ByteDance | Seed

Understanding Transformer from the Perspective of Associative Memory

Shu Zhong[*,†], **Mingyu Xu**[*], **Tenglong Ao**[*], **Guang Shi**[†]

ByteDance Seed

[*]Equal contribution, [†]Corresponding authors

Overview

Part 1. Associative Memory and Its Capacity

Part 2. Update the Associative Memory

Part 3. Exploration of Next Generation Models

Before We Start

There is a contradiction between expressiveness and parallelism.

Q1: If you can **add** any **two** numbers in parallel each time, **how much** time do we need to calculate the sum of n elements ?
The answer is. $\mathrm{O}(\log^1 n)$.

$$x_1 + x_2 + \cdots + x_n$$

Q2:If you can **add** any **two** numbers in parallel each time, **how much** time do we need to calculate the add of two matrix with size $n \times n$?
The answer is $O(1) = O(\log^0 n)$.

$$A + B$$

$Time: Q1 > Q2$

$Operation\ in\ total:\ Q1 < Q2$

Before We Start

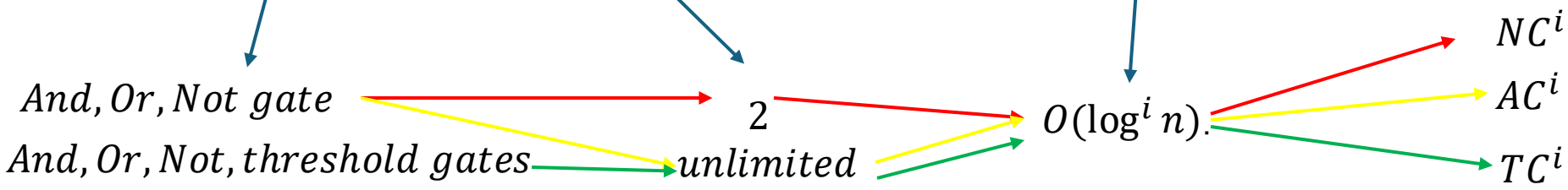
There is a contradiction between expressiveness and parallelism.

Q1: If you can **add** any **two** numbers in parallel each time, **how much** time do we need to calculate the sum of n elements ?
The answer is. $\mathrm{O}(\log^1 n)$.

$$x_1 + x_2 + \cdots + x_n$$

Q2:If you can **add** any **two** numbers in parallel each time, **how much** time do we need to calculate the add of two matrix with size $n \times n$?
The answer is $O(1) = O(\log^0 n)$.

$$A + B$$



Before We Start

There is a contradiction between expressiveness and parallelism.

Where is the popular model, such as Transformer?

Problem can solve without COT	*Parallelism*
Linear Programming	P
...	...
Matrix Inverse	NC^2
Acyclic BCQ	$LOGCFL$
$STCON$	NL
$USTCON$	SL
$S_5, Python, Chess$	NC^1
Parity Check	TC^0
Unary Language	AC^0
Add Bool Matrix	NC^0

The parallelism tradeoff: Limitations of log-precision transformers. TACL 2023.
Chain of Thought Empowers Transformers to Solve Inherently Serial Problems. Arxiv 2024.

Before We Start

There is a contradiction between expressiveness and parallelism.

Where is the popular model, such as Transformer?

TC^0 *or* AC^0 (log or constant precision)

Problem can solve without COT	*Parallelism*
Linear Programming	P
...	...
Matrix Inverse	NC^2
Acyclic BCQ	$LOGCFL$
$STCON$	NL
$USTCON$	SL
$S_5, Python, Chess$	NC^1
Parity Check	TC^0
Unary Language	AC^0
Add Bool Matrix	NC^0

The parallelism tradeoff: Limitations of log-precision transformers. TACL 2023.
Chain of Thought Empowers Transformers to Solve Inherently Serial Problems. Arxiv 2024.

Before We Start

One the most important reason that Transformer beat RNN:

GPUs + High Parallelism

Before We Start

One the most important reason that Transformer beat RNN:

GPUs + High Parallelism

However, we know that there is a contradiction between expressiveness and parallelism.

Is there a model with more expressiveness than transformer but also can be parallelized in GPUs ?

Before We Start

DeltaNet: A model which beyond TC^0

1980s and 1990s

variable weight wp_i. The output at time t is

$$p(t) = \sum_{i=1}^{n} wp_i(t)w_i(t).$$

The weights change over time according to the following equation: for i=1,...,n,

$$wp_i(t+1)=wp_i(t)+cp[z(t)-p(t-1)]x_i(t-1)$$

$$WP_{t+1} = WP_t + cp\,(Z_t - WP_{t-1}w_{t-1})x_{t-1}^{\top}$$

2020s

From the perspective of fast weight programming (Irie et al., 2022a) and test-time training (Sun et al., 2024a) and regression (Wang et al., 2025), the hidden state $\mathbf{S}$ can be interpreted as a (fast) weight matrix, with the delta rule optimizing the online regression objective $\mathcal{L}(\mathbf{S}_t) = \frac{1}{2}\|\mathbf{S}_t\boldsymbol{k}_t - \boldsymbol{v}_t\|^2$ via *test-time* stochastic gradient descent (SGD):

$$\mathbf{S}_{t+1} = \mathbf{S}_t - \beta_t\nabla\mathcal{L}(\mathbf{S}_t) = \mathbf{S}_t - \beta_t(\mathbf{S}_t\boldsymbol{k}_t - \boldsymbol{v}_t)\boldsymbol{k}_t^{\top} = \mathbf{S}_t\,(\mathbf{I} - \beta_t\boldsymbol{k}_t\boldsymbol{k}_t^{\top}) + \beta_t\boldsymbol{v}_t\boldsymbol{k}_t^{\top}$$

$$S_t = S_{t-1} + \beta_t\,(v_t - S_{t-1}k_t)k_t^{\top}$$

https://web.cs.umass.edu/publication/docs/1980/UM-CS-1980-018.pdf

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2f0becffd2f44b198d28074d01722e4c7905dae2

https://www.cs.toronto.edu/~fritz/absps/fastweights.pdf

Parallelizing Linear Transformers with the Delta Rule over Sequence Length. NeurIPS 2024.

Before We Start

DeltaNet: A model which beyond TC^0

1980s and 1990s

$$WP_{t+1} = WP_t + cp\,(Z_t - WP_{t-1}w_{t-1})x_{t-1}^{\top}$$

2020s

$$S_t = S_{t-1} + \beta_t\,(v_t - S_{t-1}k_t)k_t^{\top}$$

Expect the internal output $WP_{t-1}w_{t-1}/S_{t-1}k_t$ to be consistent with the feedback Z_t/v_t, and adjust the internal state WP/S based on the inconsistency $(Z_t - WP_{t-1}w_{t-1})/(v_t - S_{t-1}k_t)$ with a learning rate cp/β_t.

Regarding this update rule, it can have expressive power beyond TC^0 . Readers can refer *to RWKV-7 "Goose" with Expressive Dynamic State Evolution. Arxiv 2025.*

Before We Start

DeltaNet: A model beyond TC^0

1980s and 1990s

$$WP_{t+1} = WP_t + cp\,(Z_t - WP_{t-1}w_{t-1})x_{t-1}^{\top}$$

2020s

$$S_t = S_{t-1} + \beta_t\,(v_t - S_{t-1}k_t)k_t^{\top}$$

Most importantly, Songlin Yang et al find a efficient parallel strategy on GPUs, which makes this model have the potential to become a part of modern LLM

Parallelizing Linear Transformers with the Delta Rule over Sequence Length. NeurIPS 2024.

Before We Start

Limitation of limited state space.

Difficult to remember long sequences[1].

Transformers can adapt to a form of dynamic sparsity[2].

...

