Length Generalization on Multi-Digit Integer Addition with Transformers

Imran Ibrahimli
imran.ibrahimli@studium.uni-hamburg.de
Knowledge Technology, WTM
Dept. Informatik
Universität Hamburg

Advisors:
Prof. Dr. Stefan Wermter, Dr. Jae Hee Lee

19.11.2024

Outline

Outline

Motivation

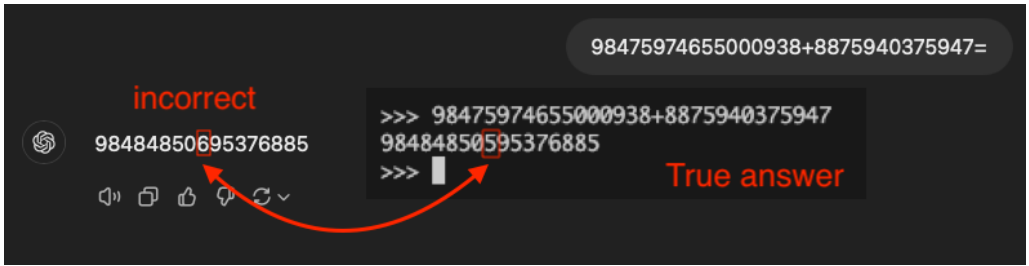
- Transformer models are being adopted across many domains from language modeling to robotics.
- For effective applications, models need to learn generalizing operations or algorithms from data.
- Generalization enhances robustness and trust in AI systems.

Motivation (cont.)

- Examine a simple toy task: integer addition.
- We don't need transformers to perform addition.
- But we care if they can learn "simple" algorithms from examples.
- Length-generalizable multi-digit addition is an open problem for transformers.
- Actively worked on with many papers in NeurIPS, ICLR, etc.

Motivation (cont.)

- Example: even SOTA model GPT-4o struggles with addition.



Motivation (cont.)

- Why is this important?
- Maybe just guardrail models by hard-coding some rules?
- Not generally, because we would not train and use the models, if we could hard-code the rules in the first place.
- We want to understand the limitations of the models and improve them.

Problem Statement

- Focus on standard decoder-only transformers with absolute positional encodings.
- Try to improve generalization without altering model architecture or task-specific modifications.
- Explore data formatting and training data diversity.

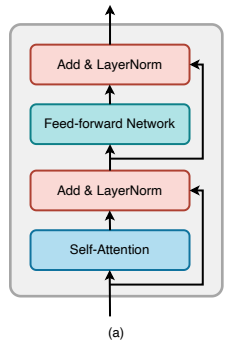
Research Questions

1. Why do transformers with absolute positional encodings fail to generalize integer addition to longer sequences?
2. How does the inclusion of sub-task data influence the model's compositionality and length generalization capabilities?

Outline

Transformers

- Utilize self-attention to process sequential data.
- Focus on decoder-only models (b).
- Capable of capturing long-range dependencies without recurrence.



Positional Encodings (PE)

- No explicit positional awareness in transformers.
- PEs inject sequence order information.
- Can be absolute or relative.
  - Absolute PEs:
    `0, 1, 2, 3, ...`
  - Relative PEs:
    `..., -3, -2, -1, 0, 1, 2, 3, ...`

Absolute Positional Encoding

- Adds fixed positional information to input embeddings.
- Sinusoidal absolute PEs introduced by Vaswani et al. (2017):

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Outline

Challenges in Length Generalization

- Easy to grok addition for in-distribution lengths.
- Transformers struggle with sequences longer than seen in training.
- Addition requires accurate *digit alignment* for all lengths.
- Positional encodings are crucial for digit alignment.

Issues with Positional Encodings

- Models struggle to select relevant tokens based on position in longer sequences.
- Failure to align digits without explicit positional cues.
- Research focuses on easing digit alignment.

Improving Length Generalization

- Reversing answer and training with scratchpad (Lee et al. 2024).
- Randomized PEs (Ruoss et al. 2023).
- Random spaces and scratchpad (Shen et al. 2023).
- Index hints `a1b2c3+a4b5c6=a5b7c9` (Y. Zhou et al. 2024)
- Task-specific PEs, e.g. Abacus (McLeish et al. 2024).

