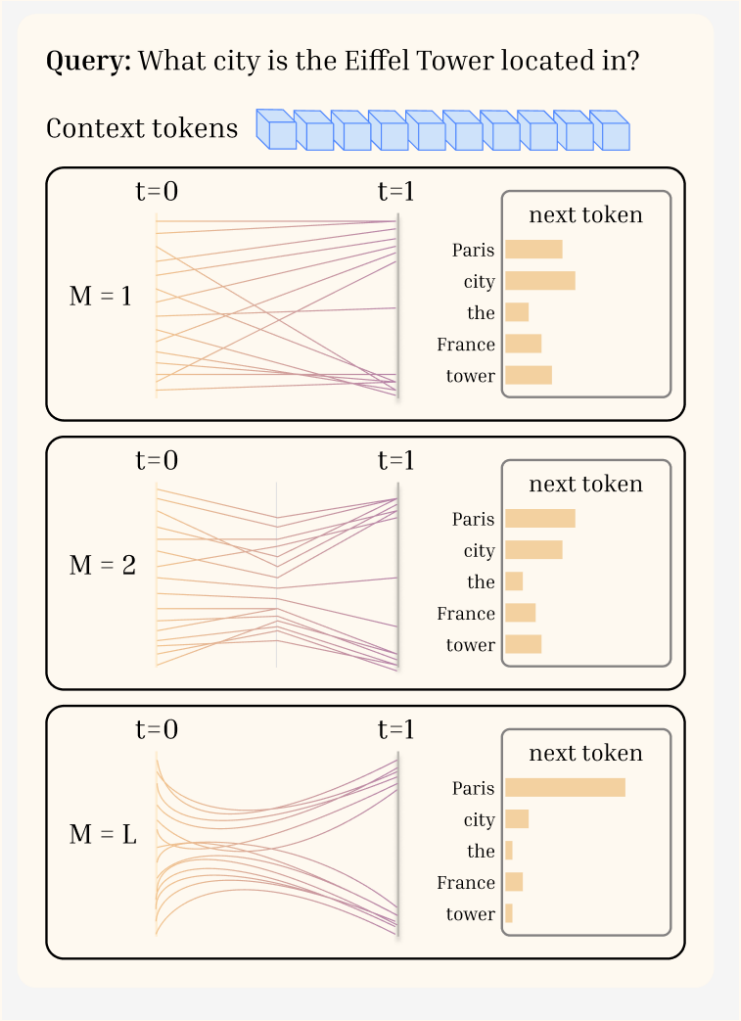
LoopFormer

Elastic-Depth Looped Transformers for Latent Reasoning via Shortcut Modulation

Ahmadreza Jeddi · Marco Ciccone · Babak Taati
ICLR 2026

Core question: can looped LMs use variable compute budgets without collapsing?



Loops are already everywhere in modern ML

Explicit or implicit, many strong models reuse the same computation multiple times.

Diffusion / iterative denoising

Sampling repeatedly applies the same denoiser across timesteps.

This is explicit looping over a time horizon.

Chain-of-thought in LLMs

Iterative autoregressive sampling using the same backbone

Explicit reasoning

Recurrent Transformers

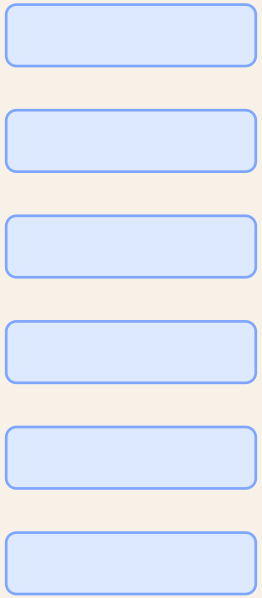
ALBERT
Universal Transformer
HRM
RLM
Recurrent Depth LLms

If you use diffusion, iterative solvers, or recurrent/shared-depth transformers, you are already using looped computation one way or another.

Background: what is a looped Transformer?

