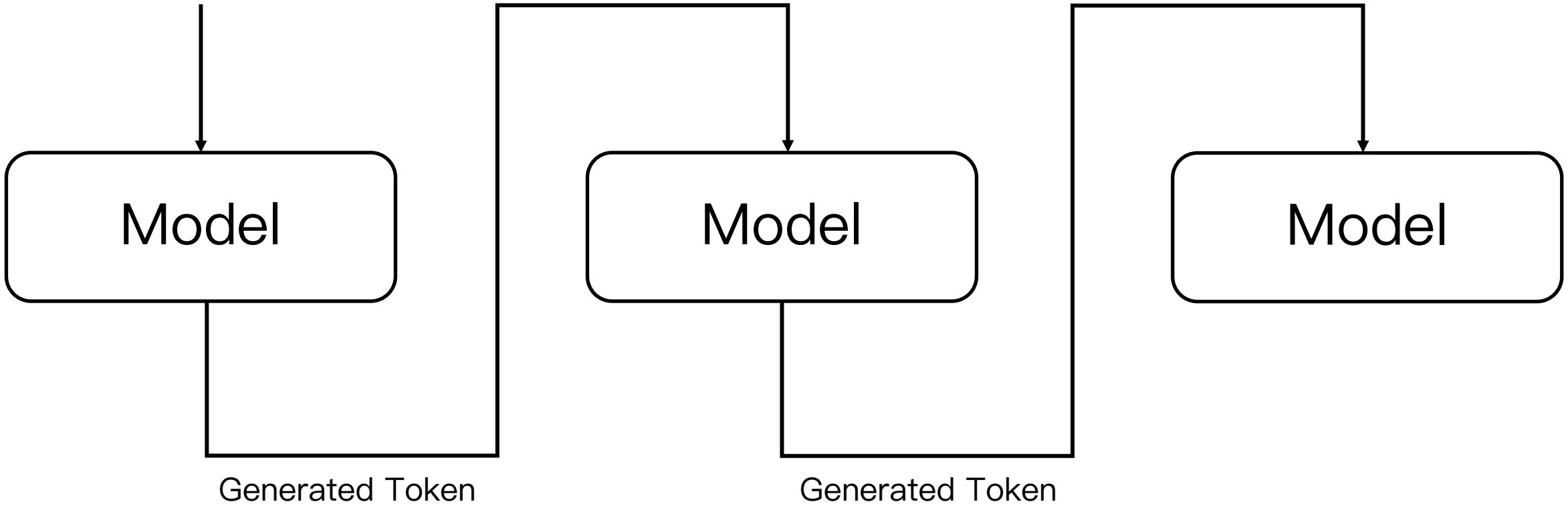
Scaling Latent Reasoning via Looped Language Models

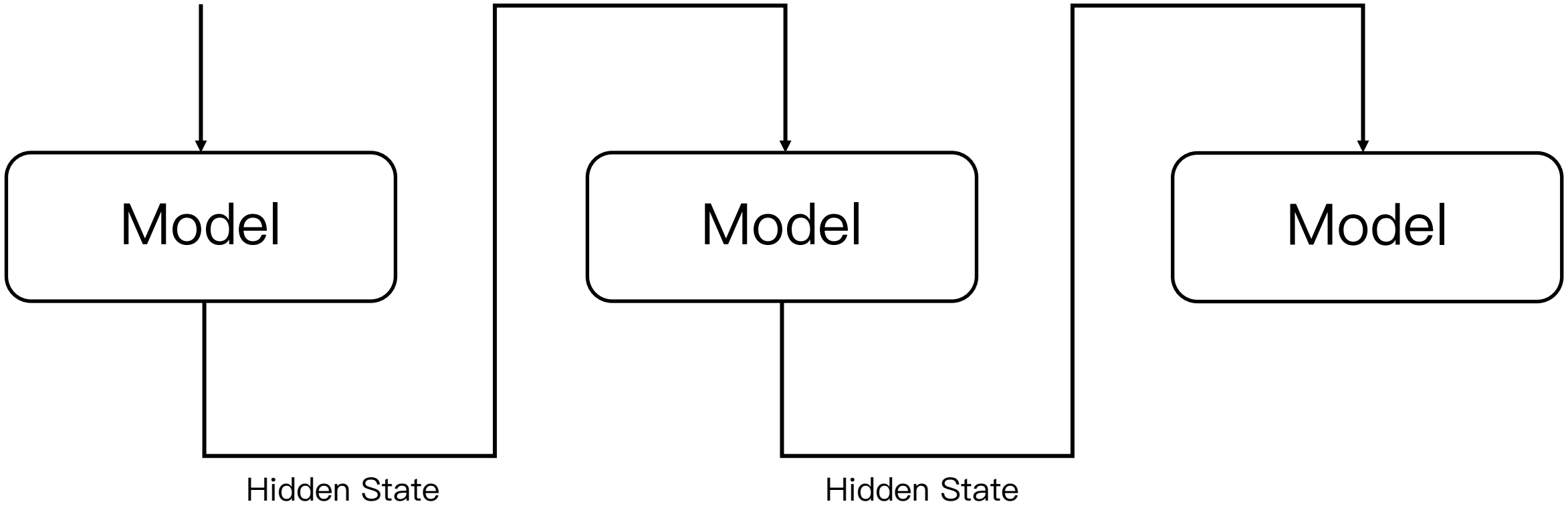
Rui-Jie Zhu

For ASAP Seminar

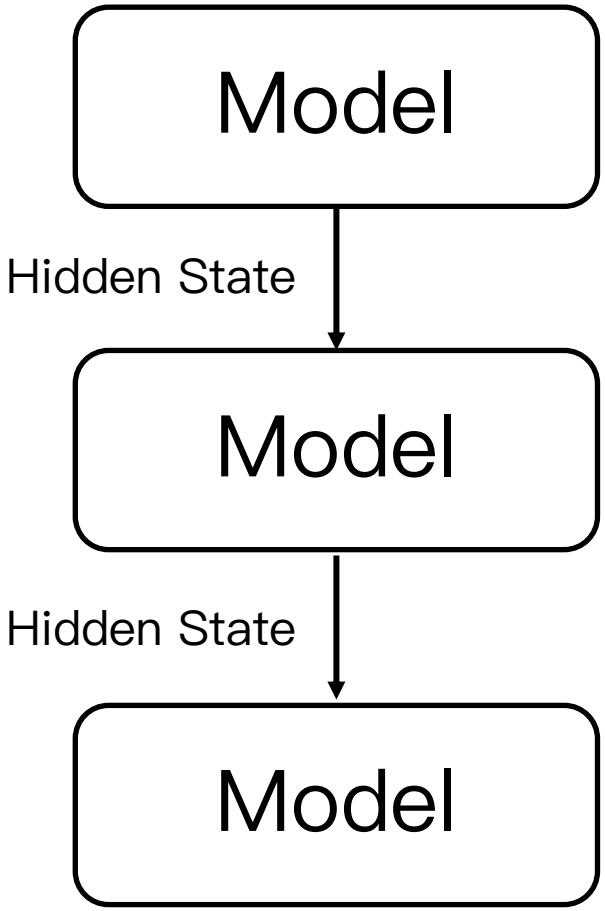
Chain-of-Thought



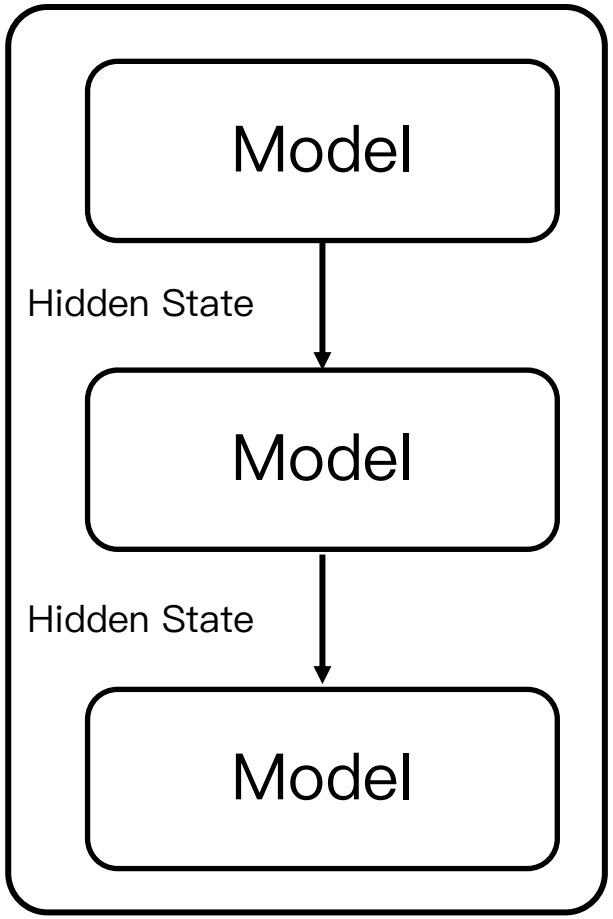
Latent Chain-of-Thought



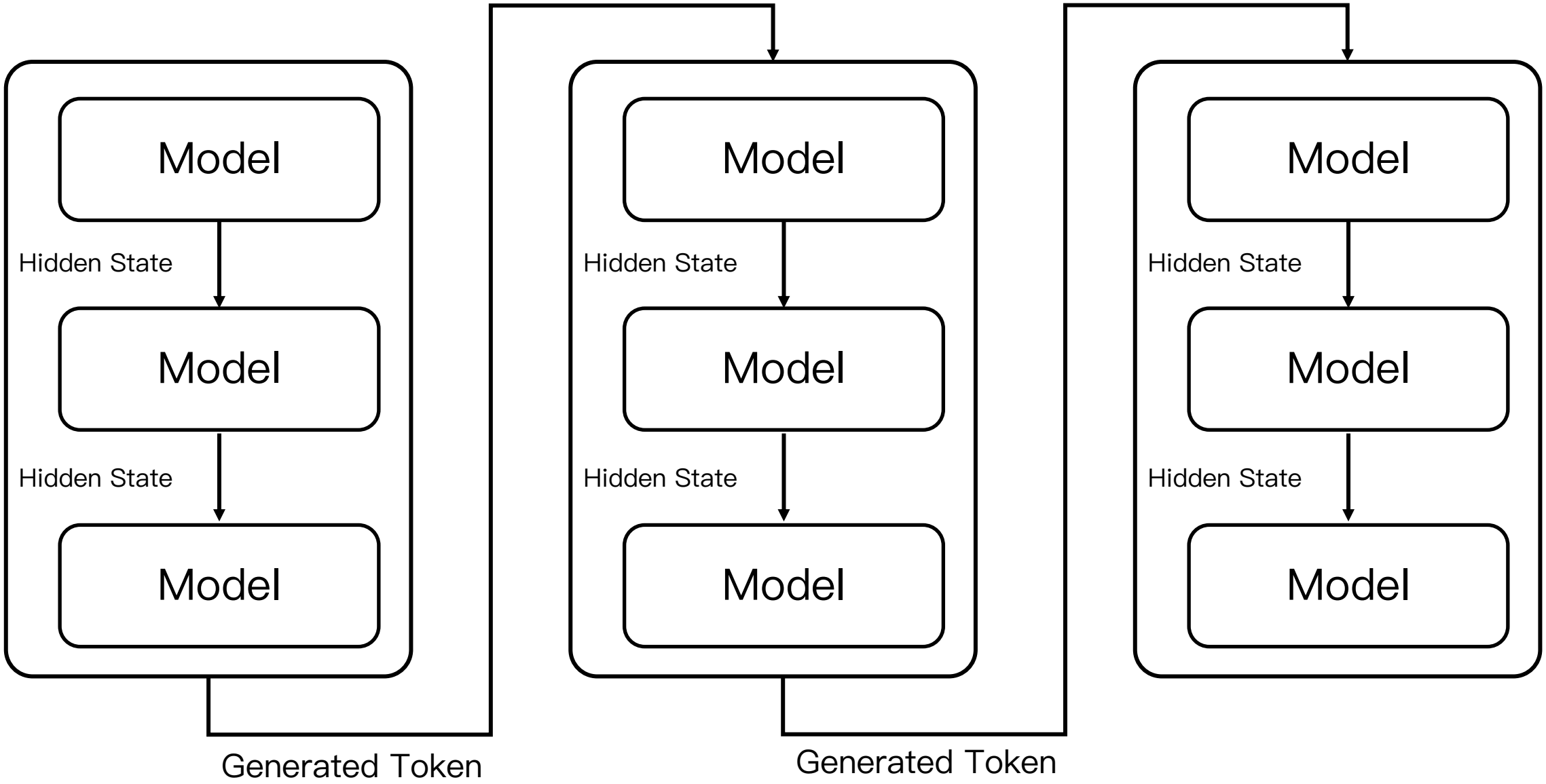
Latent Chain-of-Thought



Latent Chain-of-Thought



Loop Transformer



The origin of the Looped Transformer: Universal Transformers (2018)

Published as a conference paper at ICLR 2019

UNIVERSAL TRANSFORMERS

Mostafa Dehghani[*†]
University of Amsterdam
dehghani@uva.nl

Stephan Gouws[*]
DeepMind
sgouws@google.com

Oriol Vinyals
DeepMind
vinyals@google.com

Jakob Uszkoreit
Google Brain
usz@google.com

Łukasz Kaiser
Google Brain
lukaszkaiser@google.com

ABSTRACT

Recurrent neural networks (RNNs) sequentially process data by updating their state with each new data point, and have long been the de facto choice for sequence modeling tasks. However, their inherently sequential computation makes them slow to train. Feed-forward and convolutional architectures have recently been shown to achieve superior results on some sequence modeling tasks such as machine translation, with the added advantage that they concurrently process all inputs in the sequence, leading to easy parallelization and faster training times. Despite these successes, however, popular feed-forward sequence models like the Transformer fail to generalize in many simple tasks that recurrent models handle with ease, e.g. copying strings or even simple logical inference when the string or formula lengths exceed those observed at training time. We propose the Universal Transformer (UT), a parallel-in-time self-attentive recurrent sequence model which can be cast as a generalization of the Transformer model and which addresses these issues. UTs combine the parallelizability and global receptive field of feed-forward sequence models like the Transformer with the recurrent inductive bias of RNNs. We also add a dynamic per-position halting mechanism and find that it improves accuracy on several tasks. In contrast to the standard Transformer, under certain assumptions UTs can be shown to be Turing-complete. Our experiments show that UTs outperform standard Transformers on a wide range of algorithmic and language understanding tasks, including the challenging LAMBADA language modeling task where UTs achieve a new state of the art, and machine translation where UTs achieve a 0.9 BLEU improvement over Transformers on the WMT14 En-De dataset.

1 INTRODUCTION

Convolutional and fully-attentional feed-forward architectures like the Transformer have recently emerged as viable alternatives to recurrent neural networks (RNNs) for a range of sequence modeling tasks, notably machine translation (Gehring et al., 2017; Vaswani et al., 2017). These parallel-in-time architectures address a significant shortcoming of RNNs, namely their inherently sequential computation which prevents parallelization across elements of the input sequence, whilst still addressing the vanishing gradients problem as the sequence length gets longer (Hochreiter et al., 2003). The Transformer model in particular relies entirely on a self-attention mechanism (Parikh et al., 2016; Lin et al., 2017) to compute a series of context-informed vector-space representations of the symbols in its input and output, which are then used to predict distributions over subsequent symbols as the model predicts the output sequence symbol-by-symbol. Not only is this mechanism straightforward to parallelize, but as each symbol's representation is also directly informed by all other symbols' representations, this results in an effectively global receptive field across the whole sequence. This stands in contrast to e.g. convolutional architectures which typically only have a limited receptive field.

Notably, however, the Transformer with its fixed stack of distinct layers foregoes RNNs' inductive bias towards learning iterative or recursive transformations. Our experiments indicate that this inductive

[*] Equal contribution, alphabetically by last name.
[†] Work performed while at Google Brain.

The origin of the Looped Transformer: Universal Transformers (2018)

UNIVERSAL TRANSFORMERS

Mostafa Dehghani[*†]
University of Amsterdam
dehghani@uva.nl

Stephan Gouws[*]
DeepMind
sgouws@google.com

Oriol Vinyals
DeepMind
vinyals@google.com

Jakob Uszkoreit
Google Brain
usz@google.com

Łukasz Kaiser
Google Brain
lukaszkaiser@google.com

ABSTRACT

Recurrent neural networks (RNNs) sequentially process data by updating their state with each new data point, and have long been the de facto choice for sequence modeling tasks. However, their inherently sequential computation makes them slow to train. Feed-forward and convolutional architectures have recently been shown to achieve superior results on some sequence modeling tasks such as machine translation, with the added advantage that they concurrently process all inputs in the sequence, leading to easy parallelization and faster training times. Despite these successes, however, popular feed-forward sequence models like the Transformer fail to generalize in many simple tasks that recurrent models handle with ease, e.g. copying strings or even simple logical inference when the string or formula lengths exceed those observed at training time. We propose the Universal Transformer (UT), a parallel-in-time self-attentive recurrent sequence model which can be cast as a generalization of the Transformer model and which addresses these issues. UTs combine the parallelizability and global receptive field of feed-forward sequence models like the Transformer with the recurrent inductive bias of RNNs. We also add a dynamic per-position halting mechanism and find that it improves accuracy on several tasks. In contrast to the standard Transformer, under certain assumptions UTs can be shown to be Turing-complete. Our experiments show that UTs outperform standard Transformers on a wide range of algorithmic and language understanding tasks, including the challenging LAMBADA language modeling task where UTs achieve a new state of the art, and machine translation where UTs achieve a 0.9 BLEU improvement over Transformers on the WMT14 En-De dataset.

1 INTRODUCTION

Convolutional and fully-attentional feed-forward architectures like the Transformer have recently emerged as viable alternatives to recurrent neural networks (RNNs) for a range of sequence modeling tasks, notably machine translation (Gehring et al., 2017; Vaswani et al., 2017). These parallel-in-time architectures address a significant shortcoming of RNNs, namely their inherently sequential computation which prevents parallelization across elements of the input sequence, whilst still addressing the vanishing gradients problem as the sequence length gets longer (Hochreiter et al., 2003). The Transformer model in particular relies entirely on a self-attention mechanism (Parikh et al., 2016; Lin et al., 2017) to compute a series of context-informed vector-space representations of the symbols in its input and output, which are then used to predict distributions over subsequent symbols as the model predicts the output sequence symbol-by-symbol. Not only is this mechanism straightforward to parallelize, but as each symbol's representation is also directly informed by all other symbols' representations, this results in an effectively global receptive field across the whole sequence. This stands in contrast to e.g. convolutional architectures which typically only have a limited receptive field.

Notably, however, the Transformer with its fixed stack of distinct layers foregoes RNNs' inductive bias towards learning iterative or recursive transformations. Our experiments indicate that this inductive

[*] Equal contribution, alphabetically by last name.
[†] Work performed while at Google Brain.

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

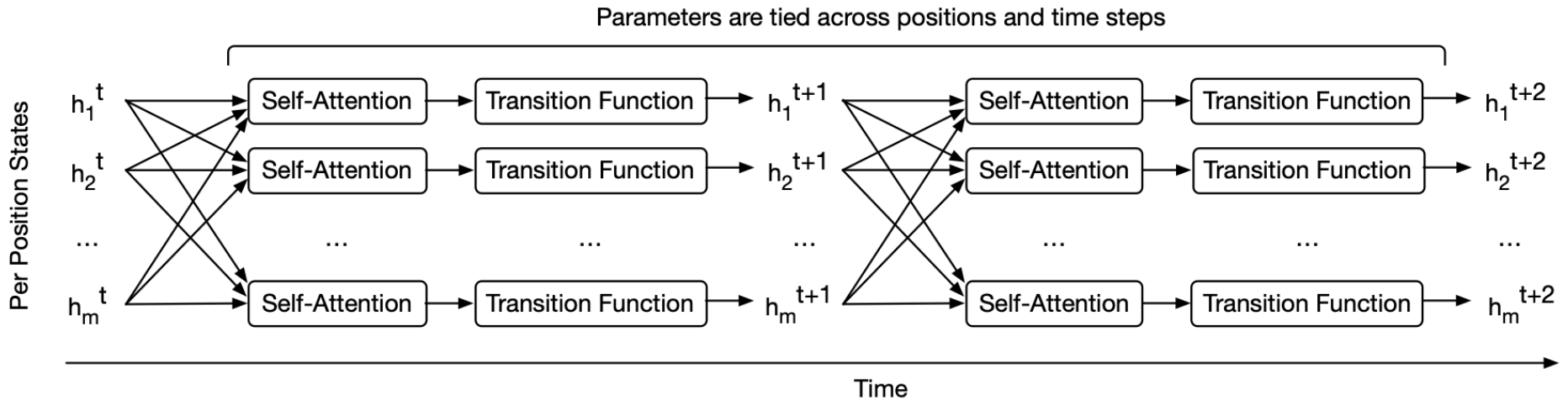
1 Introduction

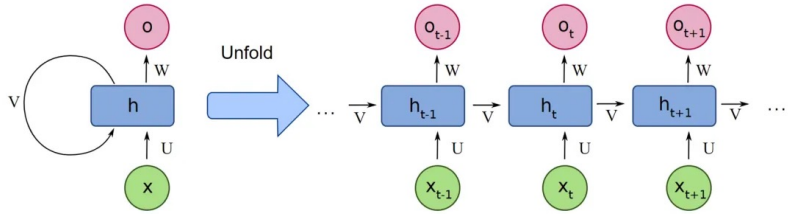
Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [31, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [34, 22, 14].

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [19] and conditional

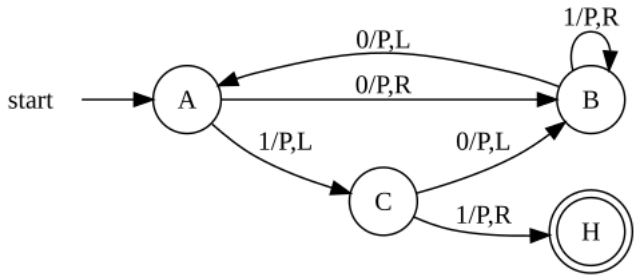
[*]Equal contribution. Listing order is random.
[†]Work performed while at Google Brain.

