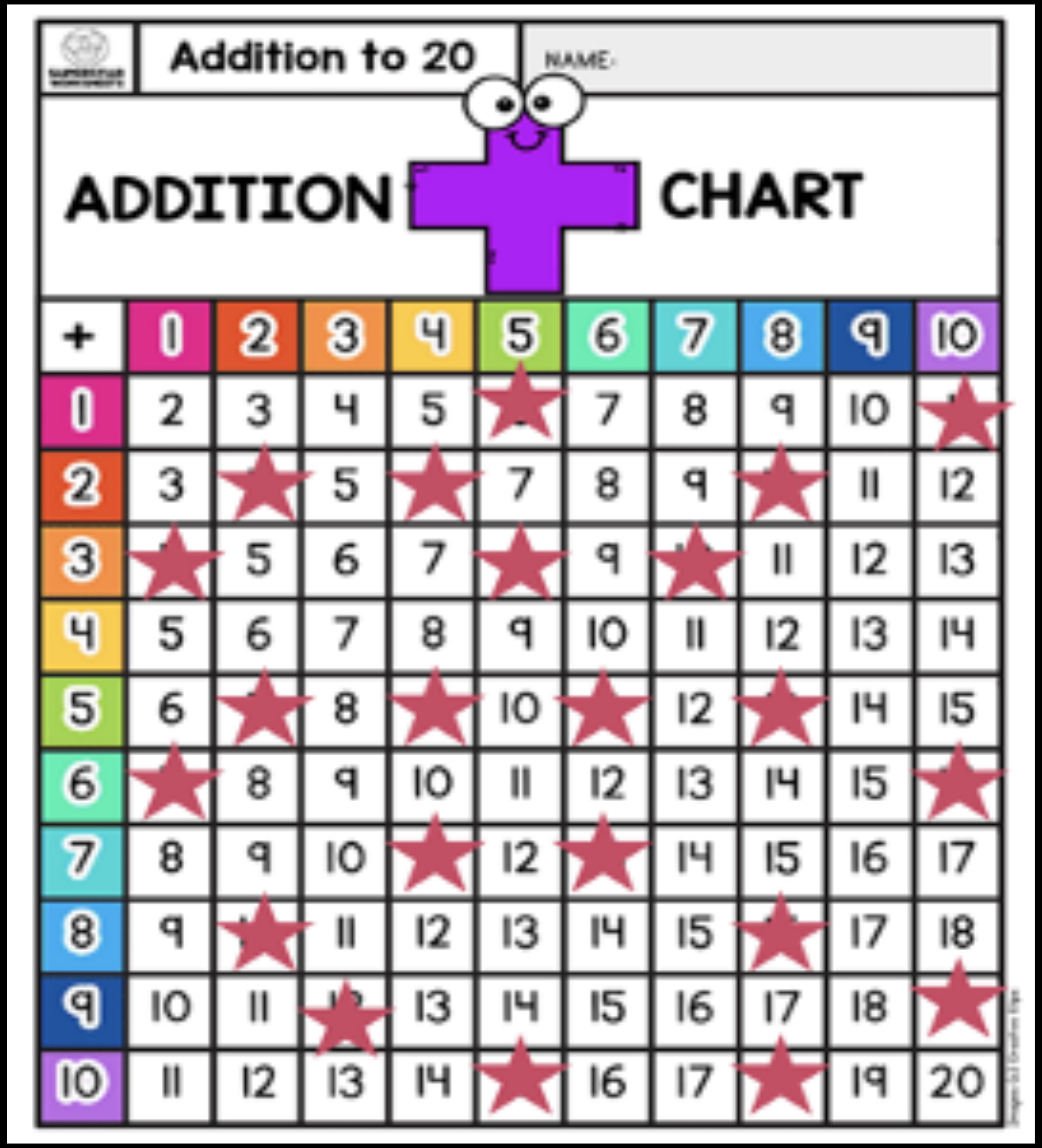


Simple formatting changes make a HUGE difference.