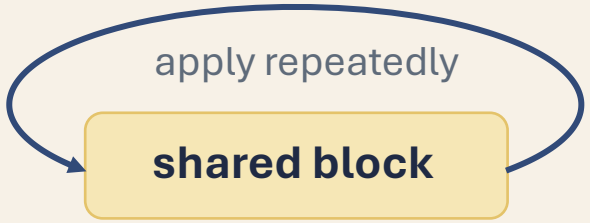
A looped model reuses the same block multiple times instead of stacking many distinct blocks.

Standard stack



Many unique layers

Looped stack



Fewer parameters

$(k \otimes L)$: k-layer block, looped L times

Almost all prior work trains and runs at one fixed loop count.

Our goal: adaptive compute for looped models

looped models are efficient and reasoning-friendly, but fixed-depth training leaves compute flexibility on the table.

Parameter sharing

One shared block can act like a much deeper network while keeping parameters fixed.

Latent reasoning

Recent work suggests looped depth helps internal multi-step computation on reasoning tasks.

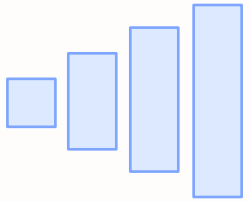
Compute budgets

Real deployments rarely want one fixed budget; we want the same model to work from cheap to expensive.

The mismatch: fixed training depth vs variable test budgets

Train at one route

Most looped LMs are trained at one unroll length L, so hidden-state trajectories specialize to that schedule.

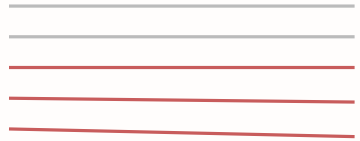


Deploy under a different budget

At inference, users may want M < L or different step schedules. Those runs are off-distribution.

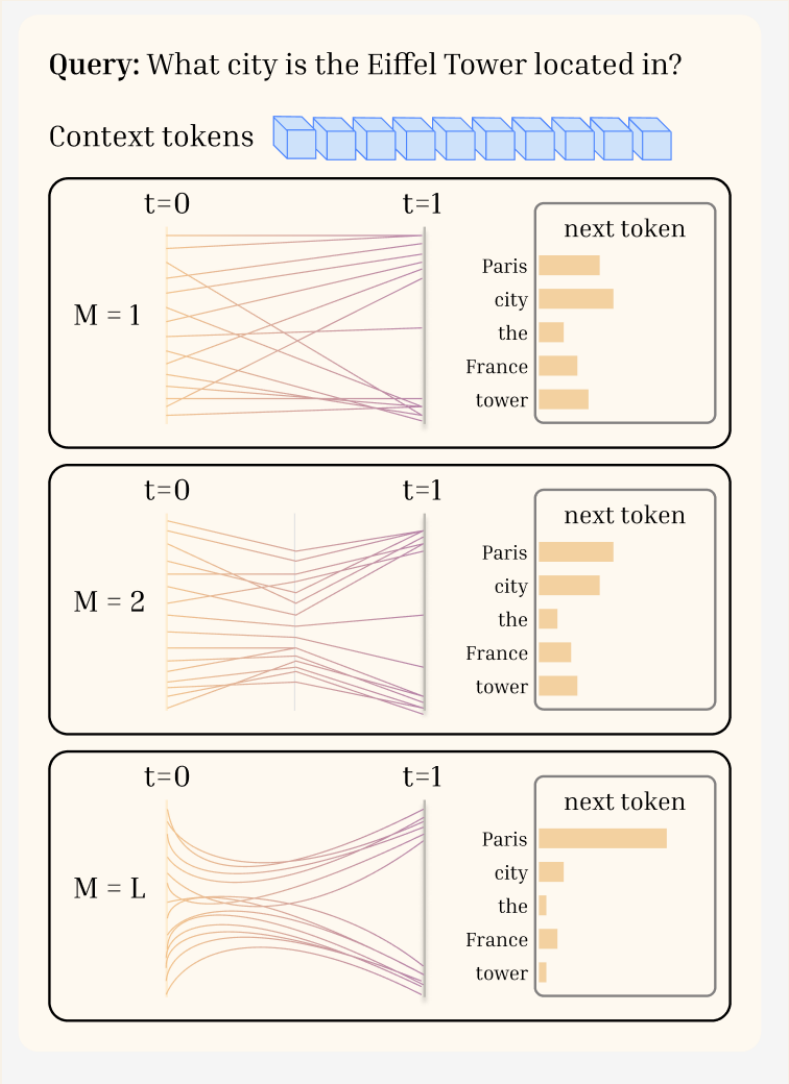
Naive early exit can stagnate

Repeated passes of the shared block can converge to very similar states, underusing depth.



