An Introduction to Latent Reasoning

Yi Hu

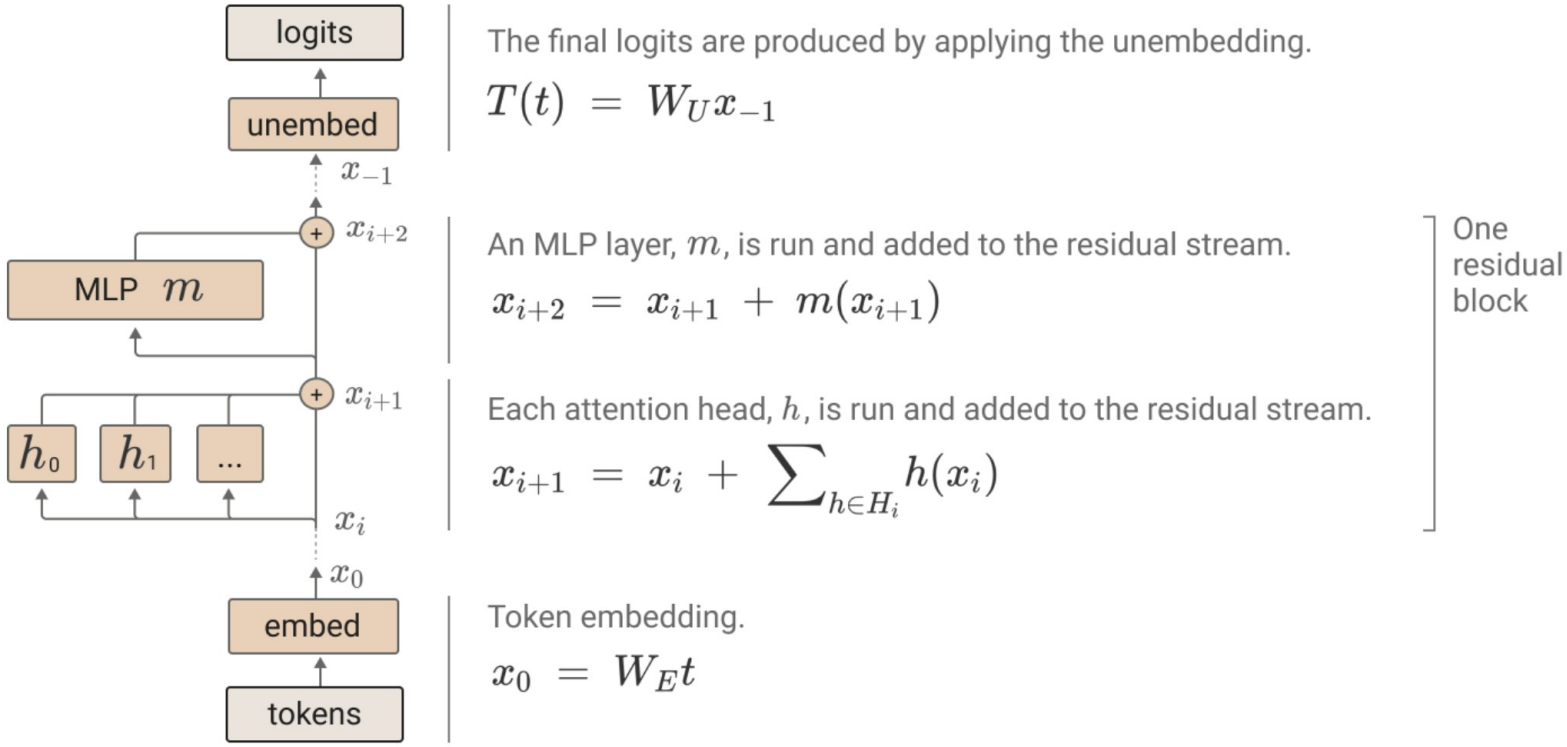
2026.2.6

Outline

- **Preliminary**
- Latent Reasoning
  - Loop/Universal Transformer Recurrence: Universal Transformer, CoTFormer, Scaling Up Looped Transformers
  - Activation with Explicit Hidden-State Feedback: Coconut, CoCoMix
- A Brief Introduction of Mechanistic Interpretability

Preliminary

- Latent CoT methods perform reasoning through continuous representations and hidden states within the model's computational graph



Activation-based Methods

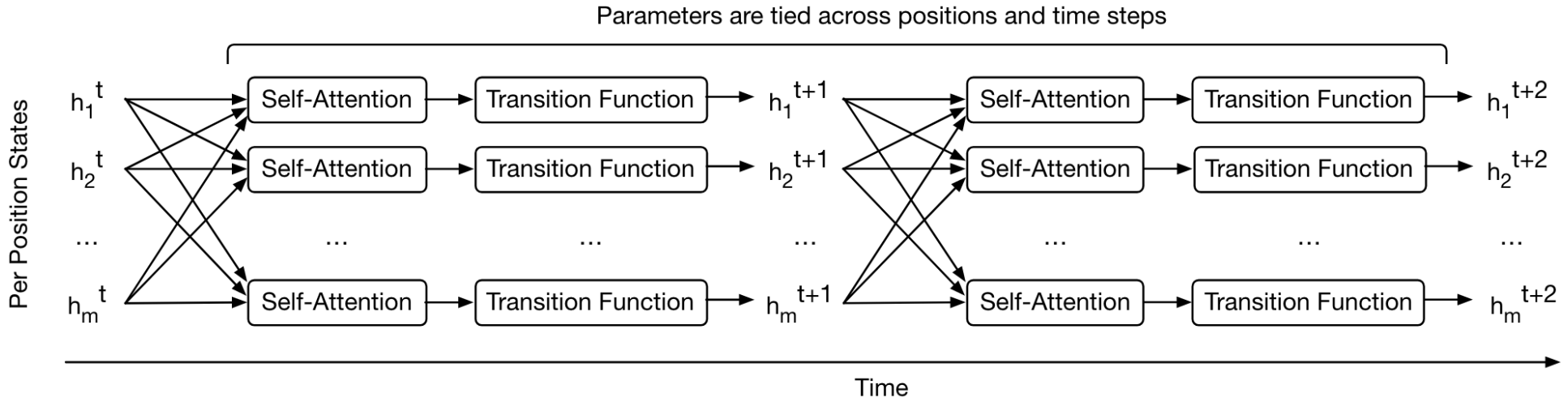
- Loop / Universal Transformer Recurrence
  - universal transformer
  - seed: Ouro
- Activation with Explicit Hidden-State Feedback
  - coconut, cocomix
  - cotformer
- Training-induced Recurrence
- Training Strategies for Recurrent Reasoning
  - MIDAS

Looped / Universal Transformer

- Loop-based architectures represent the foundational approach to activation-based latent CoT reasoning
- implementing continuous activation propagation across Transformer layers through explicit architectural modifications
- enabling iterative refinement of hidden states within a single forward pass through layer-wise recurrence

Universal Transformers

- combine the parallelizability and global receptive field of Transformer with recurrent inductive bias of RNNs
- different tokens need different compute
- repeatedly refine the representations of each token
  - parameters are fixed across positions and timesteps
  - dynamic halting: T is dynamically determined for each position



Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. arXiv preprint arXiv:1807.03819, 2018.

Universal Transformers

- at the beginning of each loop, add positional and "depth" embedding

At step t, the UT then computes revised representations $H^t \in \mathbb{R}^{m \times d}$ for all m input positions as follows

FFN

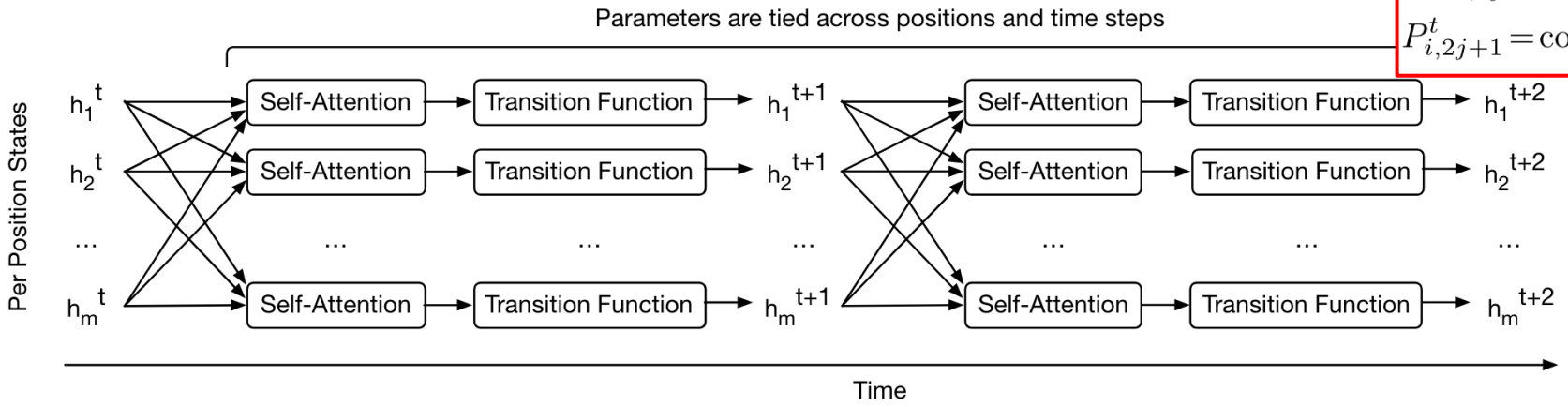
add embedding

Attention

$$H^t = \text{LayerNorm}(A^t + \text{Transition}(A^t)) \quad (4)$$

$$\text{where } A^t = \text{LayerNorm}((H^{t-1} + P^t) + \text{MultiHeadSelfAttention}(H^{t-1} + P^t)), \quad (5)$$

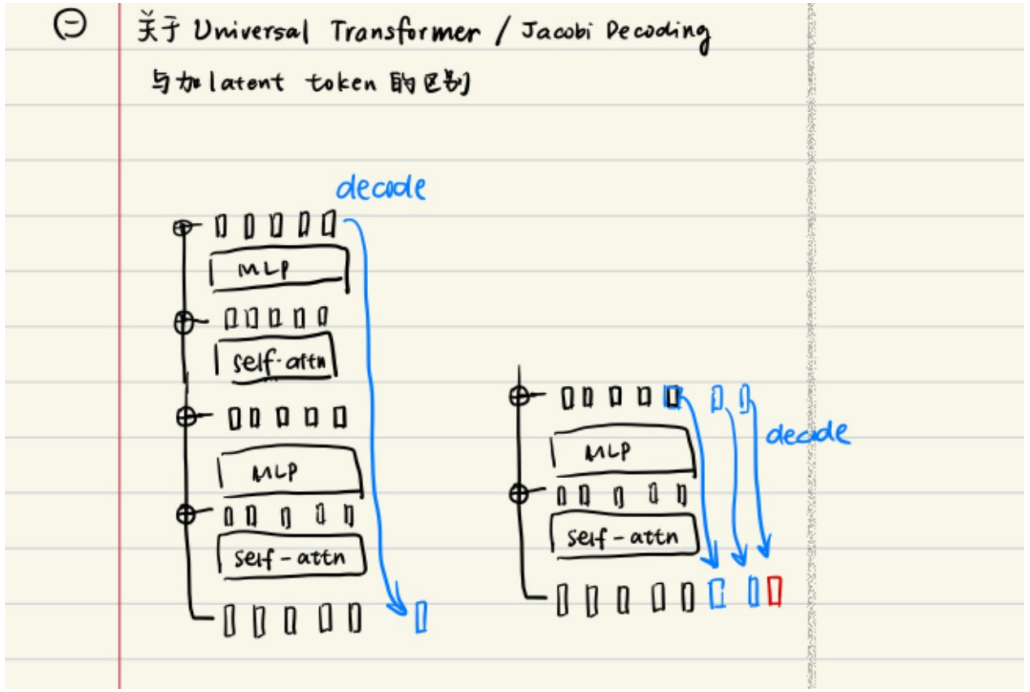
$$P^t_{i,2j} = \sin(i/10000^{2j/d}) + \sin(t/10000^{2j/d})$$
$$P^t_{i,2j+1} = \cos(i/10000^{2j/d}) + \cos(t/10000^{2j/d}).$$

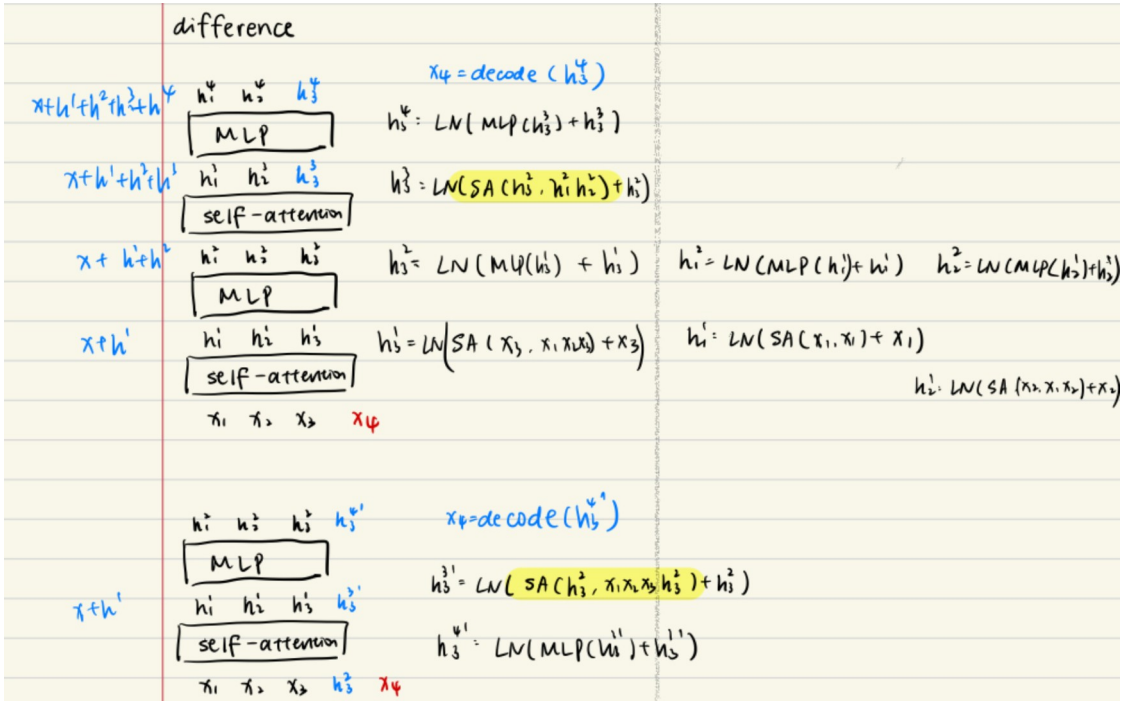


Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. arXiv preprint arXiv:1807.03819, 2018.