Length Generalization Ratio

- Ratio of the length of solved test problems to training lengths.
- 1x with data formatting (Lee et al. 2024).
- 1.1x with Random PE (Shen et al. 2023).
- 1.125x with NoPE (Kazemnejad et al. 2023).
- 1.5x with index hints and special setup (H. Zhou et al. 2023).
- 2.5x with FIRE and index hints (Y. Zhou et al. 2024).
- **6x** achieved with Abacus encoding (McLeish et al. 2024).

SOTA: Abacus Encoding

- Encode digit position relative to the start of the number.
- Task specific and requires reversing numbers.
- Use sinusoidal PE with indices specified by the Abacus encoding.
- Combined with architectural modifications like input injection and recurrent layers.
- Trained on up to 20 digits, generalizes to 120 digits.

Figure: Source: McLeish et al. 2024

Conclusions from Related Work

- No common dataset $\rightarrow$ hard to compare methods.
- Positional encodings and data formatting significantly impact length generalization.
- Task-specific design can boost generalization to unseen lengths.
- Some diverge from standard decoder unsupervised training.

Limitations of Existing Methods

- Some successful methods lose sight of the original task:
  It's not about addition, but about model capabilities.
- Task-specific modifications are not always feasible.
- For the same reason e.g. index hints or scratchpad are "hacky."
- No common benchmark datasets or setups.
- Desire for simpler methods without altering model architecture.

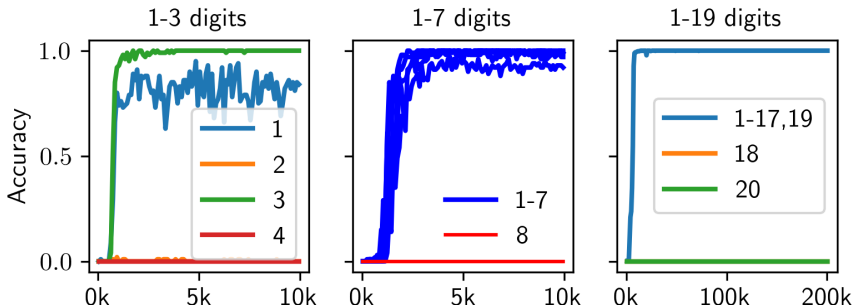
Outline

Overview of Approach

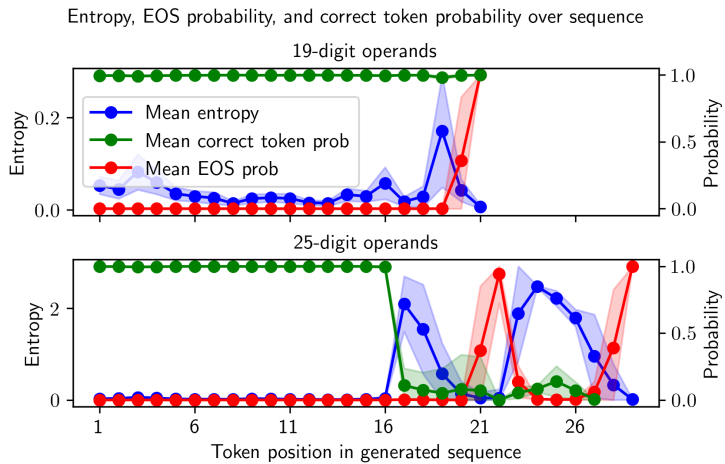
- Baseline performance (Lee et al. 2023).
- Fixed positional patterns in absolute PE.
- Length generalization with different data formats.
- Weak generalization by breaking positional patterns.
- Sub-task data to improve compositionality.

Baseline with Absolute Positional Encoding

- High accuracy on training lengths (1 and 3 digits).
- No generalization on unseen lengths (2 and 4 digits).



Next-Token Uncertainty Analysis



Next-Token Uncertainty Analysis

- For in-distribution lengths (1–17 and 19), models are confident in predictions (low entropy)
- For any OOD lengths (18 and 20+), entropy increases and model becomes uncertain.
- Next-token distribution ≈uniform for extreme OOD lengths.
- Predictions are not "confident and wrong" but "uncertain and wrong."
- Indicates that learned algorithm "hardcodes" positional patterns.