Goal: keep shorter routes useful and let additional loops refine rather than collapse.

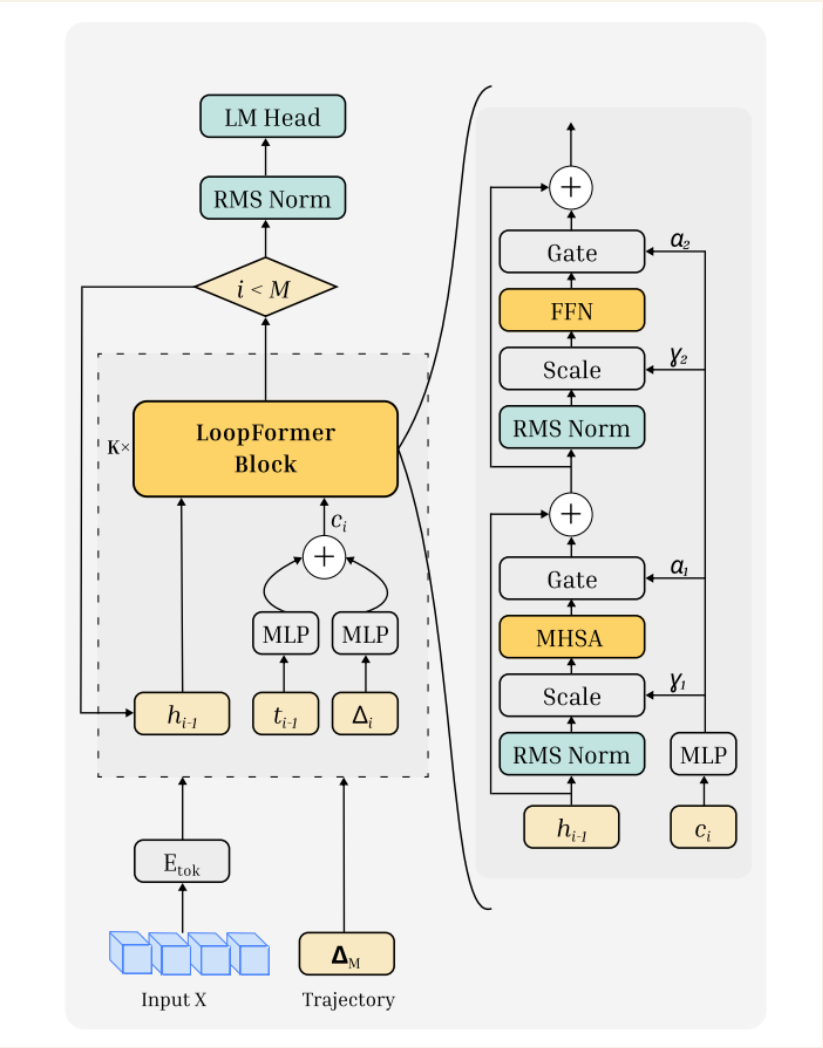
Key intuition: “latent trajectories”



- Represent looping as a trajectory over normalized time $t \in [0,1]$.
- A budget M chooses how many points (and jumps) are used on that trajectory.
- The model should remain useful at low budget, but get better as extra steps refine the representation.
- This is the paper’s “latent trajectories” view of latent reasoning.

LoopFormer architecture

Same looped backbone, but every iteration is conditioned on where it is and how big the jump is.



- Shared K-block decoder is reused across iterations.
- Condition each loop on current time t_i and step size Δt_i.
- The conditioning vector modulates RMSNorm scales and gates MHSA/FFN residuals.
- At inference: pick any budget $M \leq L$ and a schedule Δ_M.

