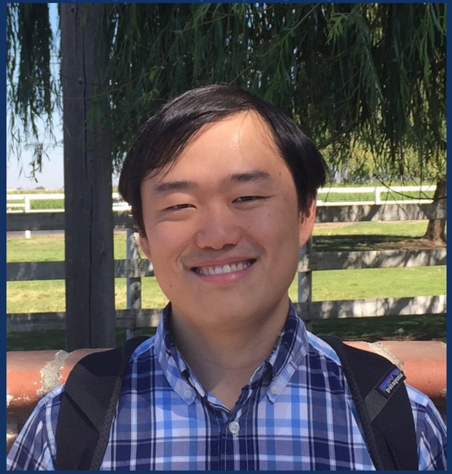
Demystifying Attention in Multi-layer Transformer and its application for Large Language Models

Yuandong Tian
Research Scientist and Manager

Meta AI (FAIR)

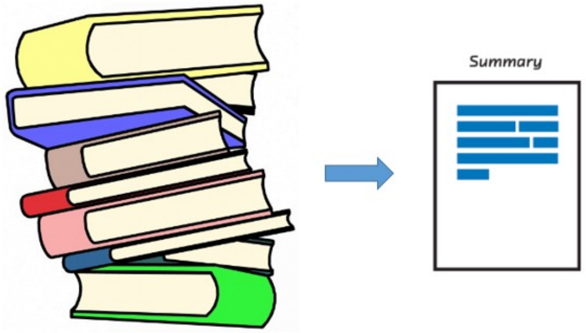


Large Language Models (LLMs)