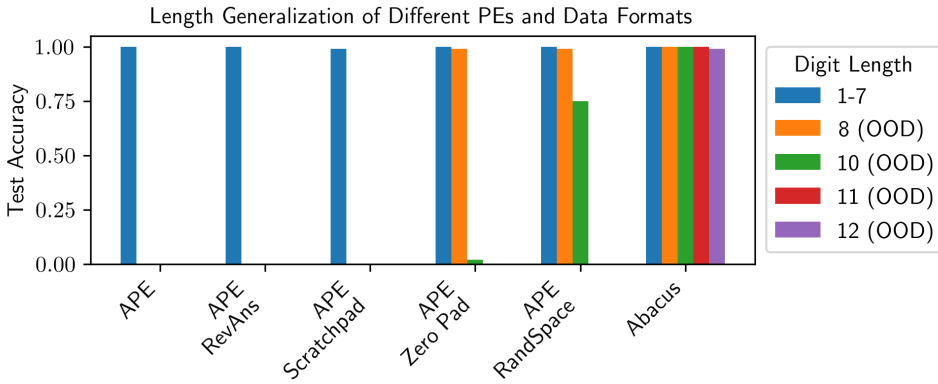
Limitations of Absolute Positional Encoding

- No generalization to unseen lengths.
- Neither in-between lengths (e.g. 18) nor longer lengths (20+).
- Learned algorithm completely breaks for OOD lengths.
- Literature suggests aligning digits is the key challenge.
- Next: try to break positional patterns with data formatting.

Data Formatting Techniques

- Standard Format
  $123+456=579$
- Zero Padding
  $00123+00456=00579$ ($N_{\text{pad}} = 5$)
- Reversing
  $321+654=975$
- Random Spaces
  $1 23 +4 5 6=579$
- Scratchpad
  $567+789=7 6 5 + 9 8 7; c=0,7+0+0=7,c=0;`
  `6+9+0=5,c=1; 5+8+1=4,c=1; 0+7+1=8,c=0|8457$

Data Formatting Results



Data Formats Summary

- Zero padding can "close the gap" in OOD lengths (7, **8**, 9 digits).
- Random spaces weakly improve generalization (x1.125).
- Other methods do not help generalization.
- Random spaces smooth attention patterns.
- Abacus generalizes well.

Sub-tasks for Compositionality

- A different direction to improve generalization.
- Encourage model to learn an *algorithm*.
- Include *sub-task* data in training.
- Sub-tasks can be composed to perform addition.

Sub-task Data

- Parts of the addition process broken down into sub-tasks.
- Sub-tasks include:
  - Digit Alignment
  - Reversing
  - Carry Detection
  - Digit-wise Modular Addition
  - Addition
- Each sub-task has a specific format.
- Add a prefix to the input sequence to indicate the sub-task.
  `$pre123+456=`

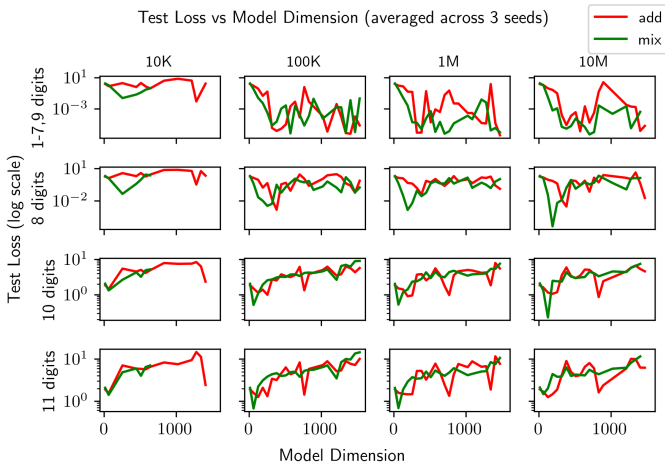
Experiment Setup

- 1-7 digits in-distribution, 8 and 10 digits OOD.
- Model dimension: 64–1536 (increments of 64).
- Dataset sizes: 10K, 100K, 1M, 10M samples.
- Addition-only and mixed-task for each.
- 3 seeds per run.