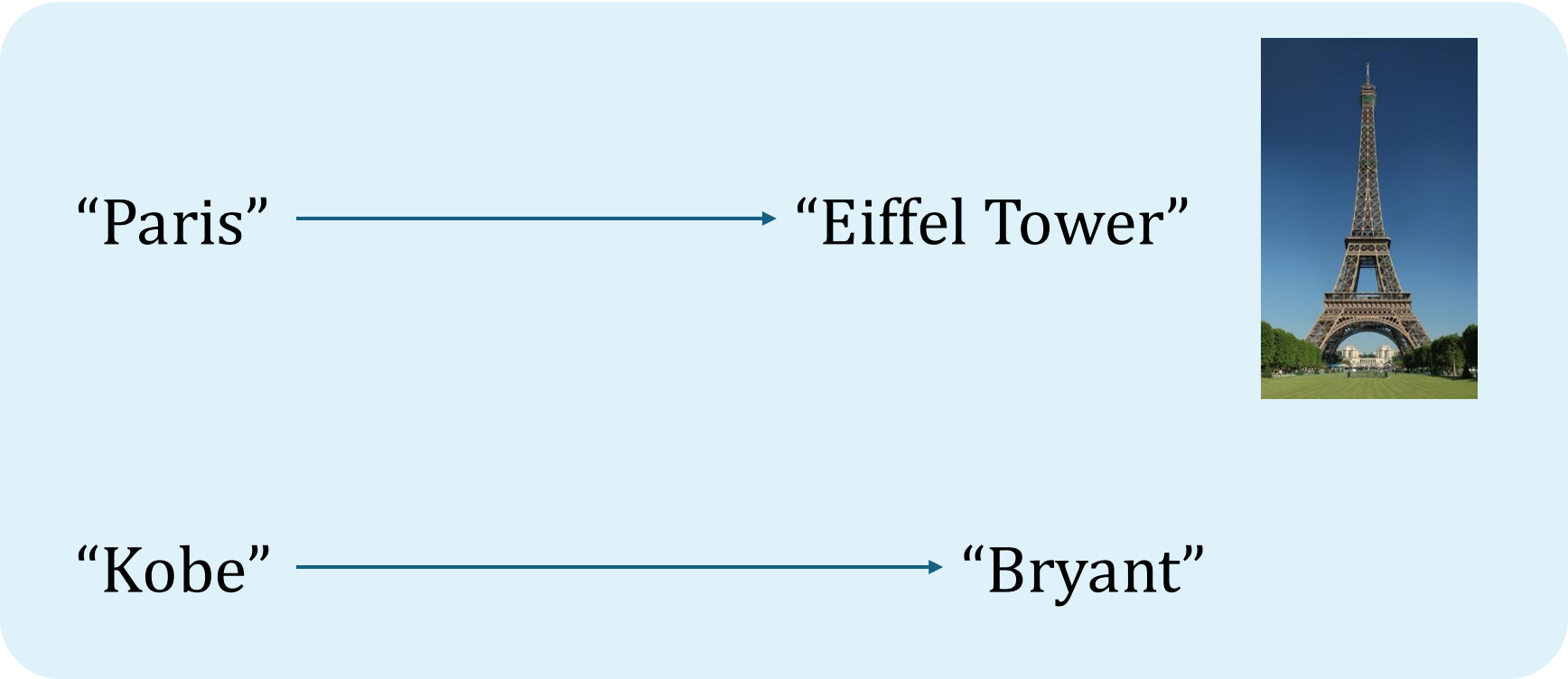
→ Combine Delta Rule with Transformer.

[1] Repeat after me: Transformers are better than state space models at copying. ICML 2024.
[2] When Do Transformers Outperform Feedforward and Recurrent Networks? A Statistical Perspective. Arxiv2025.

Part 1. Associative Memory and Its Capacity

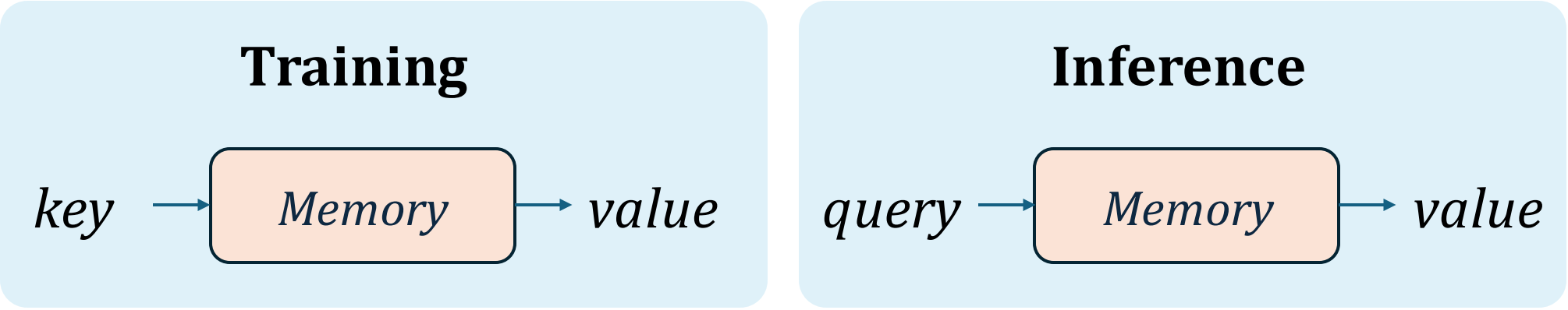
Associative Memory

- Associative memory is defined as the ability to learn and remember the relationship between ~~unrelated~~ items.



Associative Memory

- Associative memory is defined as the ability to learn and remember the relationship between ~~unrelated~~ items.
- Like a dictionary:
  - We have keys and values;
  - Learning is the process of acquiring the *fuzzy* mapping.



Simplest Model for Associative Memory

- Suppose we have key-value pairs $\{(k_i, v_i)\}_t$ where the keys are **orthogonal**. We can store these relationships into an outer-product associative memory (Kohonen 1972):

$$S_t = \sum_{i=1}^{t} v_i k_i^T$$

- Query with $q = k_i$ we get perfect retrieval:

$$o = S_t k_i = v_i k_i^T k_i + \sum_{j \neq i} v_j k_j^T k_i = v_i$$

- This kind of outer-product memory is also known as linear attention.

Measuring Retrieval Error

- What if keys are not orthonormal**?**

$$o = S_t k_i = \underbrace{v_i\left(k_i^T k_i\right)}_{Signal\ \boldsymbol{v}_i c} + \underbrace{\sum_{j \neq i} v_j k_j^T k_i}_{Noise\ \boldsymbol{r}} \approx v_i$$

- Define inverse retrieval SNR (Signal-to-Noise Ratio):

$$SNR^{-1} = \mathbb{E}_{\boldsymbol{v}_j, \boldsymbol{k}_j}\left[\frac{\|\boldsymbol{r}\|^2}{c^2\|\boldsymbol{v}_i\|^2}\right]$$

- A larger value indicates a higher noise component, resulting in lower retrieval accuracy.

Measuring Retrieval Error

- Considering keys and values are i.i.d. standard Gaussian vectors. We can quantitatively calculate the inverse SNR:

$$SNR_{Linear}^{-1} \approx \frac{N}{d}$$

- To attain a target SNR, **the linear model width d must grow linearly with the sequence length** N;

- This explains why linear attention usually has poor retrieval performance.

Kernel Trick

- Introducing a kernel function $\kappa(x, y) = \phi(x)^T \phi(y)$. $\phi(\cdot)$ maps keys into higher-dimensional space.

$$S_t = \sum_{i=1}^{t} v_i \phi(k_i)^T$$

$$o = S_t \phi(q) = \sum_{i=1}^{t} v_j \phi(k_i)^T \phi(q) = \sum_{i=1}^{t} v_i \kappa(k_i, q)$$

- Greater separability among the vectors leads to lower retrieval error.

Kernel Trick

- Also, considering $q = k_i$, and assuming keys and values are i.i.d. standard Gaussian, we get:

$$SNR_{\kappa}^{-1} = N \frac{\mathbb{E}_{\boldsymbol{k}_j}\left[\kappa^2\left(k_j, k_i\right)\right]}{\kappa^2(k_i, k_i)}$$

How much the kernel suppresses irrelevant features

How much the kernel amplifies the matched features

Exp Kernel

- For $\kappa(x, y) = \exp\left(\frac{x^T y}{\tau}\right)$ and $\tau = \sqrt{d}$, we get standard softmax attention without normalization:

$$o = S_t \phi(q) = \sum_{i=1}^{t} v_i \exp\left(\frac{k_i^T q}{\sqrt{d}}\right)$$

- Inverse SNR is:

$$SNR_{\exp}^{-1} = N \exp\left(-\frac{2(\tau - 1)}{\tau^2} d\right)$$

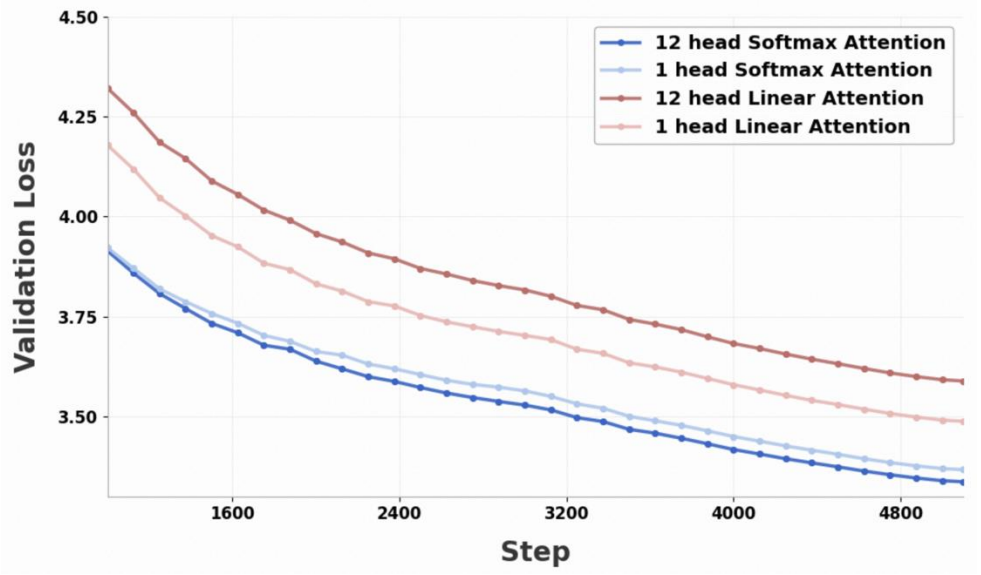
- **From $\boldsymbol{d = \mathcal{O}(N)}$ in linear attention to $\boldsymbol{d = \mathcal{O}(\log^2 N)}$.**

Understanding Transformer Reasoning Capabilities via Graph Algorithms. NeurIPS 2024.