- eg. A+B=C → A+B=reverse(C) => MUCH faster & accurate learning.

- Using CoT training data teaches compositions of functions by breaking it down to simpler ones to be learnt *helps a lot*

Hints on Foundations of Emergence?



Addition to 20 | NAME:

ADDITION CHART

+	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5		7	8	9	10	
2	3		5		7	8	9		11	12
3		5	6	7		9		11	12	13
4	5	6	7	8	9	10	11	12	13	14
5	6		8		10		12		14	15
6		8	9	10	11	12	13	14	15	
7	8	9	10		12		14	15	16	17
8	9		11	12	13	14	15		17	18
9	10	11		13	14	15	16	17	18	
10	11	12	13	14		16	17		19	20

Filling-up the addition chart $\equiv$ LRMC!

Q: why does addition emerge rapidly from 0->100% accuracy?

A: Addition maps <u>up to a fixed digit n</u>, are low-rank! $(\mathbf{M} = \mathbf{n}\mathbf{1}^T + \mathbf{1}\mathbf{n}^T)$

"Learning" fixed length addition ~ low-rank matrix completion (LRMC)

→ goes from 0 to 100% when you see $O(n)$ out of n^2 samples!

MC viewpoint doesn’t explain
some interesting generalization aspects

NanoGPT generalizes better than MC solutions!

