Universal Length Generalization with Turing Programs

Kaiying Hou*, David Brandfonbrener, Sham Kakade, Samy Jelassi†, Eran Malach†
Kempner Institute, Harvard University

Summary

Problem: Length generalization refers to the ability to extrapolate from short training sequences to long test sequences and is a challenge for current large language models. Is there a recipe to achieve length generalization for a variety of algorithmic tasks?

Solution: Turing Programs + Hard-ALiBi

- Turing Programs: CoT mimicking Turing machines. At each step, we copy the content of the tape and modify it by performing one operation.
- Hard-ALiBi: positional encoding similar to NoPE, except we mask out positions that are more than a fixed distance away.

Results:

Problem	Generalization	Accuracy
Addition (n+n)	$50 \to 100$ $(2\times)$	98%
Multiplication ($n \times 1$)	$50 \to 100$ $(2\times)$	97%
Multiplication ($n \times 3$)	$50 \to 100$ $(2\times)$	97%
SGD (n examples)	$50 \to 80$ $(1.6\times)$	95%

Turing Program

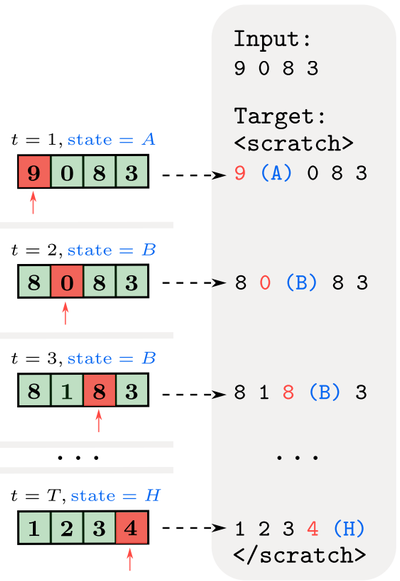

Figure: Turing Program example for simulating a Turing Machine with scratchpad.

Hard-ALiBi

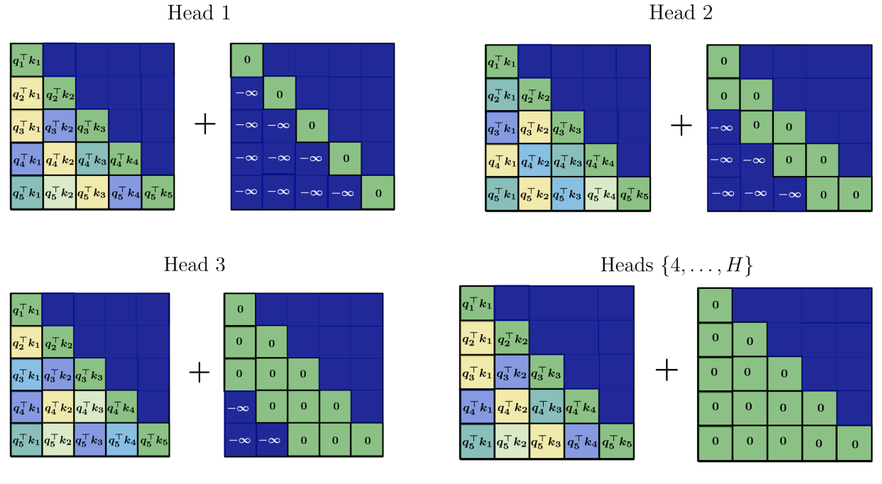

Figure: Hard-ALiBi: an additive positional encoding where the bias satisfies $b(i,j) = -\infty$ for $j \leq i - m$ and $b(i,j) = 0$ for $j > i - m$, for some hyperparameter $m > 0$. (Figure from Jelassi et al. 2024)

Addition

```
Input:
4 3 2 4 + 1 3 9

Target:
<scratch>
4 3 2 4 + 1 3 9
4 3 2 e + 1 3 j (1,3)    # added 4 + 9 = 3 carry 1
4 3 c + 1 d (0,63)       # added 2 + 3 + 1 = 6 carry 0
4 d + b (0,463)          # added 3 + 1 = 4 carry 0
e + ^ (0,4463)           # added 4 + 0 = 4 carry 0
4 4 6 3
</scratch>
```

Figure: Turing Program for addition; text in comments is not part of the input.

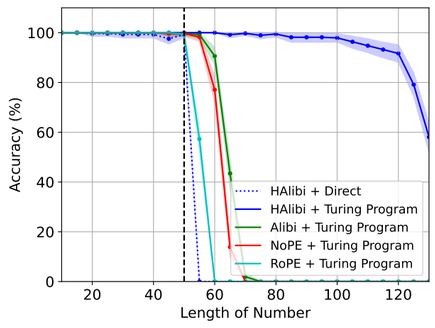

Figure: Comparison of positional encodings and data formats on addition generalization. Shade = 95 % CI.

Data: Our token space is of size 24 and made of $\mathcal{V} = \{0, \ldots, 9, +, a, \ldots, j, \hat{}, ,, \}$. All the digits are sampled uniformly as follows: we first sample the length of each operand (between 2 and $L = 50$) and then independently sample each digit. We set the training context length to 500. At test time, we evaluate our models using a sliding window.

Model: Our base model is a 150M parameter transformer with $L = 12$ layers, a $D = 1024$ hidden size, feedforward layer with a hidden dimension of 4096 and $H = 16$ attention heads. The backbone of our model is based on the GPT-NeoX architecture.

More Results

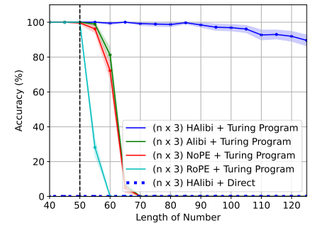

Figure: Comparison of positional encodings and data formats for length generalization on $(n \times 3)$-digit multiplication (95% CI).

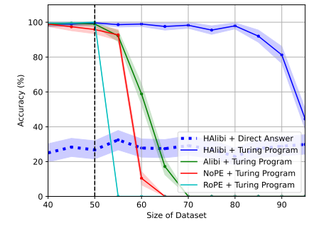

Figure: Length generalization when running SGD, varying the number of training examples.

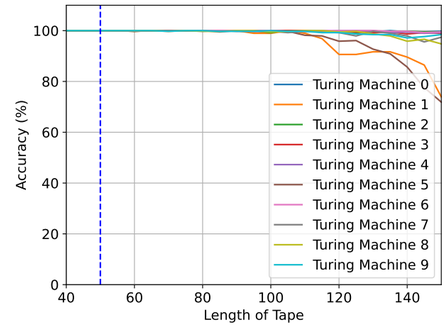

Figure: Length generalization performance on 10 different randomly generated Turing machines.

Why It Works

Intuition: In Turing Program, a task gets broken down to two subtasks: 1. modifying the tape content at a single position and 2. copying the tape content. The modification only requires the token at the head position and the positions where the Turing machine state is located, so it is length-independent. Copying is length dependent, but Jelassi et al. already showed that Hard-ALiBi could achieve length-generalization for it.

Theorem: Let T be a Turing Machine s.t. 1) T does not generate repeated n-grams and 2) T operates in-memory. Then, there exists a RASP program P of size (number of lines) $O(n)$ s.t. for every input x without repeated n-grams, P correctly simulates T for $\exp(n)$ steps.