The origin of the Looped Transformer: Universal Transformers (2018)

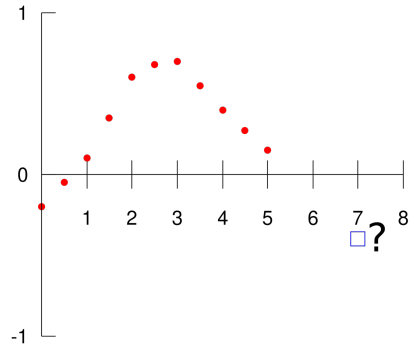




Adding Recurrent Inductive Bias into Transformers

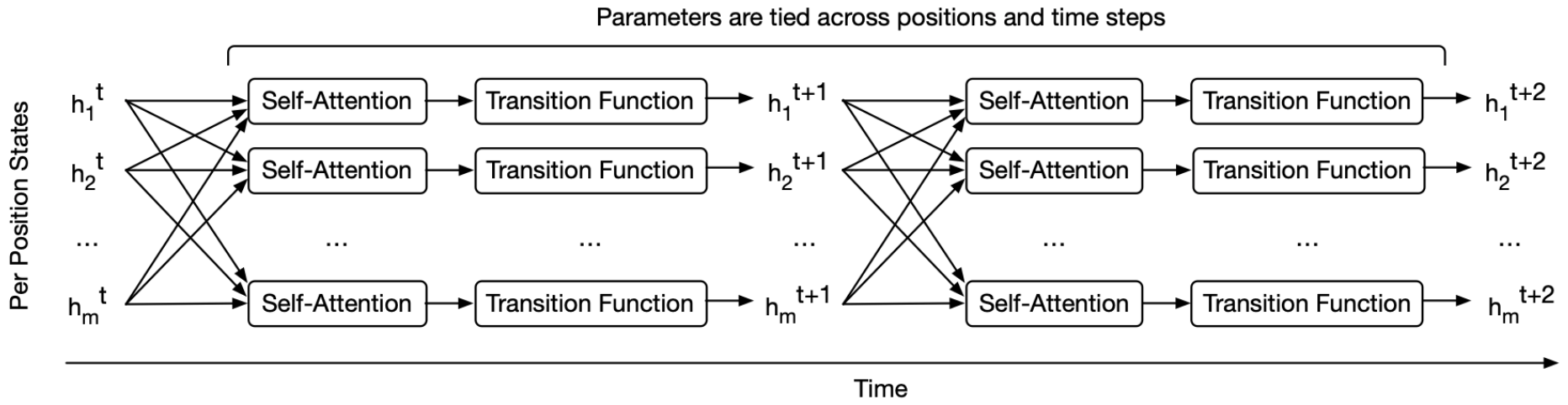


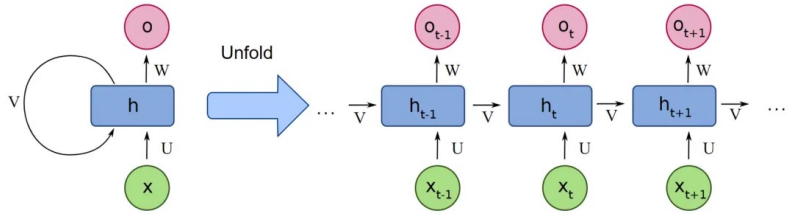
Making Transformer Turing–Complete



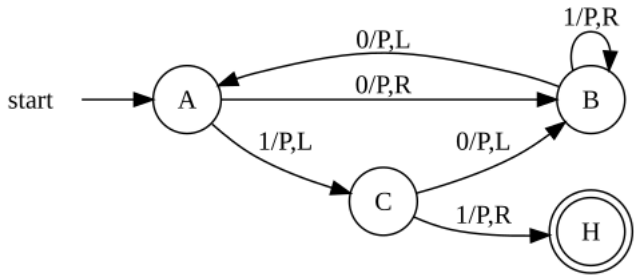
Improving Generalization of Transformers

Adaptive Computing in Universal Transformer

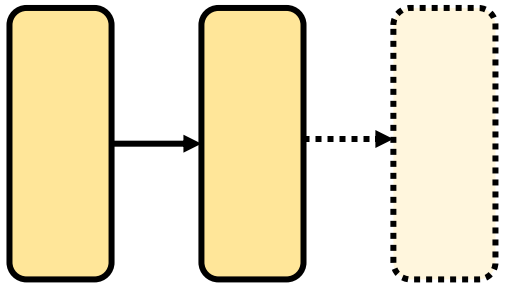




Adding Recurrent Inductive Bias into Transformers

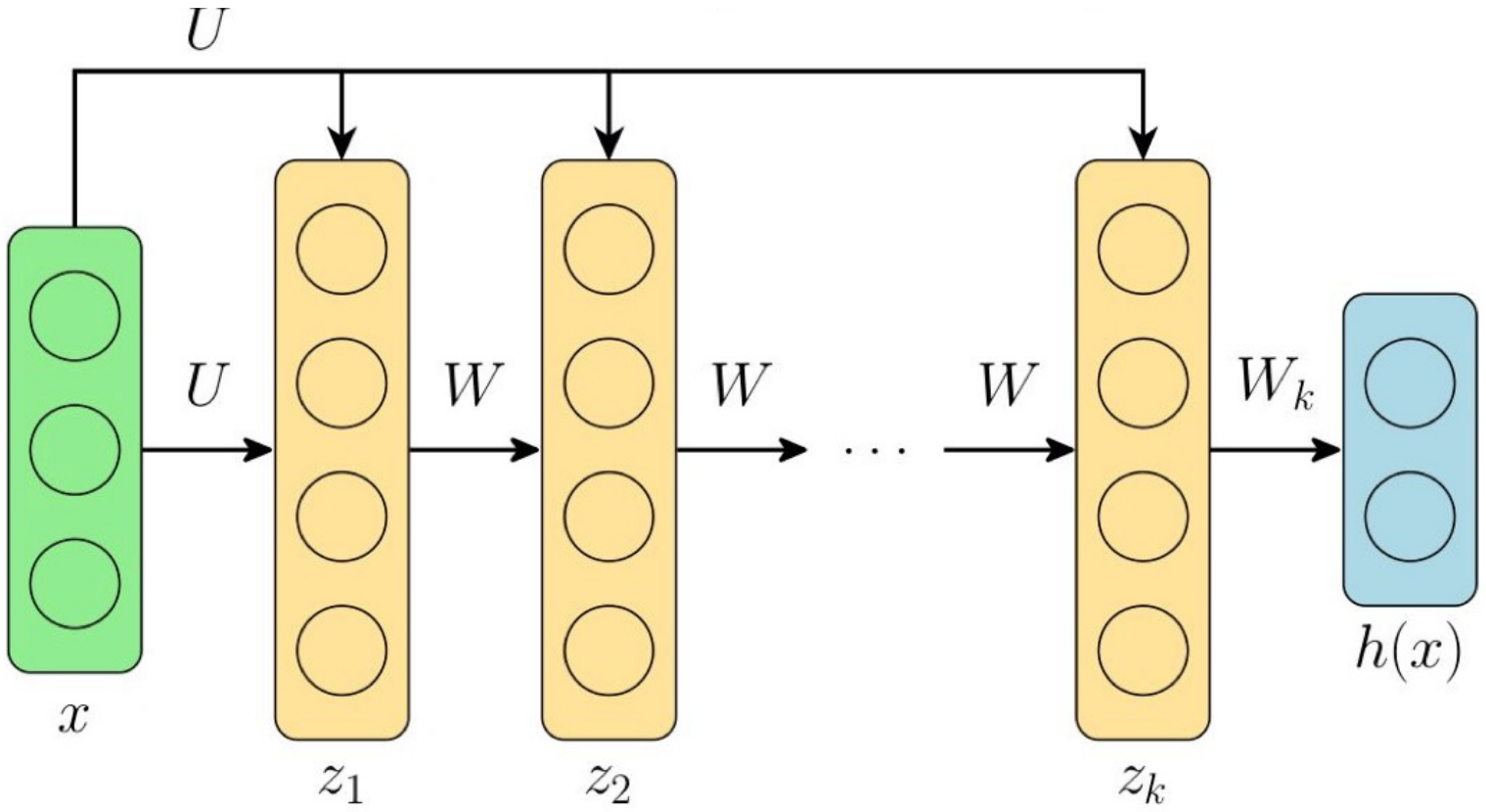


Making Transformer Turing–Complete

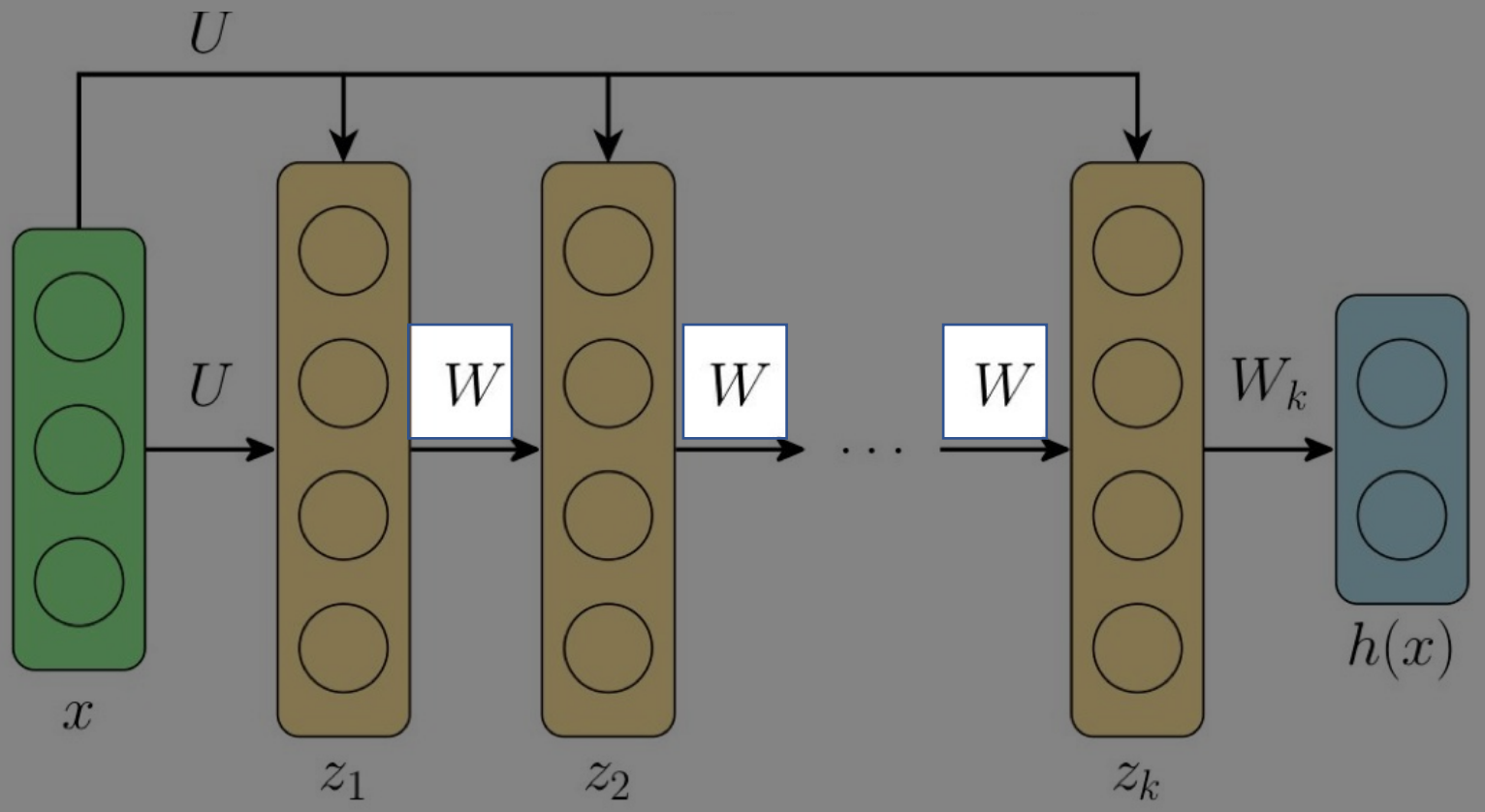


Adaptive Depth Inference

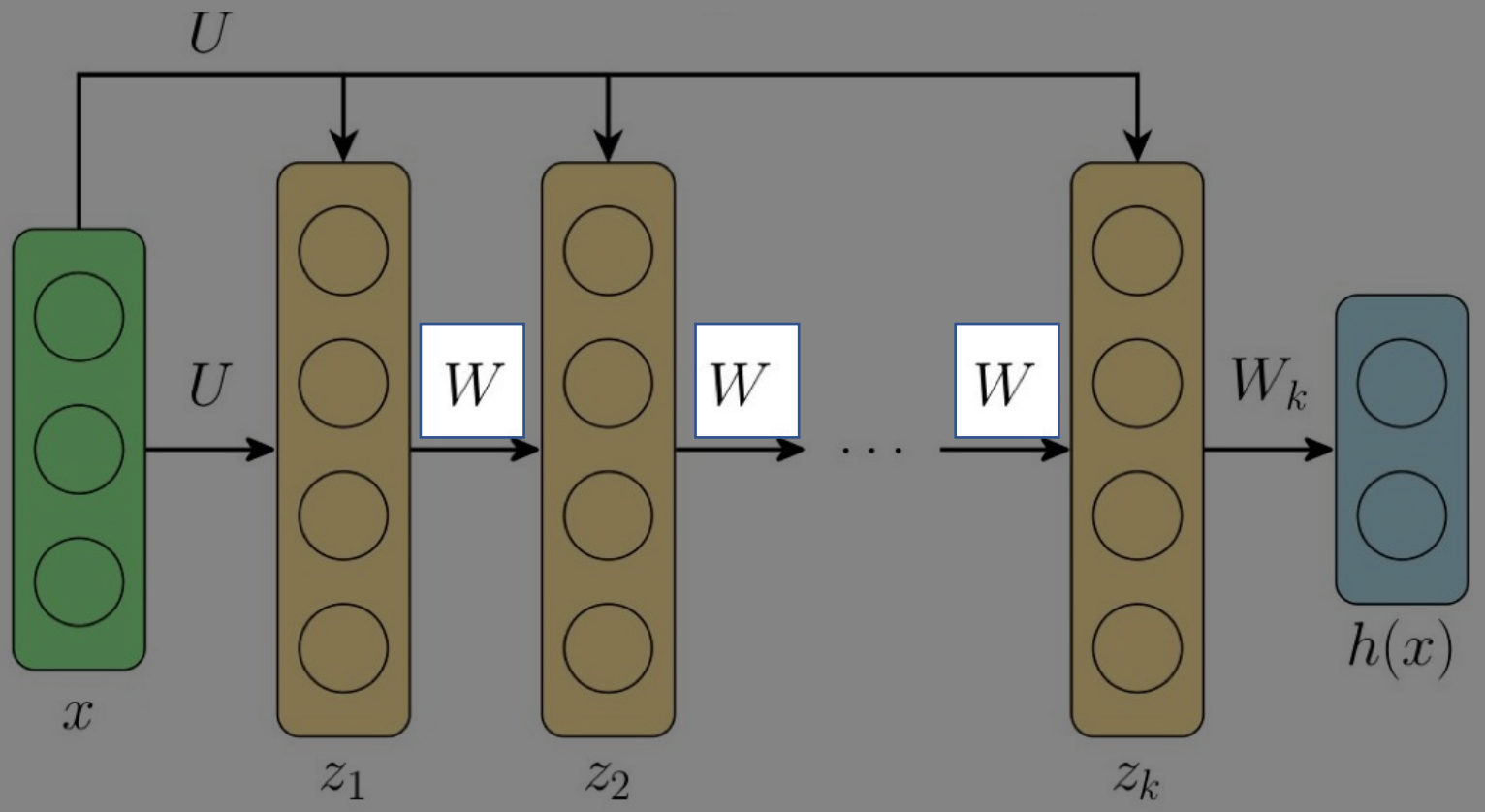
The origin of the Looped Transformer: Deep Equilibrium Models (2019)



The origin of the Looped Transformer: Deep Equilibrium Models (2019)

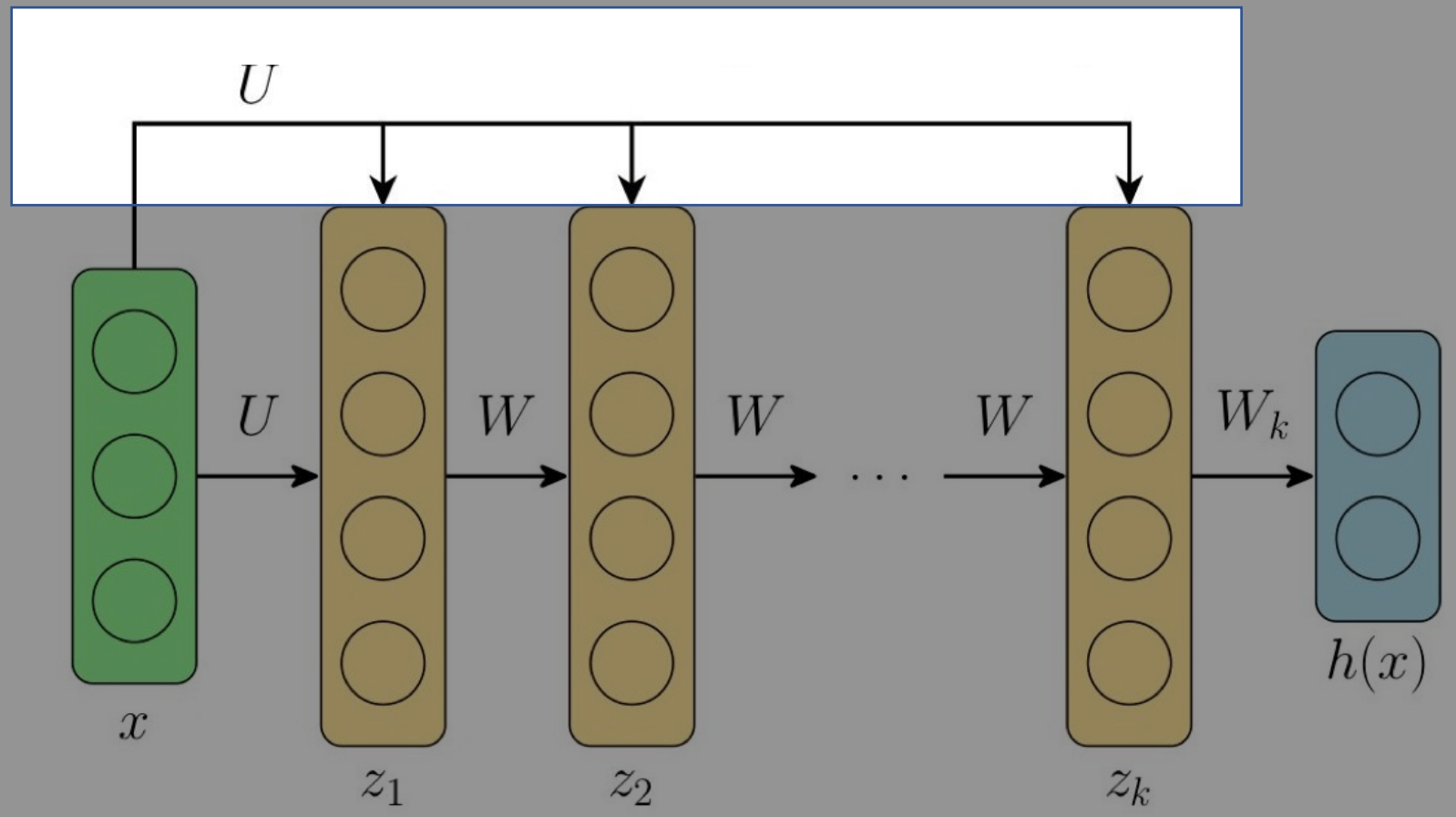


The origin of the Looped Transformer: Deep Equilibrium Models (2019)

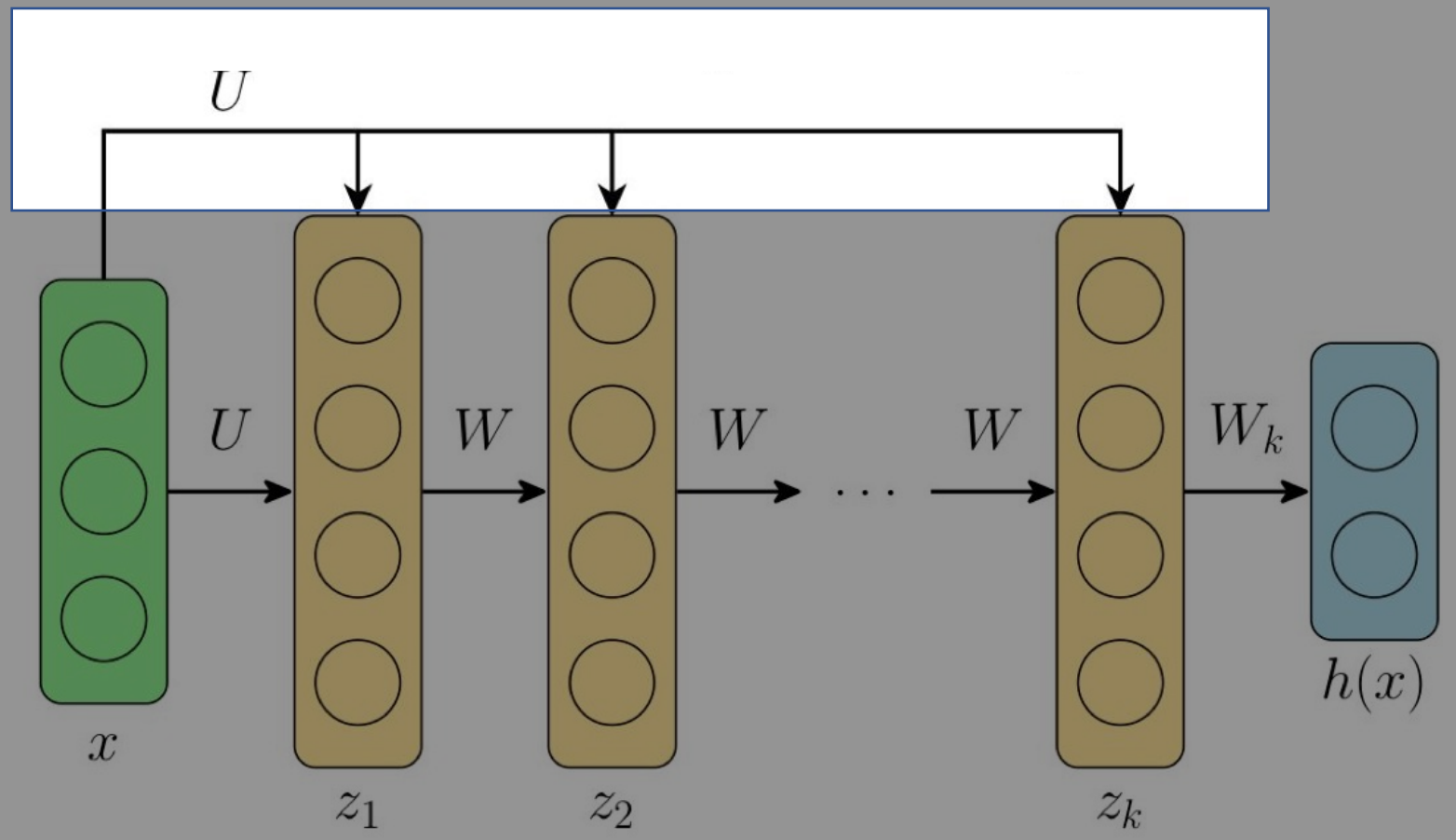


Tied–Weight Across Layers

The origin of the Looped Transformer: Deep Equilibrium Models (2019)

