First-After-Last Attention Head Improves OOD Algorithmic Generalization

Santiago Akle Serrano
NVIDIA
sakle@nvidia.com

Boris Ginsburg
NVIDIA
bginsburg@nvidia.com

Abstract

Transformers can fit to algorithmic tasks but often fail under simple distribution shifts, reflecting inappropriate inductive biases. We propose the differentiable First-After-Last (FAL) head, an extension to attention that biases models to implement reliable solutions for two synthetic benchmarks: Flip-Flop Language (FFL) and Cycle Tracing (CT). Unlike regular rotary Transformers, adding a FAL head yields perfect accuracy on FFL sequences far longer than those seen in training, and accurate extrapolation to much longer CT cycles. These results show that small architectural modifications can improve algorithmic generalization in Transformers.

1 Introduction

Transformer models are often used in settings that implicitly assume some capacity for algorithmic execution, such as maintaining memory or logical flow control. Yet despite their ability to fit training distributions for such tasks, training does not elicit robust algorithms.

Prior work shows that rather than learning the intended algorithm, Transformers converge to mechanisms that break when used out-of-distribution [7, 10][5, Section5.2]. We illustrate this with two synthetic benchmarks. Flip-Flop Language (FFL) [4] requires simulating a one-bit register by writing and later retrieving state. Cycle Tracing (CT) requires following edges in a cycle graph, testing the ability to retrieve the successor of a given key. Standard Transformers with rotary positional embeddings (RoPE) fit both training tasks but fail under simple distribution shifts, such as longer sequences or larger cycles.

A range of architectures that show evidence of algorithmic generalization and generalization to longer sequences have been proposed: Neural Turing Machine [2] and the Neural GPU [11]. More substantial modifications of the Transformer itself have also been explored: The Universal Transformer [1], the Compressive Transformer [9], hybrid state space models like S4 [3], and SWAN [8]. In contrast, we introduce a much smaller intervention: a single specialized attention head biased toward the "find last, return next" pattern required by both FFL and CT.

We introduce the First-After-Last (FAL) head, which retrieves the value following the most recent key positively associated with the query. Adding a FAL head to a Transformer suffices to induce robust generalization: perfect accuracy on FFL across sequences far longer than seen in training, and accurate extrapolation to much longer CT cycles. These results highlight the value of targeted inductive biases for eliciting robust algorithmic behavior in Transformers.

2 Task and Method

FFL [4] simulates the storage and retrieval of a single bit. Each input is a sequence of instruction–value pairs: write (w), read (r), or ignore (i), with values 0 or 1. A read should output the most recently

written bit. For example, `w1w0i0r0` writes 1, overwrites it with 0, ignores one token, and then reads 0. Instructions are sampled independently with fixed probabilities.

CT simulates simple graph traversal. The input lists edges between 3-digit node identifiers and specifies a start node. For example, `E 984,456 345,984 456,345 C 345` encodes a cycle over nodes $\{984, 456, 345\}$ starting at 345. The target output is the traversal `984 456 345 984 456 ...`, repeated until the token budget is exhausted. Task difficulty scales with cycle length. Models are scored by transition accuracy: the proportion of predicted nodes matching the ground truth, conditioned on a perfect generation so far.

Both tasks require querying positions derived from the current token and retrieving information that immediately follows. In FFL, a read token must locate the most recent write (or read) and return the subsequent bit. In CT, the current node must locate its outgoing edge and return the next node. Standard Transformers approximate these mechanisms but fail to do so robustly under distribution shift.

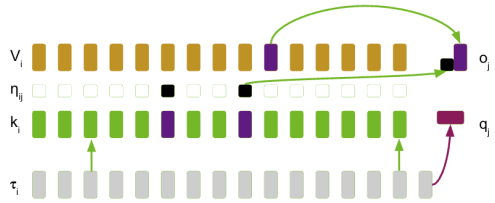

Figure 1: A First-After-Last (FAL) head selects the latest position where $q^\top k_\ell > 0$ and returns $\left(q_i^\top k_\ell\right) v_{\ell+1}$.

The FAL attention head is designed to perform this operation more directly (Figure 1 and listing 1). For query q_i, the head selects the most recent key k_ℓ with $q_i^\top k_\ell > 0$ and outputs the next value $v_{\ell+1}$ weighted by $\left(q_i^\top k_\ell\right)$. Adding FAL heads suffices to induce better generalization: In FFL, it achieves perfect accuracy on longer and differently sampled sequences, and in CT, it enables extrapolation to cycles several times longer than those seen in training (Table 1).

By focusing only on the latest positive association rather than all previous ones as in standard attention, the model is biased towards a more robust implementations of the solution. Mechanistic inspection further confirms that the head behaves as intended: in FFL, read tokens attend to the latest write or read and correctly return the most recent bit (Appendix B).

Table 1: Transformer with FAL performs perfectly OOD on Flip-Flop Language (FFL) and works on $8\times$ longer sequences in Cycle Tracing (CT). $F(p_{\text{ignore}})$ are sequences where ignore instructions are sampled with p_{ignore} probability.