Content Creation

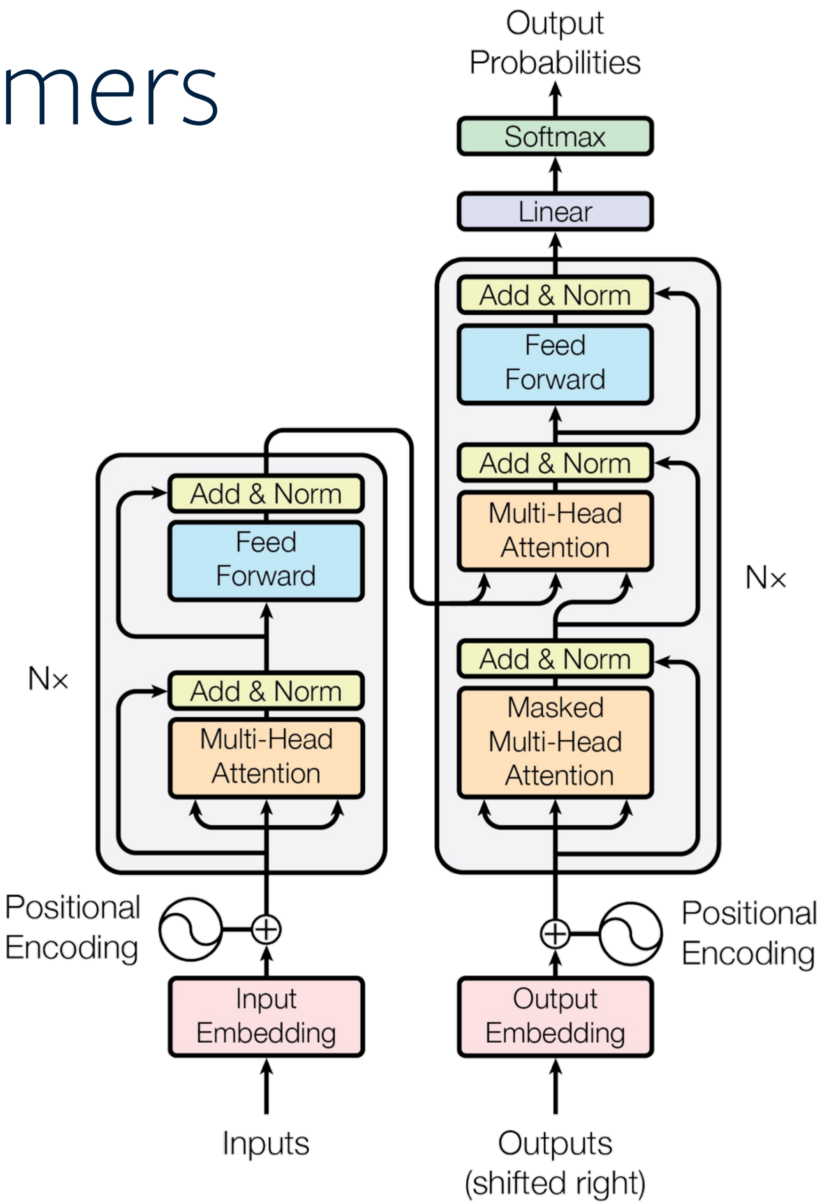
AI Agents

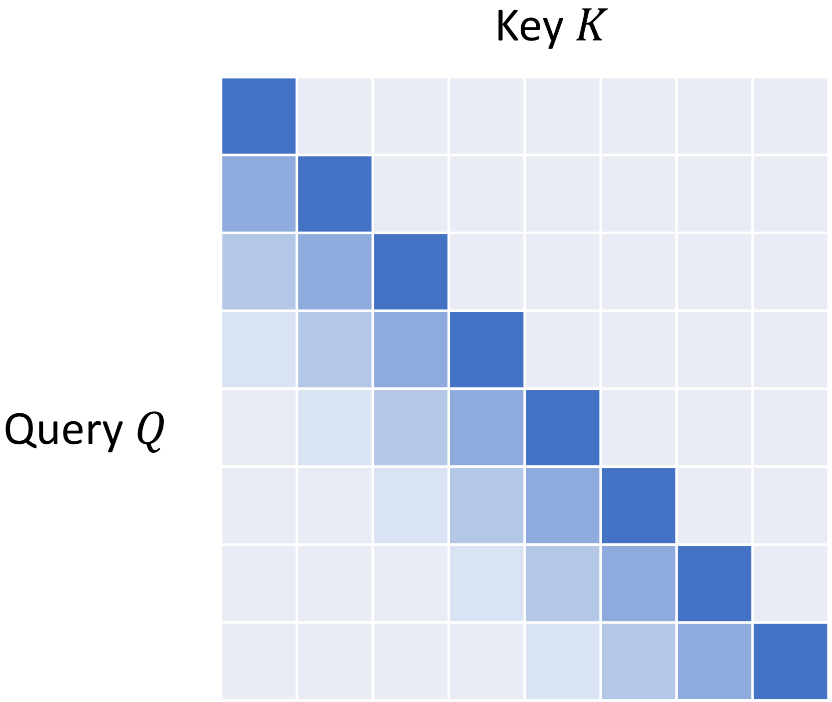
AI co-pilot



Summarization

Transformers

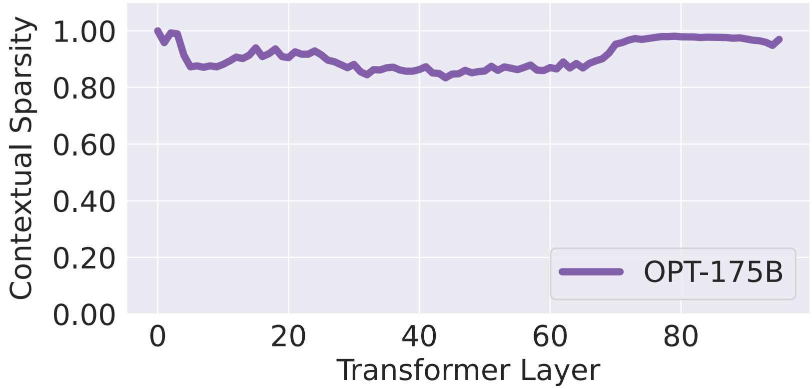


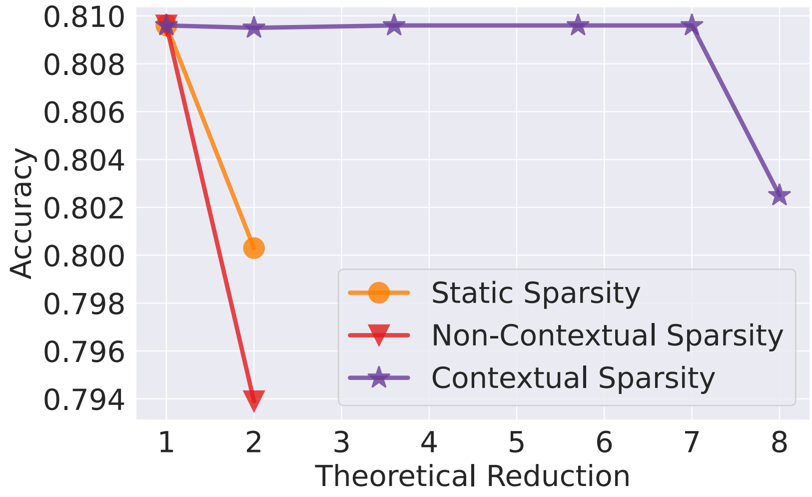


Attention mechanism

[A. Vaswani et al, Attention is all you need, NeurIPS'17]