CoTFormer

- What is the connection between constructing deeper Transformers and employing CoT?
  - both settings use 2 transformer blocks for 1 forward pass for x_4

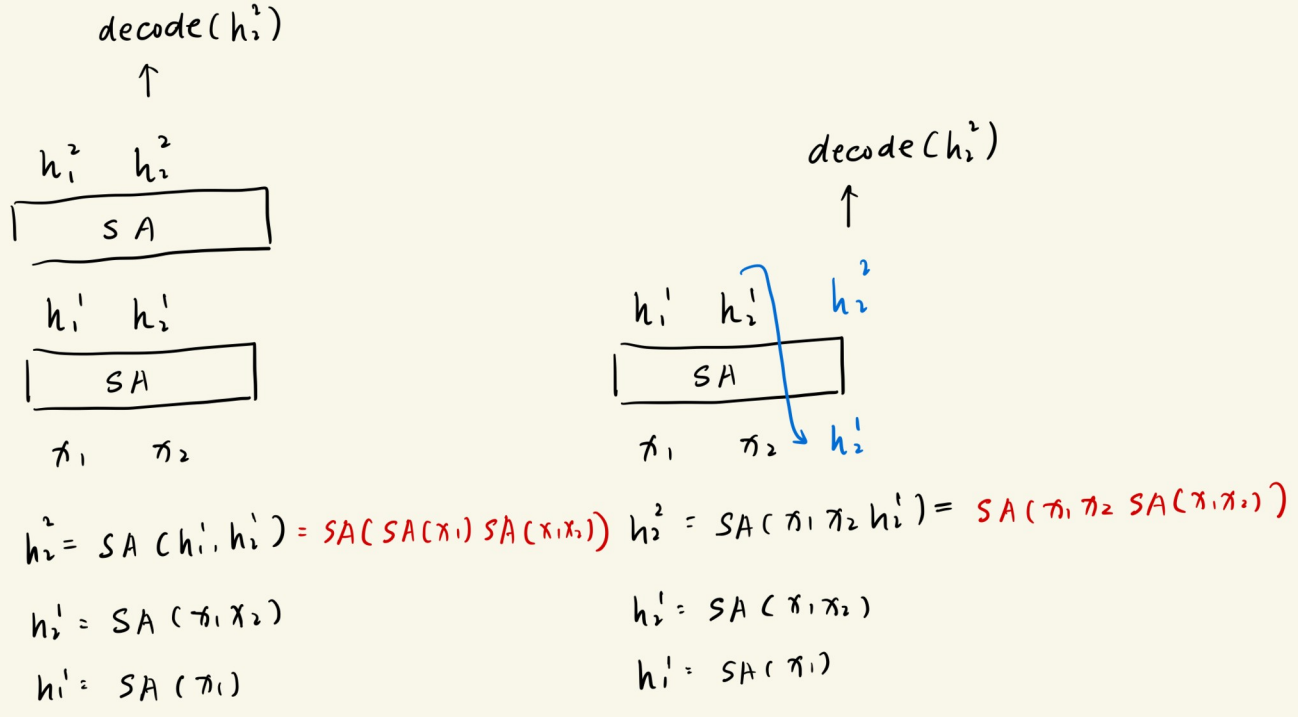




Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. Cotformer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference. arXiv preprint arXiv:2310.10845, 2023. (ICLR 2025)

CoTFormer

- What is the connection between constructing deeper Transformers and employing CoT?
  - both settings use 2 transformer blocks for 1 forward pass for x_4
  - if we only consider attention layer



in two token settings, seem to be equivalent regarding expressivity, but have not proved this

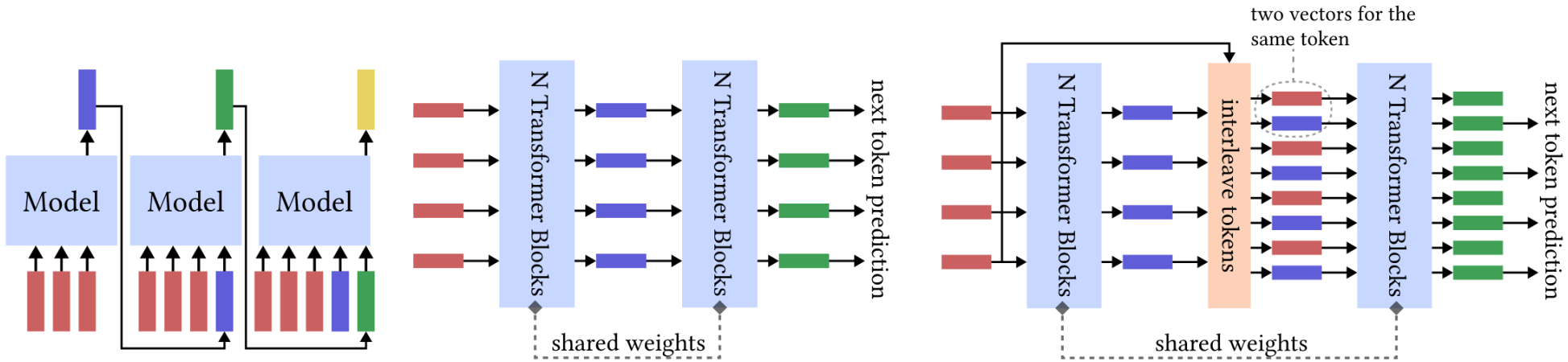
what about in multiple token settings?

what about considering layer norm, residual and more layers?

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. Cotformer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference. arXiv preprint arXiv:2310.10845, 2023. (ICLR 2025)

CoTFormer

- What is the connection between constructing deeper Transformers and employing CoT?
  - this paper says: there is a distinction between CoT and conventional weight tying.
  - More particularly, when applying CoT, the attention mechanism can access previous intermediary tokens whereas in the weight tying such access is not granted.
- CoTFormer interleaves old and new representations in between each block
  - context length grows linearly with #repeats



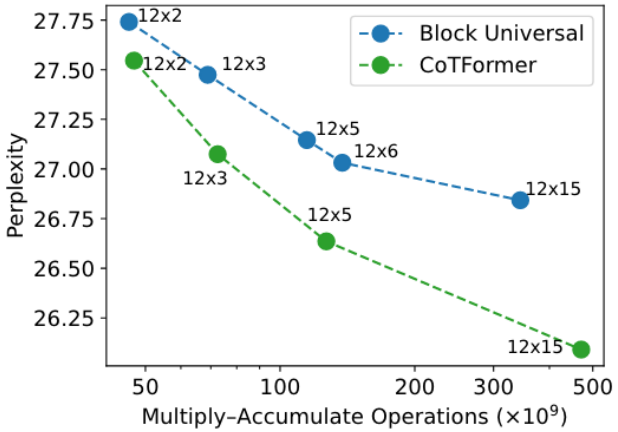
(a) CoT (b) Block Universal Transformer (c) CoTFormer

CoTFormer

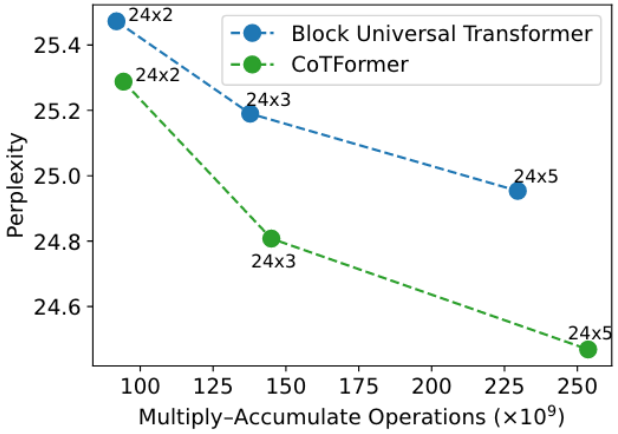
- Experiment Results
  - lower PPL than universal Transformer
  - better performance-computation trade-off

Table 1: **Performance of CoTFormer, Block Universal Transformer and Standard Transformers on OpenWebText2**. The mean perplexity of 3 runs is reported with the standard error of the mean in parenthesis. It can be seen that CoTFormers clearly outperforms Block Universal Transformers. The best perplexity for a given n_{layer}xn_{repeat} combination is marked in bold.

Model	Base Layers (n_{layer})	n_{repeat}		
		2	3	5
Standard	12	28.39 (0.01)		
Block Universal Transformer	12	27.74(0.01)	27.47(0.01)	27.15(0.02)
CoTFormer	12	**27.55(0.02)**	**27.07(0.01)**	**26.64(0.04)**
Standard	24	25.93 (0.02)		
Block Universal Transformer	24	25.47(0.01)	25.19(0.03)	24.95(0.01)
CoTFormer	24	**25.28(0.00)**	**24.85(0.04)**	**24.48(0.03)**
Standard	48	24.17 (0.00)		



(a) $n_{\text{layer}} = 12$ (x-axis is in log scale)



(b) $n_{\text{layer}} = 24$

Figure 2: **Comparison of Block Universal Transformer and CoTFormer in terms of accuracy-computation tradeoff.** It can be clearly seen that at both $n_{\text{layer}} = 12$ and $n_{\text{layer}} = 24$, CoTFormers are closer to the Pareto frontier. The gap widens with larger number of repeats, suggesting better scaling properties of CoTFormers.

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. Cotformer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference. arXiv preprint arXiv:2310.10845, 2023. (ICLR 2025)

Since Universal Transformers...

1. **The Rise of Pre/Loop/Coda Structure**
   - input encoding (Prelude); iterative reasoning (Loop blocks); output decoding (Coda)
2. Pre-iteration Input and Hidden State Management
3. The Decline of Depth Embeddings
4. Pre-iteration Input and Hidden State Management
5. Simplifications of Dynamic Stopping Mechanisms

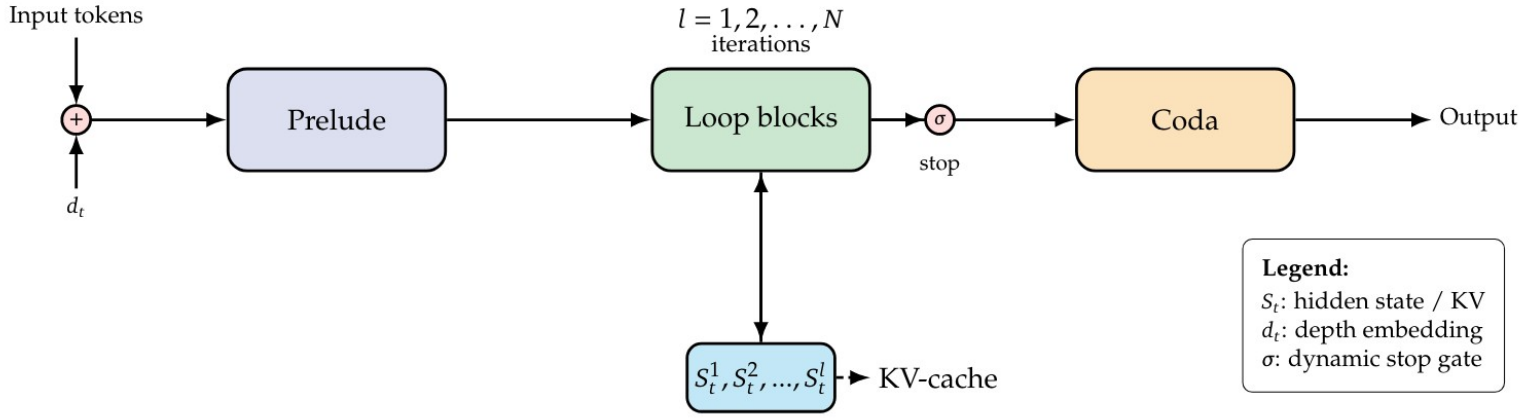


Figure 4. Conceptual diagram of a Pre/Loop/Coda architecture with per-iteration input x_t, hidden state S_t (KV-cache), depth embedding d_t, and a dynamic-stop gate.

Since Universal Transformers...

1. The Rise of Pre/Loop/Coda Structure
2. Pre-iteration Input and Hidden State Management
3. The Decline of Depth Embeddings
4. Simplifications of Dynamic Stopping Mechanisms
   - ACT (cumulative probability) → trained router → early exit based on change magnitude ($\max_t \Delta h < \epsilon$) → fixed iterations → fixed-point

Architecture	**Pre/Loop/Coda**	**Per-iter input** x_t	**Hidden state** S_t	**Dynamic stop**	**Depth-emb** d_t
Universal Transformer[22]	No	$x_t^{l-1} + d_t$	standard unroll	ACT, $\sum_t p_t > \tau$	sinusoidal d_t
CoTFormer[71]	No	S_t^{l-1}, x_t^{l-1}	standard unroll	MoR router g_i	learnable d_t
Recursive Transformer[2]	Optional	x_t^{l-1}	share/refill $\hat{h}$	early-exit, $\max_t \Delta h < \varepsilon$	none
AlgoFormer[29]	Yes	x_t^{l-1}	standard roll	fixed	none
Recurrent-Depth[31]	Yes	x_t^1, x_t^{l-1}	modulo (t mod r) reuse	fixed-point iteration	tried, dropped

What about Scaling up?

Recurrent Depth: the first attempt to scaling up

- 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

Scaling up test-time compute with latent reasoning: A recurrent depth approach. arXiv preprint arXiv:2502.05171, 2025.