Exp Kernel

$$SNR_{\exp}^{-1} = N \exp\left(-\frac{2(\tau-1)}{\tau^2} d\right)$$

- Increase d improves retrieval (but in most cases d is enough).
- Reducing τ but keeping it above 1 improves retrieval.
- Multihead is good for softmax attention, but not good for linear.



ReLU Kernel

- **FFN is associative memory with** $\kappa(x, y) = \mathrm{ReLU}(x^T y)$:

$$FFN(x) = \sum_{i=1}^{m} W_{V_i}^T \mathrm{ReLU}(W_{K_i} x)$$

- Query is hidden state x, keys and values are learnable weights.

Inverse SNR is :

$$SNR_{\mathrm{ReLU}}^{-1} \approx \frac{N}{2d}$$

We suspect a kernel with lower retrieval precision encourages a more polysemantic key–value memory, which is suitable for FFN.

Architecture Symmetry

Component	Attention	FFN
Kernel	Exp	ReLU / SiLU
Normalization	Yes	None
Multihead	Yes	None
Sparsity	None	MoE
Gating	None	SwiGLU Gating

$$[\boldsymbol{q}_{j,1}; \boldsymbol{q}_{j,2}; \ldots; \boldsymbol{q}_{j,n_h}] = \boldsymbol{q}_t,$$
$$[\boldsymbol{k}_{i,1}; \boldsymbol{k}_{i,2}; \ldots; \boldsymbol{k}_{i,n_h}] = \boldsymbol{k}_i,$$
$$[\boldsymbol{v}_{i,1}; \boldsymbol{v}_{i,2}; \ldots; \boldsymbol{v}_{i,n_h}] = \boldsymbol{v}_i,$$
$$\boldsymbol{o}_{t,.} = \sum_{i=1}^{t} \boldsymbol{v}_{i,.}\, \kappa(\boldsymbol{q}_{t,.}, \boldsymbol{k}_{i,.}),$$
$$\boldsymbol{o}_t = \boldsymbol{W}_O[\boldsymbol{o}_{t,1}; \boldsymbol{o}_{t,2}; \ldots; \boldsymbol{o}_{t,n_h}].$$

- We suspect FFN can benefit from multihead too [1].

[1] *Attention is all you need. Arxiv 2017.* ***v2***

Architecture Symmetry

Component	Attention	FFN
Kernel	Exp	ReLU / SiLU
Normalization	Yes	None
Multihead	Yes	None
Sparsity	None	MoE
Gating	None	SwiGLU Gating

- Integrating MoE into attention we get MoBA.

$$\mathrm{FFN}_{\mathrm{MoE}}(\boldsymbol{x}) = \sum_{e=1}^{E} g_e(\boldsymbol{x}) \left(\sum_{i \in \mathrm{Expert}(e)} \boldsymbol{W}_{V_i}^{(e)} \mathrm{ReLU}(\boldsymbol{W}_{K_i}^{(e)} \boldsymbol{x}) \right).$$

$$\boldsymbol{o}_{t,\mathrm{MoE}} = \sum_{e=1}^{E} g_e(\boldsymbol{q}_t, \{\boldsymbol{k}_i\}_{i \in \mathrm{Expert}(e)}) \left(\sum_{i \in \mathrm{Expert}(e)} \boldsymbol{v}_i \, \kappa(\boldsymbol{q}_t, \boldsymbol{k}_i) \right).$$

Moba: Mixture of block attention for long-context llms. Arxiv 2025

Architecture Symmetry

Component	Attention	FFN
Kernel	Exp	ReLU / SiLU
Normalization	Yes	None
Multihead	Yes	None
Sparsity	None	MoE
Gating	None	SwiGLU Gating

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(\boldsymbol{x}) = \sum_{i=1}^{m} \underbrace{\boldsymbol{W}_{V\,i}}_{\boldsymbol{v}_i} \underbrace{(\boldsymbol{W}_{G_i}^{\top}\boldsymbol{x})}_{g_i(\boldsymbol{x})} \underbrace{\mathrm{Swish}}_{\kappa(\cdot,\cdot)} \Big(\underbrace{\boldsymbol{W}_{K\,i}^{\top}}_{\boldsymbol{k}_i} \underbrace{\boldsymbol{x}}_{\boldsymbol{q}_t} \Big),$$

$$\boldsymbol{o}_{t,\mathrm{gated}} = \sum_{i=1}^{t} \boldsymbol{v}_i \, g_i(\boldsymbol{x}_{i:t}) \, \kappa(\boldsymbol{q}_t, \boldsymbol{k}_i),$$

- Gating like SwiGLU is underexplored. FoX is a special case here：

$$g_i(x_{i:t}) = \prod_{j=i+1}^{t} \alpha_j(x_j)$$

Forgetting transformer: Softmax attention with a forget gate. ICLR 2025

Part 2. Update the Associative Memory

How is associative memory updated?

- Taking the linear model as an example,

- $S_t = \underbrace{\sum_{i=1}^{t-1} v_i k_i^\top}_{S_{t-1}} + v_t k_t^\top$

$$= S_{t-1} - \underbrace{(-v_t k_t^\top)}_{\frac{\partial L_t}{\partial S_{t-1}}}$$

How is associative memory updated?

- Taking the linear model as an example,

- $S_t = \underbrace{\sum_{i=1}^{t-1} v_i k_i^\top}_{S_{t-1}} + v_t k_t^\top$

$$= S_{t-1} - \underbrace{(-v_t k_t^\top)}_{\frac{\partial L_t}{\partial S_{t-1}}}$$

$$L_t(S_{t-1}) = - < S_{t-1} k_t, v_t >$$

How is associative memory updated?

- Taking the linear model as an example,

- $S_t = \underbrace{\sum_{i=1}^{t-1} v_i k_i^\top}_{S_{t-1}} + v_t k_t^\top$

$$= S_{t-1} - \underbrace{(-v_t k_t^\top)}_{\frac{\partial L_t}{\partial S_{t-1}}}$$

$$L_t(S_{t-1}) = - < S_{t-1} k_t, v_t >$$

The loss is unbounded.

How is associative memory updated?

Solution 1: regularize $||S||_F$

- $L_t(S_{t-1}) = - < S_{t-1}k_t, v_t > + \frac{1}{2}||\sqrt{1-\lambda_t}\, S_{t-1}||_F^2$

How is associative memory updated?

Solution 1: regularize $||S||_F$

- $L_t(S_{t-1}) = -< S_{t-1}k_t, v_t > + \frac{1}{2}||\sqrt{1-\lambda_t}\, S_{t-1}||_F^2$

- Update form: $S_t = \lambda_t S_{t-1} + v_t k_t^\top$

This is gated linear model.

How is associative memory updated?

Solution 1: regularize $||S||_F$

- $L_t(S_{t-1}) = -< S_{t-1}k_t, v_t > + \frac{1}{2}||\sqrt{1-\lambda_t}\, S_{t-1}||_F^2$

- Update form: $S_t = \lambda_t S_{t-1} + v_t k_t^\top$

  This is gated linear model.

- Gating implies a bias that older information is less important.

How is associative memory updated?

Solution 2: regularize $||Sk||$

- $L_t(S_{t-1}) = - < S_{t-1}k_t, v_t > + \frac{1}{2}|| S_{t-1}k_t||^2$

$$= \frac{1}{2}||S_{t-1}k_t - v_t||^2 - \frac{1}{2}||v_t||^2$$

How is associative memory updated?

Solution 2: regularize $||Sk||$

- $L_t(S_{t-1}) = -< S_{t-1}k_t, v_t > + \frac{1}{2}|| S_{t-1}k_t||^2$

$$= \frac{1}{2}||S_{t-1}k_t - v_t||^2 - \cancel{\frac{1}{2}||v_t||^2}$$

$$:= \frac{1}{2}||S_{t-1}k_t - v_t||^2$$

How is associative memory updated?

Solution 2: regularize $||Sk||$

- $L_t(S_{t-1}) = \frac{1}{2}||S_{t-1}k_t - v_t||^2$

- Update form: $S_t = S_{t-1}(I - k_t k_t^\top) + v_t k_t^\top$

This is DeltaNet（Delta rule update）.

