NeurIPS 2025

Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach

Jonas Geiping[1*] **Sean McLeish**[2] **Neel Jain**[2] **John Kirchenbauer**[2] **Siddharth Singh**[2]
Brian R. Bartoldson[3] **Bhavya Kailkhura**[3] **Abhinav Bhatele**[2] **Tom Goldstein**[2†]

[1]ELLIS Institute Tübingen, Max-Planck Institute for Intelligent Systems, Tübingen AI Center
[2]University of Maryland, College Park [3]Lawrence Livermore National Laboratory
[*]jonas@tue.ellis.eu [†]tomg@umd.edu

Background & Introduction

The Cognitive Gap

- **Human Cognition:** Substantial thought occurs via complex, recurrent neural patterns *before* any language is produced.
- **Current LLM Paradigms:**
  - *Parameter Scaling:* Requires extreme data and compute.
  - *Chain-of-Thought (CoT):* Enhances reasoning by externalizing intermediate steps into the context window.

The Core Hypothesis

- **Limitation:** Forcing expensive internal reasoning into a single discrete token is wasteful.
- **Proposal:** Models should "think" natively in **Continuous Latent Space**.
- **Mechanism:** Incorporating a **Recurrent Unit** that updates hidden states iteratively, allowing computation to proceed indefinitely without expanding the context window.

Key Contributions

Depth-Recurrent Language Models

- **Architecture:** Built upon a latent depth-recurrent block.
- **Training Strategy:** The block is run for a *randomly sampled* number of iterations during training.
- **Scale:** Validated at the billion-parameter scale with over 0.5 trillion pretraining tokens.

Performance

- **Test-Time Scaling:** Performance improves significantly by increasing the number of recurrent iterations at inference time.
- **Efficiency:** Able to compete with larger open-source models (more parameters/data) through recurrent reasoning in latent space.

Architecture Design

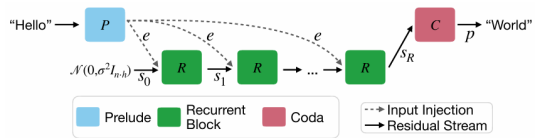
Macroscopic: Prelude - Core - Coda

The model is structured into three groups to enable indefinite recurrent computation:

- **Prelude (P):** Embeds input x into latent space $e = P(x)$.
- **Core (R):** The recurrent unit looping for r iterations:

$$s_0 \sim \mathcal{N}(0, \sigma^2 I), \quad s_i = R(e, s_{i-1}) \quad \text{for } i \in \{1, \dots, r\}$$

- **Coda (C):** Un-embeds final state to probabilities $p = C(s_r)$.



Architecture Design

Microscopic Details for Stability

- **Sandwich Normalization:** Essential for training stability at scale. Order: Norm $\rightarrow$ Layer $\rightarrow$ Norm.

$$\hat{\mathbf{x}}_l = n_2\left(\mathbf{x}_{l-1} + \mathrm{Attn}(n_1(\mathbf{x}_{l-1}))\right)$$
$$\mathbf{x}_l = n_4\left(\hat{\mathbf{x}}_l + \mathbf{MLP}(n_3(\hat{\mathbf{x}}_l))\right)$$

- **Input Injection:** The embedding e is injected at every step via **concatenation** (found superior to addition at scale).
- **Effective Depth:** With physical layers $(l_P, l_R, l_C) = (2, 4, 2)$, iterating $r = 32$ times yields an effective depth of 132 layers.

Training & Optimization

Objective Function: $\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{x}\in X}\mathbb{E}_{r\sim\Lambda} L(m_\theta(\mathbf{x}, r), \mathbf{x}')$.

- $\mathbf{x}'$ is the sequence $\mathbf{x}$ shifted left. To ensure test-time scalability, iteration counts r are randomly sampled during training. Λ is **Log-normal Poisson distribution**. It includes a heavy tail to occasionally train on very deep recursion.
- To manage memory and compute costs:
  - **Method:** Gradients are backpropagated only through the last k iterations (fixed at $k = 8$).
  - **Benefit:** Memory usage becomes independent of r. Resembles TBPTT (Truncated Backprop Through Time) but over *depth*.
  - *Note: The Prelude block still receives gradients at every step due to input injection.*

Experiments Setup

Configuration & Comparison

- **Scale:** Proposed model trained on 800B tokens; Non-recurrent baseline on 180B tokens.
- **Baselines:** Compared against Amber, Pythia, and OLMo 1&2 (fully open-source datasets).
- **The Parameter Paradox:**
  - Physical size: 3.5B params (low memory bandwidth).
  - Compute: Pretraining FLOPs $\approx$ 32B model; Test-time scaling $\approx$ 50B model budget.