Contextual Sparsity happens beyond Attention





Key Observation

Keeping only **high activation (contextual!)** in attention/MLP

- results in 85% structured sparsity
  - 80% attention, 95% MLP
- leads to 7× potential parameter reduction for each input
- maintains same accuracy

Contextual sparsity widely exists in pre-trained models, e.g., OPT /LLaMA /Bloom/GPT

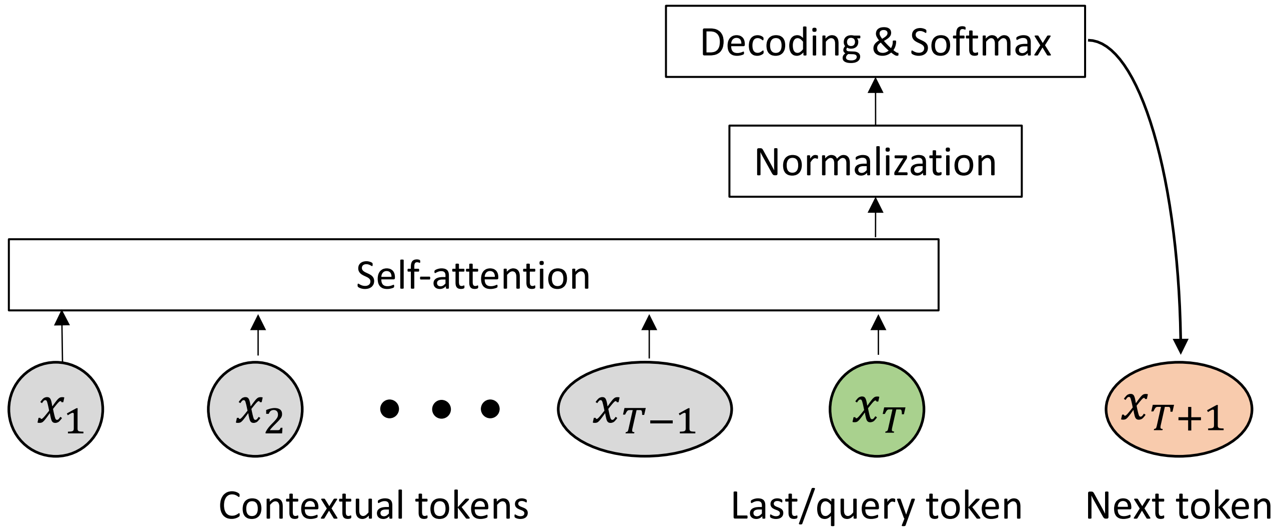
[*Z. Liu et al*, Deja vu: Contextual sparsity for efficient LLMs at inference time, *ICML'23 (oral)*]

Part I

Understanding Learning Mechanism of Transformer

Understanding Attention in 1-layer Setting

$U = [\boldsymbol{u}_1, \boldsymbol{u}_2, \dots \boldsymbol{u}_M]^T$: token embedding matrix



Self-attention

$$\hat{\boldsymbol{u}}_T = \sum_{t=1}^{T-1} b_{tT} \boldsymbol{u}_{x_t} = U^T X^T \boldsymbol{b}_T$$

$$b_{tT} := \frac{\exp(\boldsymbol{u}_{x_T}^\top W_Q W_K^\top \boldsymbol{u}_{x_t} / \sqrt{d})}{\sum_{t=1}^{T-1} \exp(\boldsymbol{u}_{x_T}^\top W_Q W_K^\top \boldsymbol{u}_{x_t} / \sqrt{d})}$$

Normalized version $\tilde{\boldsymbol{u}}_T = U^T \mathrm{LN}(X^T \boldsymbol{b}_T)$

Objective:

$$\max_{W_K, W_Q, W_V, U} J = \mathbb{E}_D \left[\boldsymbol{u}_{x_{T+1}}^T W_V \tilde{\boldsymbol{u}}_T - \log \sum_l \exp(\boldsymbol{u}_l^T W_V \tilde{\boldsymbol{u}}_T) \right]$$

[*Y. Tian et al*, Scan and Snap: Understanding Training Dynamics and Token Composition in 1-layer Transformer, *NeurIPS'23*]

Reparameterization

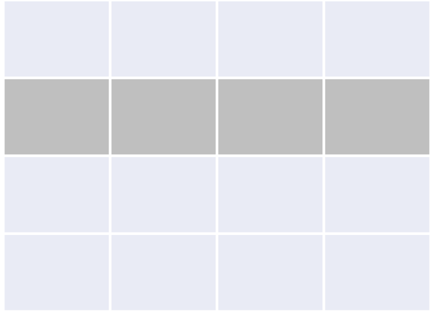
- Parameters W_K, W_Q, W_V, U makes the dynamics complicated.