How is associative memory updated?

Solution 2: regularize $||Sk||$

- $L_t(S_{t-1}) = \frac{1}{2}||S_{t-1}k_t - v_t||^2$

- Update form: $S_t = S_{t-1}(I - k_t k_t^\top) + v_t k_t^\top$

This is DeltaNet（Delta rule update）.

- How to understand the magic gate $(I - k_t k_t^\top)$?

How is associative memory updated?

Understand delta rule

- $S_t = S_{t-1}(I - k_t k_t^\top) + v_t k_t^\top = \sum_{i=1}^{t-1} v_i k_i^\top (I - k_t k_t^\top) + v_t k_t^\top$

$$= S_{t-1} + (v_t - \underbrace{\sum_{i=1}^{t-1} v_i \left(k_i^\top k_t\right)}_{\text{Info overlap with } v_t}) k_t^\top$$

- Delta rule is a smarter gate for erasing overlapping historical info.

How is associative memory updated?

Solution 3: normalization (e.g., $\sum_{i=1}^{t} \kappa(q, k_i)$)

- When t is sufficiently large:

$$L_t(S_{t-1}) \approx \frac{1}{t}(- < S_{t-1}k_t, v_t > + \frac{1}{2}||S_{t-1}||_F^2)$$

- Besides regularizing $||S||_F$, $\frac{1}{t}$ also suppresses numerical explosion.

Part 3. Exploration of Next Generation Models

What is the essential difference?

$$S_{t+1} = S_t + v_t k_t^\top \longrightarrow S_t \to \infty$$

Solution：

- 1) Decay: $S_{t+1} = S_t(\alpha_t I) + v_t k_t^\top$

  Data independent: RetNet,RWKV4
  Data dependent: Gated linear attention,Mamba2,RWKV6

- 2) Delete: $S_{t+1} = S_t(I - k_t k_t^\top) + v_t k_t^\top$

  Without decay: DeltaNet
  With decay: RWKV7, Gated DeltaNet

Intuitively speaking:

The computation of β can be treated as a **prefix sum**, which allows for an efficient implementation.
The computation of u seems to be inherently sequential, requiring an O(t) loop?

Decay

$$S_{t+1} = S_t(\alpha_t I) + v_t k_t^\top$$

$$\beta_t = \prod_{i=1}^{t} \alpha_i$$

$$S_t = \sum_{i=1}^{t} \frac{\beta_t}{\beta_i} v_i k_i^\top$$

Delete

$$S_{t+1} = S_t(I - k_t k_t^\top) + v_t k_t^\top$$

$$u_t = v_t - \sum_{i=1}^{t-1} (k_i \cdot k_t) u_i$$

$$S_t = \sum_{i=1}^{t} u_i k_i^\top$$

Mathematically speaking:

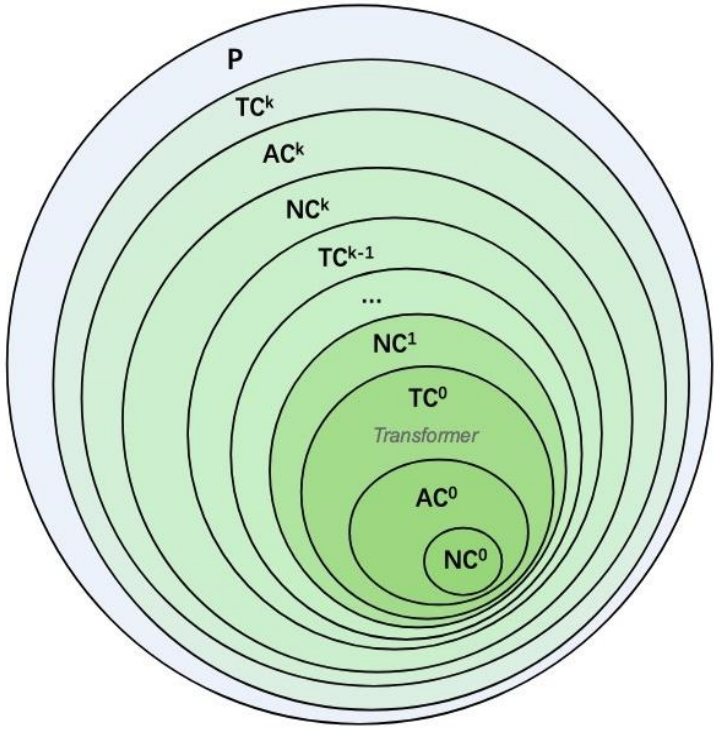
The computation of β and u have **different parallel complexity.**

Three basic questions for determining Parallel Complexity:

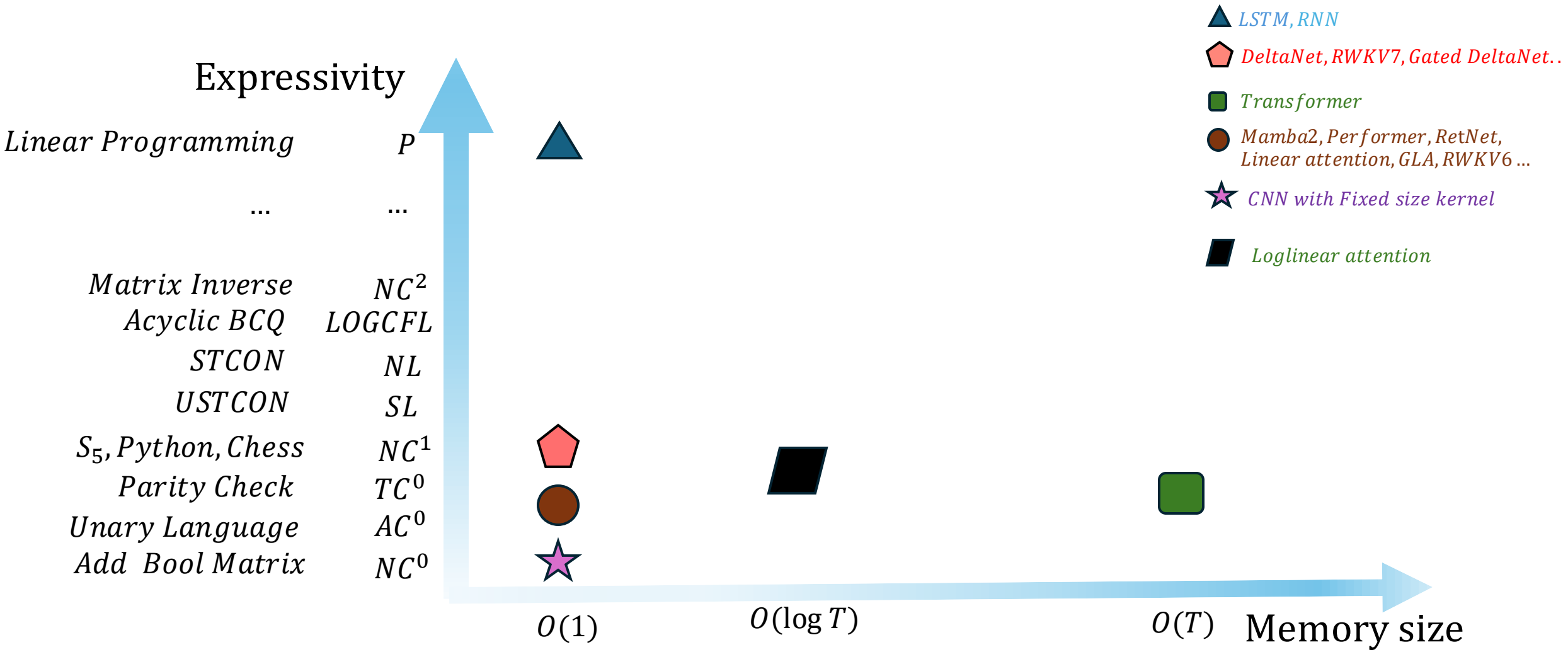
1. *What is the length of the critical path?*
$O(\log^k n) \rightarrow \{TC^k, AC^k, NC^k\}$

2. *How many fan-in one node can receive?*
$2 \rightarrow NC$
unlimit $\rightarrow \{AC, TC\}$

3.*Can we use a Threshold Gate out of OR/AND/NOT?*
Yes $\rightarrow$ TC
No $\rightarrow \{NC, AC\}$



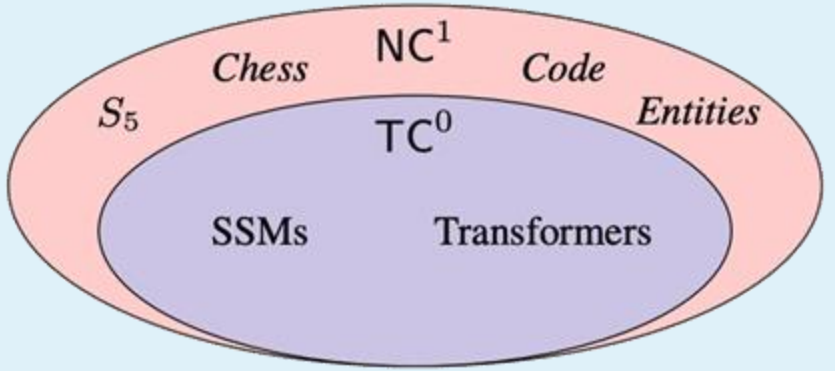
Where is the popular model?