- NanoGPT can add **unseen numbers**!
  - Hiding numbers in both operands

312+527=839
350+527=877
527+439=966

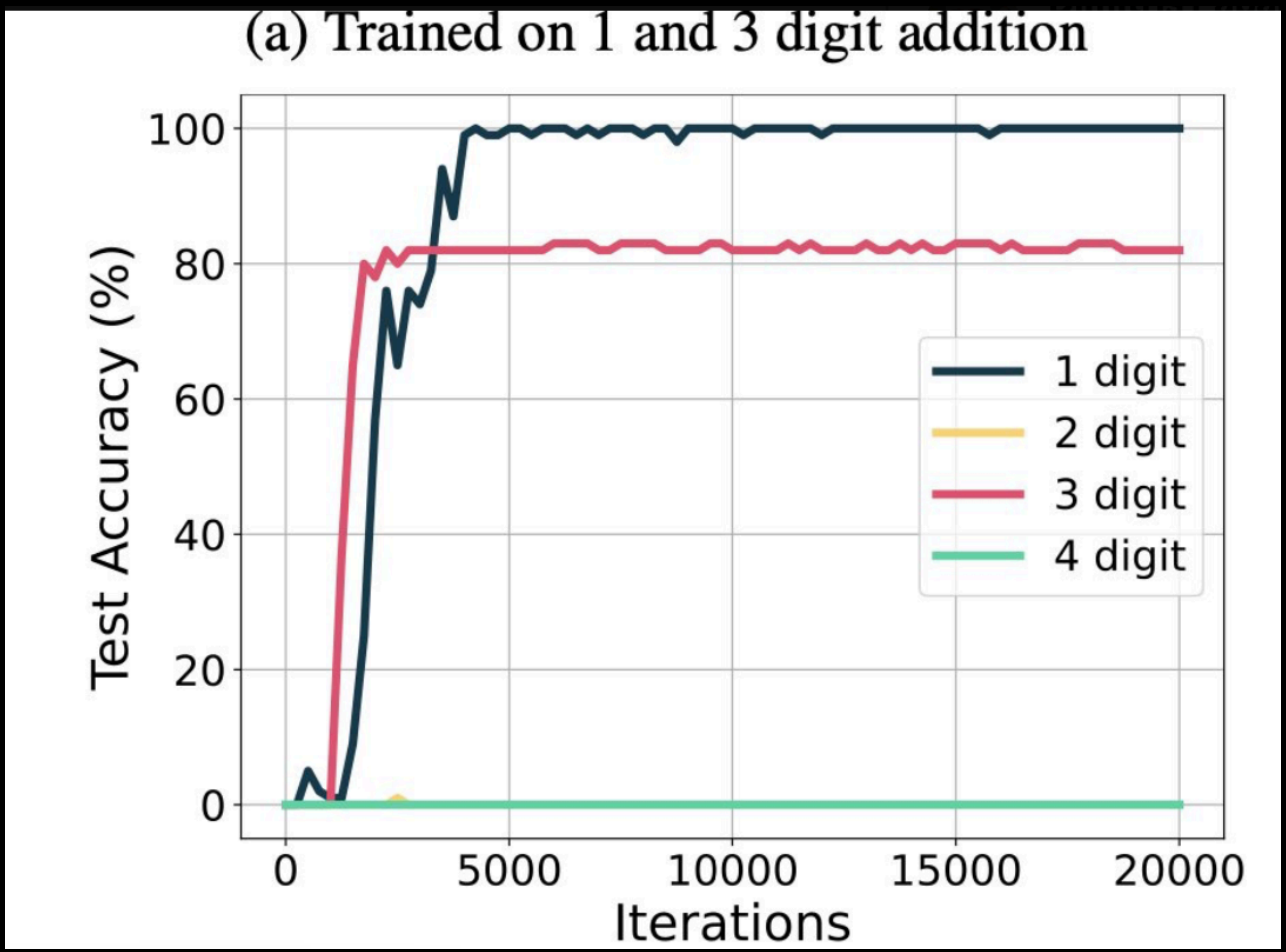
312+547=859
350+529=879
526+439=965

	No Exclusion		Excluding 100 numbers		Excluding 200 numbers		Excluding 500 numbers	
	Plain	Rev	Plain	Rev	Plain	Rev	Plain	Rev
Overall Accuracy	87.18%	99.97%	87.94%	100.00%	87.24%	99.99%	88.15%	99.99%
Exclusion Accuracy	-	-	92.55%	100.00%	92.15%	99.95%	90.85%	100%

- NanoGPT can add **unseen digits**!

The "Matrix Completion" interpretation predicts 0s
NanoGPT does not!

Wow, nanoGPT "knows" how to add??



Q: Do LMs "understand" addition? (i.e., do they implement the ADD algorithm)

Length generalization beyond trained digit lengths is HARD

Even for lengths that are smaller than the max seen during training (eg if you skip 2-digit examples)

The models don't "fully understand" addition

They implement "pseudo" algorithms. Even hints don't help

Case 1: Just asking the question

```
Input:
8465+3541
Target:
<scratch>
[8,4,6] has 3 digits.
[3,5,1] has 3 digits.
[8,4,6] + [3,5,1] , A=[] , C=0 , 6+1+0=7 , A->7 , C->0
[8,4] + [3,5] , A=[7] , C=0 , 4+5+0=9 , A->9 , C->0
[8] + [3] , A=[9,7] , C=0 , 8+3+0=11 , A->1 , C->1
[] + [] , A=[1,9,7] C=1 , END
</scratch>
1 1 9 7
```

←Randomly drops a digit

They implement "pseudo" algorithms. Even hints don't help

Case 4: Giving all but one intermediate steps

```
Input:
8465+3541
Target:
<scratch>
[8,4,6,5] has 4 digits.
[3,5,4,1] has 4 digits.
[8,4,6,5] + [3,5,4,1] , A=[] , C=0 , 5+1+0=6 , A->6 , C->0
[8,4,6] + [3,5,4] , A=[6] , C=0 , 6+4+0=10 , A->0 , C->1
[8,4] + [3,5] , A=[0,6] , C=1 , 4+5+1=10 , A->0 , C->1
[8] + [3] , A=[0,0,6] , C=1 , 8+3+1=12 , A->2 , C->1
[] + [] , A=[2,0,6] C=1 END    ←Randomly drops a digit
</scratch>
1 0 0 6
```

hints don't help, they seem to just be bad at unseen digit lengths

Many more in our paper

- beyond addition
- mixing arithmetic with text data
- few-shot prompting
- effect of noise/mistakes in prompts
- effect of scale/finetuning (nanoGPT, GPT-2, GPT-3)
- token efficiency of different formats (CoT vs plain)