- Reparameterize the problem with independent variable Y and Z
  - $Y = UW_V^T U^T$
  - $Z = UW_Q W_K^T U^T$ (pairwise logits of self-attention matrix)

- Then the dynamics becomes easier to analyze

Training dynamics of Y and Z

$Z =$



$\boldsymbol{z}_m$: All logits of the contextual tokens when attending to last token $x_T = m$

Training Dynamics:

$$\dot{Y} = \eta_Y \mathrm{LN}(X^T \boldsymbol{b}_T)(\boldsymbol{x}_{T+1} - \boldsymbol{\alpha})^T$$

$$\dot{Z} = \eta_Z \boldsymbol{x}_T (\boldsymbol{x}_{T+1} - \boldsymbol{\alpha})^T Y^T \frac{P^{\perp}_{X^T \boldsymbol{b}_T}}{\|X^T \boldsymbol{b}_T\|_2} X^T \mathrm{diag}(\boldsymbol{b}_T) X$$

Here $Z = [\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_M]^T$, each $\boldsymbol{z}_m \in \mathbb{R}^M$ is the attention score for query/last token m:

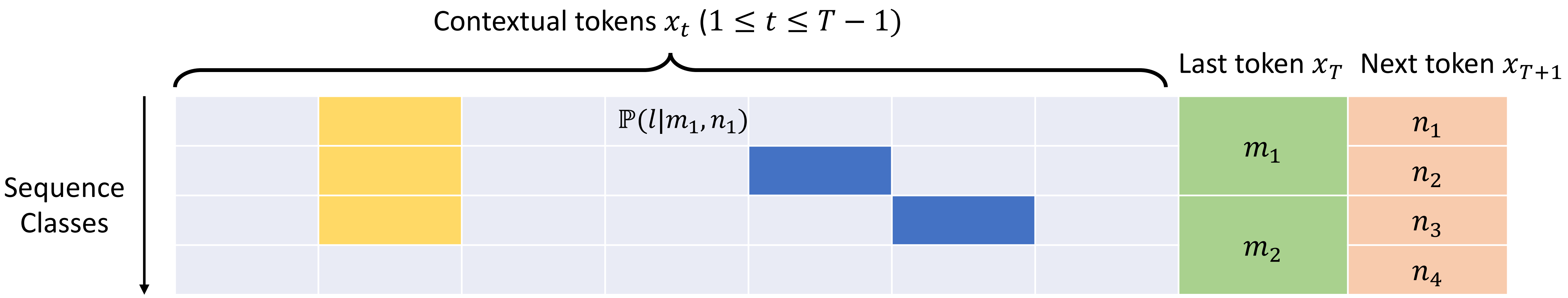
$$\dot{\boldsymbol{z}}_m = \eta_Z X^\top[i] \mathrm{diag}(\boldsymbol{b}_T[i]) X[i] \frac{P^{\perp}_{X^\top[i]\boldsymbol{b}_T[i]}}{\|X^\top[i]\boldsymbol{b}_T[i]\|_2} Y(\boldsymbol{x}_{T+1}[i] - \boldsymbol{\alpha}[i])$$

Major Assumptions

- No positional encoding
- Sequence length $T \to +\infty$
- Learning rate of decoder Y larger than self-attention layer Z ($\eta_Y \gg \eta_Z$)
- Other technical assumptions

Data Distribution

$x_t \in [M]$ for $1 \le t \le T$
$x_{T+1} \in [K]$
$K \ll M$



Distinct tokens: There exists unique n so that $\mathbb{P}(l|n) > 0$
Common tokens: There exists multiple n so that $\mathbb{P}(l|n) > 0$