The higher the parallelism, the lower the expressiveness. We list representative tasks of common complexity next to them.

Important tasks beyond TC^0

- NC^1: Python, Chess, Entities…
- Beyond NC^1:Graph connectivity …

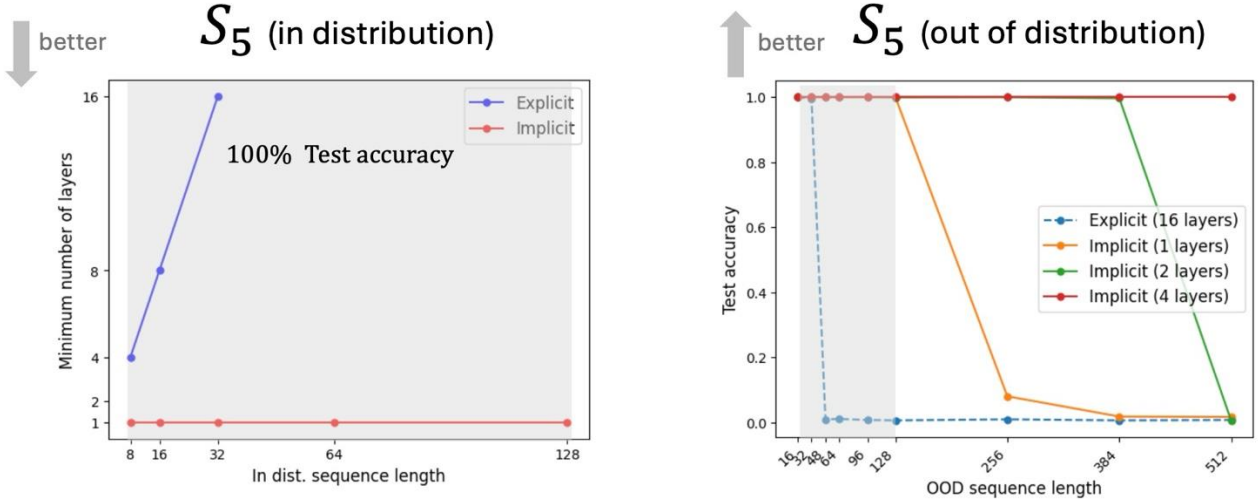


The illusion of state in state-space models. ICML 2024.

When Transformer learn task beyond TC^0

S_5: Track the swap of 5 elements, which is beyond TC^0

- Difficult to learn: context length 32 need 16 layers
- Difficult to generalize: accuracy 0 out of distribution



Implicit Language Models are RNNs: Balancing Parallelization and Expressivity. Arxiv 2025.

DeltaFormer: Make Transformer Beyond TC^0

- Rethink Transformer from delta rule:

Delta rule: $S_t = S_{t-1}(I - k_t k_t^\top) + v_t k_t^\top$

+Kernel Trick: $\kappa(k_i, k_j) = \phi(k_i)^T \phi(k_j)$

$S_t = S_{t-1}(I - \phi(k_t)\phi(k_t^\top)) + v_t \phi(k_t^\top)$

Derivation of DeltaFormer

$$\kappa(k_i, k_j) = \phi(k_i)^T \phi(k_j)$$

$$S_t = S_{t-1}(I - \phi(k_t)\phi(k_t^\top)) + v_t \phi(k_t^\top)$$

By method of undetermined coefficients

$$u_t = v_t - \sum_{i=1}^{t-1} \kappa(k_t, k_i) u_i$$

$$o_t = \sum_{i=1}^{t} \kappa(q_t, k_i) u_i$$

General form

$$u_t = \alpha_t v_t - \beta_t \sum_{i=1}^{t-1} \kappa_1(w_t, k_i) u_i$$

$$o_t = \sum_{i=1}^{t} \kappa_2(q_t, k_i) u_i$$

Efficient chunk-wise implementation

- Recurrent: $u_t = v_t - \sum_{i=1}^{t-1} \kappa(k_t, k_i) u_i$
- Parallelize : $U = V - AU \Rightarrow U = (I + A)^{-1} V$
- Chunk-wise: $U_C = V_c - A_c U_c - A_{c,p} U_p$

$\Rightarrow U_C = (I + A_c)^{-1}(V_c - A_{c,p} U_p)$

Efficient chunk-wise implementation

Current Implementation

- Recurrent (Serial): O(T) loop
- Parallelize(Torch trsv):
  Too much I/0.
- Chunk-wise(dfpa): O(T/C) loop
  with $O(T^2D + TCD + TC^2)$ flops

Performance

Expected performance results on H800:

- [B, H, T, D] = 2, 32, 8192, 128

Forward:

```
serial          time 279.908 ms
Torch trsv      time 102.156 ms
dfpa            time 12.705 ms
```

Backward:

```
Torch trsv backward time 275.659 ms
dfpa       backward time 25.713 ms
```

Reference time consumption in training

```
Typical normal attention    time 46.839 ms
MLP hto4h GeMM              time 2.918 ms
FlashAttention forward      time 3.406 ms
FlashAttention backward     time 9.348 ms
```

Make Transformer beyond TC^0

- More expressive:

Can track the exchange of n elements with $d = O(\log n)$.

In the Lemma 2 of rwkv7, they tracking 5 elements used 5 dimensions

- Native compression:

If we read out and rewrite KU cache every O(n) step, the actually KU cache is $O(nlogn)$.

Theorem 1 (State Exchange)

Assumption 1: *There exist n state points on a d-dimensional unit sphere, and the absolute value of the inner product of any two distinct state points is less than or equal to $\epsilon(d,n)$, which means:*

$$\exists \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d \quad s.t. \quad \|\boldsymbol{x}_i\|_2 = 1 (\forall i), \quad \max_{i \neq j} |\boldsymbol{x}_i^\top \boldsymbol{x}_j| \leq \epsilon(d,n) < \frac{1}{8}.$$

Assumption 2: *There is a function f satisfies:*

$$\forall \boldsymbol{x} \in \{-1, 0, 1, 2\}, \forall \tilde{\boldsymbol{x}} \in U(\boldsymbol{x}, 4\epsilon(d,n)) : \quad f(\tilde{\boldsymbol{x}}) = \boldsymbol{x}.$$

Consider initializing n key-value pairs as $\{(\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_n, \boldsymbol{v}_n)\}$. The keys $\{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n\}$ lie on a d-dimensional unit sphere and satisfies Assumption 1, which means:

$$\forall i, j \in \{1, \ldots, n\}, i \neq j : \quad \|\boldsymbol{k}_i\|_2 = 1, \quad |\boldsymbol{k}_i^\top \boldsymbol{k}_j| \leq \epsilon(n,d).$$

Define an attention mechanism as follows:

$$\boldsymbol{u}_t = \boldsymbol{v}_t - \sum_{i=1}^{t-1} f(\boldsymbol{k}_i^\top \boldsymbol{k}_t) \boldsymbol{u}_i, \quad \boldsymbol{o}_t = \sum_{i=1}^{t} f(\boldsymbol{q}_t^\top \boldsymbol{k}_i) \boldsymbol{u}_i,$$

where $f(\cdot)$ satisfies Assumption 2 and it is noted that $\forall\, i \in \{1, \ldots, n\}$, since $f(\boldsymbol{k}_i^\top \boldsymbol{k}_i) = 0$, we have $\boldsymbol{u}_i = \boldsymbol{v}_i$.