Recurrent Depth: the first attempt to scaling up

- 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

Scaling up test-time compute with latent reasoning: A recurrent depth approach. arXiv preprint arXiv:2502.05171, 2025.

Scaling Looped Language Models

- pre-train looped language models (LoopLM), scaling to **7.7T** tokens, **1.4B / 2.6B**
- yield comparable performance to 4B / 12B LLMs

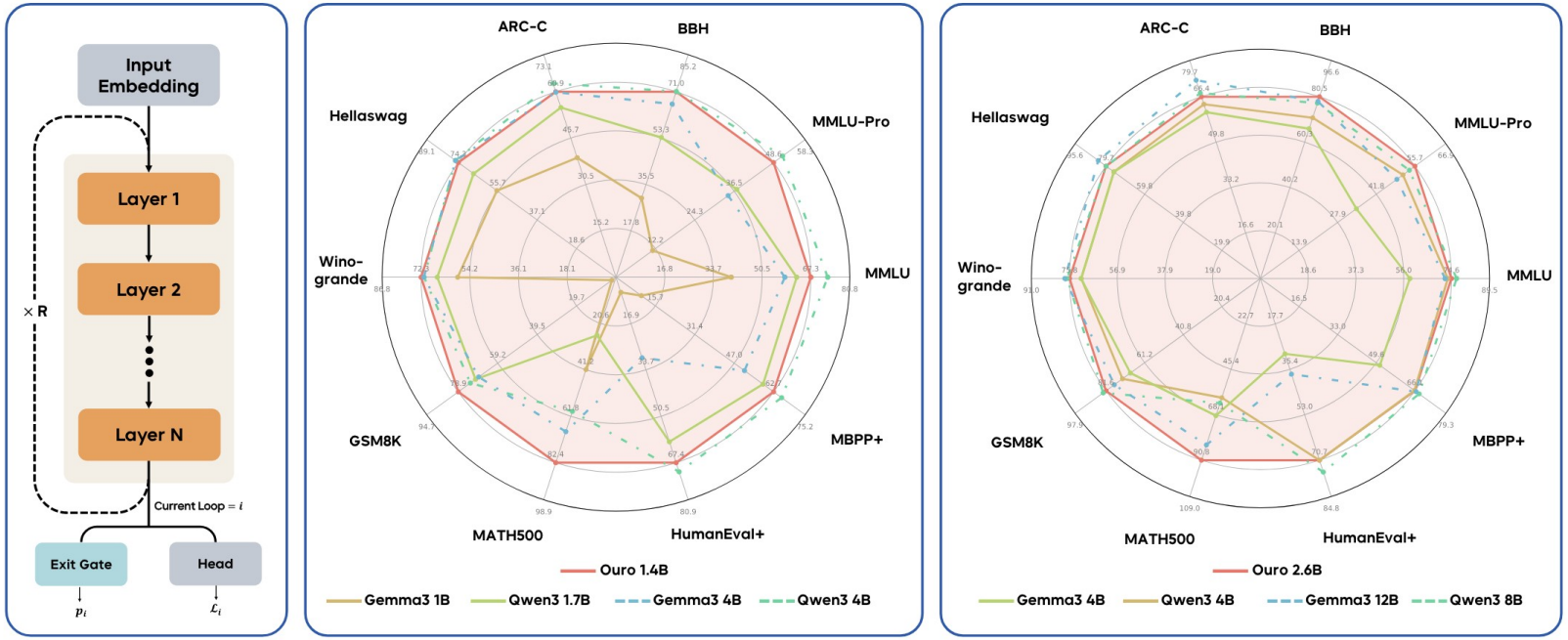
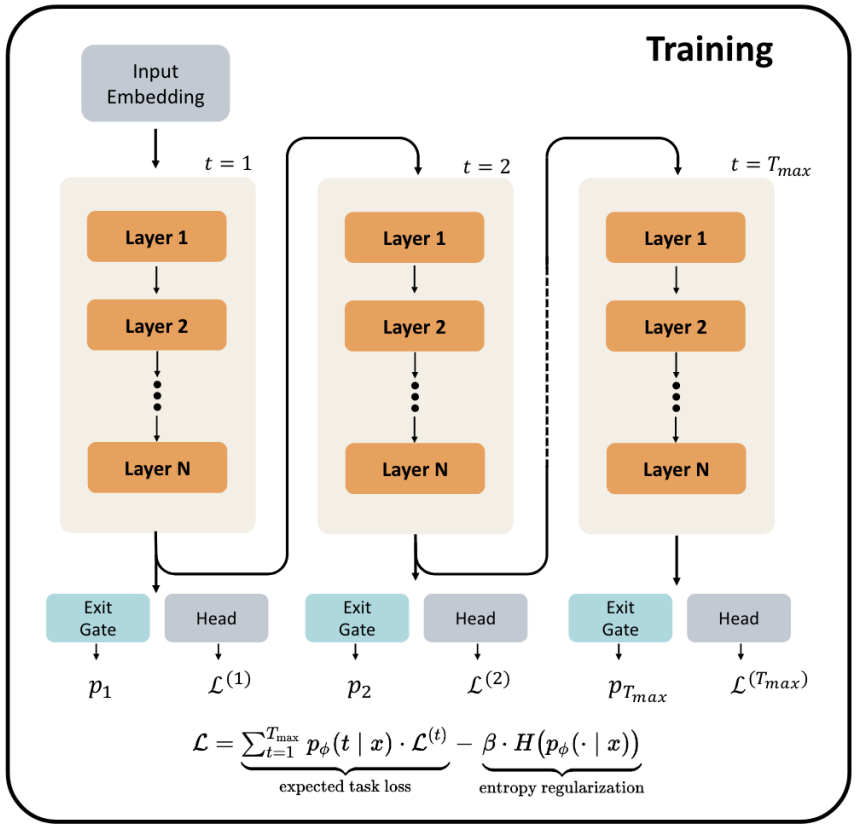


Figure 1 Ouro Looped Language Model performance. (Left) The parameter-shared looped architecture. (Middle & Right) Radar plots comparing the Ouro 1.4B and 2.6B models, both with 4 recurrent steps (red), against individual transformer baselines. Our models demonstrate strong performance comparable to or exceeding much larger baselines.

Rui-Jie Zhu, et al. Scaling Latent Reasoning via Looped Language Models. arXiv preprint arXiv: 2510.25741, 2025.

Scaling Looped Language Models: Architecture

- pretraining with loop, training with full depth

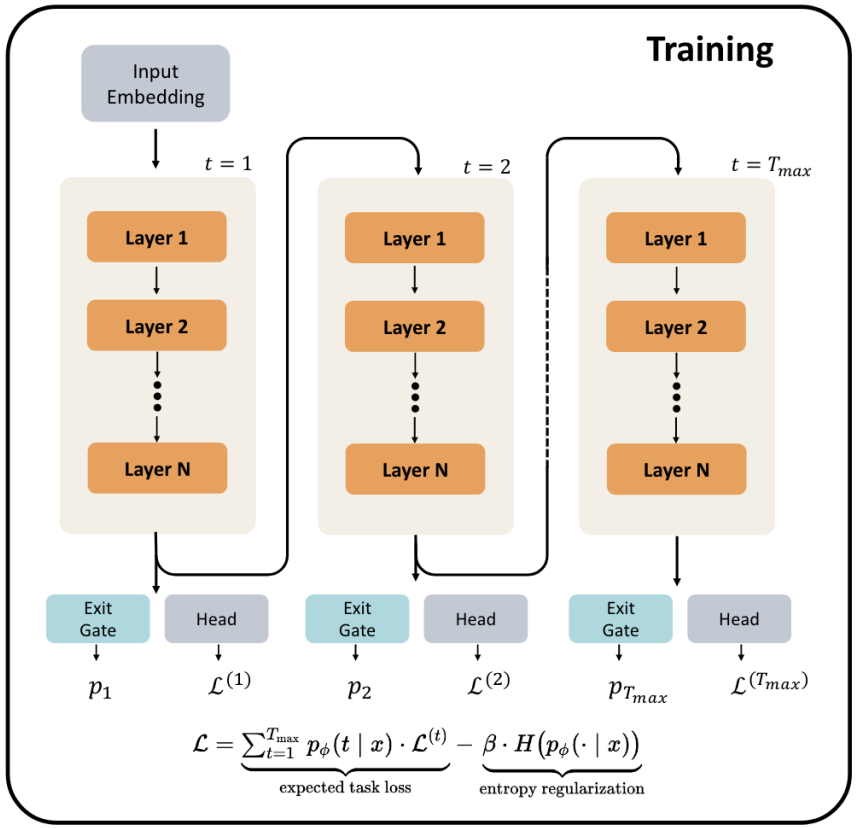


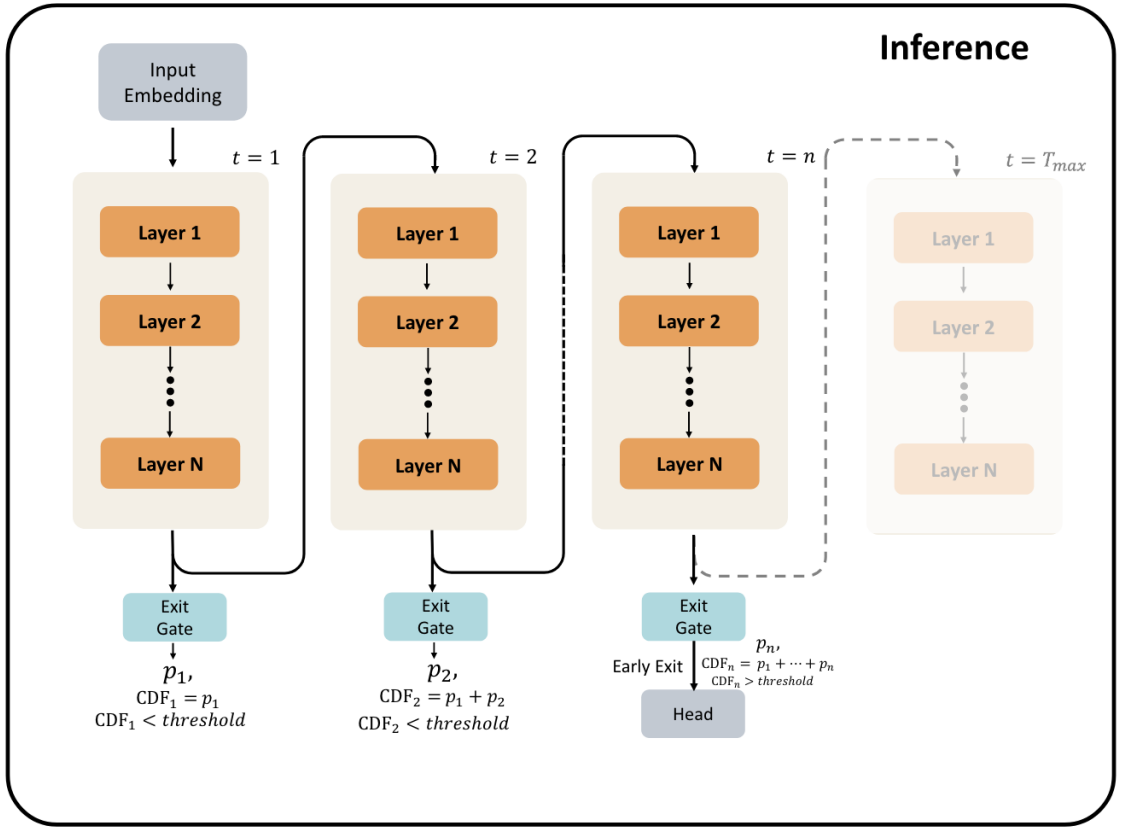
- training with full depth, calculate CE loss of each loop
- this design is questionable (as also acknowledged by the author)
  - objective: do you want to use more depth or not?, currently all loops are fitting the same distribution
  - mixture of recursion demonstrates the feasibility
- adding an entropy regularization as prior for "load balance"
  - uniform prior for all loops --- experiments show uniform is the best

Rui-Jie Zhu, et al. Scaling Latent Reasoning via Looped Language Models. arXiv preprint arXiv: 2510.25741, 2025.

Scaling Looped Language Models: Architecture

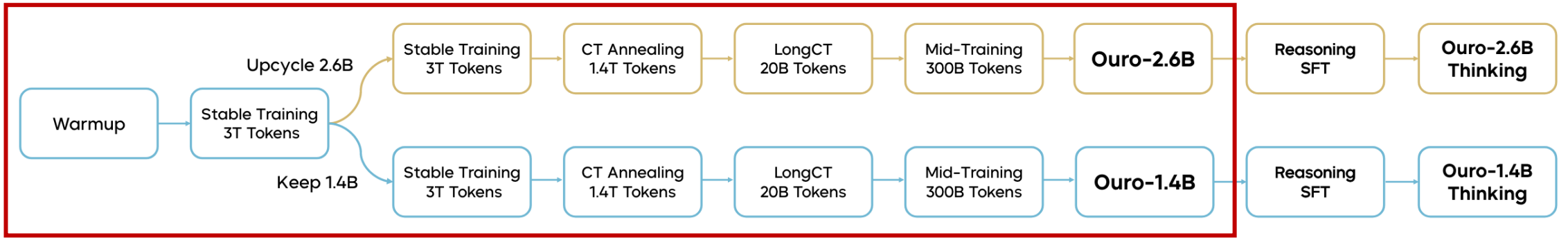
- pretraining with loop, training with full depth
- inference with early exit: based on cumulative probability





Rui-Jie Zhu, et al. Scaling Latent Reasoning via Looped Language Models. arXiv preprint arXiv: 2510.25741, 2025.