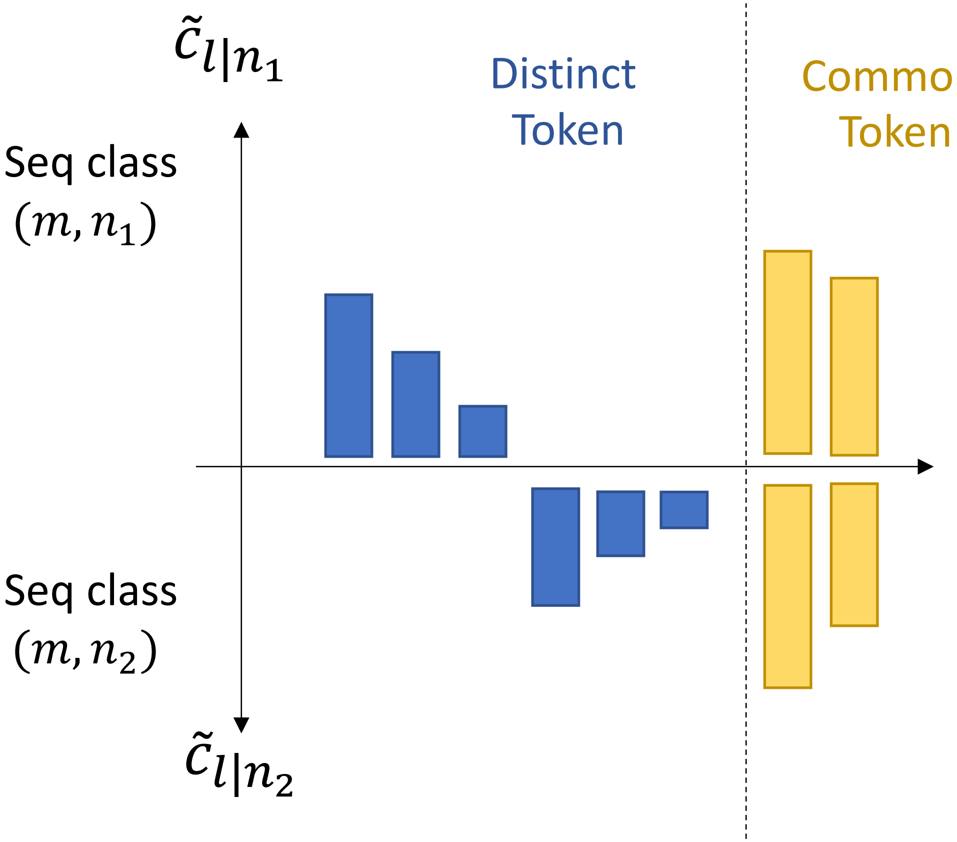
$\mathbb{P}(l|m,n) = \mathbb{P}(l|n)$ is the conditional probability of token l given last token $x_T = m$ and $x_{T+1} = n$

Assumption: $m = \psi(n)$, i.e., no next token shared among different last tokens

Question: Given the data distribution, how does the self-attention layer behave?