Novelty = trajectory conditioning not just “loop more”

Training recipe: full route + shortcut route + consistency

The model learns to keep short trajectories aligned with a stronger full route.

Sample 2 trajectories for each batch

- **Full route (L steps)**
- **Shortcut route (S < L)**

Training Loss

$$L = L_{full} + \lambda_1 L_{short} + \lambda_2 L_{consistency}$$

L_{full} : Cross-entropy on the full route

L_{short} : Cross-entropy on a sampled shorter route

$L_{consistency}$: Stop-gradient consistency from full to shortcut

Where the idea comes from: diffusion, shortcut training, and ODEs

The core intuition is to view repeated computation as a trajectory in hidden space.

Diffusion / flow view

A model follows a trajectory from one state to another.

Fewer steps give a coarser approximation; more steps refine it.

Shortcut / consistency view

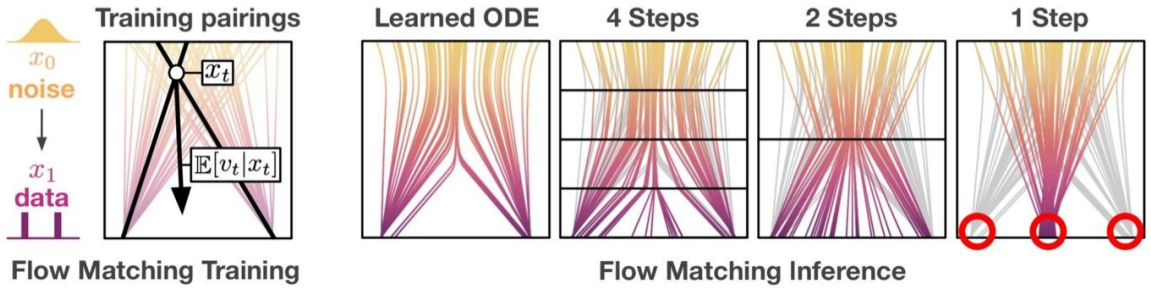
Train short routes to agree with a longer route.

That is the training recipe we import into looped language models.

Neural ODE view

Depth can be interpreted as integration over time.

Changing the discretization changes the path, but the endpoint should remain meaningful.



From “ONE STEP DIFFUSION VIA SHORTCUT MODELS”, Frans et al – ICLR2025 Oral

Baselines and the main comparison table

The key comparison is against (i) vanilla non-looped Transformers, (ii) fixed-loop baselines, and (iii) elastic-depth baselines.

Baselines

- Base: vanilla decoder-only Transformer
- Base-Loop: fixed loop count
- TMLT: time-conditioned fixed-loop model
- Base-Loop-EE / TMLT-EE: naive early-exit baselines