Impact of Sub-task Learning

Dataset	8 Digits Add	8 Digits Mix	10 Digits Add	10 Digits Mix	Diff
10K	7.0 ± 22.7	35.4 ± 39.6	0 ± 0	2.4 ± 8.4	**+15.4**
100K	21.5 ± 29.7	41.8 ± 36.5	0 ± 0	1.0 ± 5.2	**+10.6**
1M	16.1 ± 24.8	39.5 ± 36.3	0 ± 0	1.5 ± 7.6	**+12.4**
10M	18.6 ± 33.1	37.3 ± 37.2	0 ± 0	3.1 ± 10.1	**+10.9**
Diff		**+22.7**		**+2.0**	

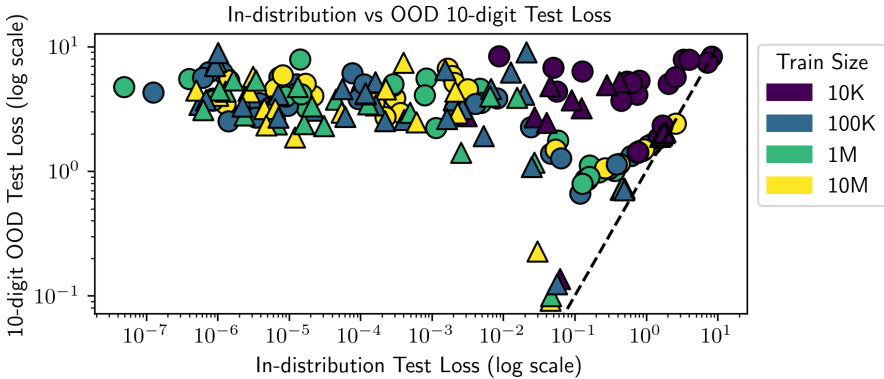
Test Loss vs. Model Dimension



Test Loss vs. Model Dimension

- Smaller models benefit more from mixed-task training.
- Effect diminishes as model size increases.
- Mixed-task training never hurts performance.

In-Distribution vs. OOD Test Loss



In-Distribution vs. OOD Test Loss

- Mixed-task can be a Pareto-improvement.
- Allows to reduce OOD loss without increasing in-distribution loss.

Outline

Summary of Findings

- Absolute positional encodings fail to generalize.
- Generalization possible without architecture changes.
- Random spaces can improve generalization.
- Sub-task data improves generalization, especially for smaller models and datasets.

Limitations

- Focused on small models trained from scratch; applicability to large language models (LLMs) is unclear.
- Other positional encodings were not explored; alternative solutions might exist.
- Sub-task experiments may be under-trained due to limited training steps.
- Lack of mechanistic interpretability.

Future Work

- Develop mechanistic interpretability methods for algorithmic tasks.
- Apply to fine-tuning pre-trained LLMs and data curation.
- Experiment more with smaller models for sub-task learning.
- Explore active learning and curriculum learning.

Questions

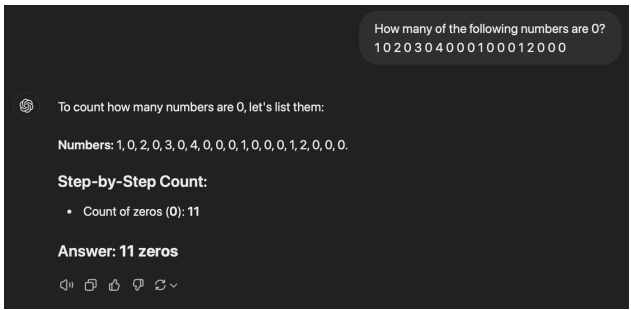
Thank you for your attention!

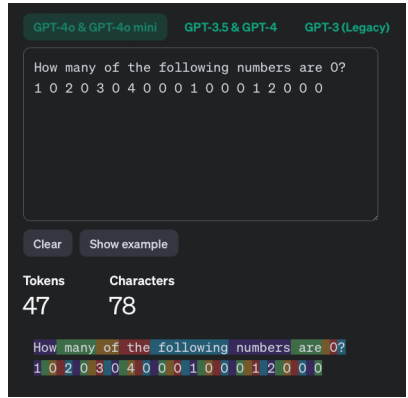
Questions?

Outline

Appendix: Counting Failure

- Example: SOTA model GPT-4o struggles with simple counting.