Overall Picture of the Training Dynamics

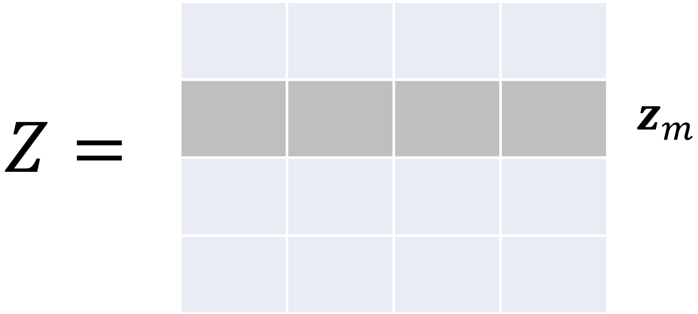
<u>At initialization</u>



Co-occurrence probability

$$\tilde{c}_{l|n_1} := \mathbb{P}(l|m, n_1) \exp(z_{ml})$$

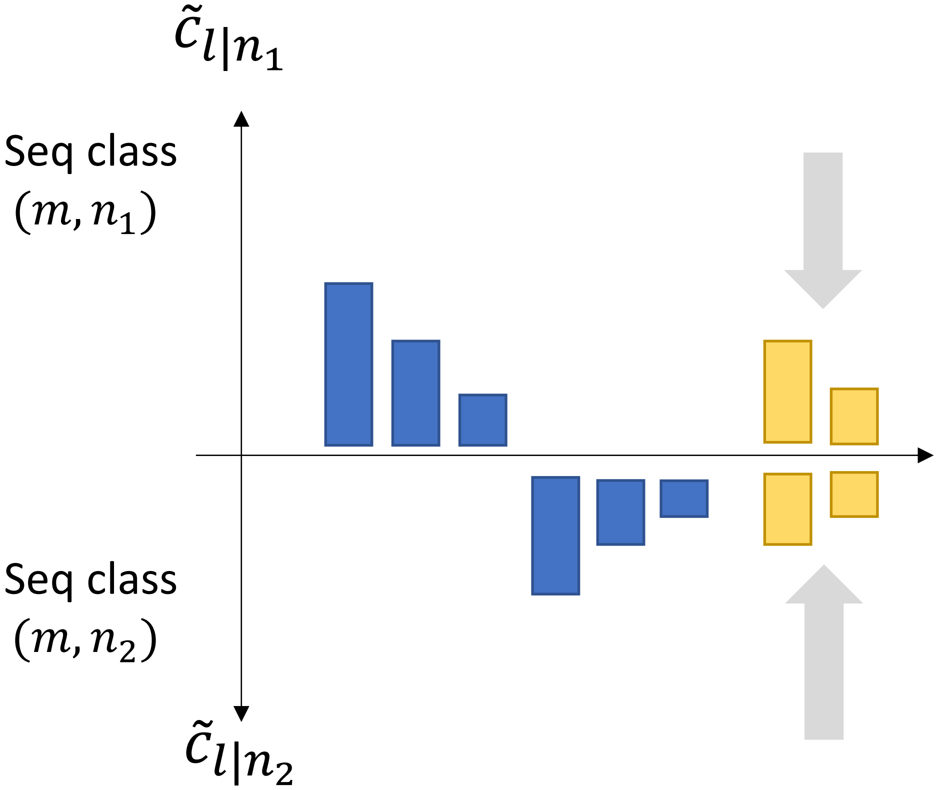
Initial condition: $z_{ml}(0) = 0$



$\boldsymbol{z}_m$: All logits of the contextual tokens when attending to last token $x_T = m$