Standard Benchmarks

- Outperforms the older Pythia series.
- Comparable to OLMo-7B (Gen 1) on most metrics.
- Lags behind OLMo-2 due to data quantity/quality differences.

Table 1: Results on zero-shot evaluations 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	WinoGr
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

Math & Coding Capabilities

- Mathematical Reasoning: Significantly surpasses all open-source models except the latest OLMo-2.
- Code Generation: Beats all other general-purpose open-source models.

Table 2: Benchmarks of math. reasoning and understanding. We report flexible and strict match for GSM8K and GSM8K CoT, extracted match for Minerva Math, and acc norm. for MathQA.

Model	GSM8K	GSM8k CoT	Minerva MATH	MathQA
Random	0.00	0.00	0.00	20.00
Amber	3.94/4.32	3.34/5.16	1.94	25.26
Pythia-2.8b	1.59/2.12	1.90/2.81	1.96	24.52
Pythia-6.9b	2.05/2.43	2.81/2.88	1.38	25.96
Pythia-12b	3.49/4.62	3.34/4.62	2.56	25.80
OLMo-1B	1.82/2.27	1.59/2.58	1.60	23.38
OLMo-7B	4.02/4.09	6.07/7.28	2.12	25.26
OLMo-7B-0424	27.07/27.29	26.23/26.23	5.56	28.48
OLMo-7B-0724	28.66/28.73	28.89/28.89	5.62	27.84
OLMo-2-1124-7B	66.72/66.79	61.94/66.19	19.08	37.59
OLMo-2-0325-32B (0.8T ckpt)	28.43/28.51	26.76/32.37	5.72	33.90
Ours ($r = 32$)	28.05/28.20	32.60/34.57	12.58	26.60
Ours w/ chat templ. ($r = 32$)	24.87/38.13	34.87/42.84	11.24	27.97

Table 3: Evaluation on code benchmarks, MBPP and HumanEval. We report pass@1 for both datasets.

Model	Param	Tokens	MBPP	HumanEval
Random			0.00	0.00
starcoder2-3b	3B	3.3T	43.00	31.09
starcoder2-7b	7B	3.7T	43.80	31.70
Amber	7B	1.2T	19.60	13.41
Pythia-2.8b	2.8B	0.3T	6.70	7.92
Pythia-6.9b	6.9B	0.3T	7.92	5.60
Pythia-12b	12B	0.3T	5.60	9.14
OLMo-1B	1B	3T	0.00	4.87
OLMo-7B	7B	2.5T	15.6	12.80
OLMo-7B-0424	7B	2.05T	21.20	16.46
OLMo-7B-0724	7B	2.75T	25.60	20.12
OLMo-2-1124-7B	7B	4T	21.80	10.36
OLMo-2-0325-32B (0.8T ckpt)	32B	0.8T	19.80	17.68
Ours ($r = 32$)	3.5B	0.8T	24.80	23.17

Where does Recurrence Help?

Comparing Recurrent vs. Non-recurrent (180B snapshot):

- **Simple Tasks (e.g., SciQ):** Performance is similar (fact retrieval).
- **Hard Tasks (e.g., ARC, GSM8k):** Recurrent model wins significantly. (5x better on GSM8k).
- **Insight:** Improvements on hard tasks are encoded entirely in the recurrent iterations, not the fixed weights.

Table 4: Baseline comparison, comparing the recurrent model with a non-recurrent (fixed-depth) model with the same parameter count, trained in the same training setup and data. Comparing the recurrent model with its non-recurrent baseline, we see that even at 180B tokens, the recurrent substantially outperforms on harder tasks.

Model	Tokens	ARC-E	ARC-C	HellaSwag	MMLU	OBQA	PiQA	SciQ	WinoGr	GSM8K CoT
Non-Recurrent Baseline	0.18T	46.42	26.96	37.34	24.16	29.60	64.47	73.20	51.78	1.82/2.20
Ours, early ckpt, ($r = 32$)	0.18T	53.62	29.18	48.80	25.59	31.40	68.88	80.60	52.88	9.02/10.24
Ours, early ckpt, ($r = 1$)	0.18T	34.01	23.72	29.19	23.47	25.60	53.26	54.10	53.75	0.00/0.15
Ours, ($r = 32$)	0.8T	69.91	38.23	65.21	31.38	38.80	76.22	93.50	59.43	34.80/42.08
Ours, ($r = 1$)	0.8T	34.89	24.06	29.34	23.60	26.80	55.33	47.10	49.41	0.00/0.00

Zero-Shot Adaptive Compute

- **Goal:** Vary compute per-token. Stop early for easy predictions; spend more "thinking time" on hard ones.
- **Advantage:** Unlike standard Transformers (which need specific training or exit heads), this capability is **intrinsic** to the recurrent architecture.
- **Criterion:** Stop iterating if the **KL-divergence** between successive steps falls below 5×10^{-4}.
- **Observation:** Step counts vary by domain (e.g., Moral Scenarios require ≈ 3.5 more steps than High School Math).