	Params / FLOPs	Perplexity ↓			Language Tasks (Accuracy) ↑										
		Pile	FineWeb-Edu	OpenWebText	COPA	HS	LB	OBQA	PIQA	Race	SciQ	ARC	SIQA	WG	Avg Acc
Budget: 24x															
Base ($24 \otimes 1$)	24x / 24x	**9.49**	**20.7**	**20.08**	61	**35.04**	**41.96**	27.6	**66**	29	**70.1**	**33.43**	38.18	50.4	45.27
Base-Loop ($3 \otimes 8$)	3x / 24x	10.91	24.53	24.53	61	30.46	34.68	27	63.22	28.8	63.7	31.71	**38.43**	49.8	42.88
TMLT ($3 \otimes 8$)	3x / 24x	10.38	22.87	21.99	65	32.34	39.06	27.8	63.11	29.67	69.8	31.67	36.95	51.54	44.69
Naive-Loop-EE ($3 \otimes 8$)	3x / 24x	11.6	26.55	25.64	**66**	29.68	31.52	27	62.24	28.33	65.5	31.64	36.64	50.59	42.91
Base-Loop-EE-Cons ($3 \otimes 8$)	3x / 24x	11.56	25.33	24.41	**66**	30.54	32.19	26.8	62.13	28.23	64.9	31.45	37.4	**52.88**	43.25
TMLT-EE ($3 \otimes 8$)	3x / 24x	10.7	24.07	23.17	65	31.03	35.14	**28.4**	63.11	28.8	68.8	31.74	37.41	50.83	44.03
LoopFormer - Ours ($3 \otimes 8$)	3x / 24x	10.28	22.87	21.98	**66**	32.3	38.27	26.8	63.33	**30.81**	68	32.71	37.97	51.94	44.81
Budget: 12x															
Base (12x1)	12x / 12x	**9.98**	**22.24**	**21.41**	67	**32.72**	**37.78**	26.2	**64.69**	**29.86**	**69.1**	**32.29**	**38.38**	51.3	44.93
Naive-Loop-EE ($3 \otimes 4$)	3x / 12x	11.66	26.74	25.81	65	29.35	31.28	26	61.75	28.32	64.4	31.15	36.8	51.85	42.59
Base-Loop-EE-Cons ($3 \otimes 4$)	3x / 12x	12.0	27.72	26.68	63	29.72	25.6	**27**	61.26	28.04	58.4	30.06	36.23	51.7	41.1
TMLT-EE ($3 \otimes 4$)	3x / 12x	12.18	28.28	27.11	60	29.88	27.73	26.4	62.02	28.8	61.4	30.59	36.69	51.54	41.5
LoopFormer - Ours ($3 \otimes 4$)	3x / 12x	11.12	25.02	24.21	**68**	31	32.35	25.4	63.06	28.23	66.3	31.78	37.77	**53.43**	43.73
Budget: 6x															
Base (6x1)	6x / 6x	**11.13**	**25.28**	**24.28**	**64**	**30.45**	**33.45**	25.6	**62.51**	**28.04**	**67.5**	**31.07**	**36.18**	48.54	42.73
Naive-Loop-EE ($3 \otimes 2$)	3x / 6x	12.61	29.36	28.38	63	28.64	27.28	24.6	61.48	26.41	62.3	29.91	35.41	50.67	40.97
Base-Loop-EE-Cons ($3 \otimes 2$)	3x / 6x	15.07	35.95	34.88	59	28.28	18.49	25.6	59.52	25.16	53.8	28.94	34.65	**52.72**	38.62
TMLT-EE ($3 \otimes 2$)	3x / 6x	15.79	37.83	37.84	57	28.18	17.09	25.08	58.68	26.41	50.01	28.26	34.9	50.27	37.59
LoopFormer - Ours ($3 \otimes 2$)	3x / 6x	14.3	33.45	32.46	63	28.69	26.14	**26.6**	60.1	25.74	58.6	29.29	35.26	50.2	40.36

LoopFormer closes the perplexity gap and is highly competitive on zero-shot reasoning, especially at higher budgets.

Main empirical message

LoopFormer narrows the perplexity gap and gives the best reasoning among looped baselines across budgets.