At the current step t, $t > n$, the value corresponding to $\boldsymbol{k}_i$ is denoted by $\tilde{\boldsymbol{v}}_i$, $i \in \{1, \ldots, n\}$. Note that, after $t - 1 - n$ exchanges, $\tilde{\boldsymbol{v}}_i$ is not necessarily equal to the initially assigned $\boldsymbol{v}_i$. $\forall\, 1 \leq t_2 < t_1 \leq n$, to exchange the stored values $\tilde{\boldsymbol{v}}_{t_1}$ and $\tilde{\boldsymbol{v}}_{t_2}$ corresponding to $\boldsymbol{k}_{t_1}$ and $\boldsymbol{k}_{t_2}$, it suffices to construct:

$$\boldsymbol{k}_t = \boldsymbol{k}_{t_1} - \boldsymbol{k}_{t_2}, \quad \boldsymbol{v}_t = 0$$

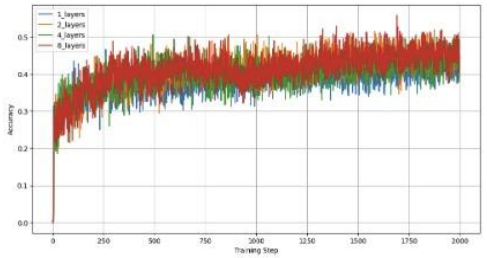
When retrieving the values:

- *Query $\boldsymbol{q}_t = \boldsymbol{k}_{t_1}$, then $\boldsymbol{o}_t = \tilde{\boldsymbol{v}}_{t_2}$;*
- *Query $\boldsymbol{q}_t = \boldsymbol{k}_{t_2}$, then $\boldsymbol{o}_t = \tilde{\boldsymbol{v}}_{t_1}$;*
- *Query $\boldsymbol{q}_t = \boldsymbol{k}_{t_3}$, $1 \leq t_3 \leq n$, $t_3 \neq t_1, t_2$, then $\boldsymbol{o}_t = \tilde{\boldsymbol{v}}_{t_3}$.*

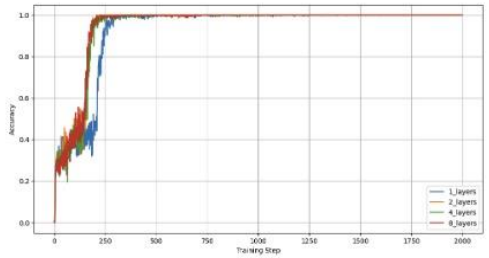
This implies the exchange of values corresponding to k_{t_1} and k_{t_2} is completed.

Learn from data -- Tracking

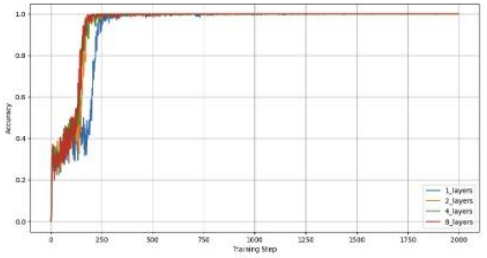
- **DeltaFormer with any kernel surpass Transformer in tracking.**
- Accuracy: 8 layers Transformer < 0.50,
  1 layers DeltaFormer can reach 1.00.



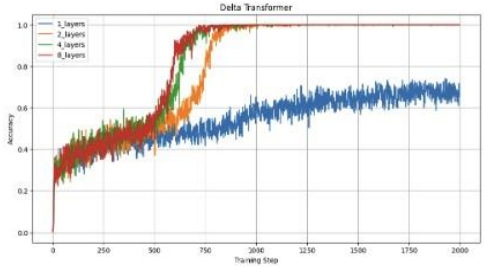
(a) Transformer.



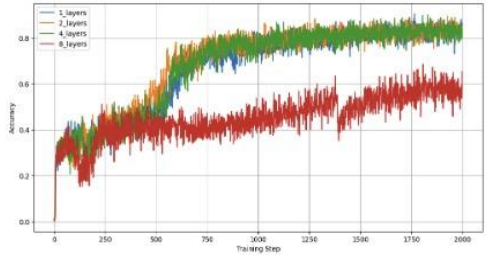
(b) DeltaFormer. ($\kappa_1(x,y) = \lfloor x^\top y \rfloor$)



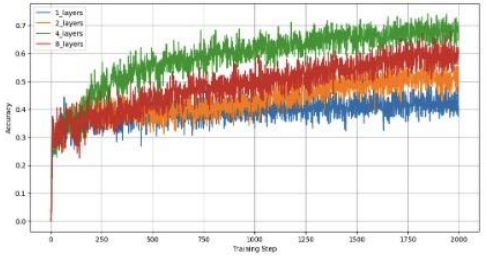
(c) DeltaFormer. ($\kappa_1(x,y) = x^\top y$)



(d) DeltaFormer. ($\kappa_1(x,y) = \max(x^\top y, 0)$)



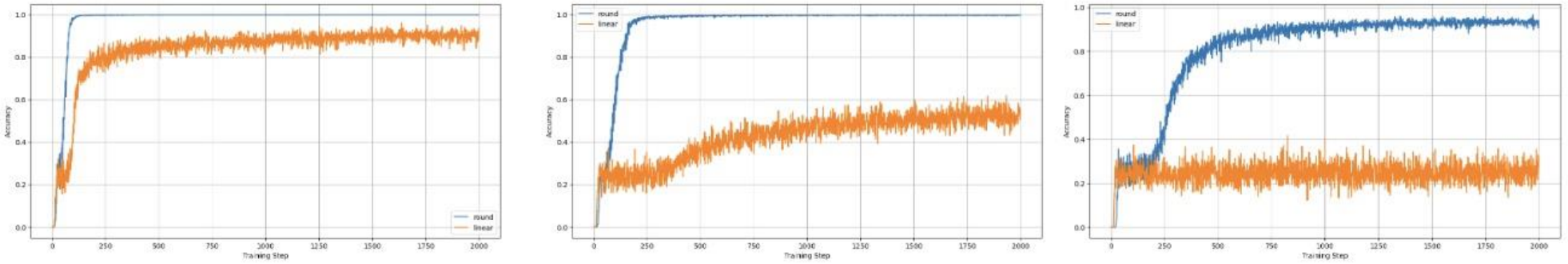
(e) DeltaFormer. ($\kappa_1(x,y) = \exp(x^\top y)$)



(f) DeltaFormer. ($\kappa_1(x,y) = softmax(x^\top y)$)

Learn from data -- Tracking

- **Nonlinear kernel has more capacity than linear.**
- Track n elements with dim d < n:
  linear → drop much , nonlinear > 0.95.



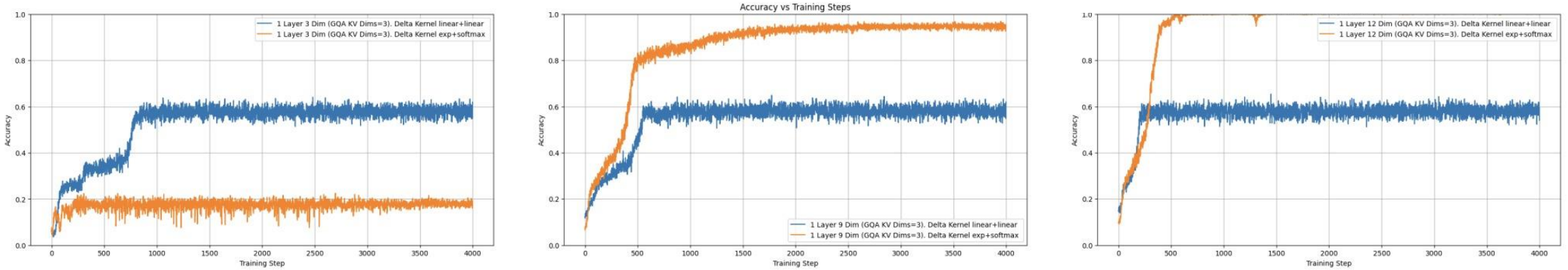
(a) $d = 128,\ n = 128.$ **(b)** $d = 128,\ n = 256.$ **(c)** $d = 128,\ n = 512.$

Figure 5 Comparison of DeltaFormer with $\kappa(x, y) = x^\top y$ and $\kappa(x, y) = \lfloor x^\top y \rfloor$.

Learn from data -- Tracking

- **GQA like method enhance nonlinear kernel with the same KU cache**
- Track 5 elements with a kv head with dim 3, but we have more "query" w.

$$\boldsymbol{u}_t = \alpha_t \boldsymbol{v}_t - \beta_t \sum_{i=1}^{t-1} \kappa_1(\boldsymbol{k}_i, \boldsymbol{w}_t)\boldsymbol{u}_i, \quad \Rightarrow \quad \boldsymbol{u}_t = \alpha_t \boldsymbol{v}_t - \beta_t \sum_{i=1}^{t-1} \sum_{j=1}^{G} a_j \kappa_1(\boldsymbol{k}_i, \boldsymbol{w}_t^j)\boldsymbol{u}_i,$$

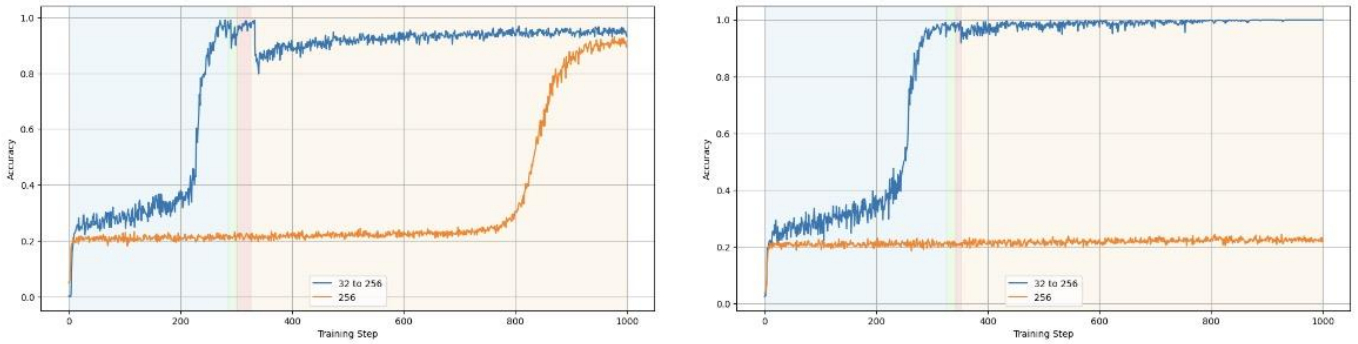


(a) Query number $= 1$. **(b)** Query number $= 3$. **(c)** Query number $= 4$.