Scaling Looped Language Models: Training



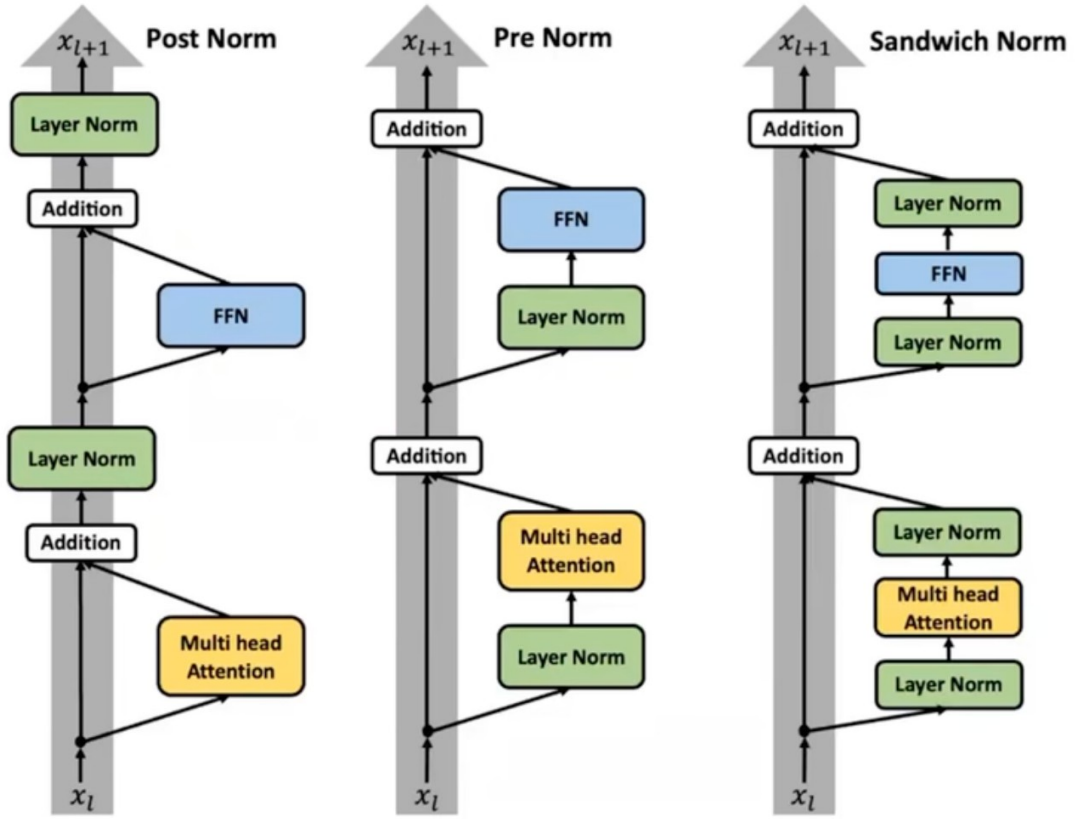
- **Stage 1a: Pre-training Phase I (Exploration)**
  - initialize pretraining with 8 recurrent steps
- **Stage 1b: Pre-training Phase II with Stability-Driven Upcycling** (due to unstability)
  - 1.4B Ouro: uses the original 24 pre-trained layers with 4 recurrent steps
  - 2.6B Ouro: upcycles 24 layers to 48 via layer duplication with 4 recurrent steps
- **Stage 2: Continue Training (CT) Annealing**
- **Stage 3: LongCT**
- **Stage 4: Mid-training**

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

Rui-Jie Zhu, et al. Scaling Latent Reasoning via Looped Language Models. arXiv preprint arXiv: 2510.25741, 2025.

Scaling Looped Language Models: Layer norm

- depth problem: sandwich norm



Sandwich Norm replace Pre–Norm

Rui-Jie Zhu, et al. Scaling Latent Reasoning via Looped Language Models. arXiv preprint arXiv: 2510.25741, 2025.

Scaling Looped Language Models: Performance

- performance of base models
- show better performance in multi-hop reasoning problems

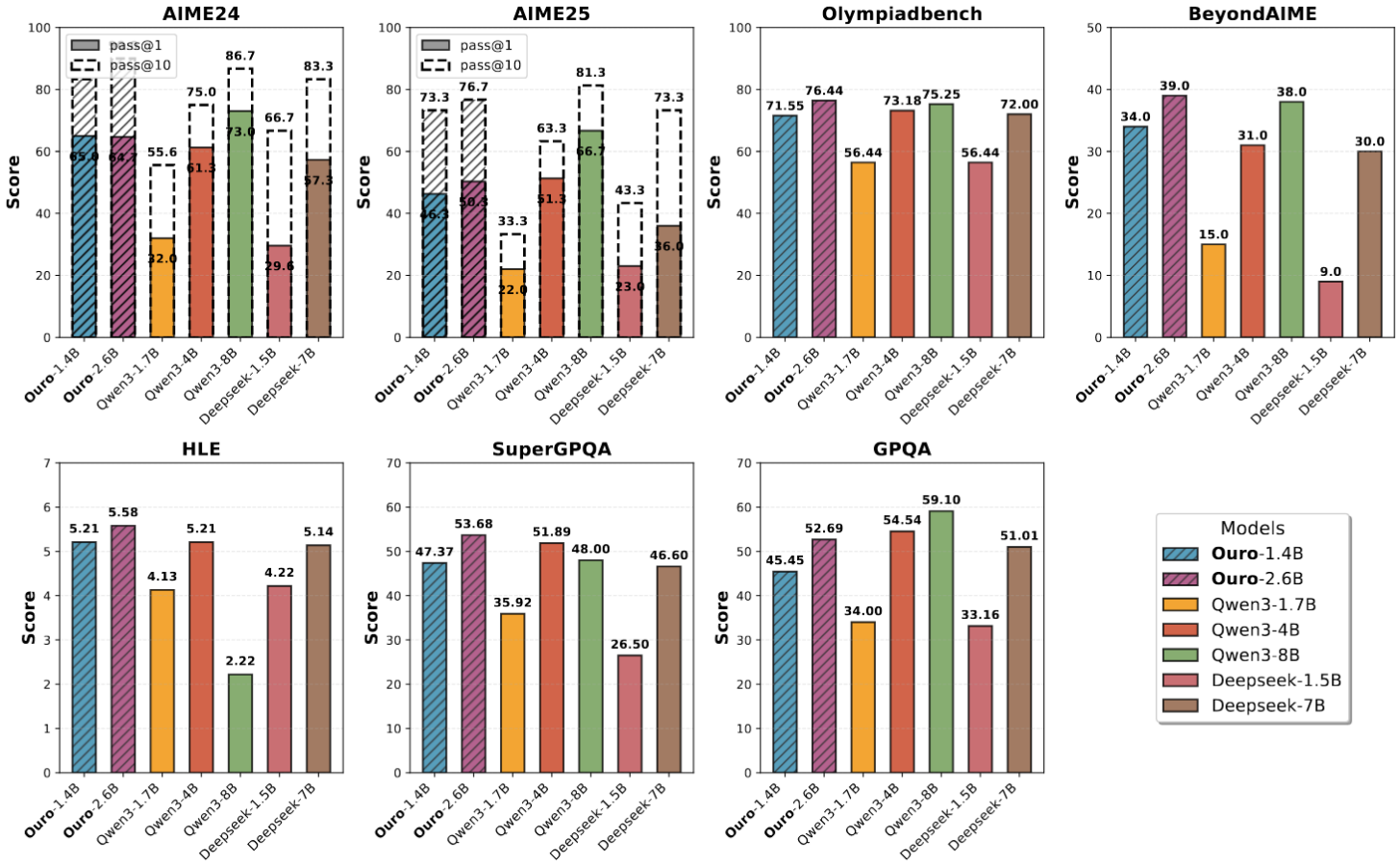
	Gemma3 1B	Llama3.2 1.2B	Qwen2.5 1.5B	Qwen3 1.7B	Qwen2.5 3B	Llama3.2 3B	Qwen3 4B	Gemma3 4B	Ouro 1.4B R4
Architecture	Dense	Dense	Dense	Dense	Dense	Dense	Dense	Dense	LoopLM
# Params	1.0B	1.0B	1.5B	1.7B	3.0B	3.0B	4.0B	4.0B	1.4B
# Tokens	2T	9T	18T	36T	18T	9T	36T	4T	7.7T
General Tasks									
MMLU	39.85	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

Rui-Jie Zhu, et al. Scaling Latent Reasoning via Looped Language Models. arXiv preprint arXiv: 2510.25741, 2025.

Scaling Looped Language Models: Performance

- performance of reasoning models: did not do RL



Rui-Jie Zhu, et al. Scaling Latent Reasoning via Looped Language Models. arXiv preprint arXiv: 2510.25741, 2025.

Why does LoopLM work?

- Two hypothesis:
  - **increased knowledge capacity?** – LoopLM might be able to memorize more facts even with the same number of parameters
  - **better knowledge extraction / composition?** – LoopLM might not store more facts but might use the knowledge in the parameters more effectively through repeated reasoning.

Q1: Does looping increase knowledge capacity?

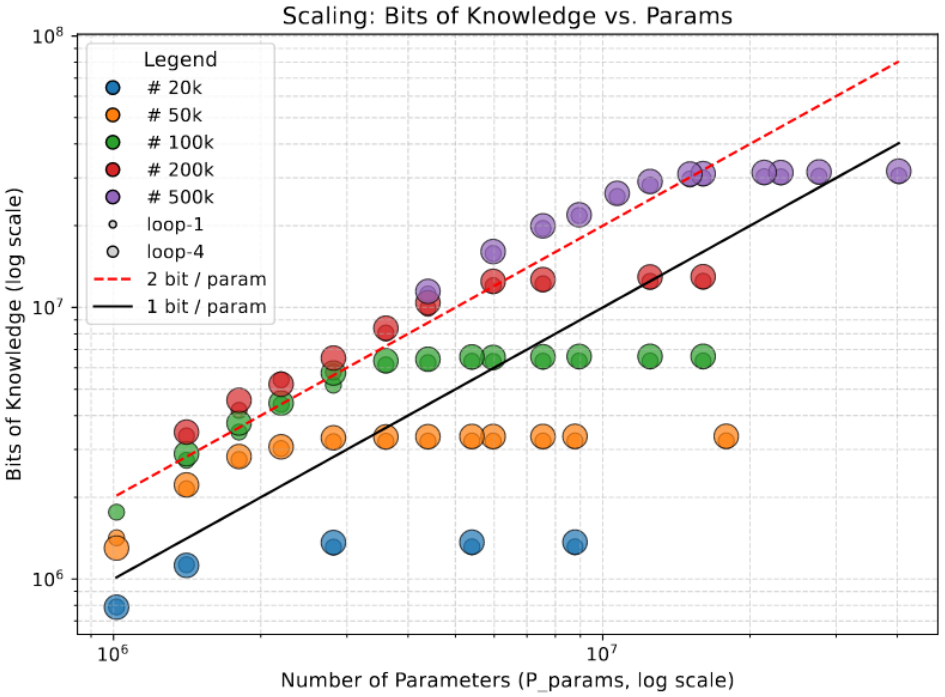
Capo Task

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.

- **Settings:**
  - Dataset: synthetic biographies (following Phys of LLMs)
  - Each biography contains name + 5 attributes: gender, birthdate, university, major, employer
  - The names and the attributes are randomly selected from a pre-defined set and combined into natural language via templates
- **Knowledge capacity metric**: sum of CE loss per attribute token → convert to bits
  - Based on random generation, we have an information-theoretic lower bound for the model in the minimum bits required to encode all the names and attributes (refer to Phys of LLMs)

Q1: Does looping increase knowledge capacity?

- **Results:**
  - looping does not increase knowledge capacity nor improve capacity scaling
  - **the number of parameters itself** can be seen as a direct indicator of knowledge capacity, and **merely increasing looping does not help enhance knowledge capacity**.



Q2: Does looping enhance knowledge manipulation?

- **Arithmetic operations**
  - `<bos> + * a b c <eos>` requires the model to directly output (a * b) + c mod 23
  - no CoT, answer directly
  - denote expression length as L: train on $\ell \in [1, L]$, teat on maximum length L

	$L = 10$	$L = 16$	$L = 24$
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

LoopLM has a better inductive bias towards knowledge manipulation

Q2: Does looping enhance knowledge manipulation?

- **Multi-hop QA**
  - 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

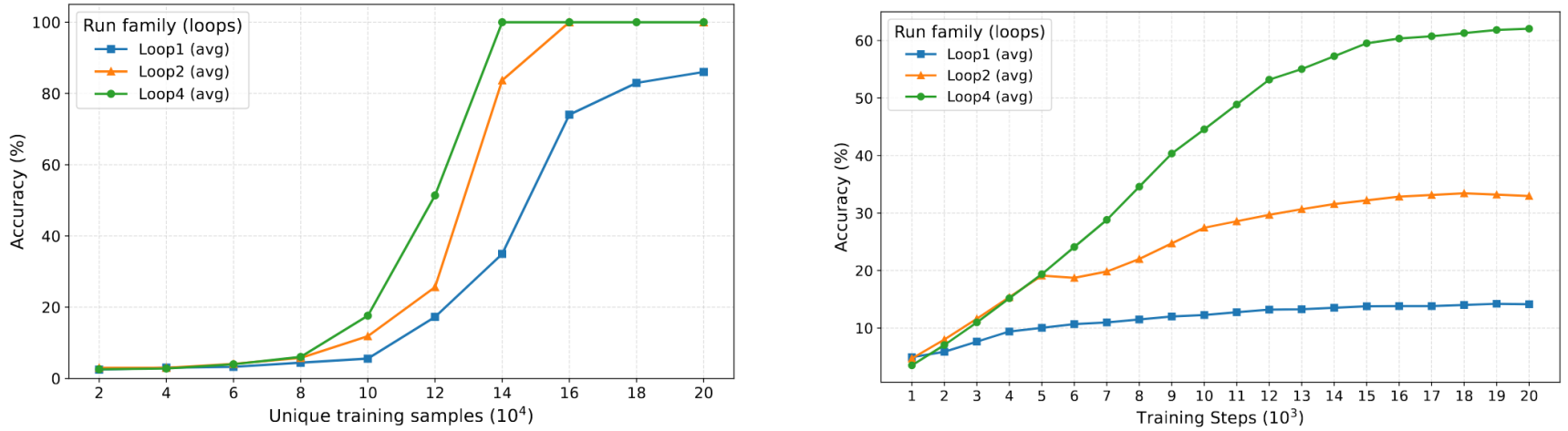


Figure 7 We trained LoopLMs and standard transformer baselines with the same parameters on Multi-hop QA tasks. To investigate the *sample efficiency* of LoopLMs, we **vary the number of unique training samples** (from 2.5% to 25% all possible QA pairs) for models with different loops. We compare the final performance using **the same compute budget in total training tokens**. **Left.** As shown, models with more loops requires fewer samples to learn the 3-hop QA task. **Right.** As an example, we train with 15% of all possible QA pairs (12000 unique samples) for 20000 steps with context length 1024 and batch size 2048. Models with more loops learn faster and achieve better performance comparing with models without loops.