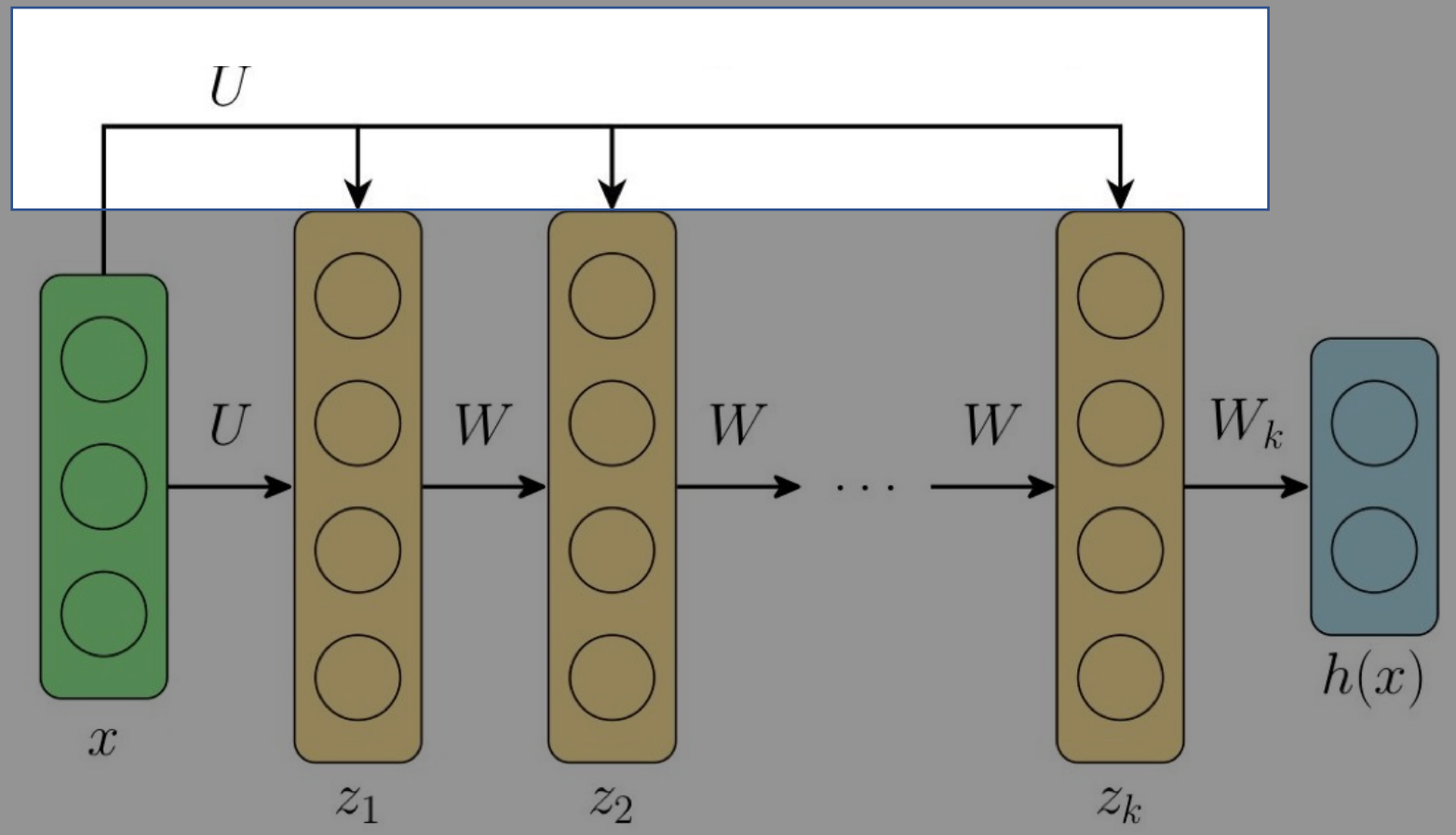


The origin of the Looped Transformer: Deep Equilibrium Models (2019)



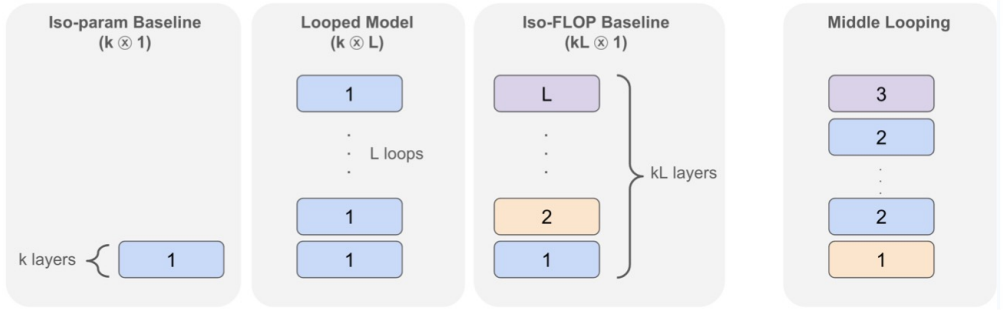
Input Injection in Each Loop

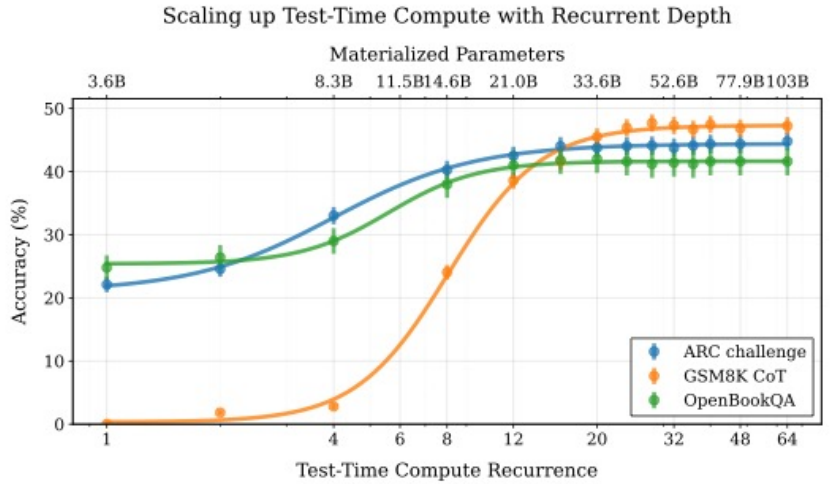
The origin of the Looped Transformer: Deep Equilibrium Models (2019)



Input Injection in Each Loop

Attempts in Language Modeling





Addition of n numbers					
	Params / FLOPs	$n=8$	$n=16$	$n=24$	$n=32$
Base $(12 \otimes 1)$	12x / 12x	100.0	100.0	100.0	100.0
1 layer model					
Base $(1 \otimes 1)$	1x / 1x	0.1	0.1	0.1	0.0
Loop $(1 \otimes 12)$	1x / 12x	99.9	100.0	99.9	99.6
2 layer model					
Base $(2 \otimes 1)$	2x / 2x	85.8	71.5	49.3	38.8
Loop $(2 \otimes 6)$	2x / 12x	100.0	99.8	99.7	99.5
3 layer model					
Base $(3 \otimes 1)$	3x / 3x	97.2	78.5	69.2	60.7
Loop $(3 \otimes 4)$	3x / 12x	100.0	99.1	97.0	96.6

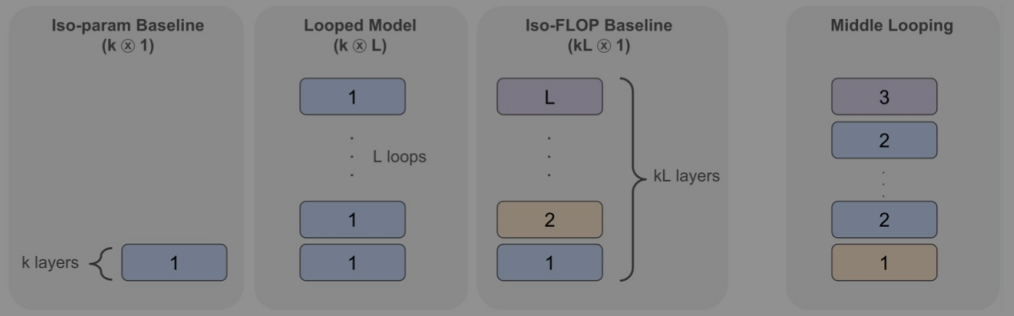
p-hop with n tokens			
	Params / FLOPs	$p=16$ $n=256$	$p=32$ $n=256$
Base $(6 \otimes 1)$	6x / 6x	**99.9**	**99.6**
1 layer model			
Base $(1 \otimes 1)$	1x / 1x	48.9	49.0
Loop $(1 \otimes 6)$	1x / 6x	**99.9**	**99.5**
2 layer model			
Base $(2 \otimes 1)$	2x / 2x	68.8	59.4
Loop $(2 \otimes 3)$	2x / 6x	**99.9**	**99.8**
3 layer model			
Base $(3 \otimes 1)$	3x / 3x	97.2	73.0
Loop $(3 \otimes 2)$	3x / 6x	**99.9**	**99.5**

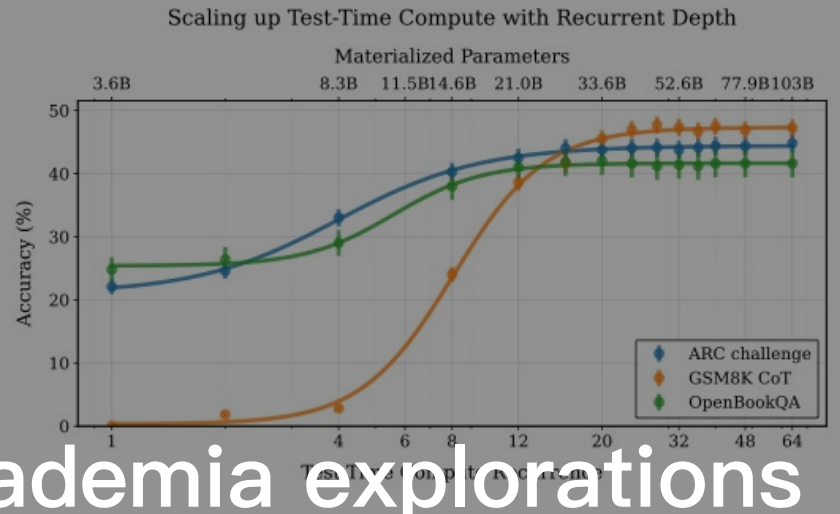
Table 1: Results on lm-eval-harness tasks zero-shot across various open-source models. We show ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021a), OpenBookQA (Mihaylov et al., 2018), PiQA (Bisk et al., 2020), SciQ (Johannes Welbl, 2017), and WinoGrande (Sakaguchi et al., 2021). We report normalized accuracy when provided.

Model	Param	Tokens	ARC-E	ARC-C	HellaSwag	MMLU	OBQA	PiQA	SciQ	WinoGrande
random			25.0	25.0	25.0	25.0	25.0	50.0	25.0	50.0
Amber	7B	1.2T	65.70	37.20	72.54	26.77	41.00	78.73	88.50	63.22
Pythia-2.8b	2.8B	0.3T	58.00	32.51	59.17	25.05	35.40	73.29	83.60	57.85
Pythia-6.9b	6.9B	0.3T	60.48	34.64	63.32	25.74	37.20	75.79	82.90	61.40
Pythia-12b	12B	0.3T	63.22	34.64	66.72	24.01	35.40	75.84	84.40	63.06
OLMo-1B	1B	3T	57.28	30.72	63.00	24.33	36.40	75.24	78.70	59.19
OLMo-7B	7B	2.5T	68.81	40.27	75.52	28.39	42.20	80.03	88.50	67.09
OLMo-7B-0424	7B	2.05T	75.13	45.05	77.24	47.46	41.60	80.09	96.00	68.19
OLMo-7B-0724	7B	2.75T	74.28	43.43	77.76	50.18	41.60	80.69	95.70	67.17
OLMo-2-1124	7B	4T	82.79	57.42	80.50	60.56	46.20	81.18	96.40	74.74
Ours, $(r=4)$	3.5B	0.8T	49.07	27.99	43.46	23.39	28.20	64.96	80.00	55.24
Ours, $(r=8)$	3.5B	0.8T	65.11	35.15	58.54	25.29	35.40	73.45	92.10	55.64
Ours, $(r=16)$	3.5B	0.8T	69.49	37.71	64.67	31.25	37.60	75.79	93.90	57.77
Ours, $(r=32)$	3.5B	0.8T	69.91	38.23	65.21	31.38	38.80	76.22	93.50	59.43

Showing advantage in reasoning–heavy tasks

Attempts in Language Modeling





But… Most of them are academia explorations

Addition of n numbers	Params / FLOPs	$n=8$	$n=16$	$n=24$	$n=32$
Base $(12 \otimes 1)$	12x / 12x	100.0	100.0	100.0	100.0
1 layer model					
Base $(1 \otimes 1)$	1x / 1x	0.1	0.1	0.1	0.0
Loop $(1 \otimes 12)$	1x / 12x	99.9	100.0	99.9	99.6
2 layer model					
Base $(2 \otimes 1)$	2x / 2x	85.8	71.5	49.3	38.8
Loop $(2 \otimes 6)$	2x / 12x	100.0	99.8	99.7	99.5
3 layer model					
Base $(3 \otimes 1)$	3x / 3x	97.2	78.5	69.2	60.7
Loop $(3 \otimes 4)$	3x / 12x	100.0	99.1	97.0	96.6

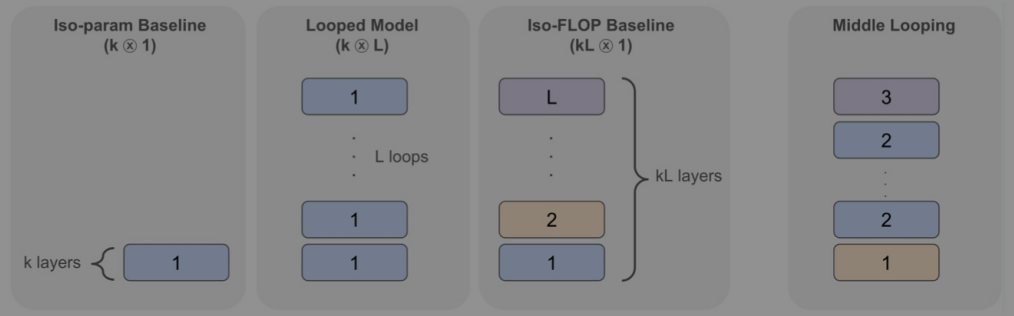
p-hop with n tokens	Params / FLOPs	$p=16$ $n=256$	$p=32$ $n=256$
Base $(6 \otimes 1)$	6x / 6x	**99.9**	**99.6**
1 layer model			
Base $(1 \otimes 1)$	1x / 1x	48.9	49.0
Loop $(1 \otimes 6)$	1x / 6x	**99.9**	**99.5**
2 layer model			
Base $(2 \otimes 1)$	2x / 2x	68.8	59.4
Loop $(2 \otimes 3)$	2x / 6x	**99.9**	**99.8**
3 layer model			
Base $(3 \otimes 1)$	3x / 3x	97.2	73.0
Loop $(3 \otimes 2)$	3x / 6x	**99.9**	**99.5**

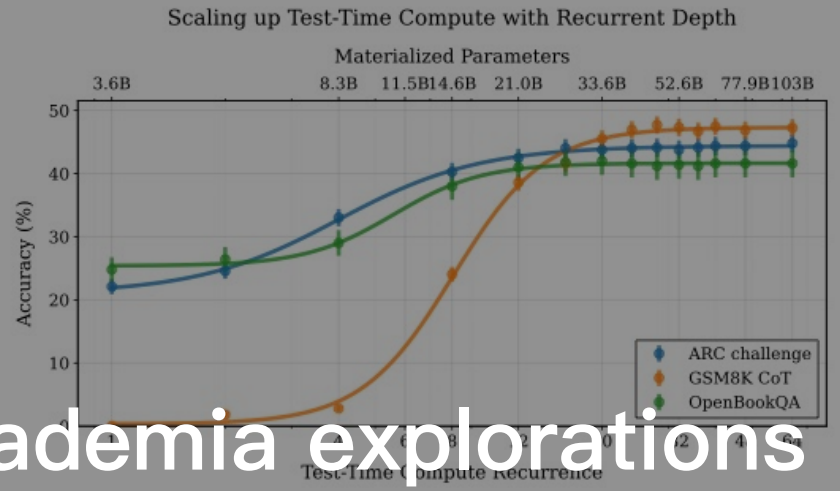
Table 1: Results on lm-eval-harness tasks zero-shot across various open-source models. We show ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021a), OpenBookQA (Mihaylov et al., 2018), PiQA (Bisk et al., 2020), SciQ (Johannes Welbl, 2017), and WinoGrande (Sakaguchi et al., 2021). We report normalized accuracy when provided.

Model	Param	Tokens	ARC-E	ARC-C	HellaSwag	MMLU	OBQA	PiQA	SciQ	WinoGrande
random			25.0	25.0	25.0	25.0	25.0	50.0	25.0	50.0
Amber	7B	1.2T	65.70	37.20	72.54	26.77	41.00	78.73	88.50	63.22
Pythia-2.8b	2.8B	0.3T	58.00	32.51	59.17	25.05	35.40	73.29	83.60	57.85
Pythia-6.9b	6.9B	0.3T	60.48	34.64	63.32	25.74	37.20	75.79	82.90	61.40
Pythia-12b	12B	0.3T	63.22	34.64	66.72	24.01	35.40	75.84	84.40	63.06
OLMo-1B	1B	3T	57.28	30.72	63.00	24.33	36.40	75.24	78.70	59.19
OLMo-7B	7B	2.5T	68.81	40.27	75.52	28.39	42.20	80.03	88.50	67.09
OLMo-7B-0424	7B	2.05T	75.13	45.05	77.24	47.46	41.60	80.09	96.00	68.19
OLMo-7B-0724	7B	2.75T	74.28	43.43	77.76	50.18	41.60	80.69	95.70	67.17
OLMo-2-1124	7B	4T	82.79	57.42	80.50	60.56	46.20	81.18	96.40	74.74
Ours, $(r=4)$	3.5B	0.8T	49.07	27.99	43.46	23.39	28.20	64.96	80.00	55.24
Ours, $(r=8)$	3.5B	0.8T	65.11	35.15	58.54	25.29	35.40	73.45	92.10	55.64
Ours, $(r=16)$	3.5B	0.8T	69.49	37.71	64.67	31.25	37.60	75.79	93.90	57.77
Ours, $(r=32)$	3.5B	0.8T	69.91	38.23	65.21	31.38	38.80	76.22	93.50	59.43

Showing advantage in reasoning–heavy tasks

Attempts in Language Modeling





But… Most of them are academia explorations

Common Question: How about scale it up?

	Params / FLOPs	$n=8$	$n=16$	$n=24$	$n=32$
Base $(12 \otimes 1)$	12x / 12x	100.0	100.0	100.0	100.0
1 layer model					
Base $(1 \otimes 1)$	1x / 1x	0.1	0.1	0.1	0.0
Loop $(1 \otimes 12)$	1x / 12x	99.9	100.0	99.9	99.6
2 layer model					
Base $(2 \otimes 1)$	2x / 2x	85.8	71.5	49.3	38.8
Loop $(2 \otimes 6)$	2x / 12x	100.0	99.8	99.7	99.5
3 layer model					
Base $(3 \otimes 1)$	3x / 3x	97.2	78.5	69.2	60.7
Loop $(3 \otimes 4)$	3x / 12x	100.0	99.1	97.0	96.6

	Params / FLOPs	$p=16$ $n=256$	$p=32$ $n=256$
Base $(6 \otimes 1)$	6x / 6x	**99.9**	**99.6**
1 layer model			
Base $(1 \otimes 1)$	1x / 1x	48.9	49.0
Loop $(1 \otimes 6)$	1x / 6x	**99.9**	**99.5**
2 layer model			
Base $(2 \otimes 1)$	2x / 2x	68.8	59.4
Loop $(2 \otimes 3)$	2x / 6x	**99.9**	**99.8**
3 layer model			
Base $(3 \otimes 1)$	3x / 3x	97.2	73.0
Loop $(3 \otimes 2)$	3x / 6x	**99.9**	**99.5**

Model	Param	Tokens	ARC-E	ARC-C	HellaSwag	MMLU	OBQA	PiQA	SciQ	WinoGrande
random			25.0	25.0	25.0	25.0	25.0	50.0	25.0	50.0
Amber	7B	1.2T	65.70	37.20	72.54	26.77	41.00	78.73	88.50	63.22
Pythia-2.8b	2.8B	0.3T	58.00	32.51	59.17	25.05	35.40	73.29	83.60	57.85
Pythia-6.9b	6.9B	0.3T	60.48	34.64	63.32	25.74	37.20	75.79	82.90	61.40
Pythia-12b	12B	0.3T	63.22	34.64	66.72	24.01	35.40	75.84	84.40	63.06
OLMo-1B	1B	3T	57.28	30.72	63.00	24.33	36.40	75.24	78.70	59.19
OLMo-7B	7B	2.5T	68.81	40.27	75.52	28.39	42.20	80.03	88.50	67.09
OLMo-7B-0424	7B	2.05T	75.13	45.05	77.24	47.46	41.60	80.09	96.00	68.19
OLMo-7B-0724	7B	2.75T	74.28	43.43	77.76	50.18	41.60	80.69	95.70	67.17
OLMo-2-1124	7B	4T	82.79	57.42	80.50	60.56	46.20	81.18	96.40	74.74
Ours, $(r=4)$	3.5B	0.8T	49.07	27.99	43.46	23.39	28.20	64.96	80.00	55.24
Ours, $(r=8)$	3.5B	0.8T	65.11	35.15	58.54	25.29	35.40	73.45	92.10	55.64
Ours, $(r=16)$	3.5B	0.8T	69.49	37.71	64.67	31.25	37.60	75.79	93.90	57.77
Ours, $(r=32)$	3.5B	0.8T	69.91	38.23	65.21	31.38	38.80	76.22	93.50	59.43

Showing advantage in reasoning–heavy tasks

The Challenge of Scaling up: Too many recurrent steps

Table 1: Results on lm-eval-harness tasks zero-shot across various open-source models. We show ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021a), OpenBookQA (Mihaylov et al., 2018), PiQA (Bisk et al., 2020), SciQ (Johannes Welbl, 2017), and WinoGrande (Sakaguchi et al., 2021). We report normalized accuracy when provided.