50 pages worth of ablations :)

Teaching Arithmetic to Small Transformers

Nayoung Lee*
University of Wisconsin-Madison
nayoung.lee@wisc.edu

Kartik Sreenivasan*
University of Wisconsin-Madison
ksreenivasa2@wisc.edu

Jason D. Lee
Princeton University
jasonlee@princeton.edu

Kangwook Lee
University of Wisconsin-Madison
kangwook.lee@wisc.edu

Dimitris Papailiopoulos
University of Wisconsin-Madison
dimitris@papail.io

Abstract

Large language models like GPT-4 exhibit emergent capabilities across general-purpose tasks, such as basic arithmetic, when trained on extensive text data, even though these tasks are not explicitly encoded by the unsupervised, next-token prediction objective. This study investigates how small transformers, trained from random initialization, can efficiently learn arithmetic operations such as addition, multiplication, and elementary functions like square root, using the next-token prediction objective. We first demonstrate that conventional training data is not the most effective for arithmetic learning, and simple formatting changes can significantly improve accuracy. This leads to sharp phase transitions as a function of training data scale, which, in some cases, can be explained through connections to low-rank matrix completion. Building on prior work, we then train on chain-of-thought style data that includes intermediate step results. Even in the complete absence of pretraining, this approach significantly and simultaneously improves accuracy, sample complexity, and convergence speed. We also study the interplay between arithmetic and text data during training and examine the effects of few-shot prompting, pretraining, and model scale. Additionally, we discuss length generalization challenges. Our work highlights the importance of high-quality, instructive data that considers the particular characteristics of the next-word prediction objective for rapidly eliciting arithmetic capabilities.[2]

Contents

Key Take-aways

- Data formatting and sampling matters

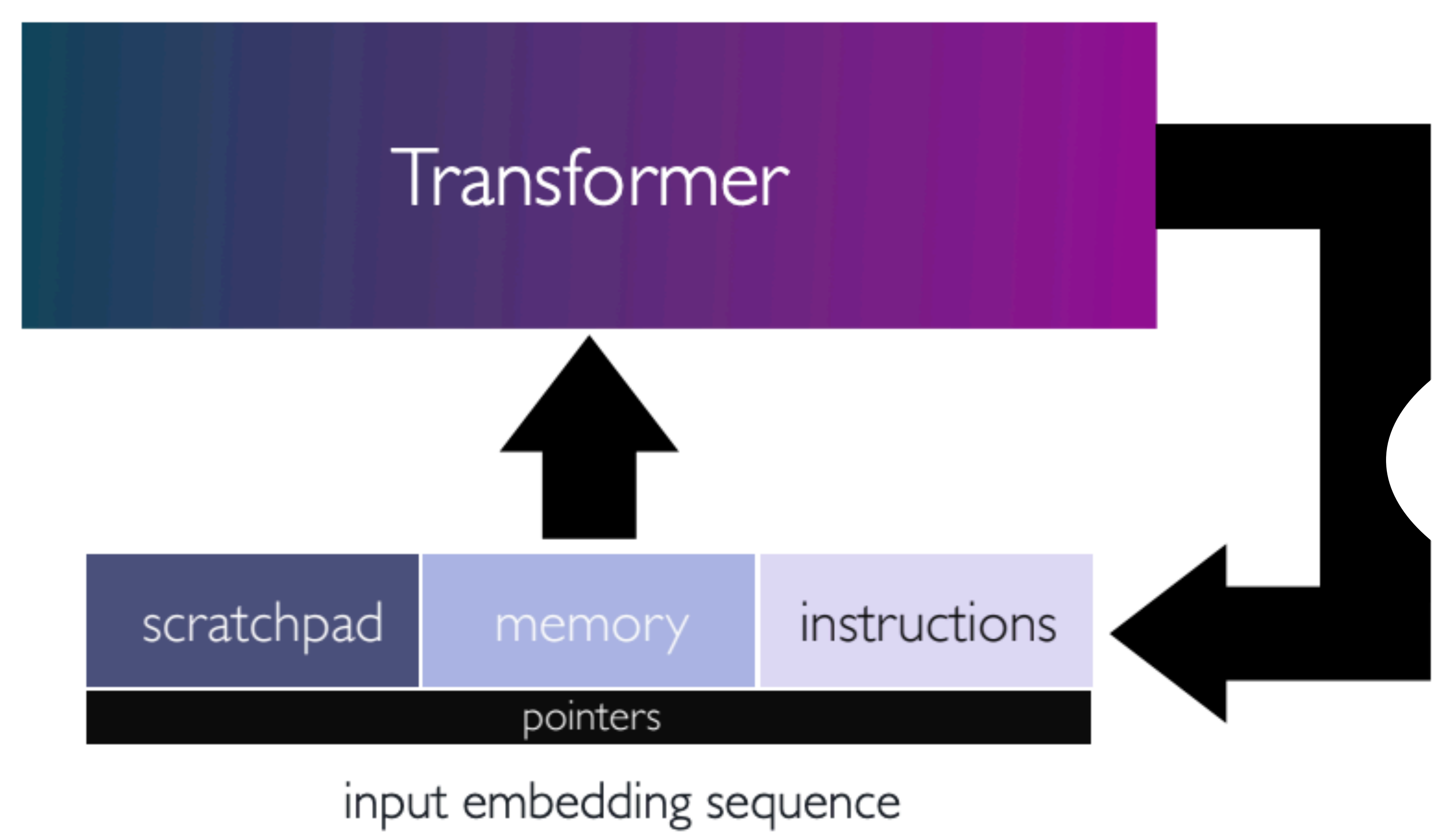
- Low-rank matrix completion partially explains the emergence of addition (0% to 100% accuracy), but transformers generalize better