Overall Picture of the Training Dynamics

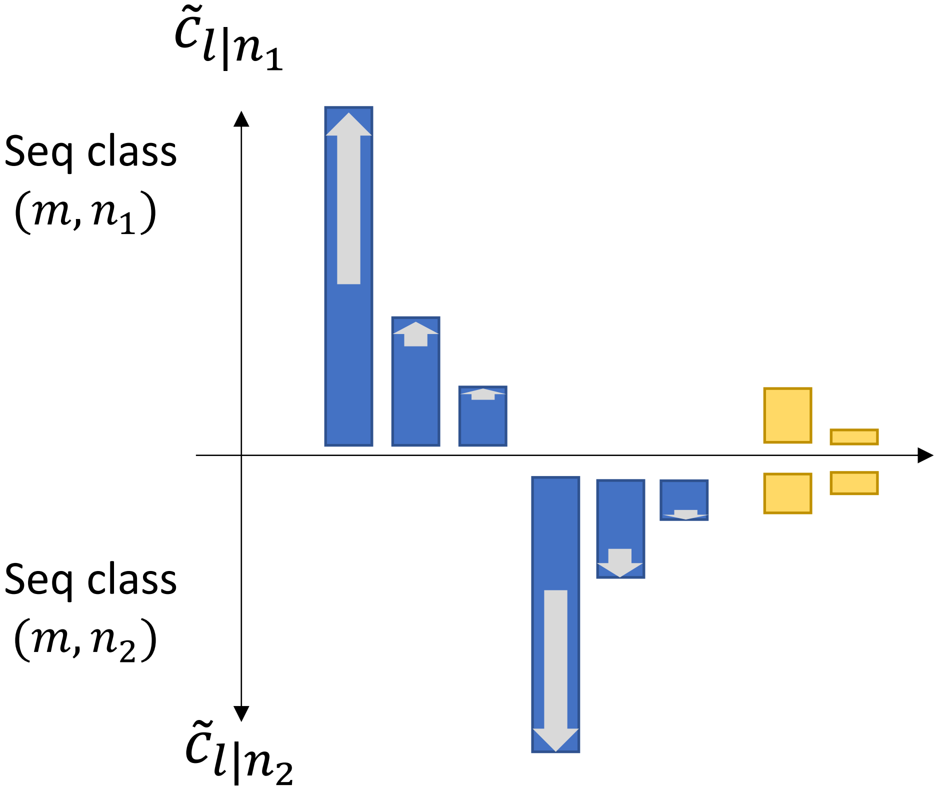
<u>Common Token Suppression</u>



(a) $\dot{z}_{ml} < 0$, for common token l

Overall Picture of the Training Dynamics

<u>Winners-emergence</u>



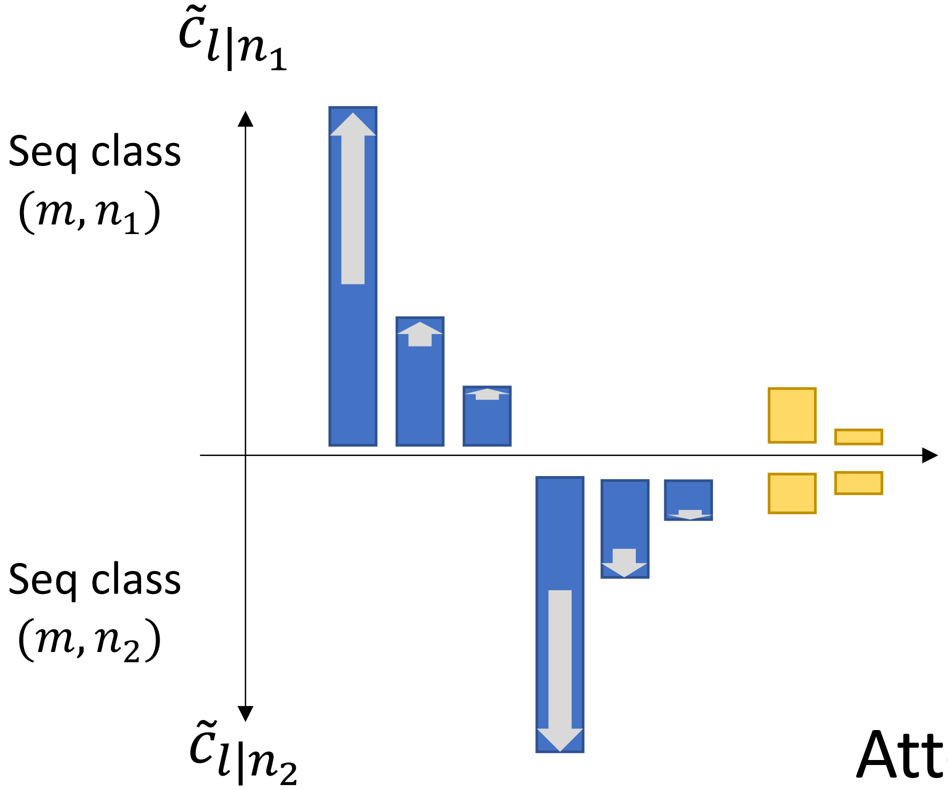
(a) $\dot{z_{ml}} < 0$, for common token l

(b) $\dot{z_{ml}} > 0$, for distinct token l

Learnable TF-IDF (Term Frequency, Inverse Document Frequency)

Overall Picture of the Training Dynamics

<u>Winners-emergence</u>



(a) $\dot{z}_{ml} < 0$, for common token l

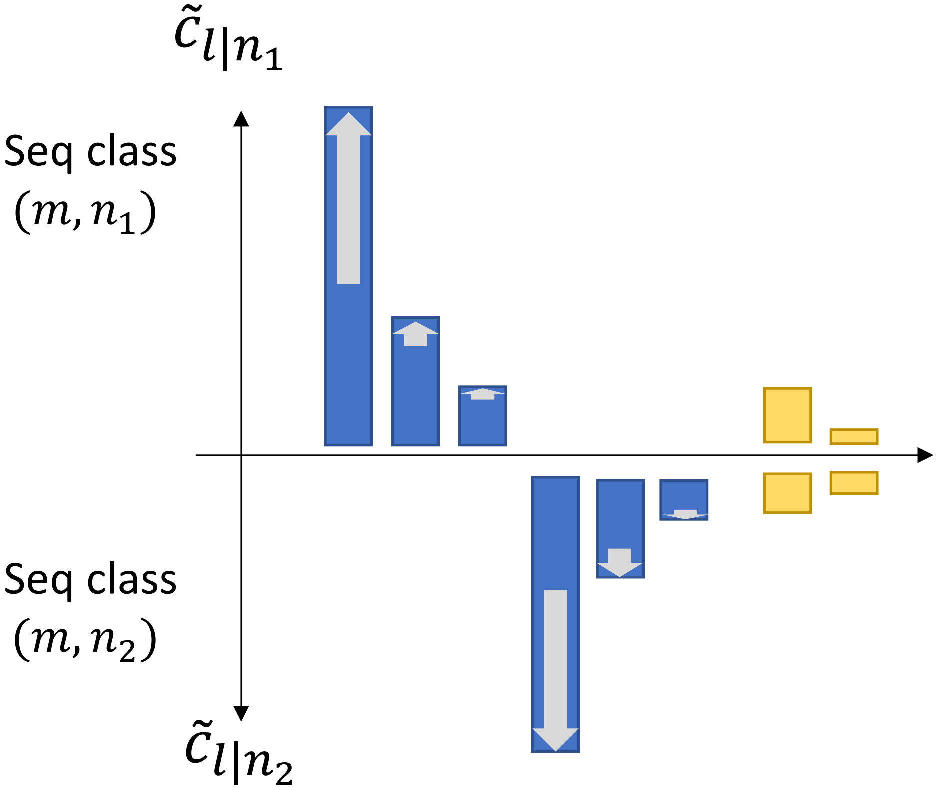
(b) $\dot{z}_{ml} > 0$, for distinct token l

(c) $z_{ml}(t)$ grows faster with larger $\mathbb{P}(l|m, n)$

Attention looks for **discriminative** tokens that **frequently co-occur** with the query.