Model	Param	Tokens	ARC-E	ARC-C	HellaSwag	MMLU	OBQA	PiQA	SciQ	WinoGrande
random			25.0	25.0	25.0	25.0	25.0	50.0	25.0	50.0
Amber	7B	1.2T	65.70	37.20	72.54	26.77	41.00	78.73	88.50	63.22
Pythia-2.8b	2.8B	0.3T	58.00	32.51	59.17	25.05	35.40	73.29	83.60	57.85
Pythia-6.9b	6.9B	0.3T	60.48	34.64	63.32	25.74	37.20	75.79	82.90	61.40
Pythia-12b	12B	0.3T	63.22	34.64	66.72	24.01	35.40	75.84	84.40	63.06
OLMo-1B	1B	3T	57.28	30.72	63.00	24.33	36.40	75.24	78.70	59.19
OLMo-7B	7B	2.5T	68.81	40.27	75.52	28.39	42.20	80.03	88.50	67.09
OLMo-7B-0424	7B	2.05T	75.13	45.05	77.24	47.46	41.60	80.09	96.00	68.19
OLMo-7B-0724	7B	2.75T	74.28	43.43	77.76	50.18	41.60	80.69	95.70	67.17
OLMo-2-1124	7B	4T	82.79	57.42	80.50	60.56	46.20	81.18	96.40	74.74
Ours, ($r = 4$)	3.5B	0.8T	49.07	27.99	43.46	23.39	28.20	64.96	80.00	55.24
Ours, ($r = 8$)	3.5B	0.8T	65.11	35.15	58.54	25.29	35.40	73.45	92.10	55.64
Ours, ($r = 16$)	3.5B	0.8T	69.49	37.71	64.67	31.25	37.60	75.79	93.90	57.77
Ours, ($r = 32$)	3.5B	0.8T	69.91	38.23	65.21	31.38	38.80	76.22	93.50	59.43

Recurrent 32 times ≈ 100+B Params

The Challenge of Scaling up: Too many recurrent steps

Table 1: Results on lm-eval-harness tasks zero-shot across various open-source models. We show ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021a), OpenBookQA (Mihaylov et al., 2018), PiQA (Bisk et al., 2020), SciQ (Johannes Welbl, 2017), and WinoGrande (Sakaguchi et al., 2021). We report normalized accuracy when provided.

Model	Param	Tokens	ARC-E	ARC-C	HellaSwag	MMLU	OBQA	PiQA	SciQ	WinoGrande
random			25.0	25.0	25.0	25.0	25.0	50.0	25.0	50.0
Amber	7B	1.2T	65.70	37.20	72.54	26.77	41.00	78.73	88.50	63.22
Pythia-2.8b	2.8B	0.3T	58.00	32.51	59.17	25.05	35.40	73.29	83.60	57.85
Pythia-6.9b	6.9B	0.3T	60.48	34.64	63.32	25.74	37.20	75.79	82.90	61.40
Pythia-12b	12B	0.3T	63.22	34.64	66.72	24.01	35.40	75.84	84.40	63.06
OLMo-1B	1B	3T	57.28	30.72	63.00	24.33	36.40	75.24	78.70	59.19
OLMo-7B	7B	2.5T	68.81	40.27	75.52	28.39	42.20	80.03	88.50	67.09
OLMo-7B-0424	7B	2.05T	75.13	45.05	77.24	47.46	41.60	80.09	96.00	68.19
OLMo-7B-0724	7B	2.75T	74.28	43.43	77.76	50.18	41.60	80.69	95.70	67.17
OLMo-2-1124	7B	4T	82.79	57.42	80.50	60.56	46.20	81.18	96.40	74.74
Ours, ($r = 4$)	3.5B	0.8T	49.07	27.99	43.46	23.39	28.20	64.96	80.00	55.24
Ours, ($r = 8$)	3.5B	0.8T	65.11	35.15	58.54	25.29	35.40	73.45	92.10	55.64
Ours, ($r = 16$)	3.5B	0.8T	69.49	37.71	64.67	31.25	37.60	75.79	93.90	57.77
Ours, ($r = 32$)	3.5B	0.8T	69.91	38.23	65.21	31.38	38.80	76.22	93.50	59.43

Recurrent 32 times ≈ 100+B Params

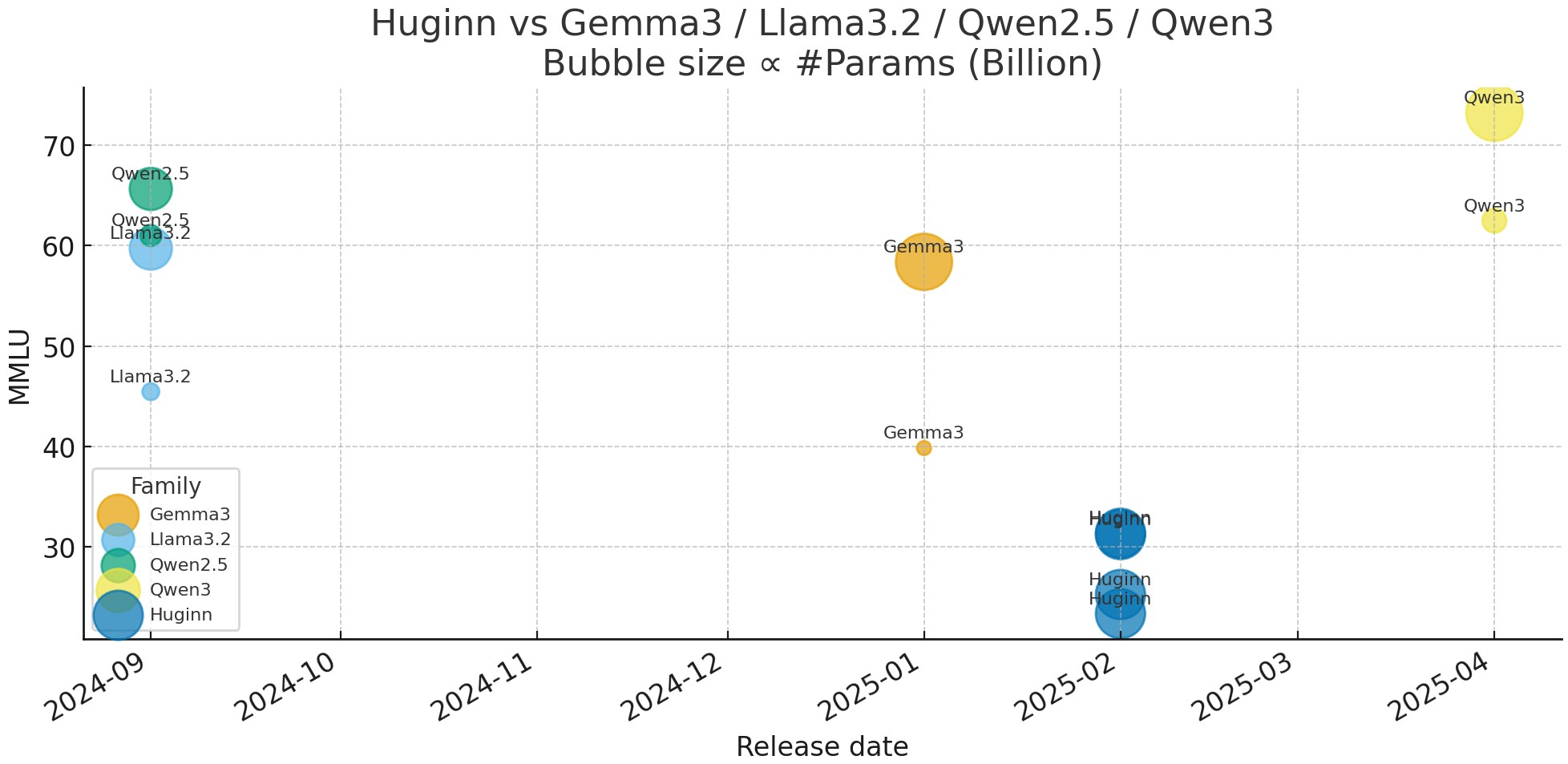
In our model, we set the maximum recurrent step to 4

The Challenge of Scaling up: Performance gap

Table 1: Results on lm-eval-harness tasks zero-shot across various open-source models. We show ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021a), OpenBookQA (Mihaylov et al., 2018), PiQA (Bisk et al., 2020), SciQ (Johannes Welbl, 2017), and WinoGrande (Sakaguchi et al., 2021). We report normalized accuracy when provided.

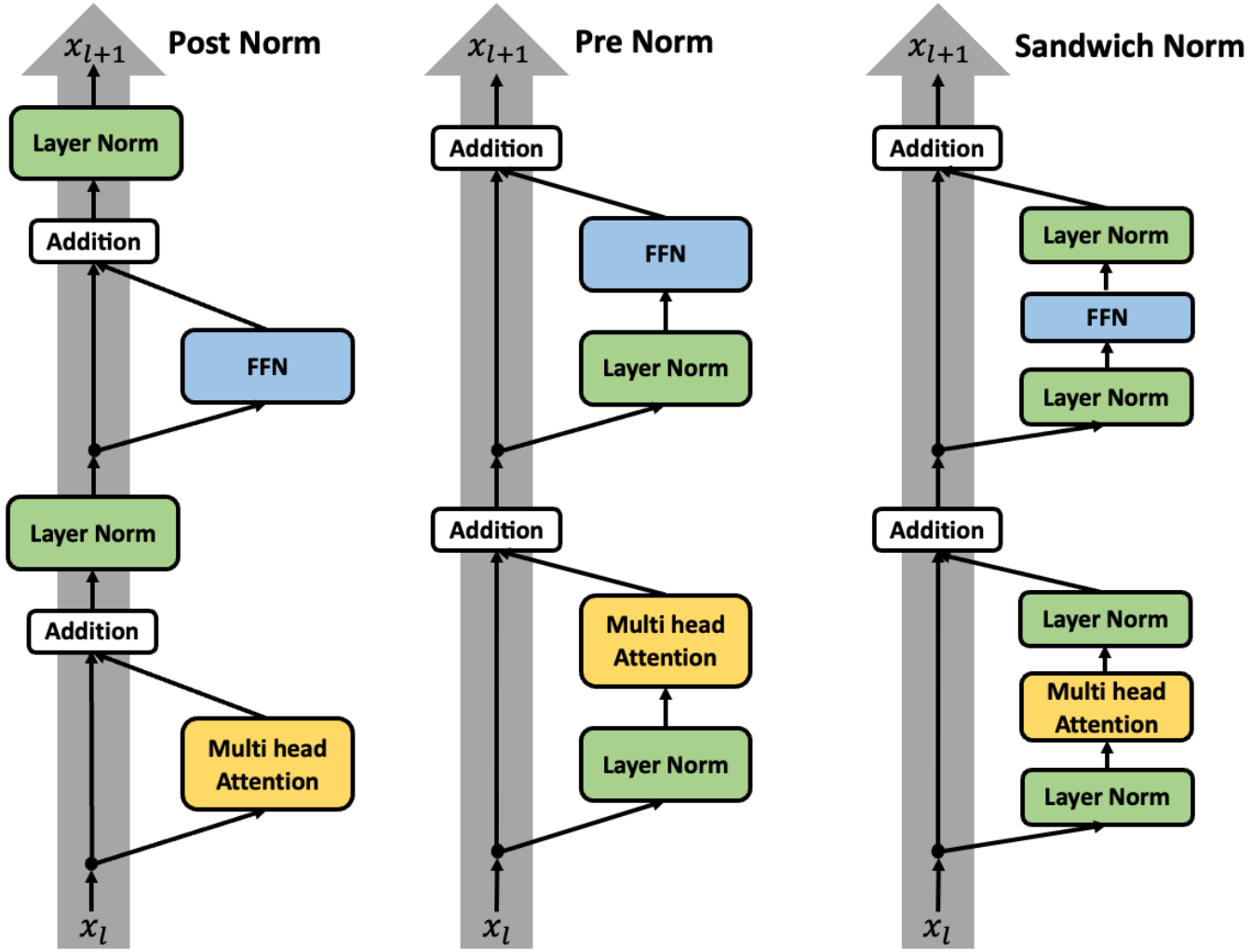
Model	Param	Tokens	ARC-E	ARC-C	HellaSwag	MMLU	OBQA	PiQA	SciQ	WinoGrande
random			25.0	25.0	25.0	25.0	25.0	50.0	25.0	50.0
Amber	7B	1.2T	65.70	37.20	72.54	26.77	41.00	78.73	88.50	63.22
Pythia-2.8b	2.8B	0.3T	58.00	32.51	59.17	25.05	35.40	73.29	83.60	57.85
Pythia-6.9b	6.9B	0.3T	60.48	34.64	63.32	25.74	37.20	75.79	82.90	61.40
Pythia-12b	12B	0.3T	63.22	34.64	66.72	24.01	35.40	75.84	84.40	63.06
OLMo-1B	1B	3T	57.28	30.72	63.00	24.33	36.40	75.24	78.70	59.19
OLMo-7B	7B	2.5T	68.81	40.27	75.52	28.39	42.20	80.03	88.50	67.09
OLMo-7B-0424	7B	2.05T	75.13	45.05	77.24	47.46	41.60	80.09	96.00	68.19
OLMo-7B-0724	7B	2.75T	74.28	43.43	77.76	50.18	41.60	80.69	95.70	67.17
OLMo-2-1124	7B	4T	82.79	57.42	80.50	60.56	46.20	81.18	96.40	74.74
Ours, ($r = 4$)	3.5B	0.8T	49.07	27.99	43.46	23.39	28.20	64.96	80.00	55.24
Ours, ($r = 8$)	3.5B	0.8T	65.11	35.15	58.54	25.29	35.40	73.45	92.10	55.64
Ours, ($r = 16$)	3.5B	0.8T	69.49	37.71	64.67	31.25	37.60	75.79	93.90	57.77
Ours, ($r = 32$)	3.5B	0.8T	69.91	38.23	65.21	31.38	38.80	76.22	93.50	59.43

The Challenge of Scaling up



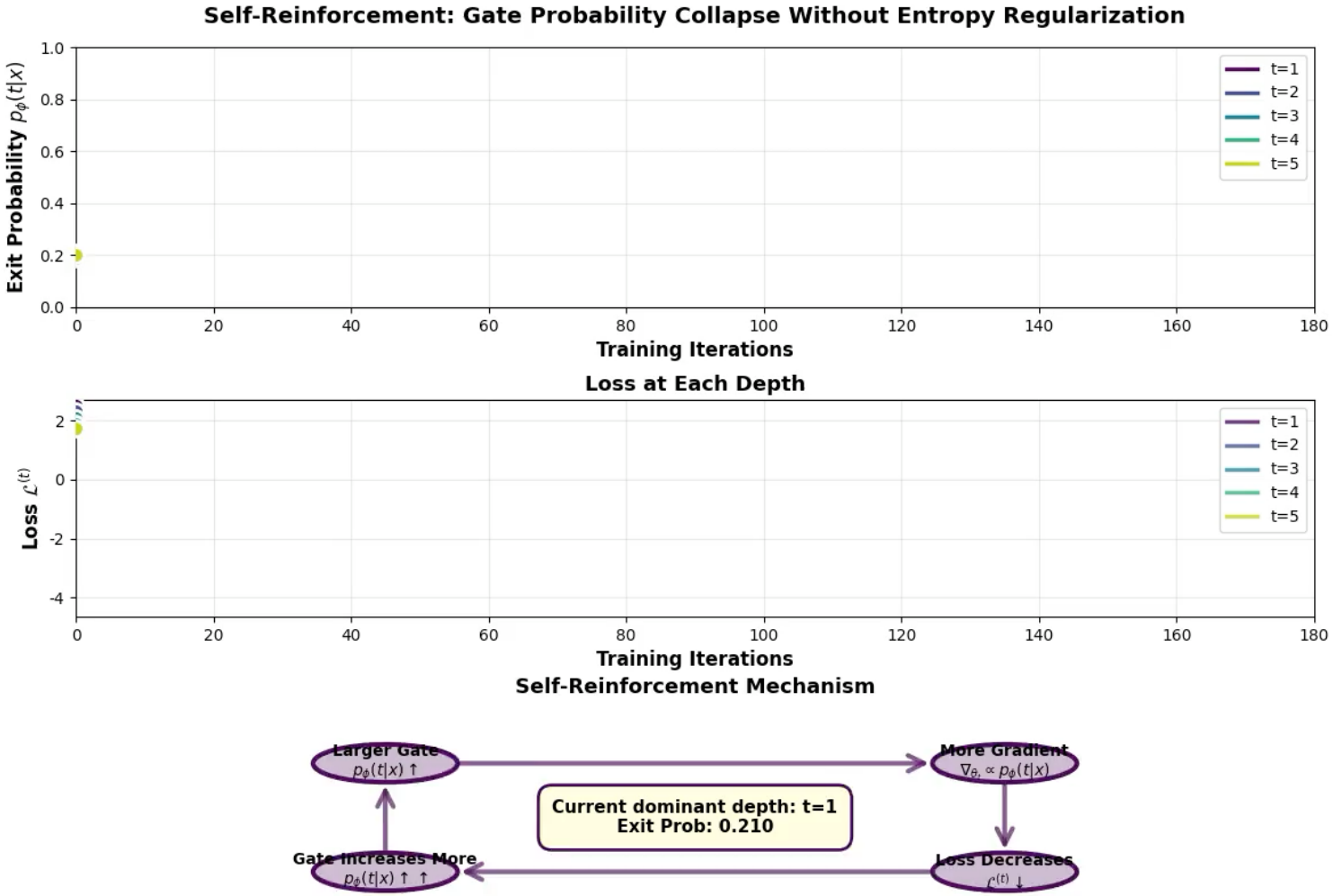
There is a noticeable performance gap between Loop (Huginn) and leading LLMs

The Challenge of Scaling up: Depth Problem

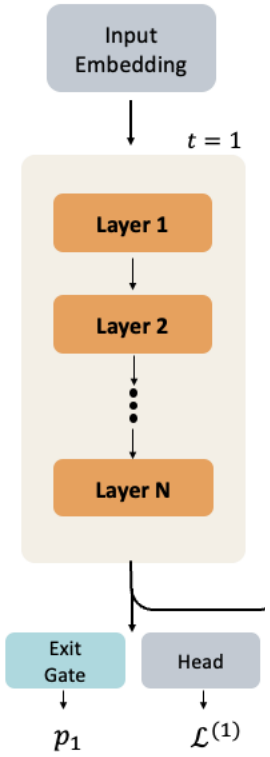


Sandwich Norm replace Pre-Norm

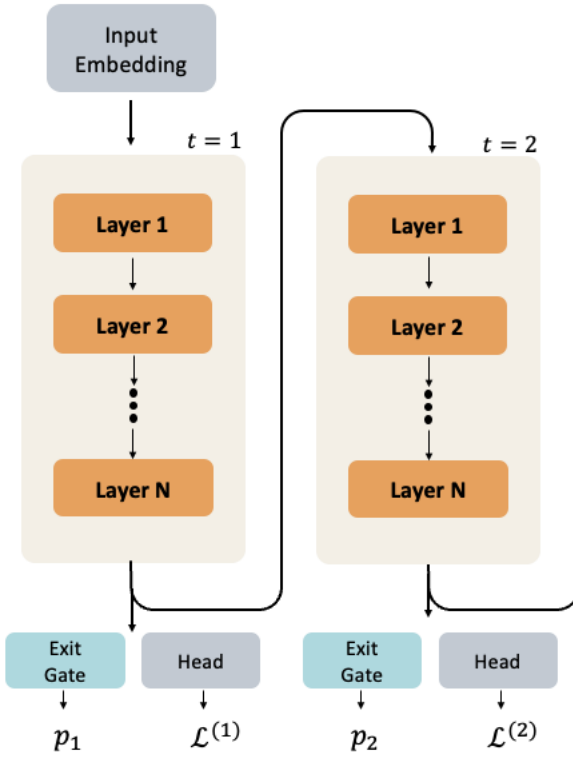
Solving Unstable Training: Early Exit Problem



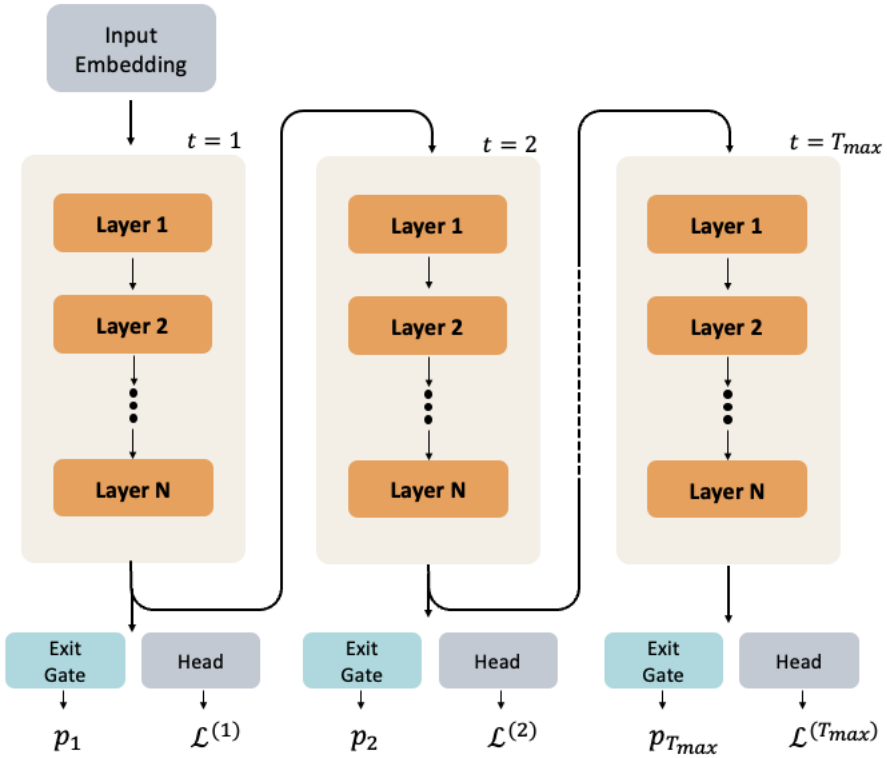
Pre-training with Loop, Inference with Early Exit



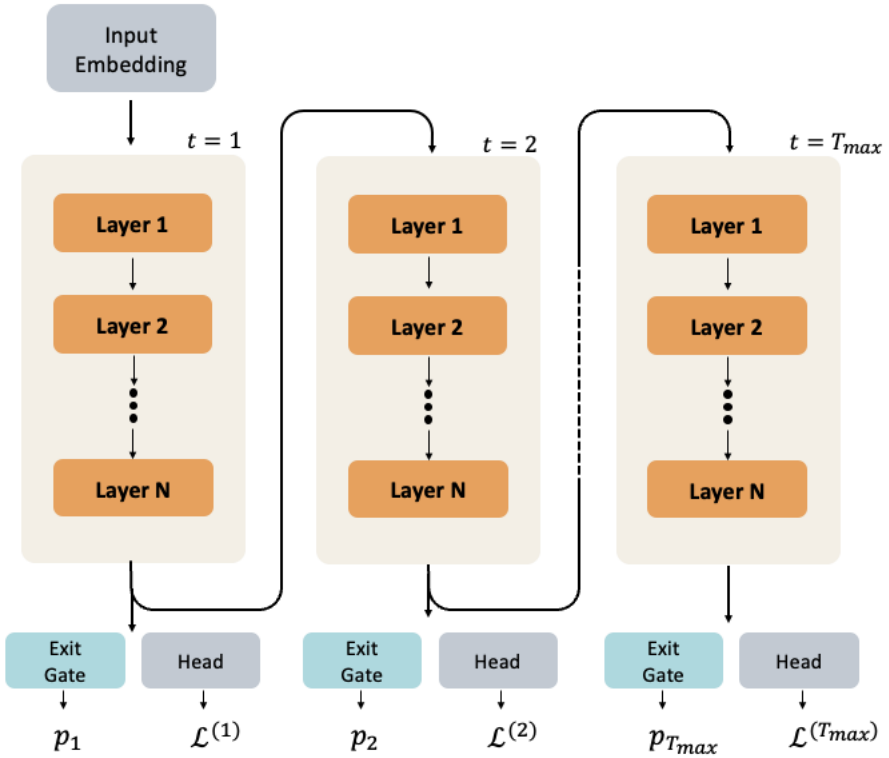
Pre-training with Loop, Inference with Early Exit