- Length generalization is still challenging!

Open Problem:
Can we teach LLMs *using samples* to implement *algorithms*, not just *approximate functions*?

Looped Transformers as Programmable Computers

Angeliki Giannou[w*], Shashank Rajput[w*], Jy-yong Sohn[w],
Kangwook Lee[w], Jason D. Lee[p], Dimitris Papailiopoulos[w]



Yes!
(If we hardcode)

But just from samples, using next-token prediction seems hard!

Thank you

Teaching Arithmetic to Small Transformers

Nayoung Lee*
University of Wisconsin-Madison
nayoung.lee@wisc.edu

Kartik Sreenivasan*
University of Wisconsin-Madison
ksreenivasa2@wisc.edu

Jason D. Lee
Princeton University
jasonlee@princeton.edu

Kangwook Lee
University of Wisconsin-Madison
kangwook.lee@wisc.edu

Dimitris Papailiopoulos
University of Wisconsin-Madison
dimitris@papail.io

Abstract

Large language models like GPT-4 exhibit emergent capabilities across general-purpose tasks, such as basic arithmetic, when trained on extensive text data, even though these tasks are not explicitly encoded by the unsupervised, next-token prediction objective. This study investigates how small transformers, trained from random initialization, can efficiently learn arithmetic operations such as addition, multiplication, and elementary functions like square root, using the next-token prediction objective. We first demonstrate that conventional training data is not the most effective for arithmetic learning, and simple formatting changes can significantly improve accuracy. This leads to sharp phase transitions as a function of training data scale, which, in some cases, can be explained through connections to low-rank matrix completion. Building on prior work, we then train on chain-of-thought style data that includes intermediate step results. Even in the complete absence of pretraining, this approach significantly and simultaneously improves accuracy, sample complexity, and convergence speed. We also study the interplay between arithmetic and text data during training and examine the effects of few-shot prompting, pretraining, and model scale. Additionally, we discuss length generalization challenges. Our work highlights the importance of high-quality, instructive data that considers the particular characteristics of the next-word prediction objective for rapidly eliciting arithmetic capabilities.[2]

Contents