Takeaways for Loop Transformers

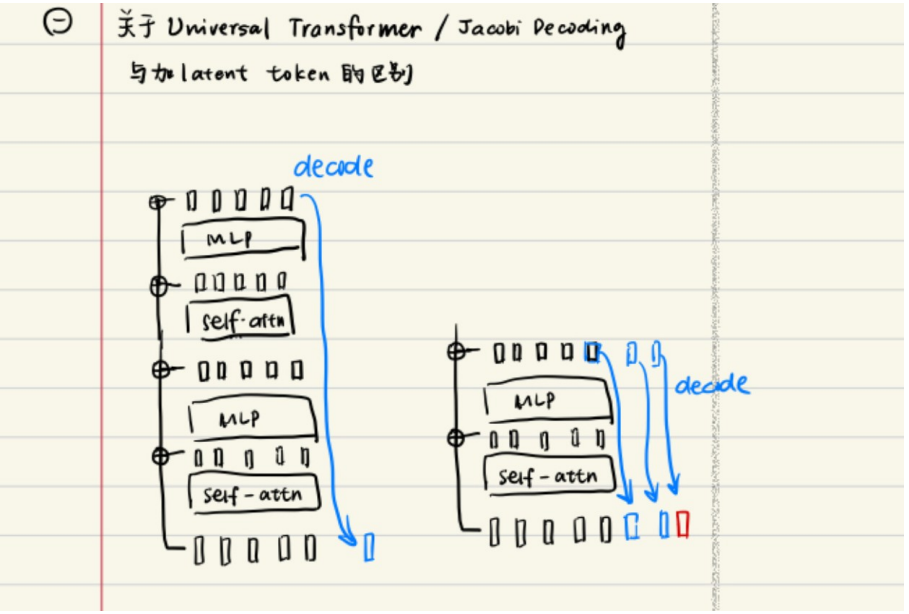
- Universal Transformer
- CoTFormer
- Scaling up looped transformer
  - Recurrent Depth
  - Ouro

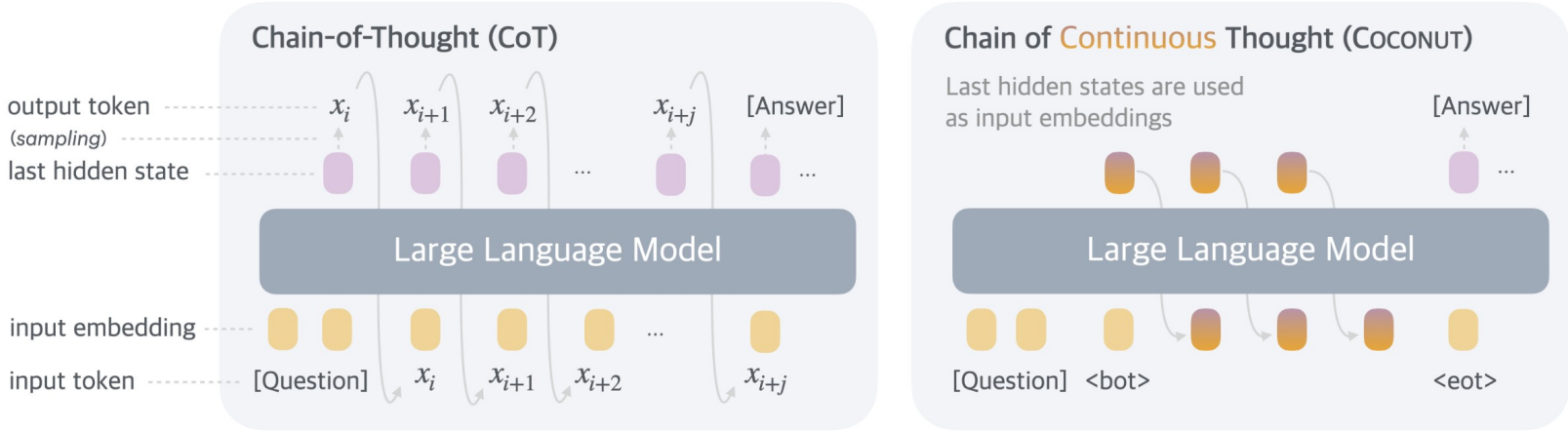
Outline

- Preliminary: Latent CoT
- Latent Reasoning
  - Activation-based Recurrent Methods
    - Loop/Universal Transformer Recurrence: Universal Transformer, CoTFormer, Recursive Transformer
    - **Activation with Explicit Hidden-State Feedback: Coconut, CODI, [xxx]-Tokens**
    - Training Strategies for Recurrent Reasoning: MIDAS,
  - Temporal Hidden-state Methods
    - Linear-State Recurrence
    - Gradient-State Recurrence
    - Training-induced Hidden-state Conversion
- Mechanical Interpretability

Activation with Explicit Hidden-State Feedback

- loop-based architectures: rerunning the same set of layers
- a distinct family of models feeds hidden states back into the input stream between iterations

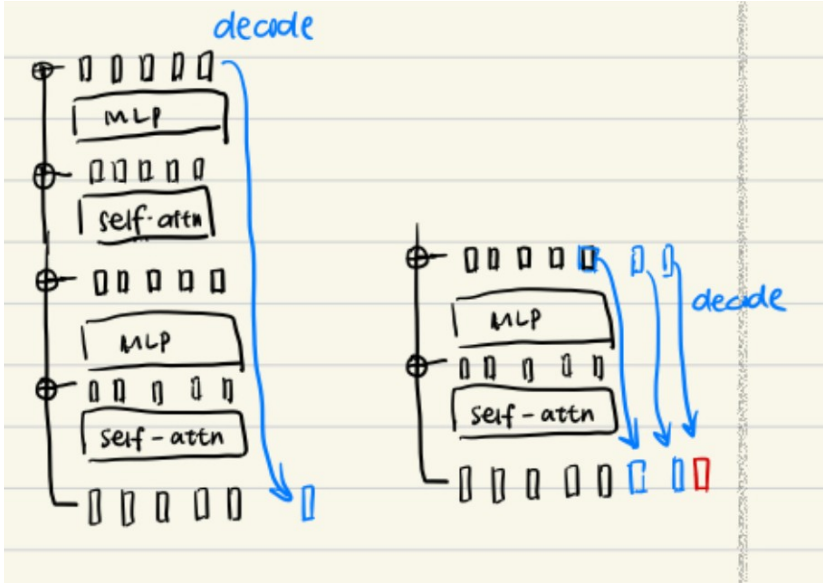




Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Activation with Explicit Hidden-State Feedback

- The core principle unifying these methods is the creation of *implicit loops*, by:
  - **continuous recurrence:** feeding activations back into the model
  - **compressed states:** compressing multi-step reasoning into iteratively-processed representations
  - **expanded iterations:** extending the effective computation depth through strategic token insertion



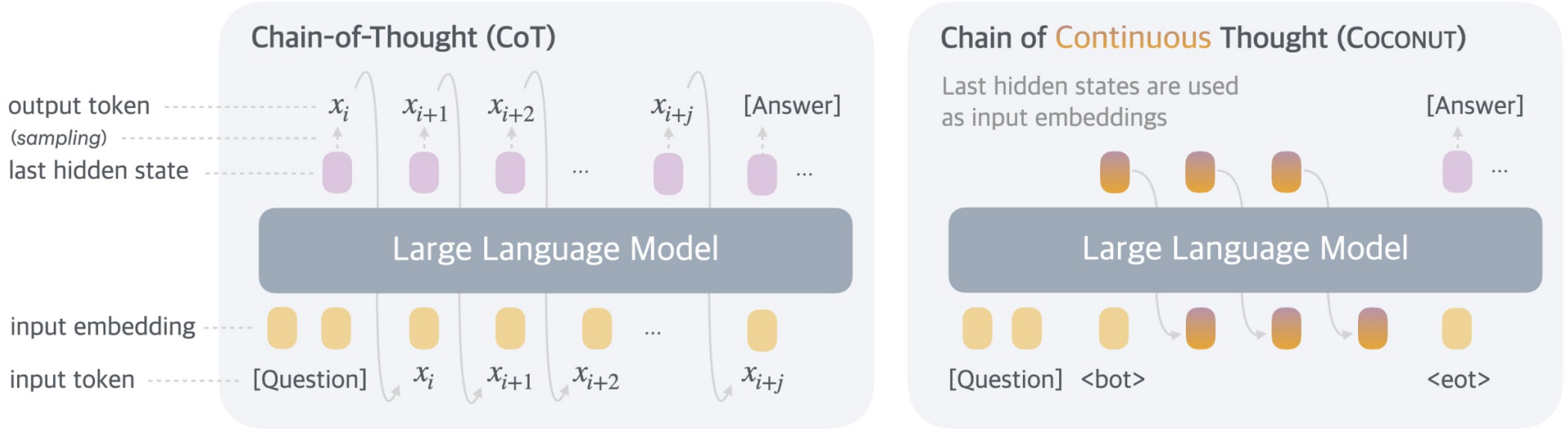
Coconut: Training Large Language Models to Reason in a Continuous Latent Space

- Motivation
  - Language space may not always be optimal for reasoning
    - **Human language is optimized for communication rather than reasoning**: Neuroimaging studies have consistently shown that the language network – a set of brain regions responsible for language comprehension and production – remains largely inactive during various reasoning tasks
    - **For LLMs, "the amount of reasoning" required for each particular reasoning token varies greatly**, yet current LLM architectures allocate nearly the same computing budget for predicting every token

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Coconut: **Architecture**

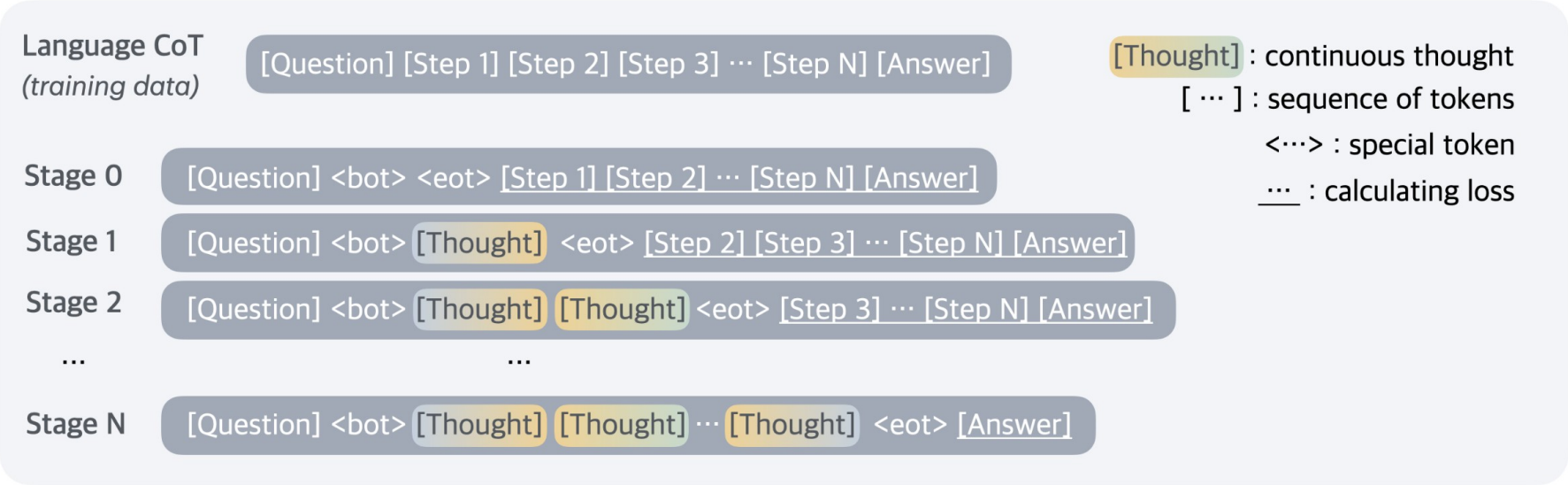
- the LLM switches between the “language mode” and “latent mode”
- special tokens are employed to mark the beginning and end of the latent thought mode, respectively



Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Coconut: **Training**

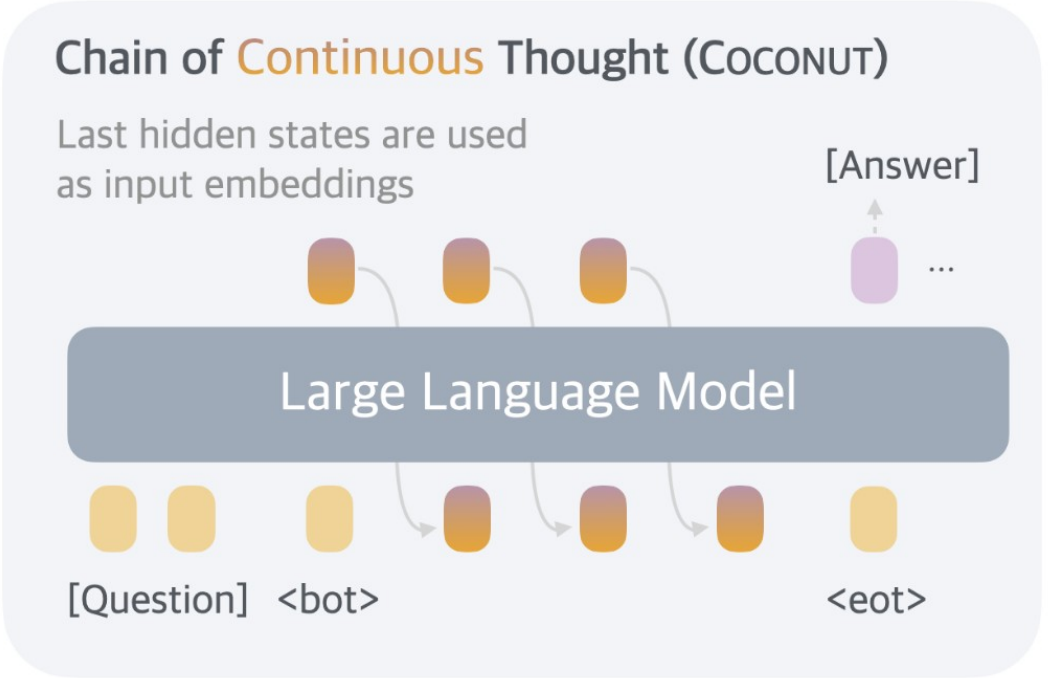
- **supervise only language tokens**: mask loss on questions and continuous tokens
- **curriculum training**:
  - in the initial stage, the model is trained on regular CoT instances.
  - at the k-th stage, the first k reasoning steps in the CoT are replaced with $k \times c$ continuous thoughts1, where c is a hyperparameter



Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Coconut: **Inference**

- do not change much from standard LM
- the challenging part is **when to switch back to the discrete token**
  - train a binary classifier on latent thoughts
  - always pad the latent thoughts to a constant length



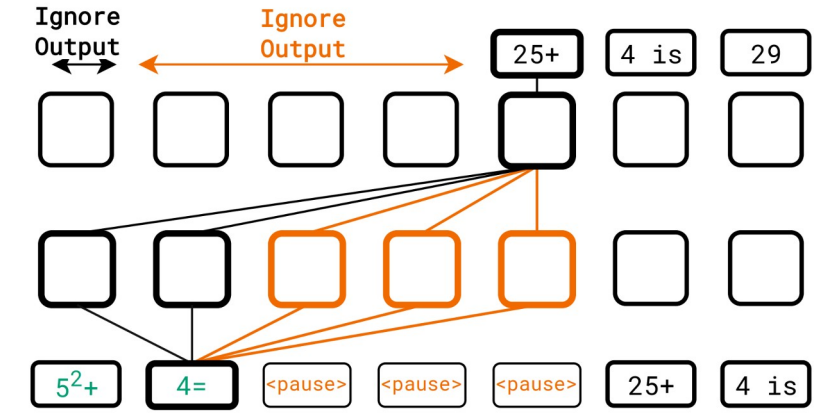
Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Coconut: **Results**

- Math: **2 latent tokens** for each reasoning step, go through **3 stages** besides the initial stage
- Logical: **1 latent token** for each reasoning, go through **6 stages** besides the initial stage
- use pretrained GPT-2 as the base model

Method	GSM8k		Pronto
	Acc. (%)	# Tokens	Acc. (%)
CoT	42.9 ±0.2	25.0	98.8 ±0.8
No-CoT	16.5 ±0.5	2.2	93.8 ±0.7
iCoT	30.0*	2.2	99.8 ±0.3
Pause Token	16.4 ±1.8	2.2	77.7 ±21.0
COCONUT (Ours)	34.1 ±1.5	8.2	99.8 ±0.2
- *w/o curriculum*	14.4 ±0.8	8.2	52.4 ±0.4
- *w/o thought*	21.6 ±0.5	2.3	99.9 ±0.1
- *pause as thought*	24.1 ±0.7	2.2	100.0 ±0.1

Pause Token: add pause token before CoT



(b) Pause-inference and finetuning

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Coconut: **Results**

- **Latent reasoning outperforms** language reasoning **in planning-intensive tasks**
- The LLM still **needs guidance** to learn latent reasoning: curriculum learning is necessary
- **Continuous thoughts are efficient representations** of reasoning.

Method	GSM8k		ProntoQA		ProsQA	
	Acc. (%)	# Tokens	Acc. (%)	# Tokens	Acc. (%)	# Toke
CoT	42.9 ±0.2	25.0	98.8 ±0.8	92.5	77.5 ±1.9	49.4
No-CoT	16.5 ±0.5	2.2	93.8 ±0.7	3.0	76.7 ±1.0	8.2
iCoT	30.0*	2.2	99.8 ±0.3	3.0	98.2 ±0.3	8.2
Pause Token	16.4 ±1.8	2.2	77.7 ±21.0	3.0	75.9 ±0.7	8.2
COCONUT (Ours)	34.1 ±1.5	8.2	99.8 ±0.2	9.0	97.0 ±0.3	14.2
- *w/o curriculum*	14.4 ±0.8	8.2	52.4 ±0.4	9.0	76.1 ±0.2	14.2
- *w/o thought*	21.6 ±0.5	2.3	99.9 ±0.1	3.0	95.5 ±1.1	8.2
- *pause as thought*	24.1 ±0.7	2.2	100.0 ±0.1	3.0	96.6 ±0.8	8.2

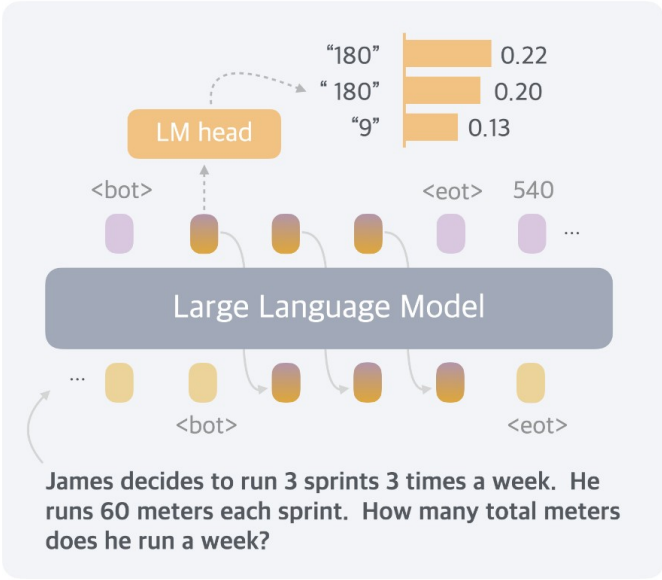


Figure 4 A case study where we decode the continuous thought into language tokens.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Are the

last-layer hidden-states of the previous token

the optimal choice for representing continuous tokens?

LLM Pretraining with Continuous Concepts

Jihoon Tack[1,2,†], **Jack Lanchantin**[1], **Jane Yu**[1], **Andrew Cohen**[1], **Ilia Kulikov**[1], **Janice Lan**[1], **Shibo Hao**[1,3,†], **Yuandong Tian**[1], **Jason Weston**[1], **Xian Li**[1]

[1]FAIR at Meta, [2]KAIST, [3]UC San Diego
[†]Work done during an internship at Meta

Next token prediction has been the standard training objective used in large language model pretraining. Representations are learned as a result of optimizing for token-level perplexity. We propose Continuous Concept Mixing (CoCoMix), a novel pretraining framework that combines discrete next token prediction with continuous concepts. Specifically, CoCoMix predicts "continuous concepts" learned from a pretrained sparse autoencoder and mixes them into the model's hidden state by interleaving with token hidden representations. Through experiments on multiple benchmarks, including language modeling and downstream reasoning tasks, we show that CoCoMix is more sample efficient and consistently outperforms standard next token prediction, knowledge distillation and inserting pause tokens. We find that combining both concept learning and interleaving in an end-to-end framework is critical to performance gains. Furthermore, CoCoMix enhances interpretability and steerability by allowing direct inspection and modification of the predicted concept, offering a transparent way to guide the model's internal reasoning process.

Date: February 13, 2025
Correspondence: Jihoon Tack at jihoontack@kaist.ac.kr, Xian Li at xianl@meta.com
Code: https://github.com/facebookresearch/RAM/tree/main/projects/cocomix

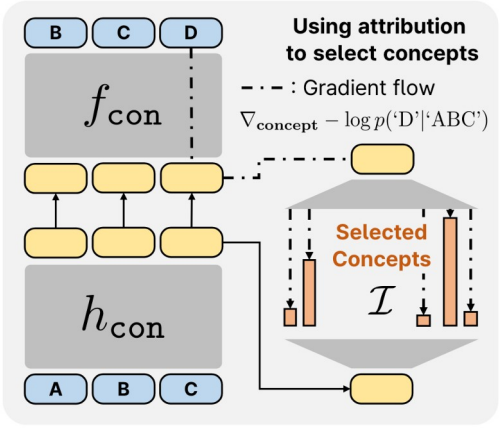
Meta

CoCoMix: LLM Pretraining with Continuous Concepts

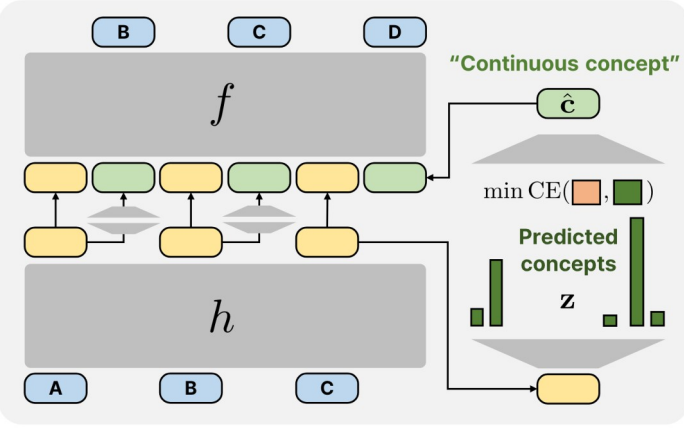
- Motivation
  - **a forward pass can store richer information** than merely predicting the next token, like MTP
  - LLMs inherently **encode high-level concepts and reasoning processes in latent representations**
  - Can we utilize these information better?
- Overview
  - a **pretraining** framework which combines discrete next token prediction with **continuous concepts**
  - CoCoMix predicts **"continuous concepts" learned from a pretrained sparse autoencoder (SAE)** and mixes them into the model's hidden state by interleaving with token hidden representations.

Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, Xian Li. LLM Pretraining with Continuous Concepts. arXiv preprint arXiv: 2502.08524, 2025.

CoCoMix: **Architecture**



Extracting concepts from a pretrained SAE model's hidden state



Learning to predict concepts & Mixing/Interleaving continuous concepts into the hidden state

1. Extracting Concepts from SAEs

linear encoder, linear decoder, TopK activation

model's hidden state at position t: $h_t^{con} \in \mathbb{R}^{d_{con}}$, linear encoder E: $\mathbb{R}^{d_{con}} \to \mathbb{R}^{C}$, and a linear decoder D: $\mathbb{R}^{C} \to \mathbb{R}^{d_{con}}$

$$\mathbf{c}_t^{\text{pre}} = E\left(\mathbf{h}_t^{\text{con}}\right), \quad \mathbf{c}_t = \text{TopK}\left(\mathbf{c}_t^{\text{pre}}\right), \quad \widehat{\mathbf{h}}_t^{\text{con}} = D\left(\mathbf{c}_t\right),$$

SAE is trained by minimizing the reconstruction loss: $\|\mathbf{h}_t^{\text{con}} - \widehat{\mathbf{h}}_t^{\text{con}}\|_2^2$

Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, Xian Li. LLM Pretraining with Continuous Concepts. arXiv preprint arXiv: 2502.08524, 2025.

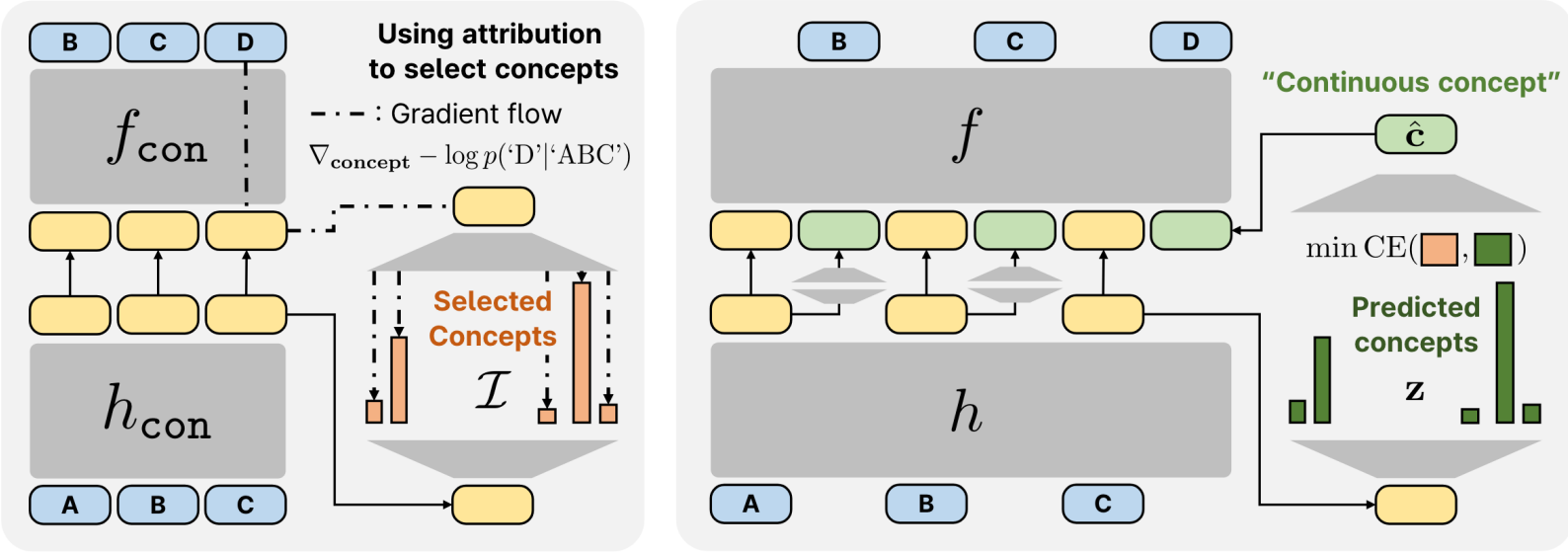
CoCoMix: **Architecture**

1. Extracting concepts from SAEs

Target concept selection using attribution

multiply pre-activation c_t^{pre} with **attribution score** $\mathbf{a}_t = \mathbf{c}_t^{\text{pre}} \odot \nabla_{\mathbf{c}_t} - \log f_{\text{con}}\left(x_{t+1} | D(\mathbf{c}_t), \mathbf{h}_{<t}\right)$

select Top K_{attr} values of a_t



Extracting concepts from a pretrained SAE model's hidden state

Learning to predict concepts & Mixing/Interleaving continuous concepts into the hidden state