Pre-training with Loop, Inference with Early Exit



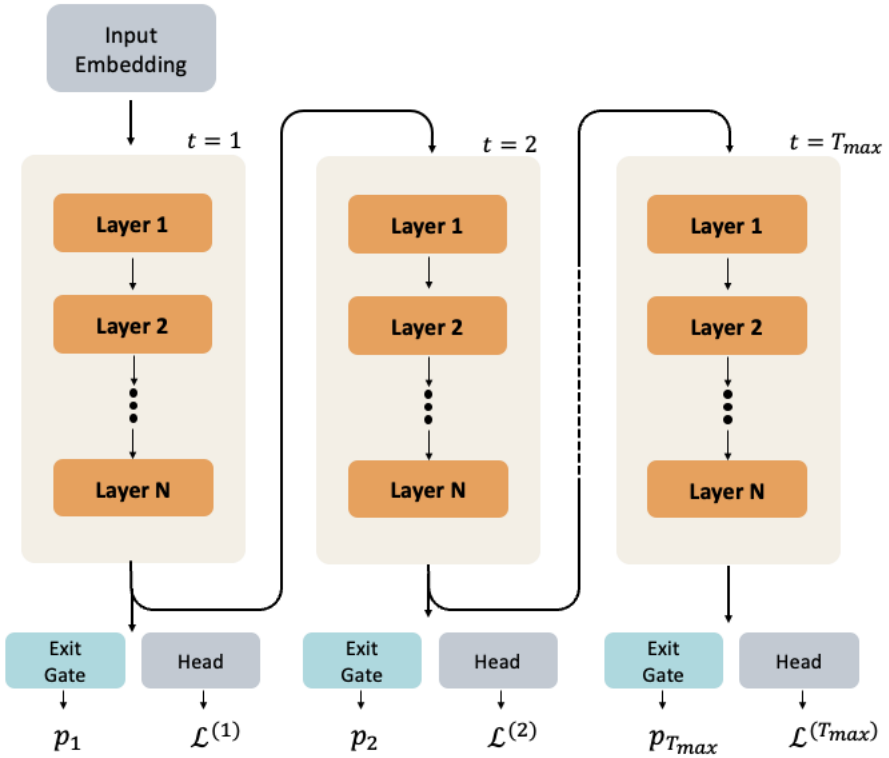
Pre–training with Loop, Inference with Early Exit



$$\mathcal{L} = \underbrace{\sum_{t=1}^{T_{\max}} p_\phi(t \mid x) \cdot \mathcal{L}^{(t)}}_{\text{expected task loss}}$$

- Training with full depth
- First compute the expectation loss of each early–exit step

Pre–training with Loop, Inference with Early Exit



$$\mathcal{L} = \underbrace{\sum_{t=1}^{T_{\max}} p_\phi(t \mid x) \cdot \mathcal{L}^{(t)}}_{\text{expected task loss}} - \underbrace{\beta \cdot H\big(p_\phi(\cdot \mid x)\big)}_{\text{entropy regularization}}$$

- Training with full depth
- First compute the expectation loss of each early–exit step
- Adding a entropy regularization as prior!

The Prior of Depth

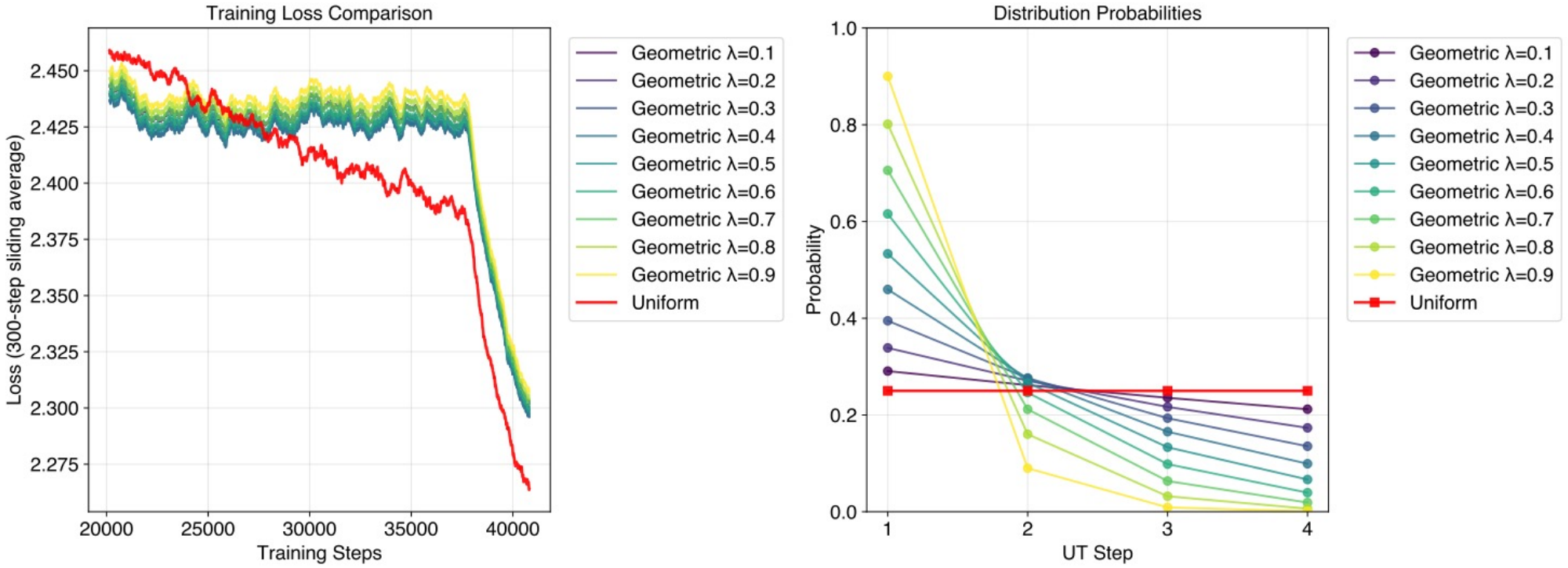
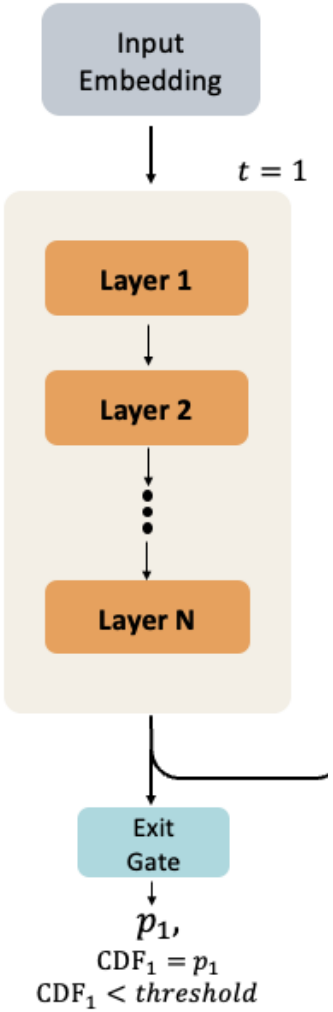
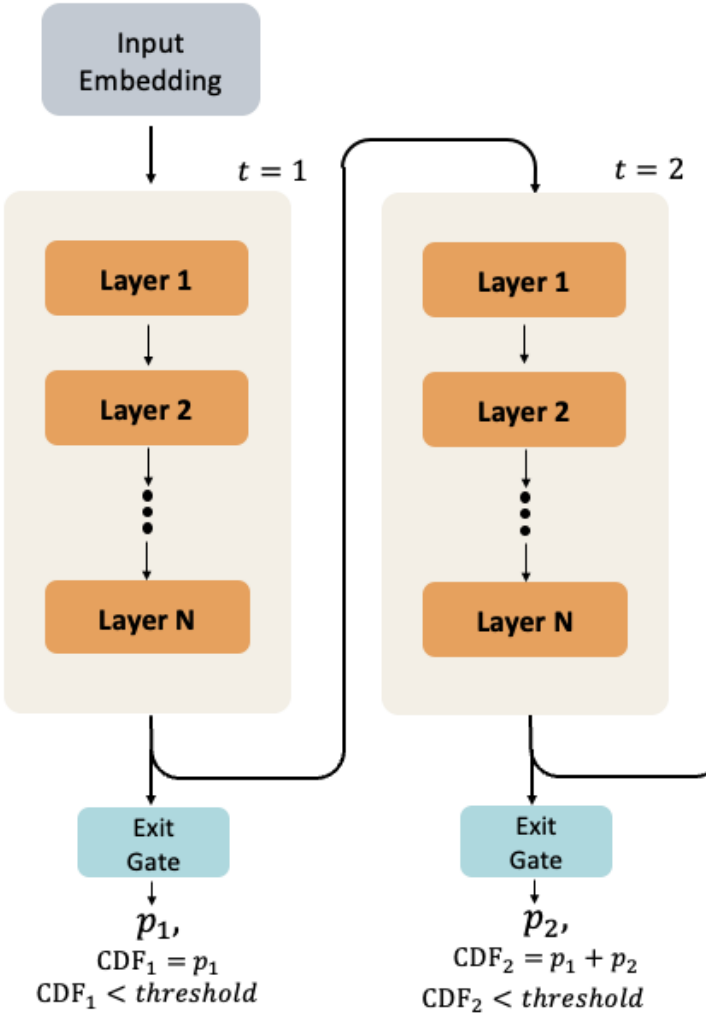


Figure 10 Effect of the prior over exit steps. Left: training loss (300-step sliding average) for a LoopLM with $T_{\max} = 4$ under different priors on z. Colored curves correspond to geometric priors with parameter $\eta \in \{0.1, \ldots, 0.9\}$; the red curve uses a uniform prior. Shaded regions indicate variability across runs. Right: prior probability over LoopLM steps induced by each η (uniform shown in red). Stronger geometric bias (larger η) concentrates mass on shallow steps, reducing credit assignment to deeper computation.