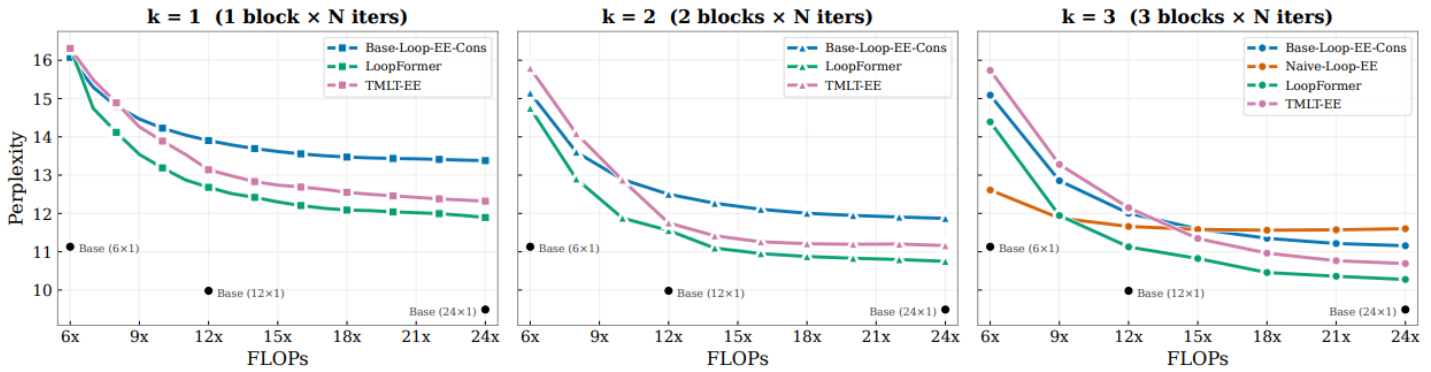


Figure 2: Scaling with layers and loops. Perplexity on The Pile across (k, L) and inference budgets $M \leq L$; larger k lowers perplexity at fixed M, and additional loops further reduce it.

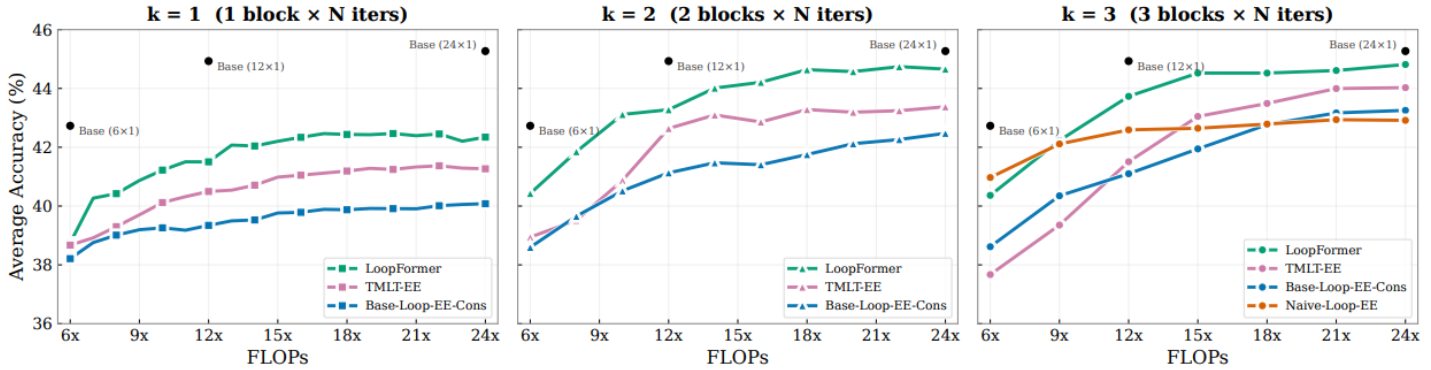


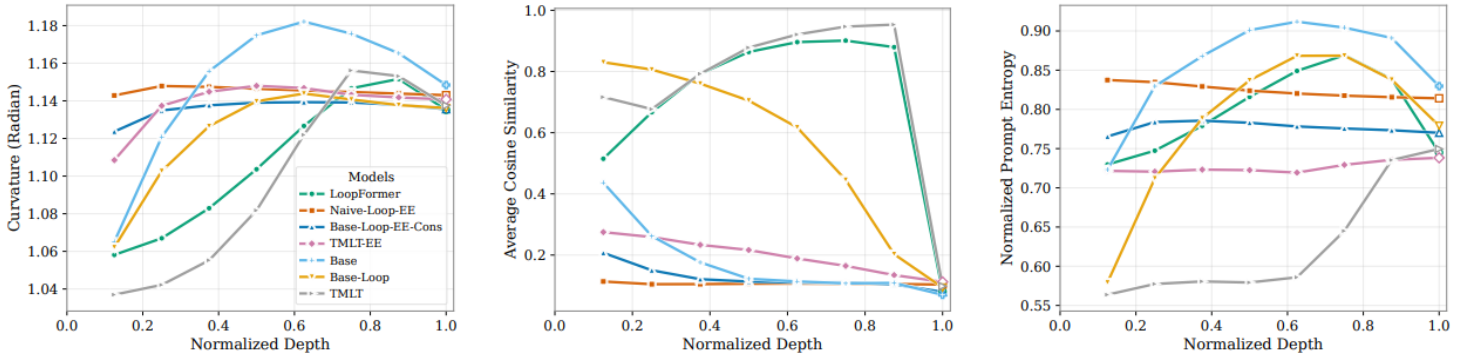
Figure 3: Scaling with layers and loops. Average zero-shot reasoning accuracy on 10 tasks for the same settings; more loops improve reasoning at fixed k. Across budgets, LoopFormer scales smoothly without collapse and consistently outperforms other looped baselines.