Appendix: Data Generation Process

- Operands are randomly sampled positive integers
- Each operand has exactly the specified number of digits (no leading zeros)
- For digit lengths 1 to 3:
  - 1-digit operands: All 100 combinations (0 to 9)
  - 2-digit operands: 900 samples randomly selected
  - 3-digit operands: 9,000 samples randomly selected
- For digit lengths 4 and above:
  - Equal number of samples per digit length to fill the dataset size
- Test sets contain unique samples not present in the training set

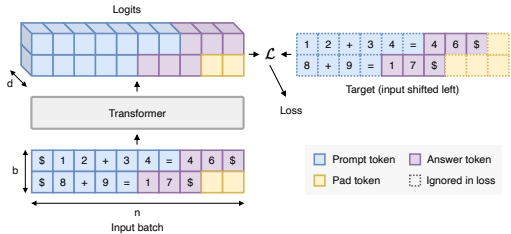
Appendix: Self-Attention Mechanism

- Computes attention weights between all token pairs.
- Allows the model to focus on relevant parts of the sequence.
- Attention function:

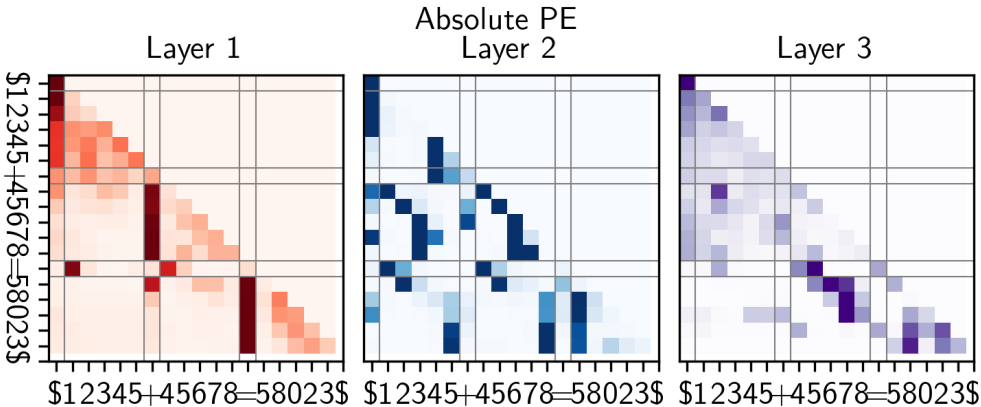
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- Where Q, K, V are query, key, and value matrices.