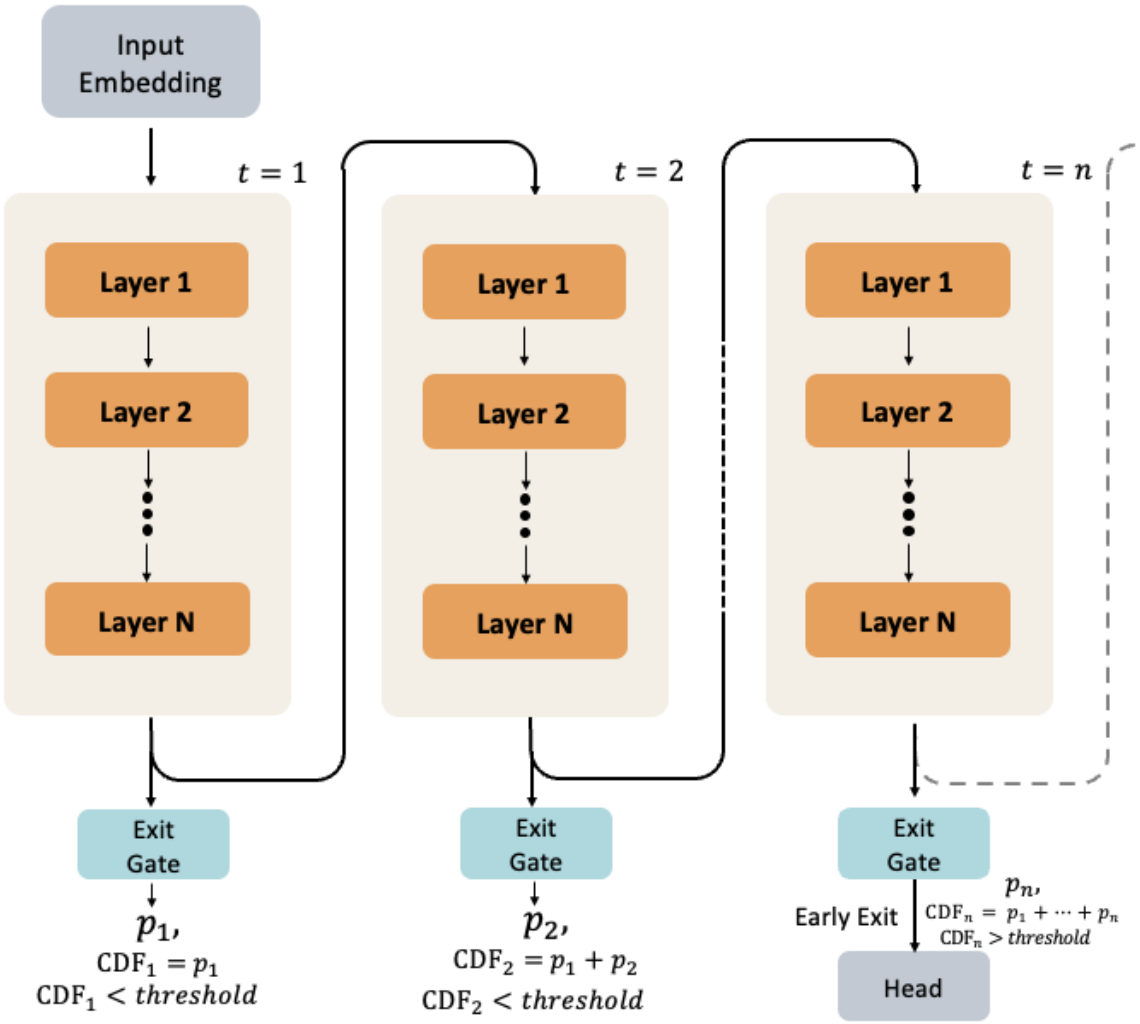
Inference with Early Exit



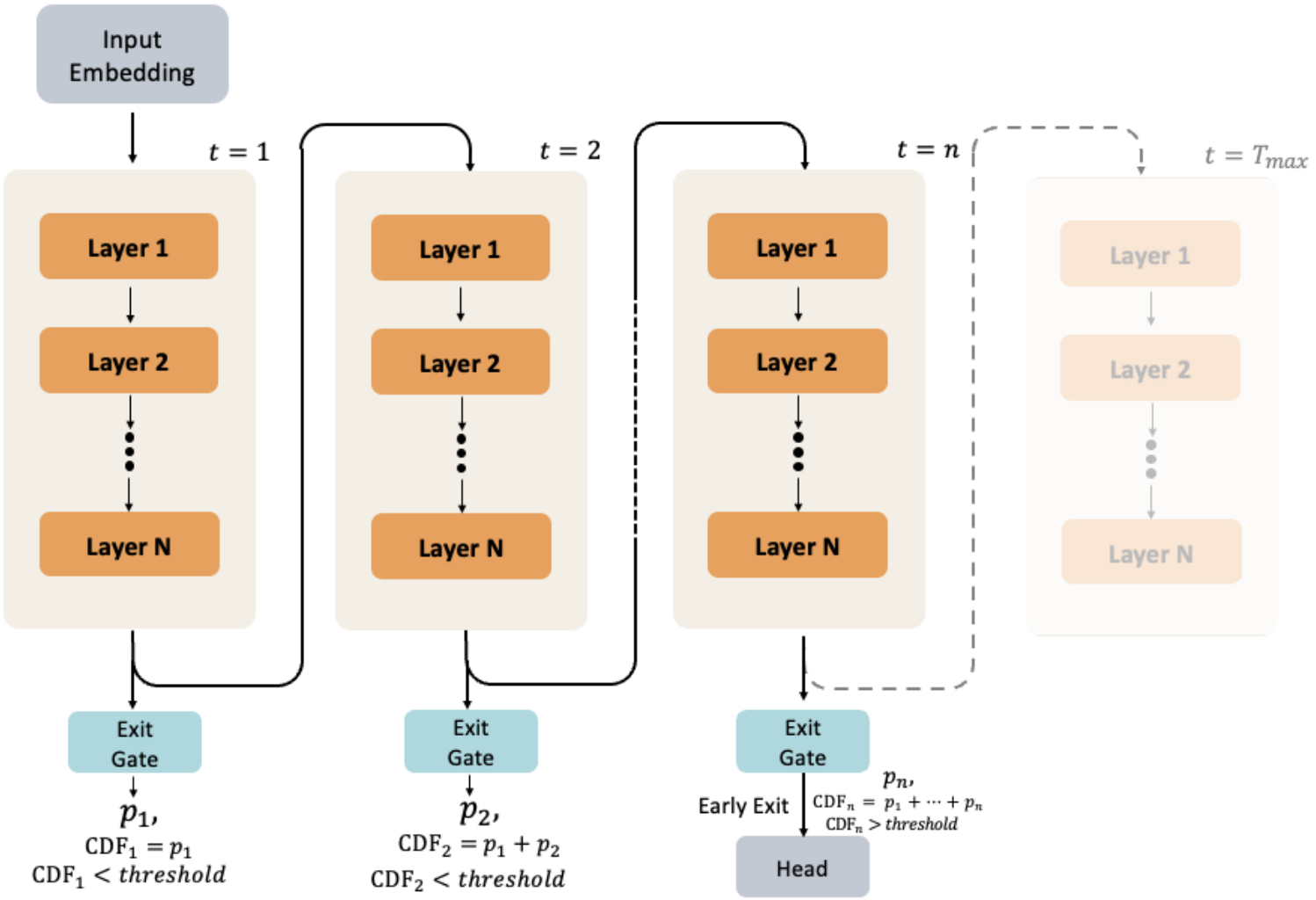
Inference with Early Exit



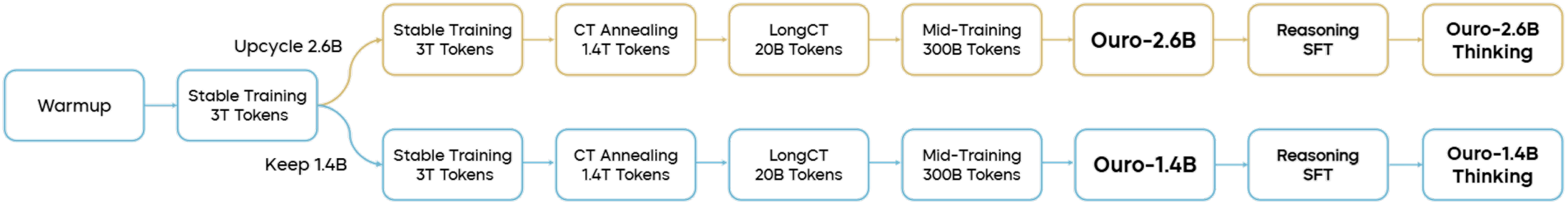
Inference with Early Exit

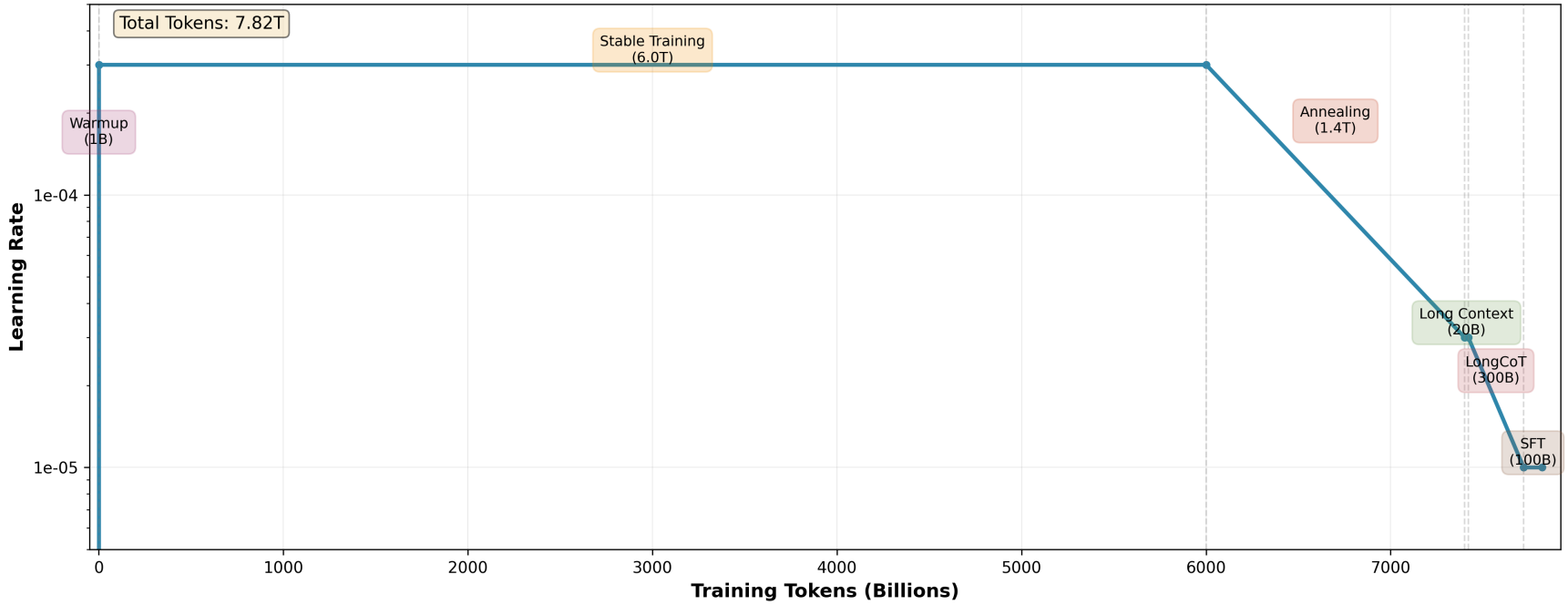


Inference with Early Exit



How do we train such a model: Training recipe

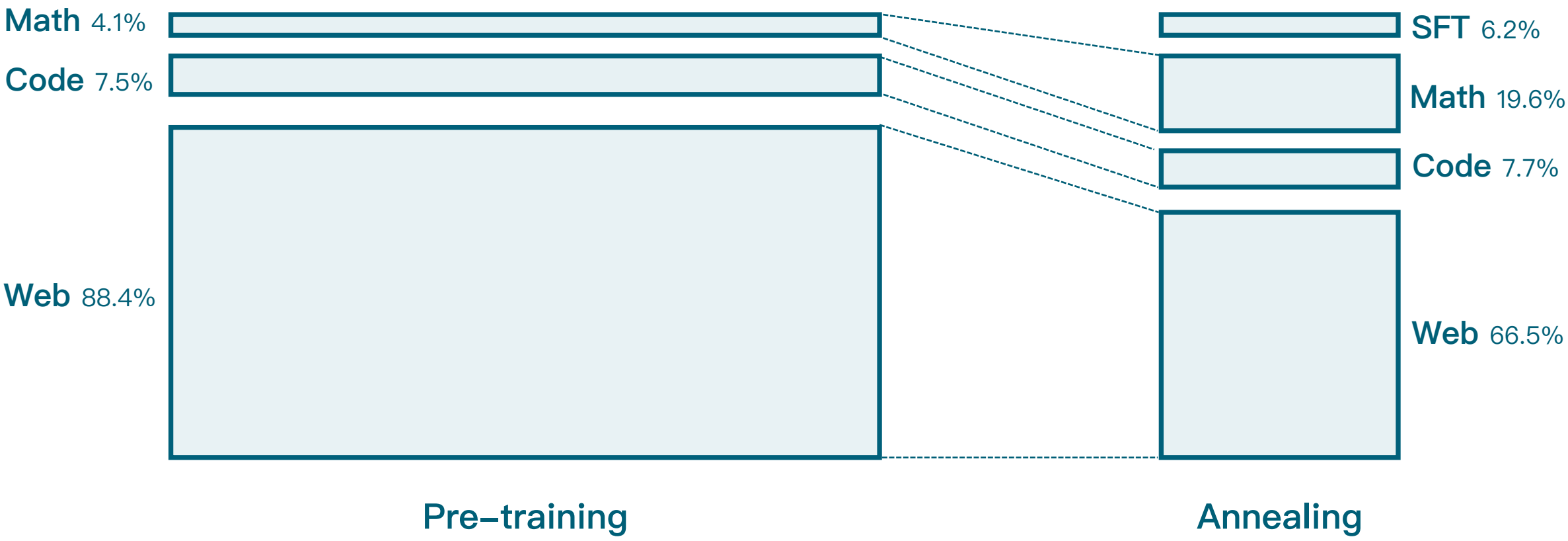




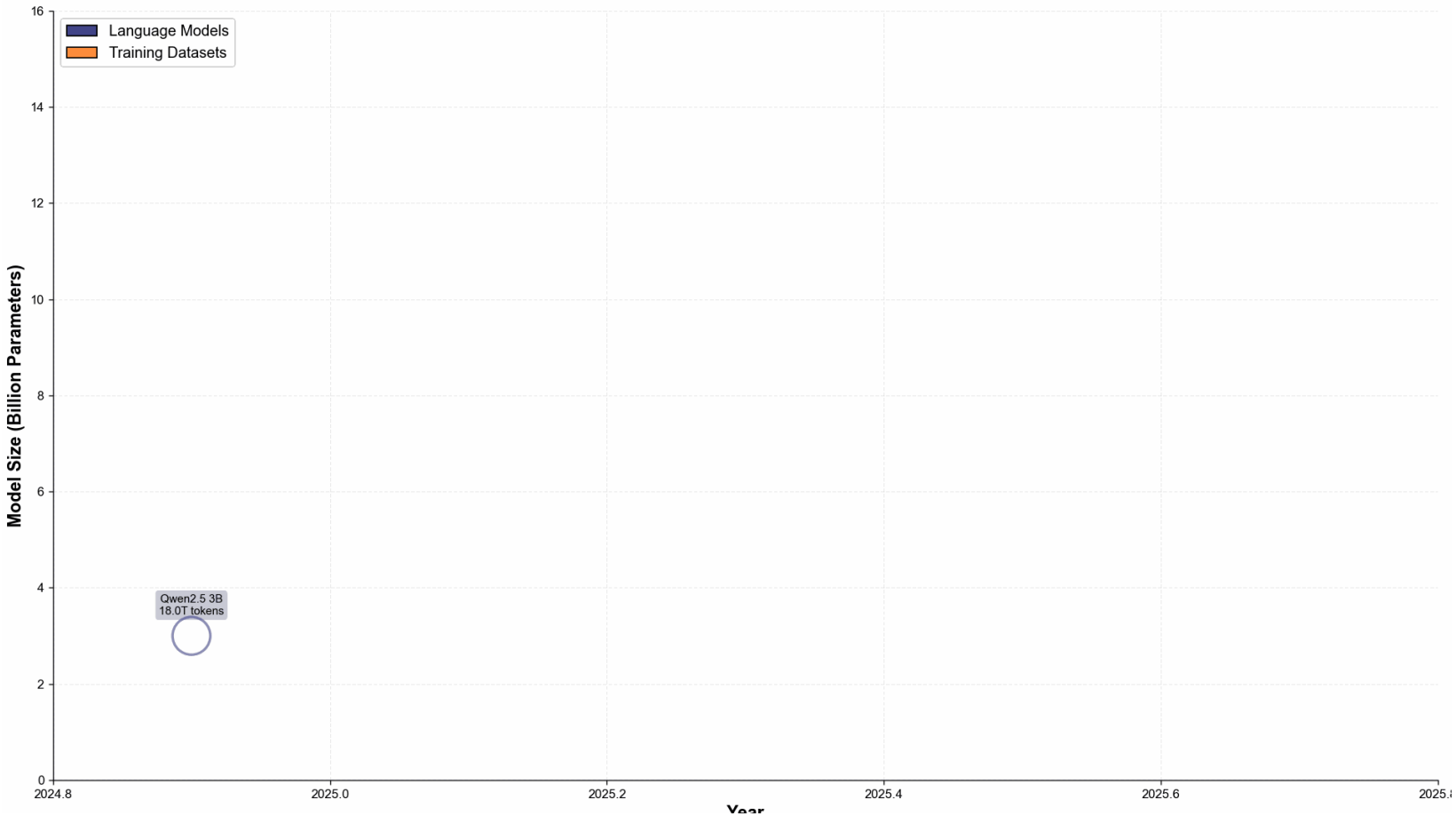
How do we train such a model: Data

Data Source	Stage	# Tokens (B)	# Used Tokens (B)
Nemotron-CC (Web Data)	Stage 1	6386	4404
MAP-CC (Web Data)	Stage 1	800	780
Ultra-FineWeb-zh (Web Data)	Stage 1	120	120
OpenCoder-pretrain	Stage 1	450	450
MegaMath-web	Stage 1	247	246
MegaMath-high-quailty	Stage 2	64	64
Nemotron-CC-Math-v1	Stage 2	210	210
Nemotron-Code	Stage 2	53	53
Nemotron-SFT-Code	Stage 2	48	48
Nemotron-SFT-General	Stage 2	87	87
OpenCoder-Annealing	Stage 2	7	7
ProLong-64K	Stage 3	20	20
Mid-training SFT Mix	Stage 4	182	90

How do we train such a model: Data

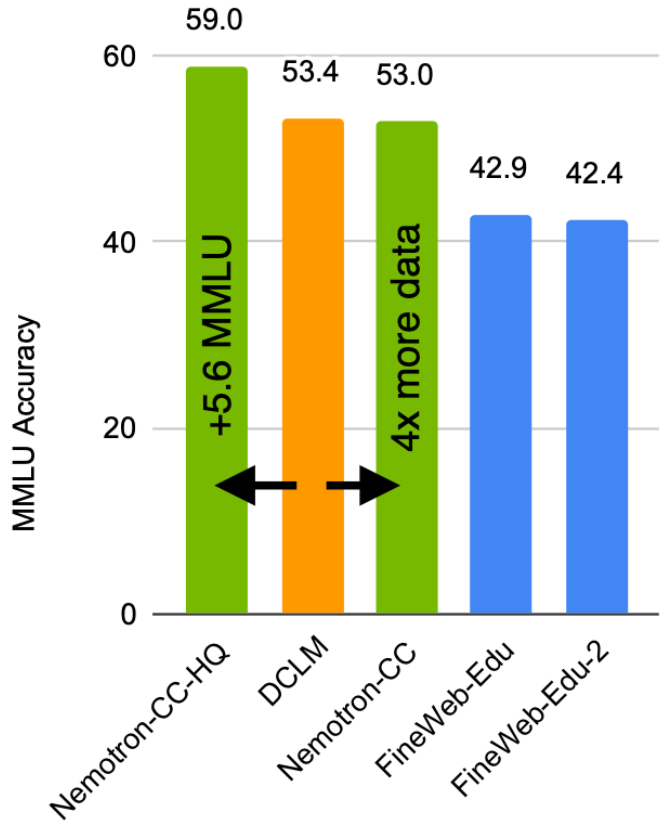


Why Nemotron-CC?



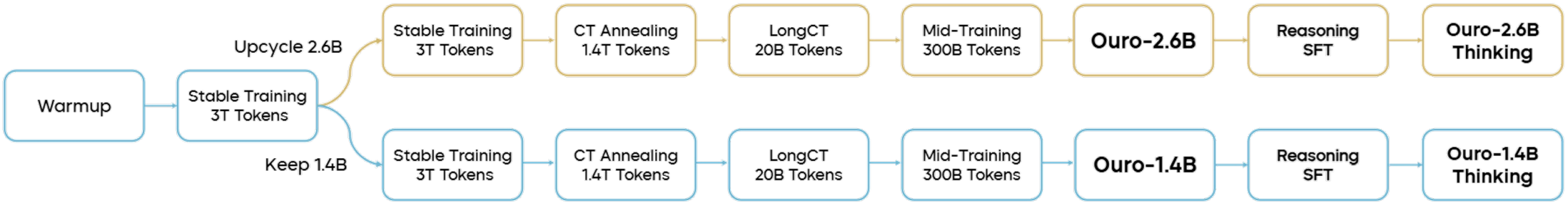
Why Nemotron-CC?

Why Nemotron-CC?



Trade off between Size and Quality

How do we train such a model: training stage



① **Warmup**: Initial model warmup phase

② **Stable Training Phase 1**: 3T tokens on standard pre-training data

③ **Model Branching**: Creating 1.4B and 2.6B variants (via upcycling)

④ **Stable Training Phase 2**: Additional 3T tokens for both model sizes

⑤ **CT Annealing**: 1.4T tokens with chain-of-thought annealing

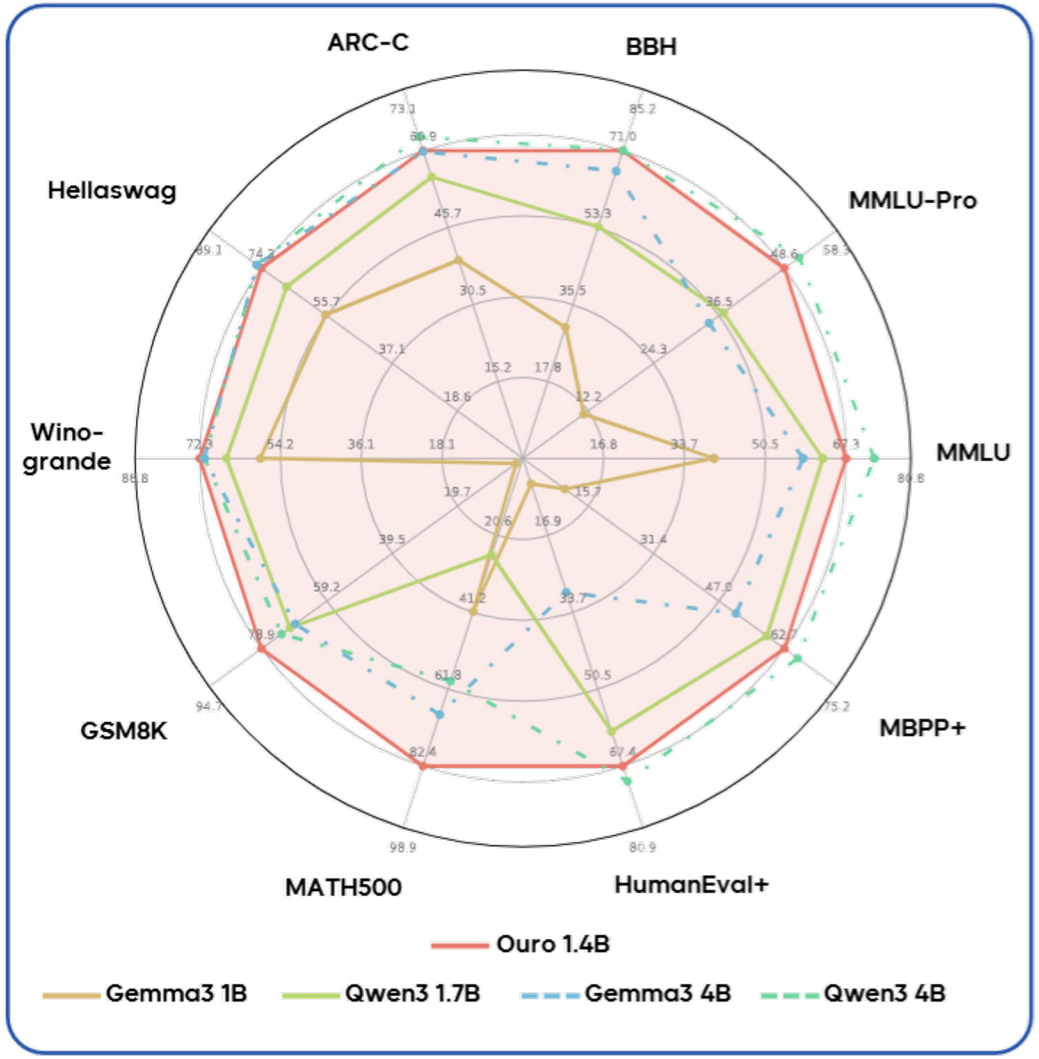
⑥ **LongCT**: 20B tokens of long-context chain-of-thought training

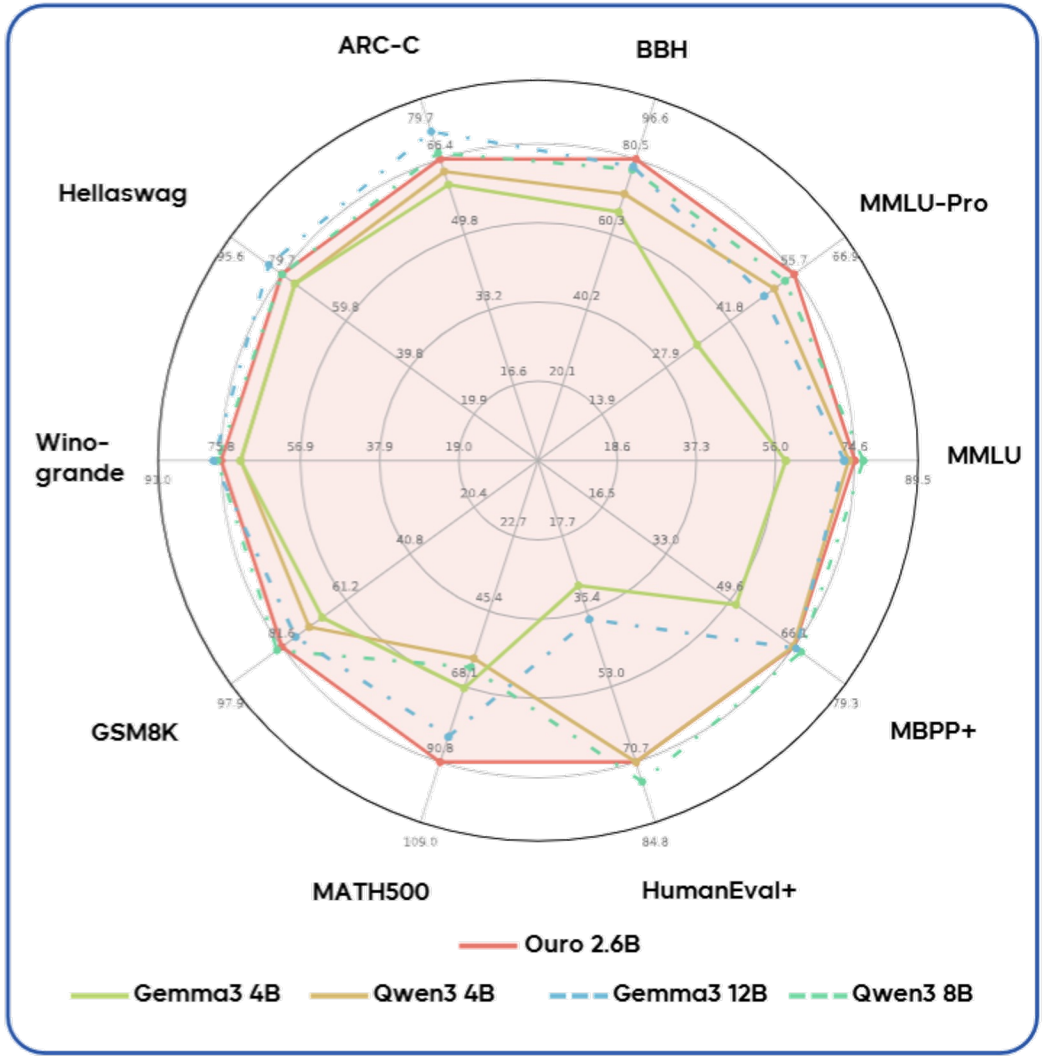
⑦ **Mid-Training**: 300B tokens of targeted mid-training

⑧ **Reasoning SFT**: Supervised fine-tuning for reasoning-focused models (Ouro-Thinking variants)

Model	Parameters	Layers	Hidden Size (d_{model})	Attention	FFN	Pos. Embed.	Vocab Size
Ouro 1.4B	1.4B	24	2048	MHA	SwiGLU	RoPE	49,152
Ouro 2.6B	2.6B	48	2048	MHA	SwiGLU	RoPE	49,152

Final Performance: Base Models



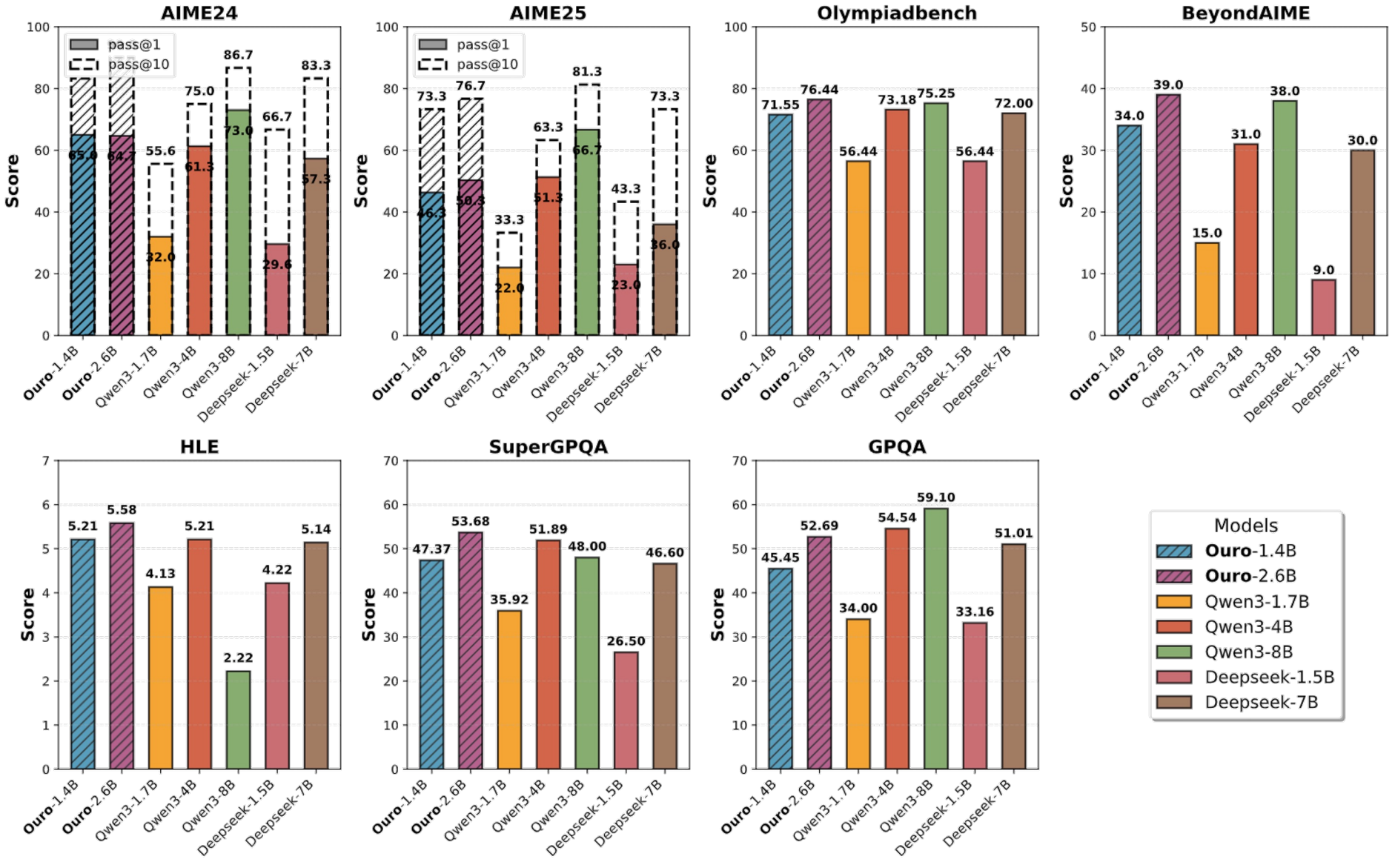


Final Performance: Base Models

	Gemma3 1B	Llama3.2 1.2B	Qwen2.5 1.5B	Qwen3 1.7B	Qwen2.5 3B	Llama3.2 3B	Qwen3 4B	Gemma3 4B	Ouro 1.4B R4
Architecture	Dense	Dense	Dense	Dense	Dense	Dense	Dense	Dense	LoopLM
# Params	1.0B	1.0B	1.5B	1.7B	3.0B	3.0B	4.0B	4.0B	1.4B
# Tokens	2T	9T	18T	36T	18T	9T	36T	4T	7.7T
General Tasks									
MMLU	39.85	45.46	60.99	62.46	65.62	59.69	**73.19**	58.37	67.35
MMLU-Pro	11.31	11.80	29.11	37.27	37.87	33.34	**51.40**	34.61	48.62
BBH	30.26	30.72	43.66	53.51	55.37	39.45	70.95	66.32	**71.02**
ARC-C	39.25	41.98	54.44	55.72	55.46	52.47	**63.65**	60.92	60.92
HellaSwag	56.12	59.35	67.73	67.09	74.54	73.09	**75.66**	75.58	74.29
Winogrande	58.72	62.75	66.77	66.30	70.17	69.14	71.19	71.07	**72.30**
Math & Coding Tasks									
GSM8K	2.05	7.05	60.73	70.28	74.60	67.20	72.86	68.69	**78.92**
MATH500	41.00	7.40	17.60	25.80	42.60	40.80	59.60	68.60	**82.40**
HumanEval	6.70	19.50	52.40	66.50	68.90	29.90	**77.40**	34.80	74.40
HumanEval+	5.50	17.40	46.30	59.80	62.20	26.20	**70.70**	29.30	67.40
MBPP	12.40	35.70	60.30	68.00	63.00	50.30	**78.80**	60.60	73.00
MBPP+	10.10	29.10	50.00	58.50	54.20	39.70	**65.90**	51.10	62.70

	Qwen2.5 3B	Llama3.2 3B	Qwen3 4B	Gemma3 4B	Qwen2.5 7B	Llama3.1 8B	Qwen3 8B	Gemma3 12B	Ouro 2.6B R4
Architecture	Dense	Dense	Dense	Dense	Dense	Dense	Dense	Dense	LoopLM
# Total Params	3.0B	3.0B	4.0B	4.0B	7.0B	8.0B	8.0B	12.0B	2.6B
# Trained Tokens	18T	9T	36T	4T	18T	15T	36T	12T	7.7T
General Tasks									
MMLU	65.62	59.69	73.19	58.37	74.20	73.02	**76.63**	72.14	74.60
MMLU-Pro	37.87	33.34	51.40	34.61	43.55	43.24	53.72	49.21	**55.73**
BBH	55.37	39.45	71.14	66.32	53.72	71.56	77.65	78.41	**80.46**
ARC-C	55.46	52.47	63.65	60.75	63.65	60.75	66.10	**72.44**	66.40
HellaSwag	74.54	73.09	75.66	75.58	79.98	81.97	79.60	**83.68**	79.69
Winogrande	70.17	69.14	71.19	71.27	76.48	77.11	76.80	**77.74**	75.85
Math & Coding Tasks									
GSM8K	74.60	67.20	72.86	68.69	81.50	78.17	**83.09**	77.18	81.58
MATH500	42.60	40.80	59.60	68.60	61.20	52.90	62.30	83.20	**90.85**
HumanEval	68.90	29.90	77.70	34.80	79.30	38.40	**84.80**	46.30	78.70
HumanEval+	62.20	26.20	70.70	29.30	70.60	31.10	**75.30**	37.20	70.70
MBPP	63.00	50.30	78.80	60.60	73.80	62.40	79.00	73.50	**80.40**
MBPP+	54.20	39.70	65.90	51.10	63.50	51.60	**67.90**	66.10	66.60

Final Performance: Reasoning Models



Wait, but why LoopLM can achieve this?