- As k grows, perplexity improves; as loops grow, reasoning improves.
- The key result is not just more compute helps — it is that a single model works well over a range of budgets M ≤ L.
- LoopFormer stays smooth under budget reduction and consistently beats other elastic looped baselines.

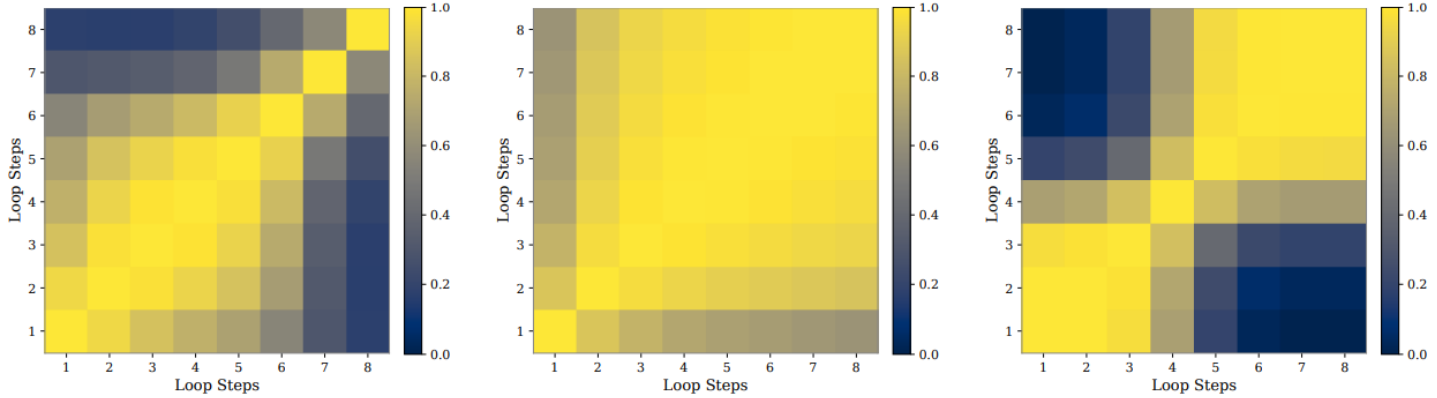
Takeaway: elastic depth can be useful, not just possible.

Why does it help? LoopFormer keeps representations moving

Diagnostics suggest naive early-exit baselines stagnate, while LoopFormer continues to evolve across steps.



(a) Curvature (b) Anisotropy (c) Prompt Entropy



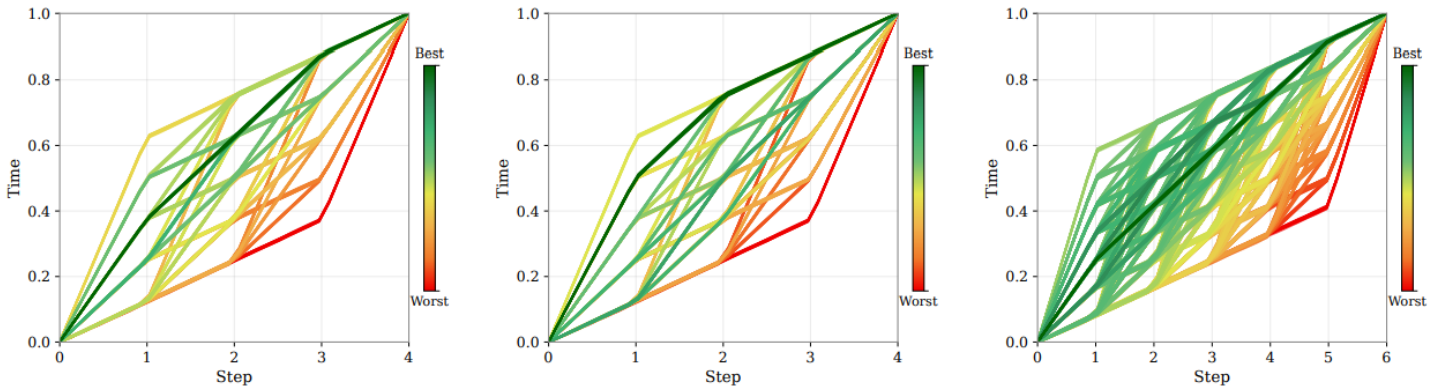
(a) LoopFormer (b) Base-Loop-EE-Cons (c) TMLT-EE