CoCoMix: **Architecture**

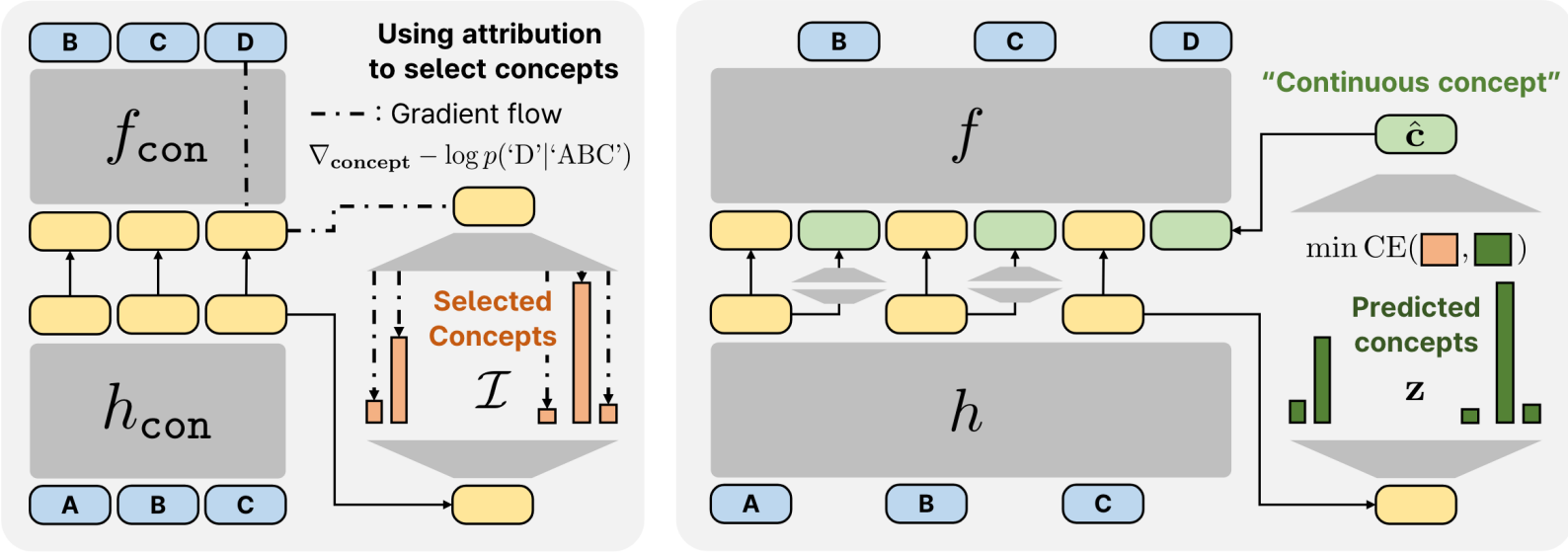
2. Predicting the selected concepts

$$\mathbf{a}_t = \mathbf{c}_t^{\texttt{pre}} \odot \nabla_{\mathbf{c}_t} - \log f_{\texttt{con}}\left(x_{t+1} | D(\mathbf{c}_t), \mathbf{h}_{<t}\right)$$

Let $I = \{i_1, \ldots, i_{K_{attr}}\}$ be the set of indices corresponding to the top K_{attr} values of a_t.

Then we learn to predict these labels using a linear prediction head M that outputs logit

$z_t = M(h_t) \in \mathbb{R}^C$ by minimizing: $\mathcal{L}_{\text{concept}}(\mathbf{a}_t) = \frac{1}{K_{\text{attr}}} \sum_{i \in \mathcal{I}} \text{CE}\left(\mathbf{z}_t, i\right).$



Extracting concepts from a pretrained SAE model's hidden state

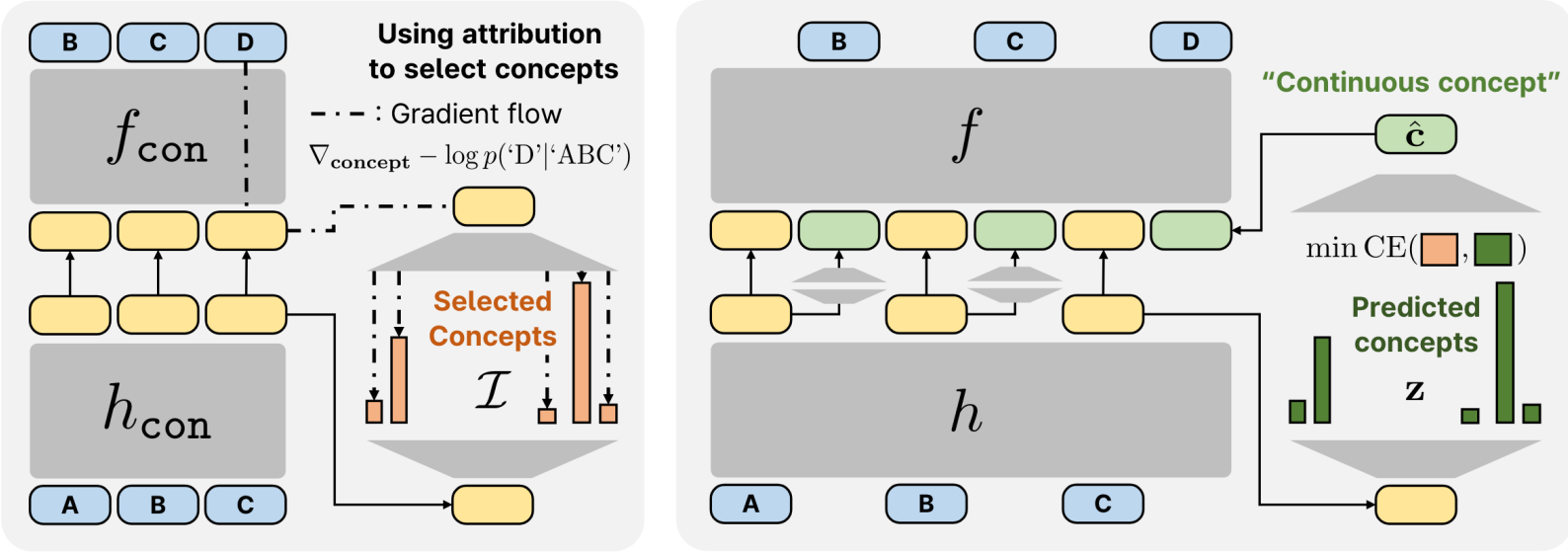
Learning to predict concepts & Mixing/Interleaving continuous concepts into the hidden state

CoCoMix: **Architecture**

3. Mixing continuous concepts with token embeddings

For a given concept prediction logit z_t, we sparsify the logit using TopK to predict the concept, then **compress them into a continuous concept vector** $\hat{c}_t = W\,\mathrm{TopK}(z_t) + b$, $\hat{c}_t \in \mathbb{R}^d$.

We then append $\hat{c}_t$ to the model's hidden sequence as $(h_1, \hat{c}_1, \cdots, h_t, \hat{c}_t)$.



Extracting concepts from a pretrained SAE model's hidden state

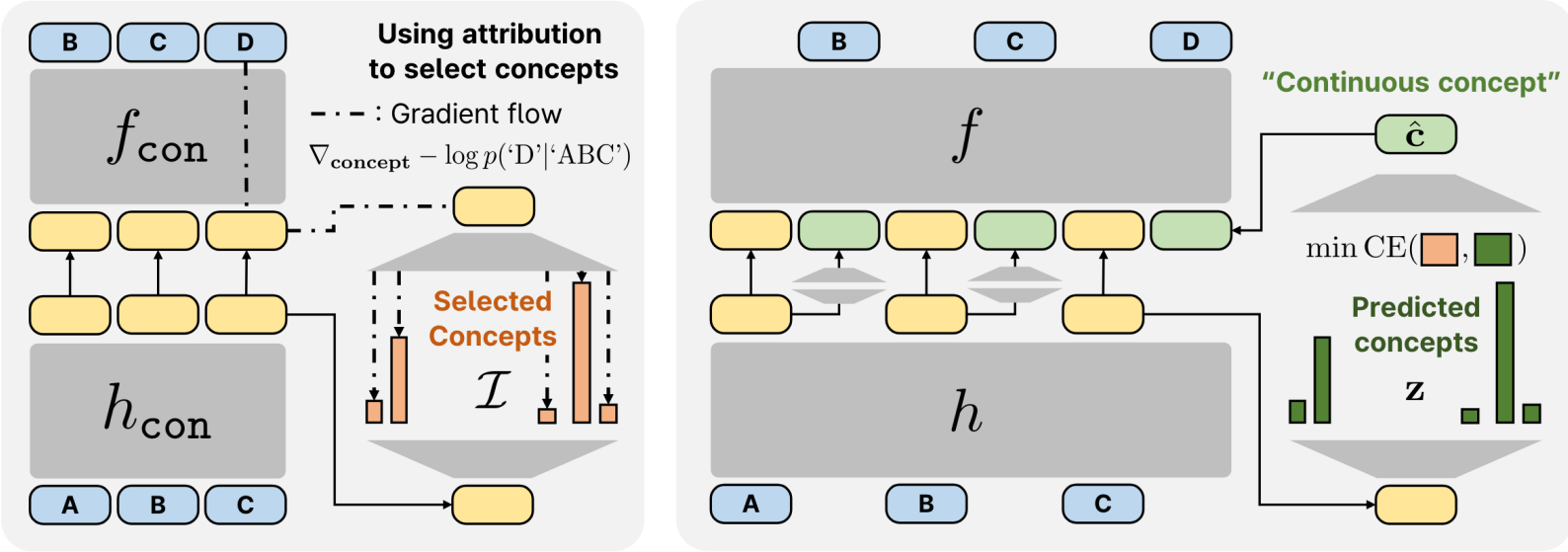
Learning to predict concepts & Mixing/Interleaving continuous concepts into the hidden state

CoCoMix: **Architecture**

4. Training objective

The final training objective combines the standard NTP loss and the concept prediction term

$$\sum_{t=1}^{T-1} -\log f\left(x_{t+1} \mid \mathbf{h}_{\leq t}, \hat{\mathbf{c}}_{\leq t}\right) + \lambda \mathcal{L}_{\text{concept}}(\mathbf{a}_t), \qquad \mathcal{L}_{\text{concept}}(\mathbf{a}_t) = \frac{1}{K_{\text{attr}}} \sum_{i \in \mathcal{I}} \text{CE}\left(\mathbf{z}_t,\ i\right).$$



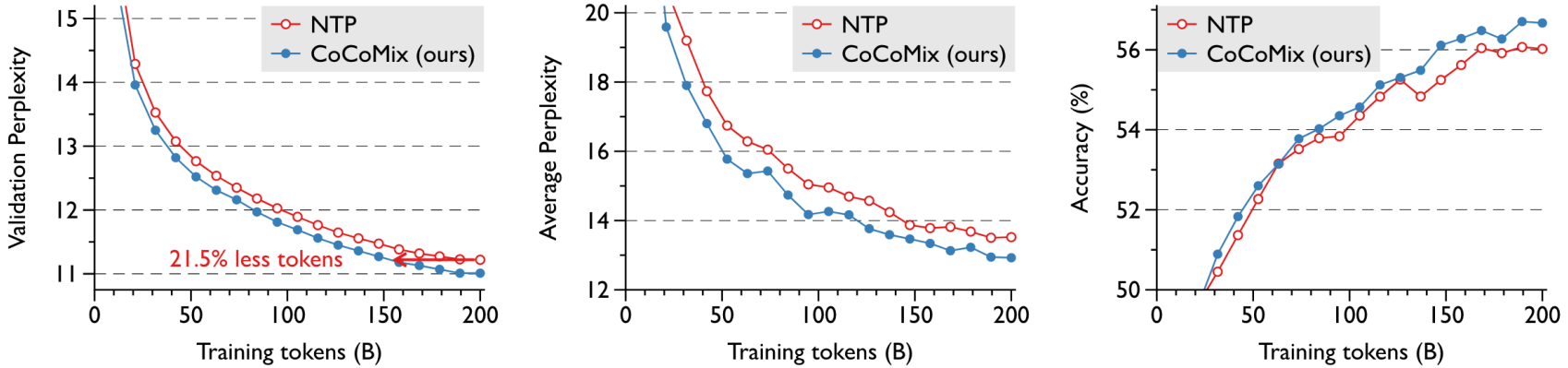
Extracting concepts from a pretrained SAE model's hidden state

Learning to predict concepts & Mixing/Interleaving continuous concepts into the hidden state

CoCoMix: **Experiments**

Q1: Can CoCoMix improve the performance of next token prediction in LLM pretraining?

- NTP can be improved with CoCoMix
- high sample efficiency



(a) Validation perplexity **(b)** Average downstream task perplexity **(c)** Average downstream task accuracy

Figure 2 CoCoMix vs. NTP performance at different training checkpoints. Each model contains a total of 1.38B parameters. Each model is trained on the OpenWebText dataset. For CoCoMix, the concepts are extracted from a 124M-sized model (10× smaller than the base model). The plots show improvements in: (a) validation perplexity, (b) average perplexity on LAMBADA, WikiText-103, and (c) average accuracy on HellaSwag, PIQA, SIQA, Arc-Easy, and WinoGrande.

Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, Xian Li. LLM Pretraining with Continuous Concepts. arXiv preprint arXiv: 2502.08524, 2025.

CoCoMix: **Experiments**

Q2: Does CoCoMix show improvement in weak-to-strong supervision setup compared to other knowledge distillation methods?