Physics of LoopLMs

Understanding LoopLMs Superiority from a Parametric Knowledge Viewpoint

Two hypotheses:

① **Increased knowledge capacity:** LoopLMs might be able to memorize more facts even with the same number of parameters.

② **Better knowledge extraction/composition:** LoopLMs might not store more facts but might use the knowledge in the parameters more effectively through repeated reasoning.

Question 1: Does looping increase knowledge capacity?

Capo Task

Layla Jack Beasley celebrates their birthday on January 24, 1914. They spent formative years in Portland, ME. They focused on Business Analytics. They supported operations for Delta Air Lines Inc. in Atlanta, GA. They received their education at Pepperdine University.

Question 1: Does looping increase knowledge capacity ?

Capo Task

Layla Jack Beasley celebrates their birthday on January 24, 1914. They spent formative years in Portland, ME. They focused on Business Analytics. They supported operations for Delta Air Lines Inc. in Atlanta, GA. They received their education at Pepperdine University.

Dataset: Synthetic biographies bioS(N) [Zeyuan et al., 2025],

- Each biography contains name + 5 attributes: gender, birth date, university, major, employer
- Names and attributes randomly sampled from predefined sets, combined into natural language via templates

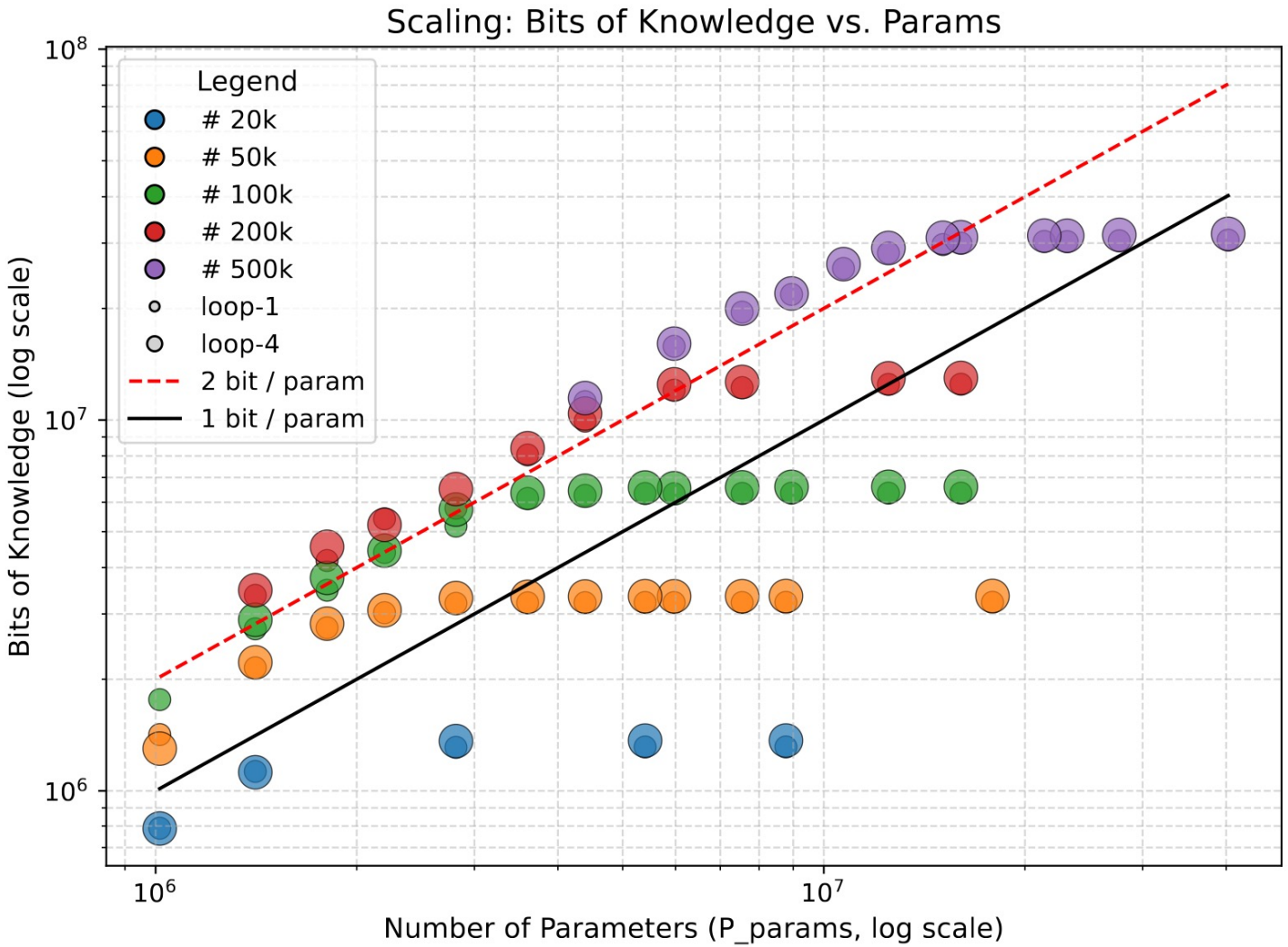
Knowledge capacity metric: Sum of cross–entropy loss per attribute token → converted to bits

Models: GPT–2 style + LoopLM

- Loop steps: 1 (non–loop baseline) vs 4 (looped model)
- Parameter sizes: 1M — 40M, varying depth and hidden dimensions

Training: Each data sample seen 1000 times

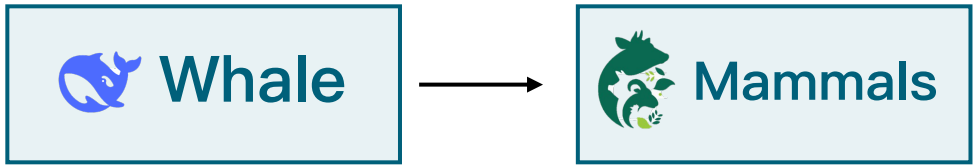
Question 1: Does looping increase **knowledge capacity ?**



Knowledge capacity:
Same as non–loop model

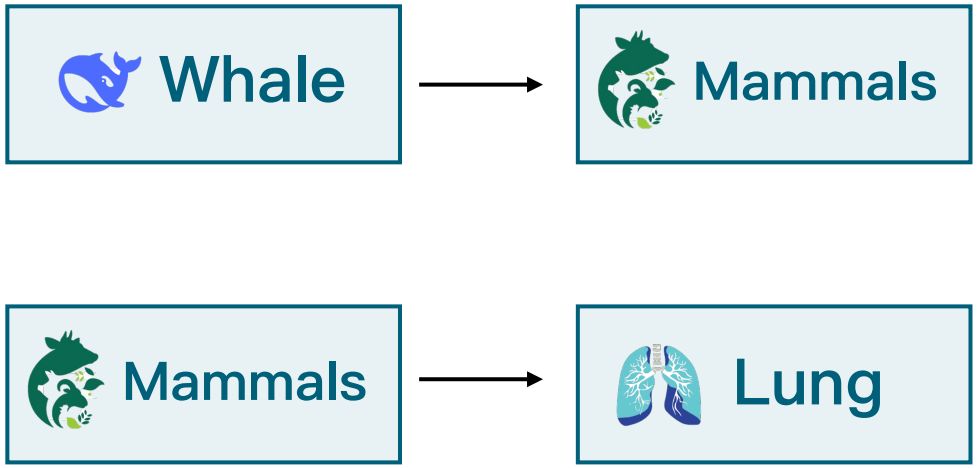
Question 2: Does looping improve knowledge manipulation?

What is knowledge manipulation?

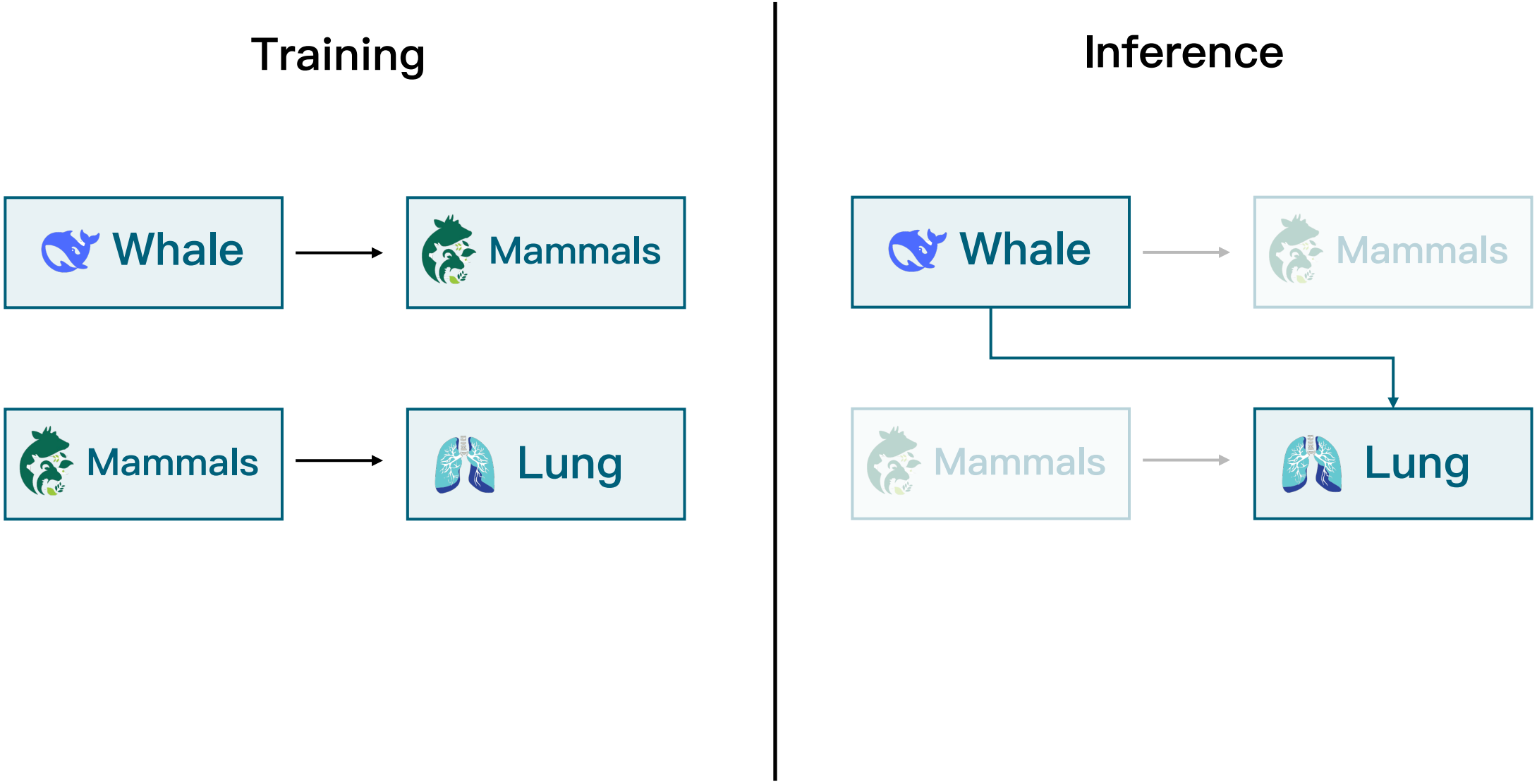


What is knowledge manipulation?

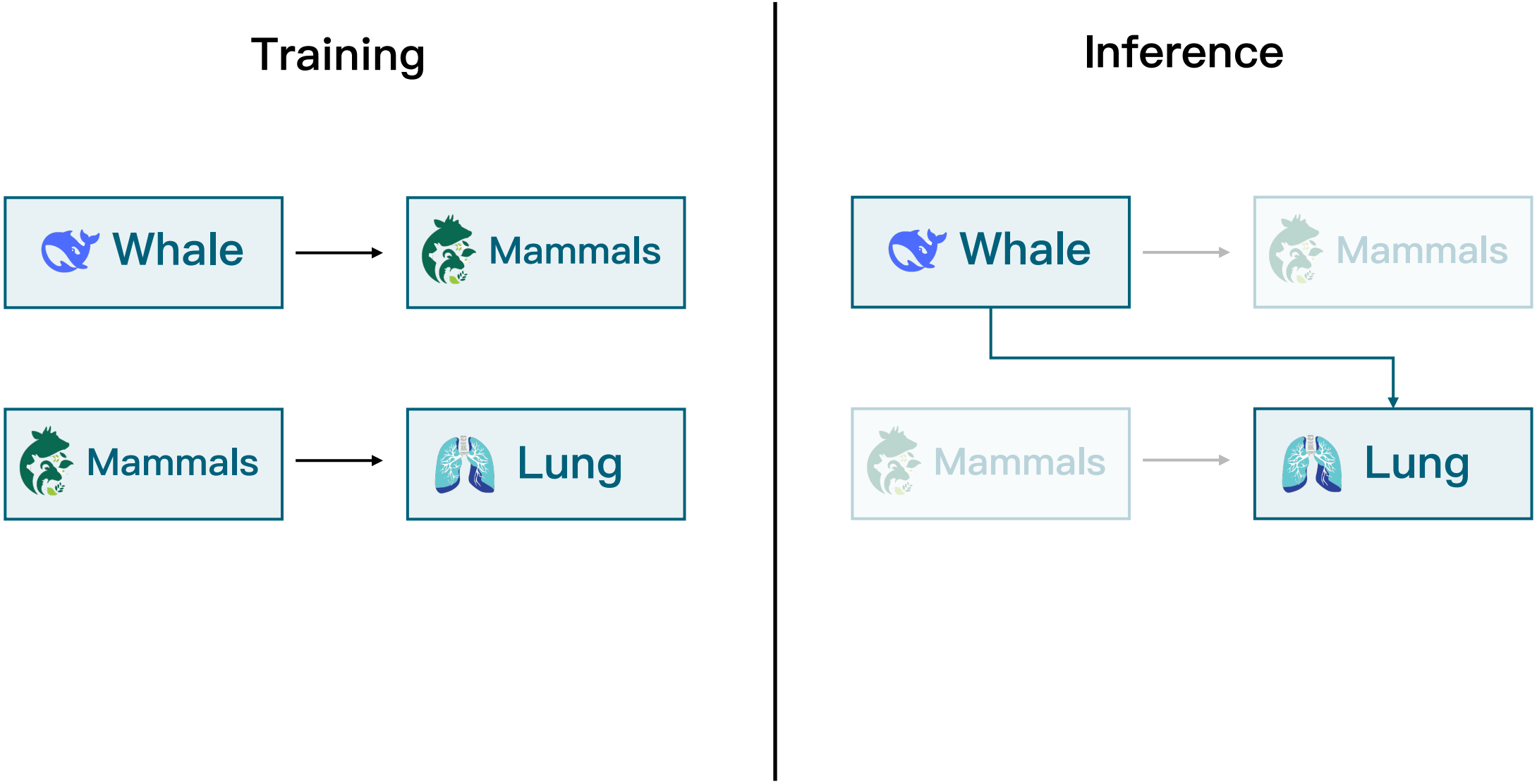
Training



What is knowledge manipulation?



What is knowledge manipulation?



Task 1: Multi–hop QA Task

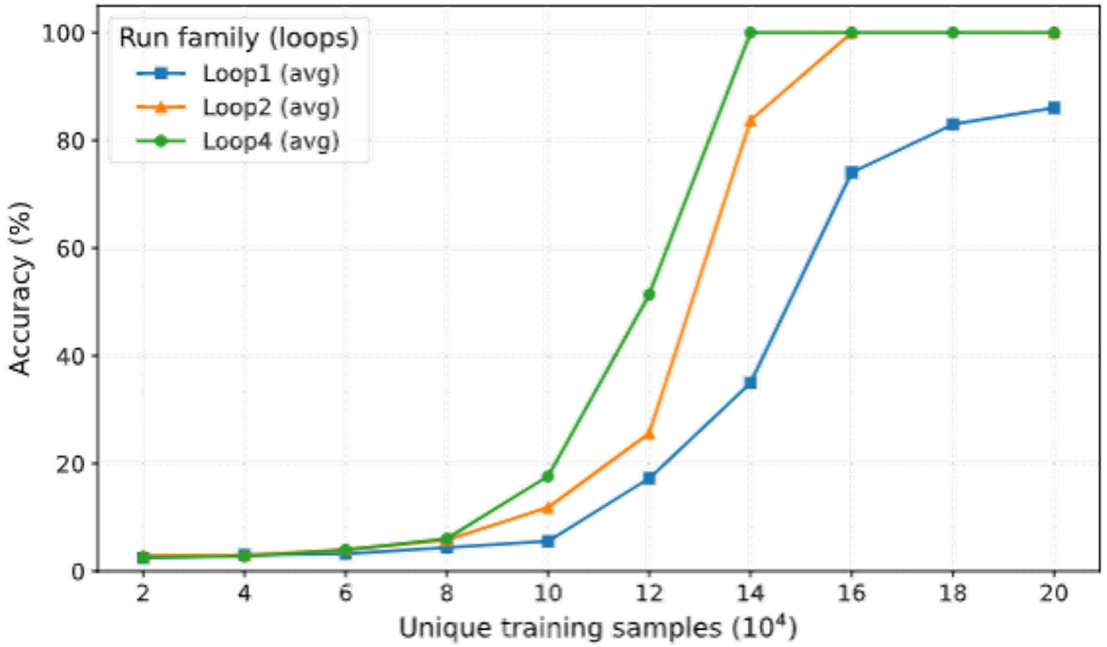
Dataset:

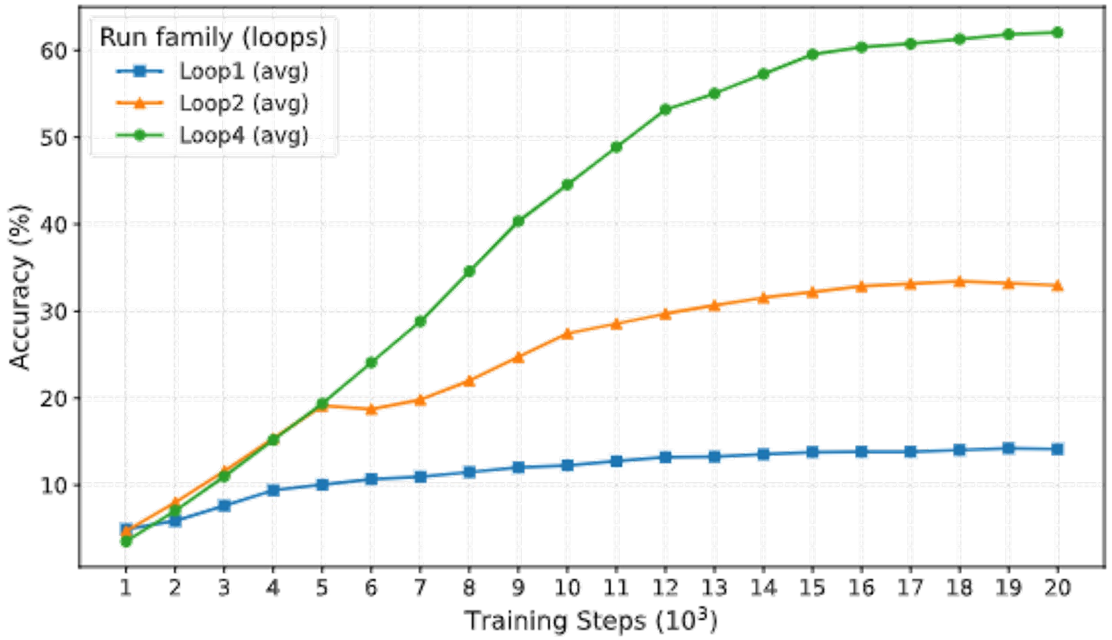
- |E| entities, each with a unique name, and N relation types.
- 500 entity names (e.g., Jennifer) and 20 relation names (e.g., instructor), from [Yao et al., 2025].
- Multi–hop questions generated via a K=5 hierarchical structure: 100 entities per layer, each connecting to |R| random entities in the next layer.
- Produces |E|/5 × |R|^k k–hop questions; we focus on 3–hop (~8×10^5 questions).

Training & Evaluation:

- Train on a subset of 3–hop QA pairs; test on 3,000 held–out questions.
- Answers decoded greedily (e.g., “Who is the instructor of the teacher of Bob?”).
- Metric: exact match accuracy.

Task 1: Multi–hop QA Task





- Models with more loops require fewer samples to learn the 3–hop QA task.
- Models with more loops learn faster and achieve better performance compared to models without loops.

Increasing the number of iterative steps significantly improves performance on **reasoning–intensive** tasks, but yields limited gains on **knowledge–intensive** tasks.