- Three probes: curvature, anisotropy, and prompt entropy.
- Plus cross-step CKA to ask: are later loop states really different?
- Early-exit baselines stay flat and highly similar across steps.
- LoopFormer does not stagnate

Interpretation: shorter runs stay useful; extra loops add refinement instead of collapse.

Which trajectories are good under a fixed budget?

Even at the same compute, different schedules can perform quite differently.



(a) Perplexity ($3 \otimes 8$; $M = 4$) (b) Accuracy ($3 \otimes 8$; $M = 4$) (c) Perplexity ($2 \otimes 12$; $M = 6$)

Figure 6: Performance across length-M trajectories. **(a,b)** enumerate all M=4 schedules for LoopFormer $(3 \otimes 8)$, reporting perplexity and average zero-shot accuracy; **(c)** shows perplexity for M=6 schedules of $(2 \otimes 12)$. Even at fixed budget, trajectory choice matters: spreads are large, and top schedules allocate coarser steps early and finer steps late.

- For fixed M, enumerate all schedules in small settings.
- Result: some trajectories are much better than others even with identical FLOPs.
- Best schedules usually take coarser steps early and finer steps late.

What is still missing?

LoopFormer solves one part of the adaptive-compute puzzle, but not the whole thing.

Current limitations

- Budgets are global, not instance- or token-adaptive.
- Training is more expensive (1.5x) because each batch uses full + shortcut routes.
- Representation analyses are diagnostic, not causal.

Interesting directions

- Learn schedules conditioned on the input.
- Combine with routing or MoE/MoD ideas.
- Scale to larger backbones and instruction-tuned models.

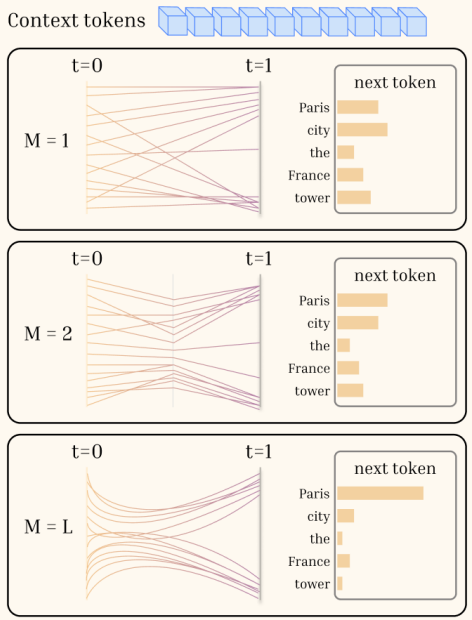
Big-picture question

- Can latent reasoning be made controllable without brittle routing?
- What does the “right” compute schedule look like for different tasks?

Closing takeaways

- Looped transformers are a compelling substrate for latent reasoning because they can trade parameters for effective depth.
- LoopFormer turns that depth into a budget-aware trajectory problem: condition on time and jump size, then train full and shortcut routes to agree.
- Empirically, that gives strong reasoning and language-model quality across budgets while avoiding the collapse seen in naive early-exit variants.

Project page: loopformer.github.io



Thanks!
Questions / discussion