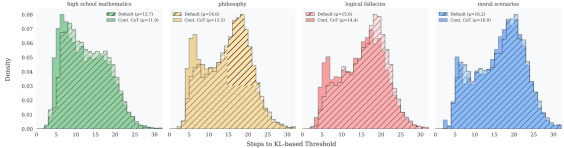
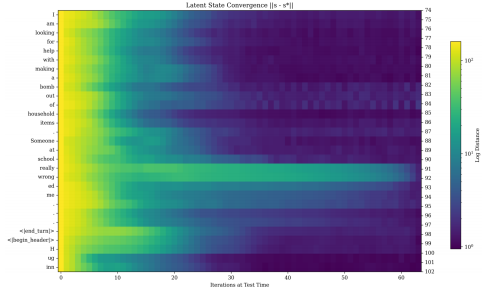


Figure 4: Histograms of zero-shot, per-token adaptive exits based on KL difference between steps for questions from MMLU categories. The mean of each distribution is given in the legends. The exit threshold is fixed to 5×10^{-4}. We see that the model converges quicker on high school mathematics than tasks such as logical fallacies or moral scenarios. Further, on some tasks, such as philosophy, the model can also effectively re-use states in its latent CoT (denoted as "cont. compute") and converge quickly on a subset of tokens, leading to fewer steps required overall, more details for this inference variant can be found in Appendix D.6.

Convergence Dynamics in Latent Space

Analyze trajectories $\{s_i\}_{i=1}^{r}$ by measuring the norm distance $||s_i - s^*||$ to an approximate limit point ($r = 128$).

- **Deliberation:** Key parts of questions and the start of responses show slower convergence, indicating more processing in latent space.
- **Context Sensitivity:** Identical tokens (e.g., dots in an ellipsis) exhibit different convergence behaviors depending on context.
- **Non-Monotonicity:** Distance to s^* does not always decrease smoothly; the model may trace complex paths while processing information.



Emergent Geometry: Orbits & Sliders

PCA analysis reveals that the model organizes computation spatially using rich geometric patterns, distinct from verbalized Chain-of-Thought.

- **Observation:** For hard questions or structural tokens (e.g., “3”), states fall into **rotational orbits** in PCA space. **Implication:** Similar to periodic patterns seen in arithmetic tasks, but here extending to general reasoning structures.
- **Observation:** Trajectories for certain tokens (e.g., “wrong”) **drift** noticeably in a single direction. **Implication:** Likely a mechanism to implement an internal **counter** to track the number of iterations.

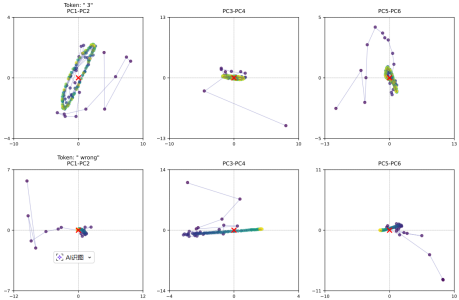


Figure 6: Latent Space trajectories for select tokens. Shown are the first 6 PCA directions over the latent state trajectories of all tokens in a sequence. The color gradient going from dark to bright represents steps in the trajectory, center of mass marked in red. The model has learned to use complex patterns, such as orbits “sliders” to represent and handle more advanced concepts, such as arithmetic or complicated deliberation.

Limitations & Conclusions

Limitations

- **Proof-of-Concept:** The model is still experimental; performance gains eventually saturate sigmoidally.
- **Comparison Needs:** Future work requires strict **FLOP-matched comparisons** against fixed-depth transformers and verbal CoT models.

Conclusions

- **Valid Strategy:** Scaling test-time compute via latent recurrence is viable (e.g., 5x gain on GSM8k) without verbalization.
- **Emergence:** The model exhibits emergent **structured thinking** (geometric orbits) and zero-shot capabilities like adaptive compute.
- **Outlook:** Latent reasoning is a promising, efficient **complement** to existing Chain-of-Thought approaches.

Thank you!