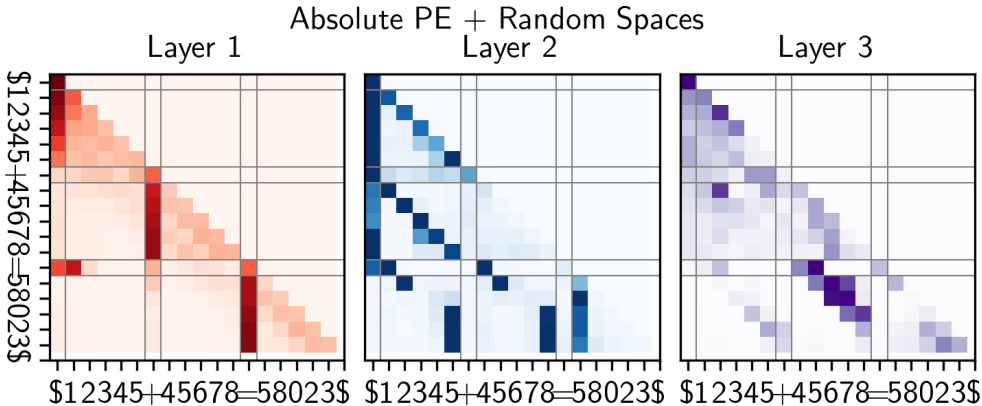
Appendix: Training and Inference



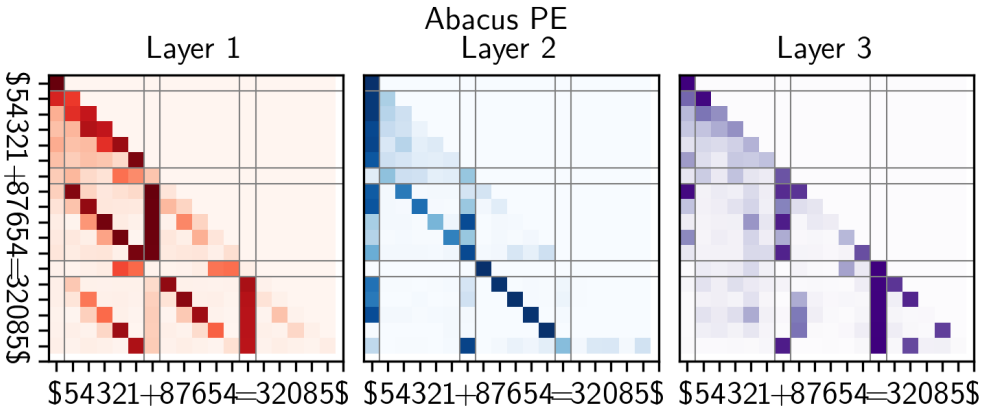
Attention Maps: Absolute PE



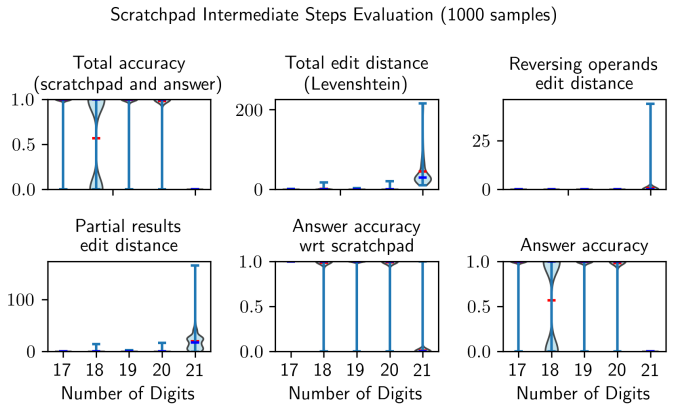
Attention Maps: Absolute PE with Random Spaces



Attention Maps: Abacus PE



Appendix: Scratchpad Evaluation



Appendix: Sub-task Prefix

- Includes a 3-letter task prefix to differentiate sub-tasks
- Format: `xxx$a+b=c$`, where `xxx` is the sub-task identifier

Example (Reversing sub-task):

`rev$123+456=321+654$`

Appendix: Sub-task: Digit Alignment (`ali`)

- Focuses on aligning digits of the operands
- Model outputs corresponding digit pairs from each operand

Example:

`ali$1234+4567=1+4,2+5,3+6,4+7$`

Appendix: Sub-task: Reversing (`rev`)

- Involves reversing the digits of each operand
- Helps model understand the reversal operation

Example:

`rev$1234+4567=4321+7654$`

Appendix: Sub-task: Carry Detection (car)

- Model identifies positions where a carry operation occurs
- Output is a string of 'c's and dashes indicating carries

Example:

`car$1234+4567=---c$`

Appendix: Sub-task: Modular Addition (mad)

- Model performs addition modulo 10 on each pair of corresponding digits
- Does not consider carries

Example:

`mad$1234+4567=5791$`

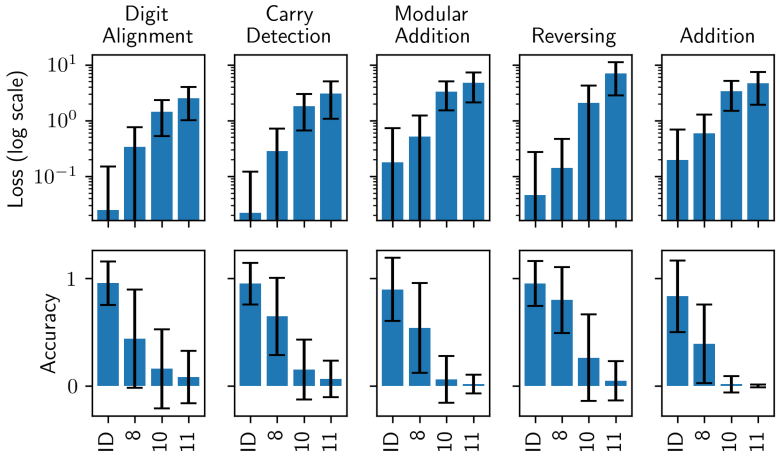
Appendix: Sub-task: Addition (add)

- Standard addition task
- Model computes the sum of the two operands

Example:

`add$1234+4567=5801$`

Sub-task Difficulty Analysis



- Modular addition and digit alignment are the most challenging