	FFL			CT		
Model	In-dist. $F(0.6)$	$F(0.98)$	Len 1024 ($F(0.6)$)	Train cycle	$4\times$ length	$8\times$ length
GPT + RoPE	1.00	0.151	0.50	0.9999	0.178	0.04
FAL-GPT + RoPE	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.995**

3 Conclusion

This work revisits the problem of generalization in algorithmic tasks using two simple algorithmic tasks. Despite achieving perfect in-distribution accuracy, standard Transformers fail under simple distribution shifts. Our differentiable FAL head enables robust out-of-distribution generalization for both tasks, highlighting the value of explicit architectural biases in guiding models toward reliable algorithmic behavior.

References

[1] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers, 2019.

[2] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.

[3] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state-space layers, 2021.

[4] Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Exposing Attention Glitches with Flip-Flop Language Modeling, October 2023. arXiv:2306.00946 [cs].

[5] Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata, 2023.

[6] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[7] Catherine Olsson, Shan Zhang, Ben Chan, Nelson Litwin, Tony Yu, Paul Christiano, and David Bowman. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. `https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html`.

[8] Krishna C. Puvvada, Faisal Ladhak, Santiago Akle Serrano, Cheng-Ping Hsieh, Shantanu Acharya, Somshubra Majumdar, Fei Jia, Samuel Kriman, Simeng Sun, Dima Rekesh, and Boris Ginsburg. Swan-gpt: An efficient and scalable approach for long-context language modeling, 2025.

[9] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019.

[10] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks, 2023.

[11] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms, 2016.

A Model specifics

All models are based on the GPT-NeoX implementation from huggingface: Each layer uses layernorm and parallel residual paths $x = x + \mathrm{attn}(\mathrm{ln1}(x)) + \mathrm{mlp}(\mathrm{ln2}(x))$. MLPs fan out to $3\times$ the hidden dimension size and use `gelu` nonlinearities. Positional encoding is handled by RoPE and, input and output embeddings are distinct.

A.1 Flip-Flop Language

Our base model contains 6 layers with 64 hidden dimensions, 4 heads with 16 dimensions per head. All head dimensions are rotated by RoPe, and the rotations are set for a maximum sequence length of 2048 tokens.

Models are trained with adamW [6] with linearly decaying learning rate with a maximum of 10^{-4} achieved after 100 warm-up steps. The rest of the parameters are set to $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$, and no weight decay is used.

Following [4] we denote $F(p_{\text{ignore}})$ the distribution of FFL sequences sampled by selecting each instruction iid with probabilities: $p_{\text{ignore}}, p_{\text{read}} = p_{\text{write}} = \frac{1-p_{\text{ignore}}}{2}$. We train Transformer models with RoPE on sequences of 512 instructions sampled from $F(0.6)$. We train for 12,000 steps with batch sizes of 128, and sequences of 512 instructions (1024 tokens).

A.2 Cycle Tracing

Our base model contains 12 layers with 768 hidden dimensions. The best base OOD precision is achieved with a single head with 768 dimensions. Therefore for our base model we use a single (768) dimensional head for each layer. For the FAL model we split this head into two: a regular attention head with 768-64 dimensions and a single FAL head with 64 dimensions.

Models are trained with adamW [6] with linearly decaying learning rate with a maximum of 5×10^{-5} achieved after 1000 warm-up steps. The rest of the parameters are set to $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$, and no weight decay is used.

To generate training graphs we sample node counts uniformly in the range $[10, 50]$, select node identifiers by sampling distinct integers in the range $[0, 999]$ and zero padding them to three digits, and we sample an initial node uniformly from the set. We shuffle the edges with a random permutation before generating the input.

B Mechanistic inspection of trained FFL models

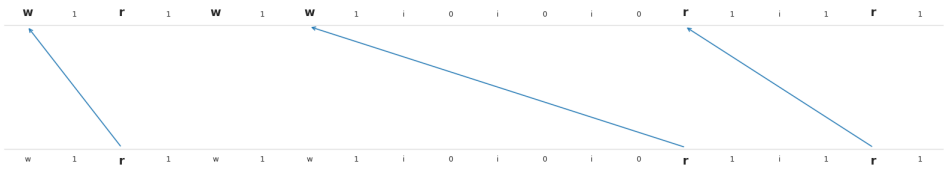

Figure 2: Attention pattern First-After-Last head on 3rd layer of model

We inspect the locations to which the First-After-Last head attends to, and observe that in fact they behave as expected. Figure 2 indicates the location of the last positive attention (top) that precedes the present token (bottom). As expected we see that the read tokens attend to the locations where the information is last contained. Either the last write operation, or the last read operation.

C The first-after-last head

Algorithm 1 First After Last Head

Require: Last-token index ℓ; KV-cache keys $\{k_1, \ldots, k_{\ell-1}\}$ and values $\{v_1, \ldots, v_{\ell-1}\}$
Require: Projection matrices W_K, W_V, W_Q, and output projection W_O

1: Compute projected vectors:

$$k_\ell \leftarrow W_K x_\ell, \qquad v_\ell \leftarrow W_V x_\ell, \qquad q_\ell \leftarrow W_Q x_\ell$$

2: Compute attention scores for all previous keys:

$$s_i \leftarrow \left(k_i^\top q_\ell\right) \quad \text{for } i = 1, \ldots, \ell - 1$$

3: Find most recent positive match:

$$m \leftarrow \max\{\, i < \ell \;:\; s_i > 0 \,\}$$

4: **if** $m \neq \emptyset$ **then**
5: **return** $s_m \cdot v_{m+1}$;
6: **else**
7: **return** 0;
8: **end if**