Task 2: Mano Task

Mano Task. We first explore the knowledge manipulation task Mano in [68], based on a complex tree structure with restricted modular arithmetic knowledge. Models need to solve the task without intermediate thinking process. As illustration, an example could be `<bos> + * a b c <eos>` requires the model to directly output $(a * b) + c \bmod 23$. To solve this task, the model needs to (1) apply the arithmetic rules modulo 23 as the factual knowledge encoded in the parameters, and (2) parse the binary tree structure of the arithmetic to compose all calculations.

	$L = 10$	$L = 16$	$L = 24$
Baseline model			
Base $(12 \otimes 1)$	93.6	94.4	34.8
2 layer model			
Base $(2 \otimes 1)$	21.5	8.4	7.5
Loop $(2 \otimes 6)$	**98.1**	**96.3**	78.0
3 layer model			
Base $(3 \otimes 1)$	75.4	29.8	11.0
Loop $(3 \otimes 4)$	**97.9**	**95.8**	**92.2**
6 layer model			
Base $(6 \otimes 1)$	84.7	59.5	20.0
Loop $(6 \otimes 2)$	93.4	88.5	35.1

Understanding Why LoopLM Enhances Knowledge Manipulation:

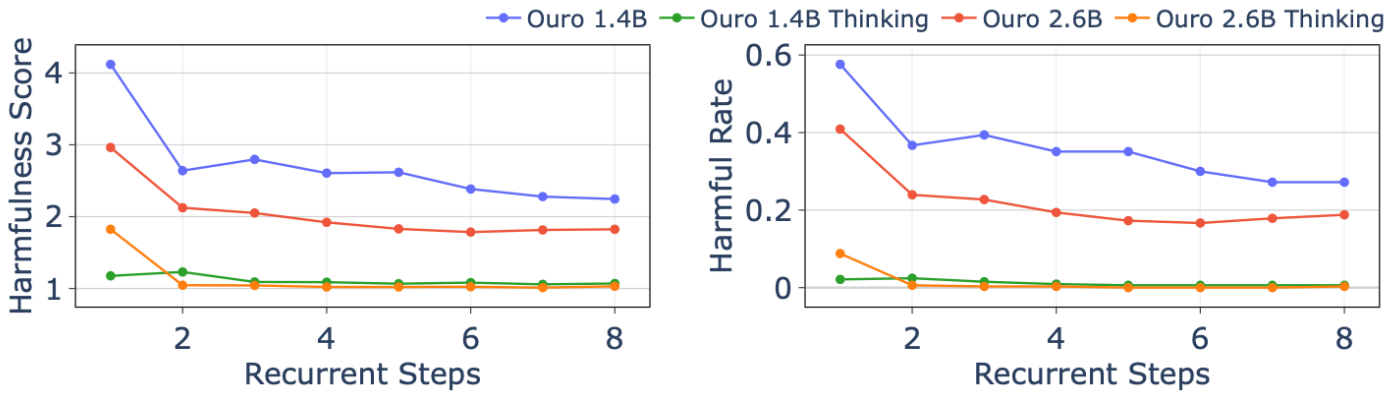
Why LoopLM helps knowledge manipulation:

- Limited parameter capacity → looping allows more efficient use of knowledge:
  - **Reuse knowledge in each looped block**
  - **Retrieve new factual information**
  - **Apply structured procedures for final prediction**

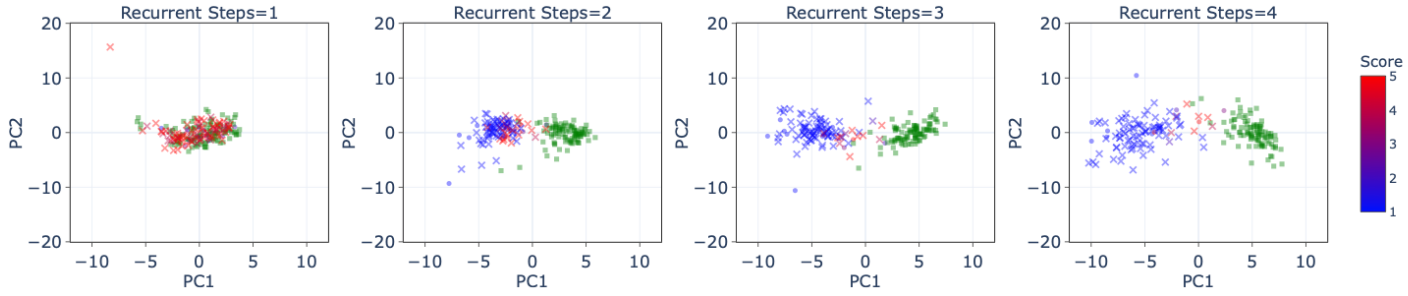
Searching on the parametric knowledge graph:

- Pre-trained LMs store large amounts of factual knowledge but often **perform shallow reasoning**.
- Complex tasks **require combining multiple pieces of knowledge** → deep search on the knowledge graph with directional dependencies.
- LoopLM's recurrent structure allows **efficient reuse of knowledge and procedures**: even if some knowledge isn't used earlier, it can be revisited in subsequent loops.

Safety Emerges from Recurrent Computation



(a) HEx-PHI evaluation



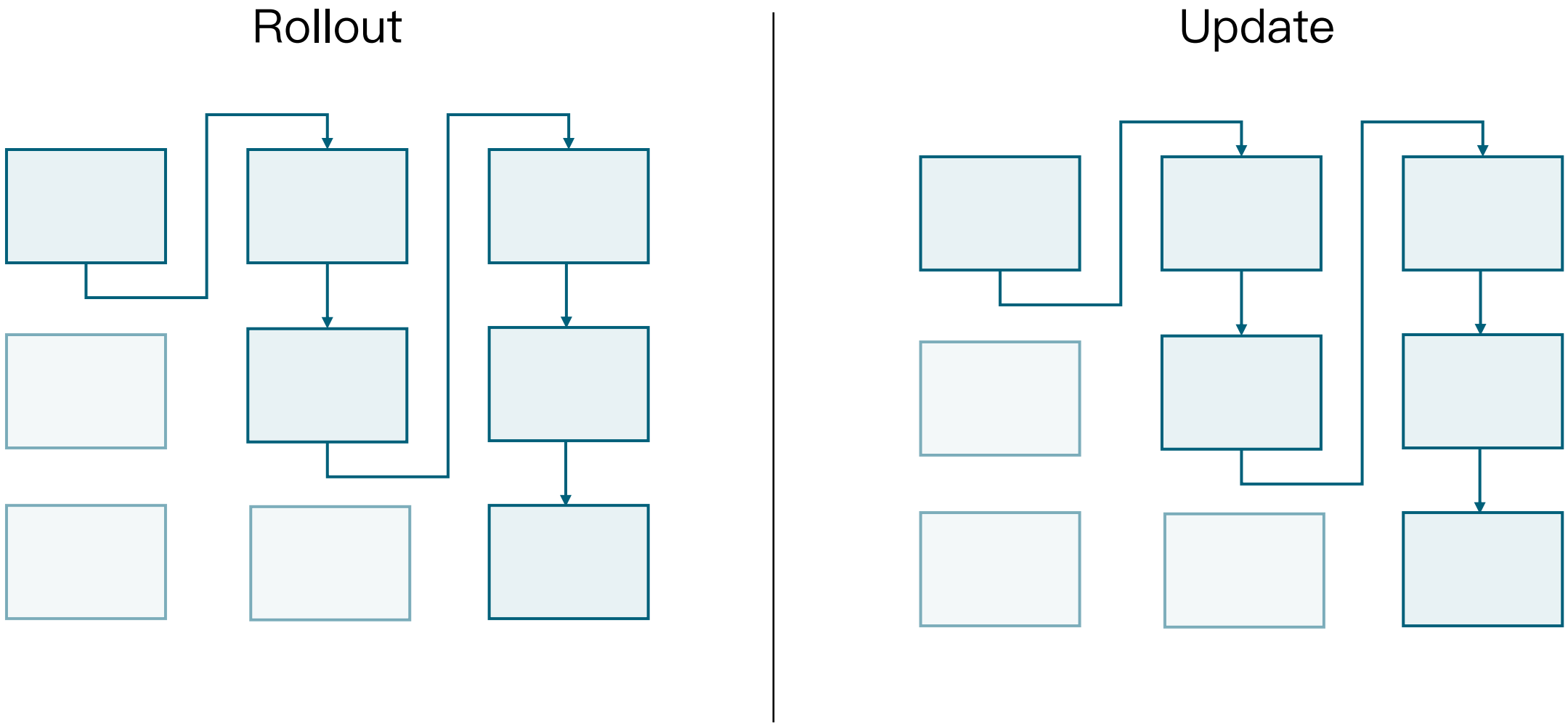
(b) PCA analysis on Ouro 1.4B

Thinking Improves Safety: Harmfulness scores and rates decline significantly as recurrent steps increase, leading to safer model responses.

Robust Extrapolation: Safety alignment continues to improve even when reasoning extends beyond the training configuration (extrapolating from 4 to 8 steps).

Latent Separation: PCA visualization (Figure b) confirms that iterative computation helps the model better distinguish between benign and harmful prompts in the latent space.

The failure of the Reinforcement Learning



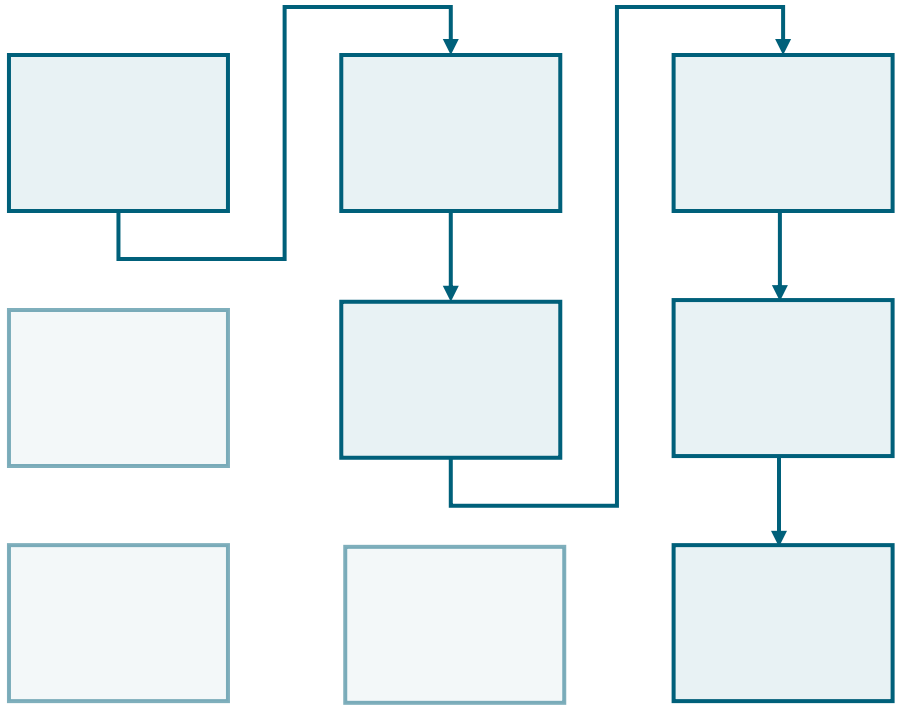
Ideally, Rollout and Update should be identical…

The failure of the Reinforcement Learning

- Data: Only use DAPO-17K
- Training-Inference Divergence

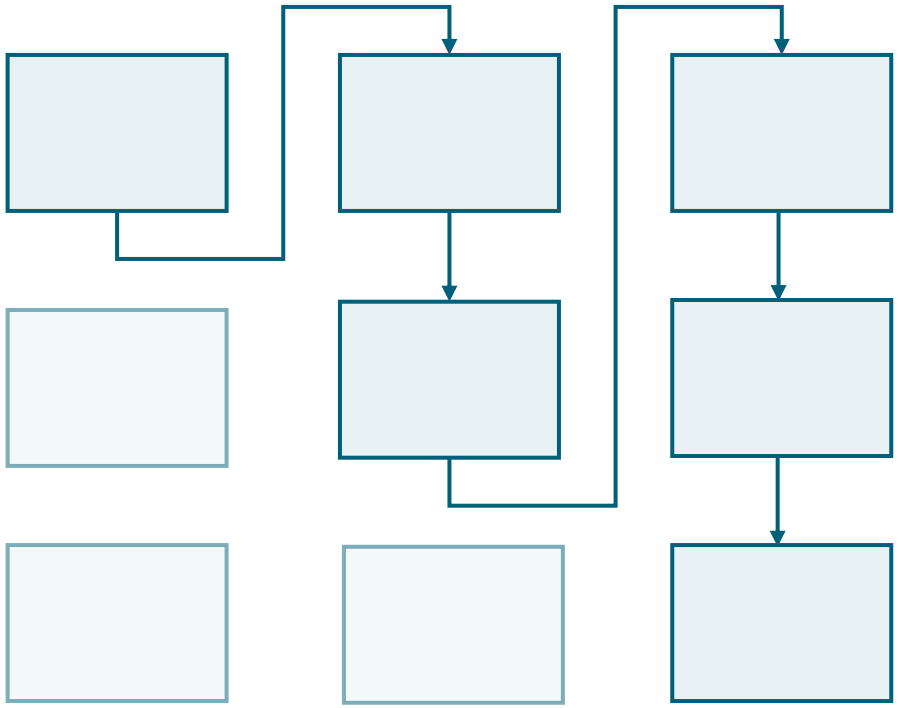
The failure of the Reinforcement Learning

Rollout



Transformers: ✅
vLLM: ❌

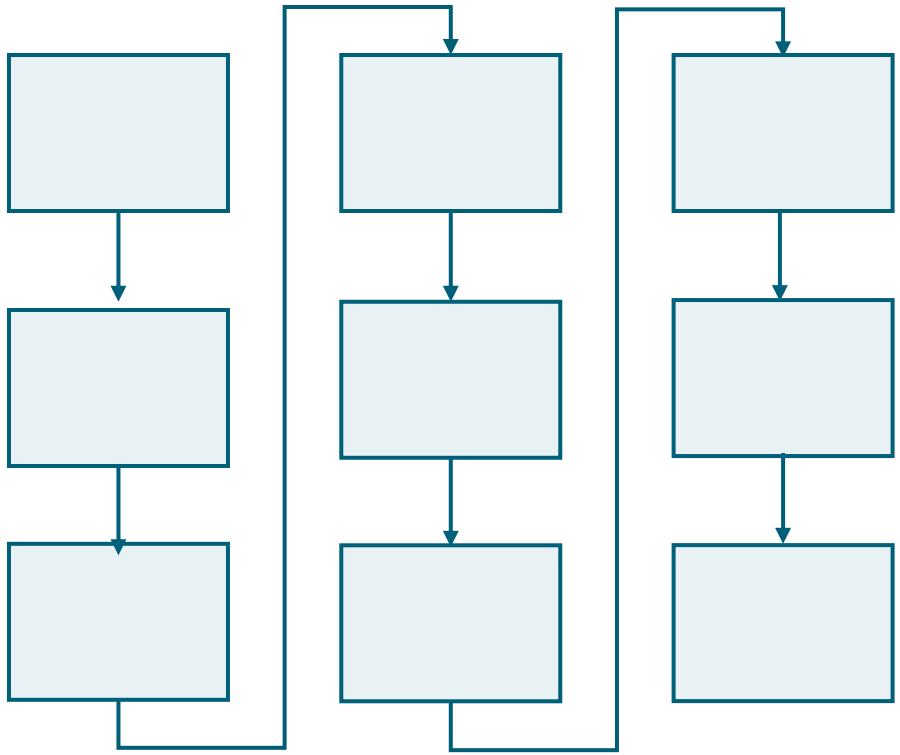
Update



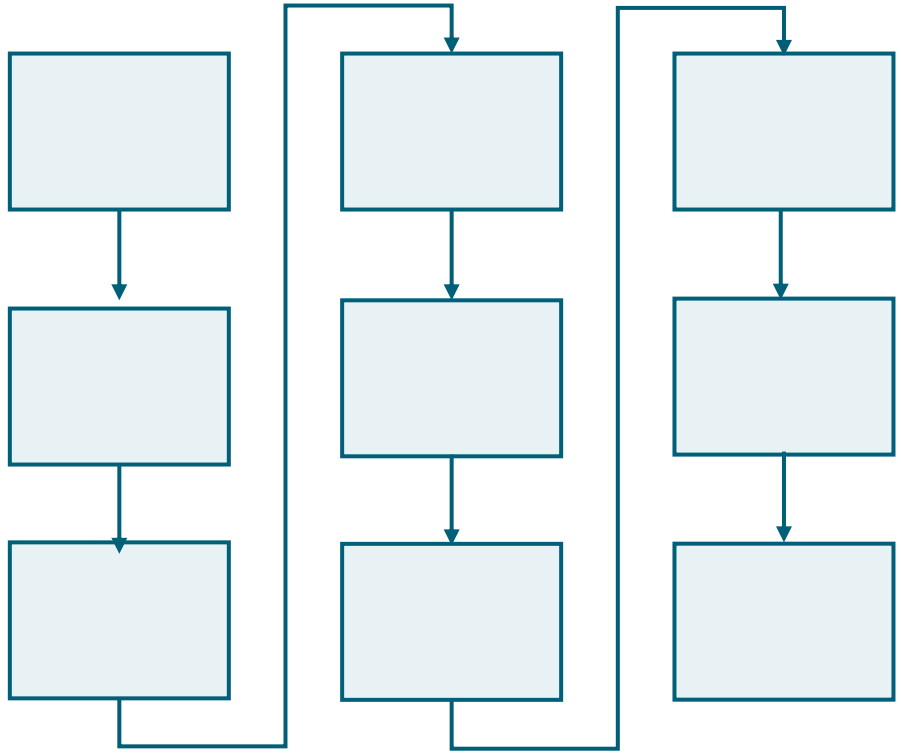
Only need Transformers: ✅

To make the rollout and update identical…

Rollout



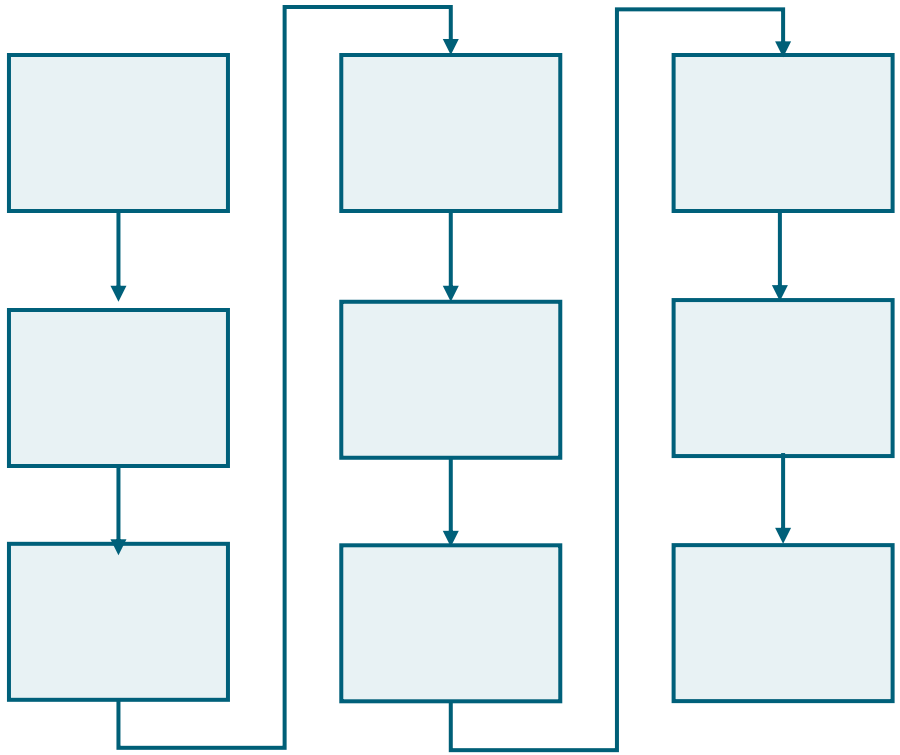
Update



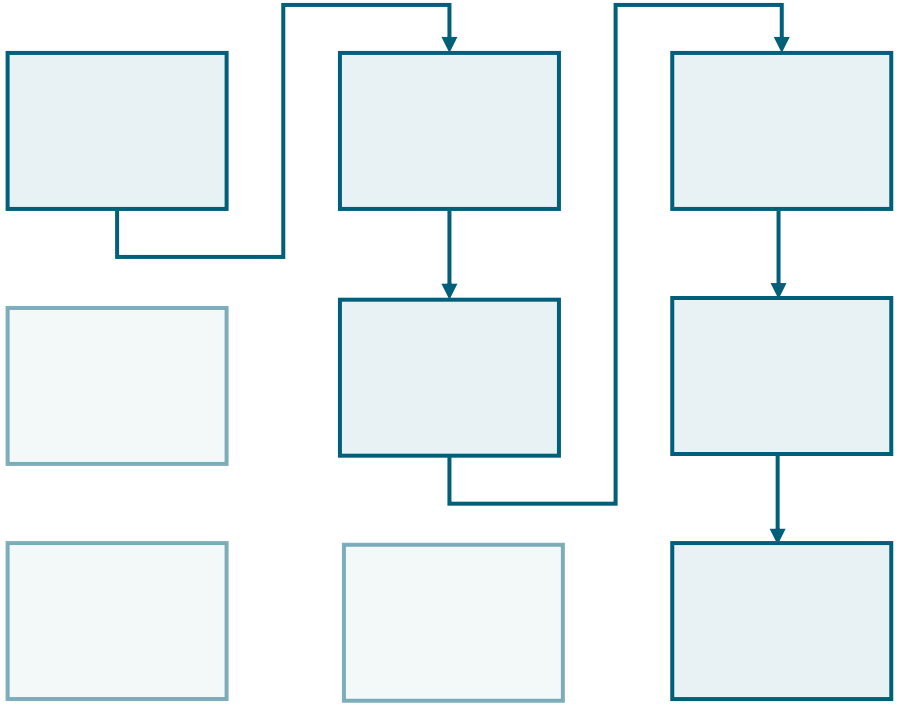
Fixed depth rollout and update with the same depth

Attempts: self–distillation?

Rollout



Update



Takeaways:

The Big Idea: Latent Reasoning

- Instead of outputting text immediately, the model loops over its own hidden states to reason internally.
- By reusing the same parameters multiple times (looping), we get better reasoning capabilities without exploding the model size.

Making it Work

- Reducing the recurrent steps from 10+ to 4.
- Stabilizing training with Sandwich Norm and providing a prior to the early exit.

Meet Ouro: Small but Mighty

- We built the Ouro models (1.4B & 2.6B) using a specialized recipe: Upcycling old models –> Stable Training –> Long–Context Reasoning.
- Our 2.6B model isn't just good for its size; it’s beating 8B and 12B models (like Qwen and Gemma) on hard math and code tasks.

The Physics of Why it Works

- We found something fascinating: Looping doesn't help the model memorize more facts (Knowledge Capacity).
- Instead, it drastically improves Knowledge Manipulation. It gives the model the "mental space" to connect the dots and manipulate the facts it already knows.

Thanks & QA