- KD baseline: 124M teacher model
- CoCoMix: concept extracted from SAE trained on 124M model

Method	Total Params	OWT PPL (↓)	LAMB PPL (↓)	Wiki PPL (↓)	HellaS Acc-n (↑)	PIQA Acc (↑)	SIQA Acc (↑)	Arc-E Acc (↑)	WinoG Acc (↑)	Avg PPL (↓)	Avg Acc (↑)
NTP	69M	25.3	107.6	52.3	27.4	59.4	36.6	39.7	50.7	61.8	42.7
KD	69M	25.2	99.3	51.0	27.4	**59.8**	36.2	**39.8**	50.7	58.5	42.8
CoCoMix	69M	**24.7**	**99.1**	**50.9**	**27.6**	59.5	**37.2**	39.3	**51.0**	**58.2**	**42.9**
NTP	386M	16.3	26.3	29.9	33.6	**64.1**	38.4	47.3	50.9	24.2	46.8
KD	386M	16.4	24.6	**29.1**	33.6	63.6	38.0	**48.5**	51.4	23.4	47.0
CoCoMix	386M	**15.9**	**19.3**	**29.1**	**34.7**	63.6	**39.2**	47.9	**52.3**	**21.4**	**47.5**
NTP	1.38B	14.3	16.6	25.0	38.1	66.1	38.9	50.5	50.0	18.6	48.7
KD	1.38B	14.2	16.6	**24.9**	37.4	66.7	39.0	50.1	52.3	18.5	49.1
CoCoMix	1.38B	**13.9**	**15.4**	**24.9**	**38.4**	**66.9**	**39.5**	**50.8**	**53.0**	**18.1**	**49.7**

Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, Xian Li. LLM Pretraining with Continuous Concepts. arXiv preprint arXiv: 2502.08524, 2025.

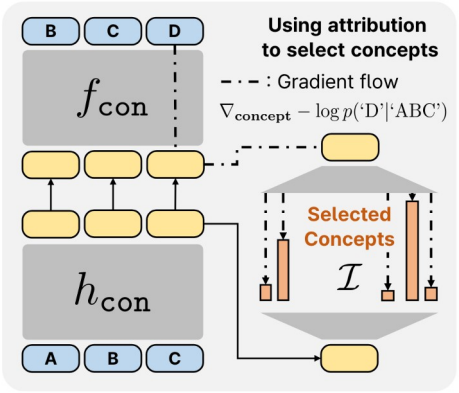
CoCoMix: **Interpretability & Steerability**

- we can control the model's behavior through modulating the magnitude of the predicted concept z_t
- multiply z_t of a desired concept element by constants ranging from -10 to 10
- CoCoMix can successfully learn the GPT-2 aligned concepts

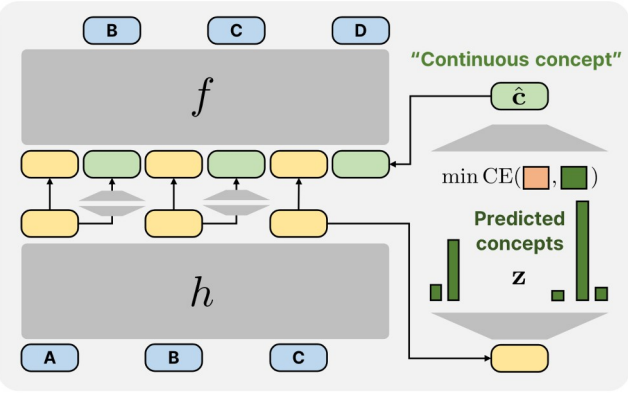
Input Prompt: The best platform for buying tickets is the

	No Steer	Steer concept 23843: site address related	Steer concept 15687: year/month related
CoCoMix (ours)	**Generation:** vernacular ticketing system," said the official. "The ticket prices are fixed and the tickets are available at the same time."	**Generation:** vernacular website, Ticketmaster.com. The site is a great resource for finding the best deals on tickets.	**Generation:** – by the end of the year – and the new platform by fans. The new ticket- by a group of fans has been known as the "ticket by 'ticket – the group
GPT2 SAE	**Generation:** iphone app.	**Generation:** urchin-cheers.com. The online store has a new, fast-growing selection of premium tickets	**Generation:** Facebook page. It's a place where you can buy tickets for free and then go to the next one. The other option is to buy a ticket for a month or two

Figure 5 Qualitative demonstration of the concept steering effect. CoCoMix and GPT2 models are 350M and 124M parameter transformers, respectively, trained on the OpenWebText dataset. For CoCoMix, we manipulate the predicted concept logit $\mathbf{z}$, while for GPT2, we adjust the SAE concept space $\mathbf{c}$ by increasing the activation of a specific concept index. This illustrates the impact of targeted concept steering on the respective model outputs.



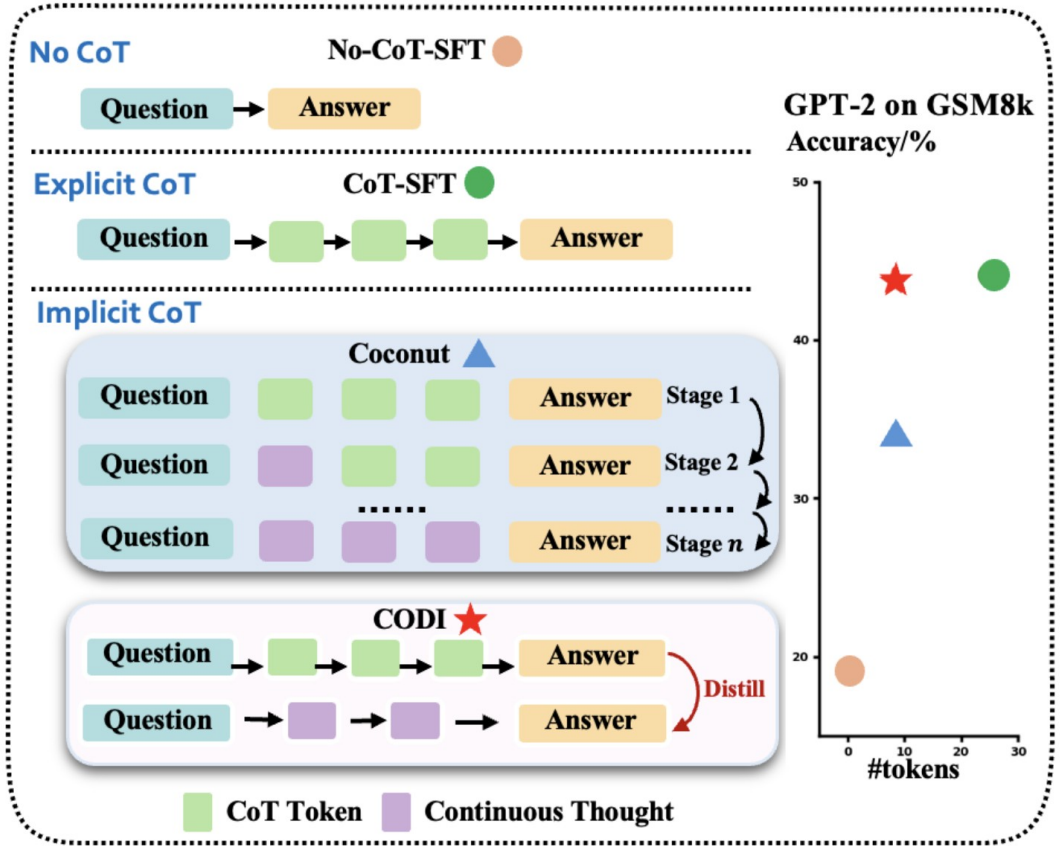
Extracting concepts from a pretrained SAE model's hidden state



Learning to predict concepts & Mixing/Interleaving continuous concepts into the hidden state

CODI: Compressing Chain-of-Thought into Continuous Space via Self-Distillation

- compress discrete tokens into less continuous tokens



Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. arXiv preprint arXiv:2502.21074, 2025.

Outline

- Preliminary
- Latent Reasoning
  - Loop/Universal Transformer Recurrence: Universal Transformer, CoTFormer, Scaling Up Looped Transformers
  - Activation with Explicit Hidden-State Feedback: Coconut, CoCoMix
- **A Brief Introduction of Mechanistic Interpretability**

Questions

In the previous work, we have acknowledged that depth is important, so we have:

- **looped transformers** to stack layers to get better performance
- **continuous token approaches** to increase the effective depth

But we haven't answered that:

1. **Do layer stacks reflect latent CoT?**
2. **How do layers function as a latent CoT?**

Do layer stacks re

What does "making Trai

- **macro level**: a close

  model

  - **Yu et al., Guo et al**

    final reasoning outc

  - **Merrill and Sabha**

    complex tasks like

- **micro level**: a clear

ng capabilities of the

emerge within some layers, the

reasoning abilities, enabling

ıs cannot achieve.

ːs within CoT reasoning

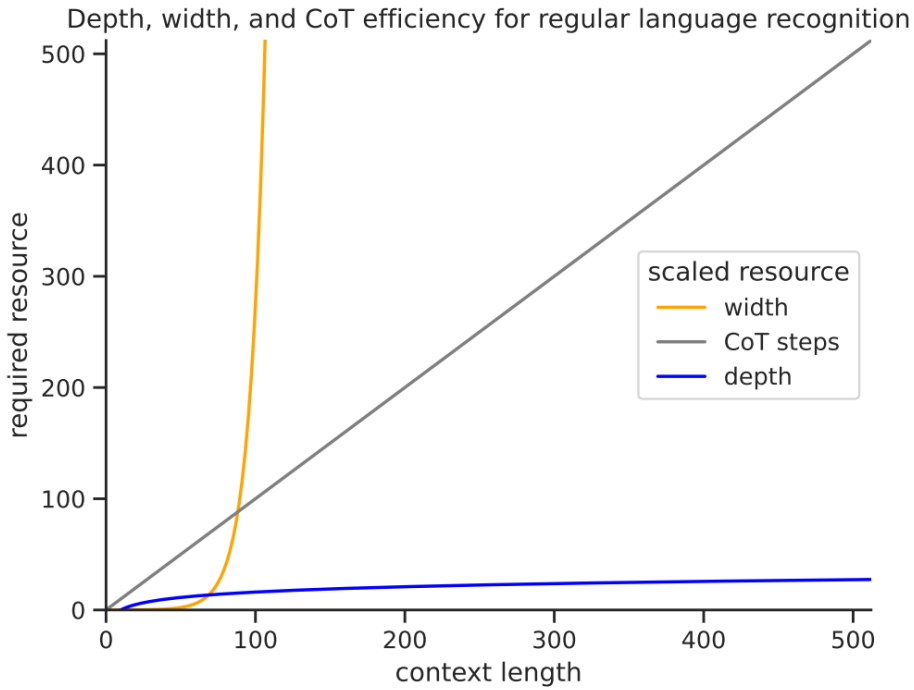


Figure 1: To recognize a regular language over inputs of length n, the depth of a universal transformer can grow $\Theta(\log n)$ by Theorem 1. On the other hand, width must grow *superpolynomially* (Theorem 3), and the number of chain-of-thought steps must be *superlogarithmic* (Theorem 4). The precise depth and width coefficients plotted here were **obtained experimentally** in Section 7.

How do layers function as a latent CoT?

- **Theory of layer specialization:** a natural assumption is that different layers play distinct roles in reasoning tasks. A series of interpretability studies are focusing on uncovering how these layers work together to build and convey the underlying CoT processes.
- **Shallow layers:** basic representational processor, primarily process local information, syntactic structures, and surface patterns, perform initial data transformations, and form early circuit primitives
- **Intermediate layers:** core of latent CoT
  - Intermediate layers form specialized sub-circuits dedicated to reasoning functions,
  - Intermediate layers exhibit superior representational capabilities,
  - Activations in intermediate layers have a decisive impact on reasoning outcomes

How do layers function as a latent CoT?

- **Shallow layers:** basic representational processor, primarily process local information, syntactic structures, and surface patterns, perform initial data transformations, and form early circuit primitives
- **Intermedi**
  - Interme
  - Interme
  - Activat
- **Deep laye**
  - layer pr
  - some at
  - some at

◦ **why is post-norm hard to train**: gradient vanishing, and output sensitive to parameter change

$$x_{t+1} = \text{Norm}(x_t + F_t(x_t)) \tag{12}$$

这我们可以称为Post Norm结构，它也是原版Transformer和BERT所使用的设计。然而，这种做法虽然稳定了前向传播的方差，但事实上已经严重削弱了残差的恒等分支，所以反而失去了残差“易于训练”的优点，通常要warmup并设置足够小的学习率才能使它收敛。

怎么理解这一点呢？假设初始状态下$x, F(x)$的方差均为1，那么$x + F(x)$的方差就是2，而Normalization操作负责将方差重新降为1，这就说明初始阶段Post Norm相当于

$$x_{t+1} = \frac{x_t + F_t(x_t)}{\sqrt{2}} \tag{13}$$

递归下去，我们得到

$$\begin{aligned} x_l &= \frac{x_{l-1}}{\sqrt{2}} + \frac{F_{l-1}(x_{l-1})}{\sqrt{2}} \\ &= \frac{x_{l-2}}{2} + \frac{F_{l-2}(x_{l-2})}{2} + \frac{F_{l-1}(x_{l-1})}{\sqrt{2}} \\ &= \cdots \\ &= \frac{x_0}{2^{l/2}} + \frac{F_0(x_0)}{2^{l/2}} + \frac{F_1(x_1)}{2^{(l-1)/2}} + \frac{F_2(x_2)}{2^{(l-2)/2}} + \cdots + \frac{F_{l-1}(x_{l-1})}{2^{1/2}} \end{aligned} \tag{14}$$

看到问题了没？本来残差的意思是给前面的层搞一条“绿色通道”，让梯度可以更直接地回传，但是在Post Norm中，这条“绿色通道”被严重削弱了，越靠近前面的通道反而权重越小，残差“名存实亡”，因此还是不容易训练。相关的分析还可以参考论文《On Layer Normalization in the Transforme r Architecture》。

Takeaways

- Looped Transformers
- Continuous Tokens
- Mechanistic Understandings