Figure 8 Comparison of DeltaFormer with different κ_1 and κ_2 under different query heads numbers.

Learn from data -- Tracking

- Curriculum learning: when accuracy reach 0.99, context length × 2
- With curriculum learning: RoPE can NOT reach 1.00, NoPE can reach 1.00.
- Without curriculum learning: RoPE can learn, NoPE can NOT learn.

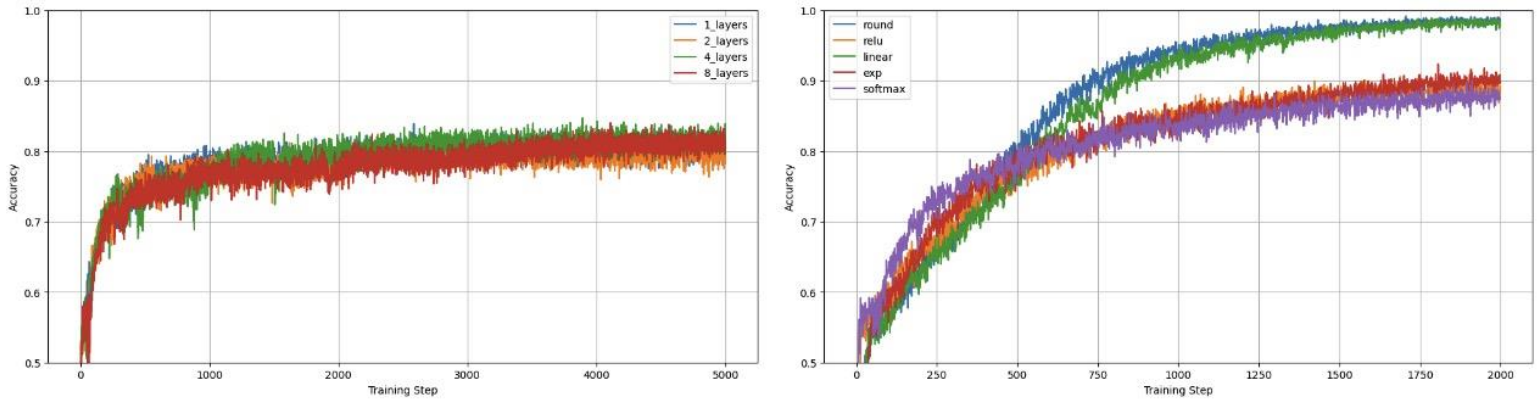


(a) DeltaFormer with RoPE. **(b)** DeltaFormer with NoPE.

Figure 6 Comparison of DeltaFormer using different learning strategy and position embedding. Each use $\kappa_1(x, y) = \lfloor x^\top y \rfloor$. "32 to 256" means that the initial training length is 32, which means the number of swaps is 32. When the accuracy reaches 0.99, the training length will be doubled until it reaches 256. Each color gradient in the image represents a doubling of the training length. And "256" means that the model is trained on a training length of 256 from the beginning. The y-axis reflects the accuracy at the current training length.

Learn from data – Reachability of DAG

- The reachability of directed acyclic graphs:
  Transformer get 0.8, DeltaFormer can reach 1.00.

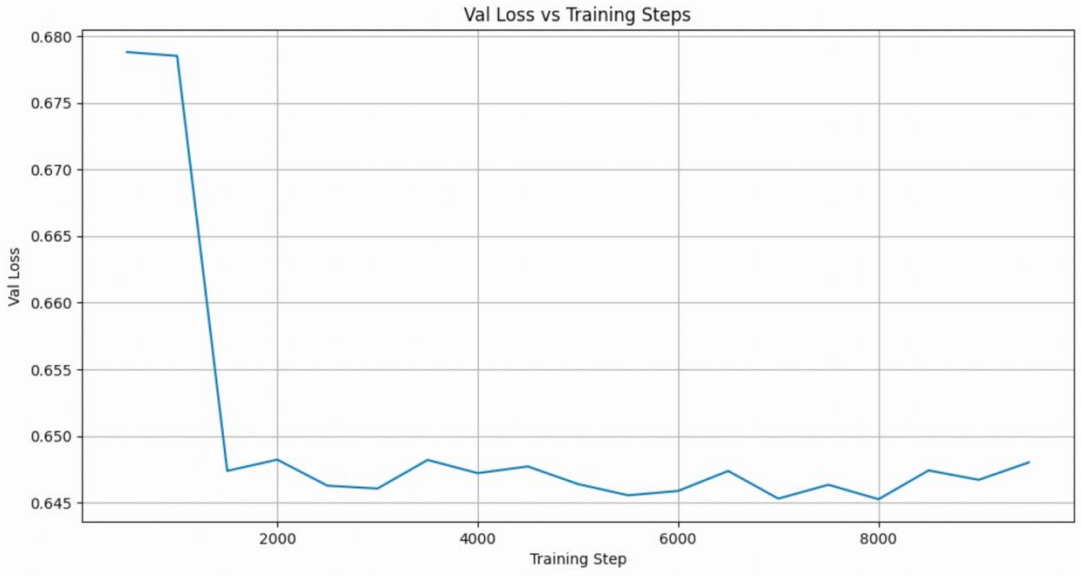


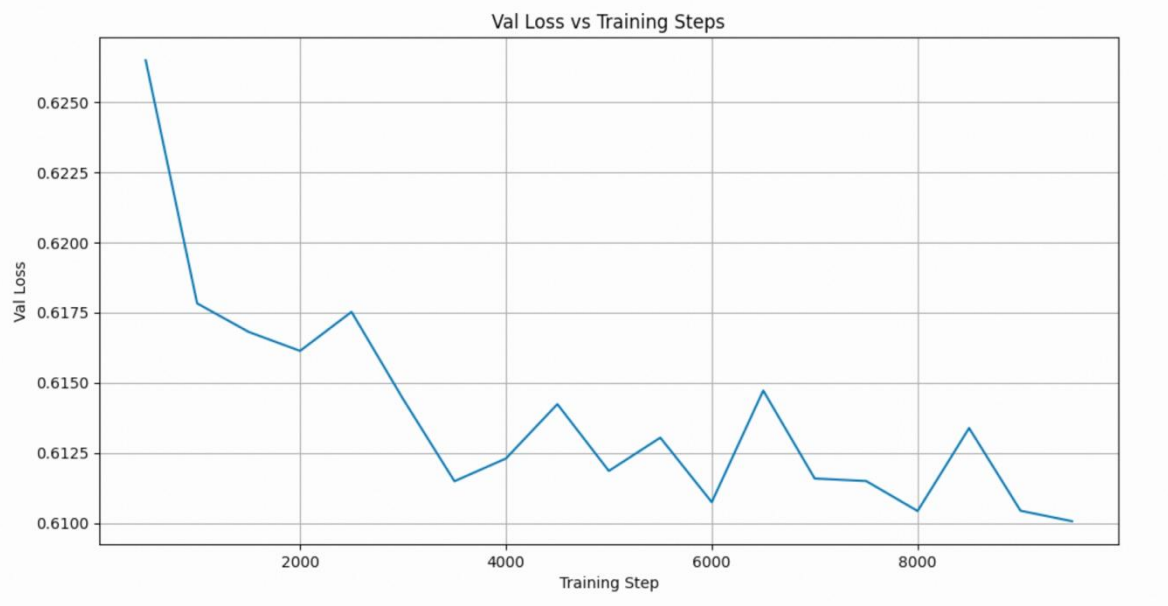
(a) Transformer. **(b)** DeltaFormer.

Figure 7 Comparison of Transformer and DeltaFormer using different similarity functions $\kappa_1(\cdot)$ for performing swapping tasks. For $\kappa_2(\cdot)$, we use the softmax function to maintain consistency with Transformer. Pay attention to the scale of the y-axis.

Learn from data – Dyck grammar

- Example: “(()())(())”



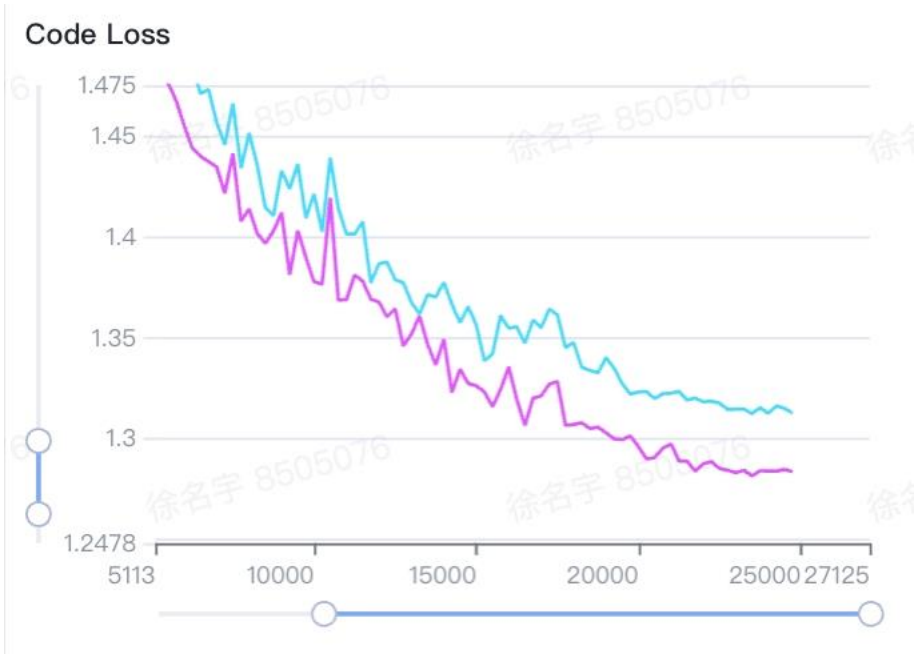


Context length = 64, next token prediction, compress Dyck grammar → Better compression ratio

$u_t = v_t + u_{t-1}$ Can record whether there are more left parentheses than right parentheses at present.

Learn from data – From Dyck to real code

- 14B total parameters model, 500B training token

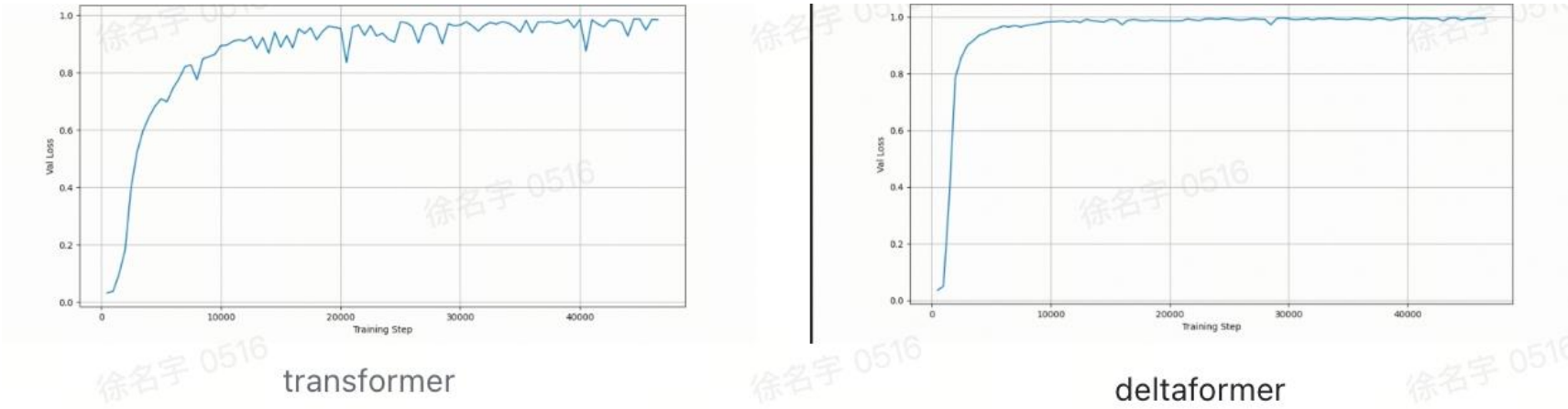


Total loss: DeltaFormer lead baseline by 0.005
Code loss: DeltaFormer lead baseline by 0.03

This implies that the model‘s ability to compress more complex knowledge exceeds that of the Transformer.

Learn from data – Induction heads

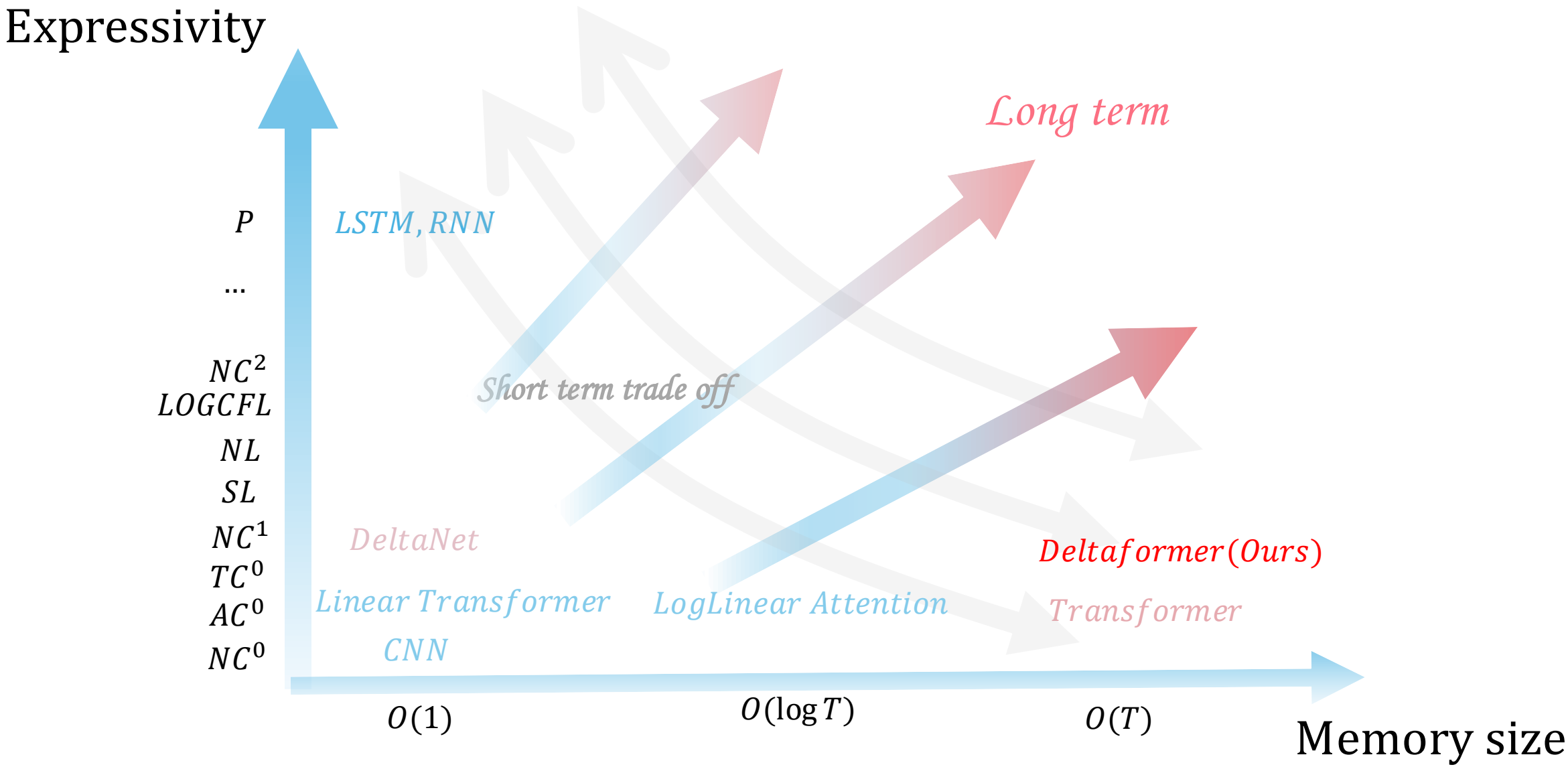
- Better learn induction heads: "… AB… A" predict "B"



The calculation of u can to some extent replace the function of copy heads, thereby promoting the formation of induction heads

DeltaFormer – Future work

- **Optimization**. -- Stability training and scaling ..
- **Fine grained design**. --GQA / Different kernel / Head Nums/ Head Dims / PE / Sparse…
- **Implementation**. -- Fast..
- **Practical**. -- Expressivity/ KU cache/ Flops Trade off..
- More expressive but also practical model.


Expressivity
P
...
NC^2
LOGCFL
NL
SL
NC^1
TC^0
AC^0
NC^0
LSTM, RNN
Long term
Short term trade off
DeltaNet
Deltaformer(Ours)
Linear Transformer
LogLinear Attention
Transformer
CNN
$O(1)$
$O(\log T)$
$O(T)$
Memory size

Thanks for listening!

Q & A