Overall Picture of the Training Dynamics

Winners-emergence



(c) $z_{ml}(t)$ grows faster with larger $\mathbb{P}(l|m,n)$

Theorem 3 Relative gain $r_{l/l'|n}(t) := \frac{\tilde{c}^2_{l|n}(t)}{\tilde{c}^2_{l'|n}(t)} - 1$ has a close form:

$$r_{l/l'|n}(t) = r_{l/l'|n}(0)\chi_l(t)$$

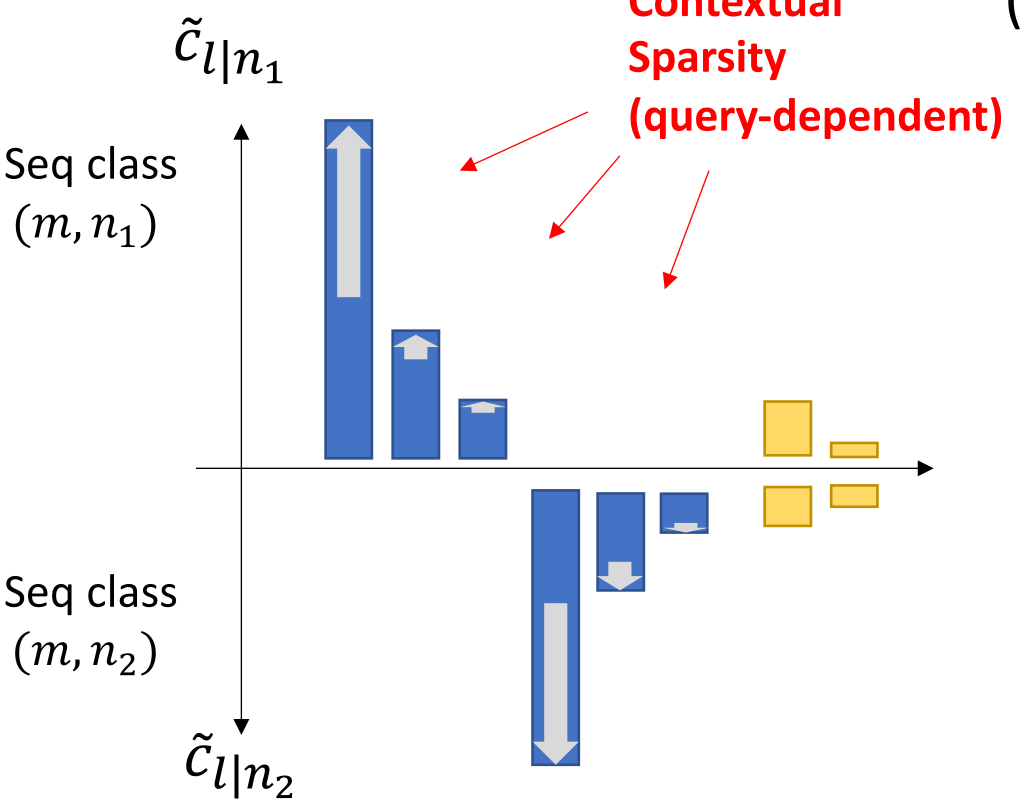
If l_0 is the dominant token: $r_{l_0/l|n}(0) > 0$ for all $l \neq l_0$ then

$$e^{2f^2_{nl_0}(0)B_n(t)} \leq \chi_{l_0}(t) \leq e^{2B_n(t)}$$

where $B_n(t) \geq 0$ monotonously increases, $B_n(0) = 0$

Overall Picture of the Training Dynamics

Winners-emergence



(c) $z_{ml}(t)$ grows faster with larger $\mathbb{P}(l|m,n)$

Theorem 3 Relative gain $r_{l/l'|n}(t) := \frac{\tilde{c}^2_{l|n}(t)}{\tilde{c}^2_{l'|n}(t)} - 1$ has a close form:

$$r_{l/l'|n}(t) = r_{l/l'|n}(0)\chi_l(t)$$

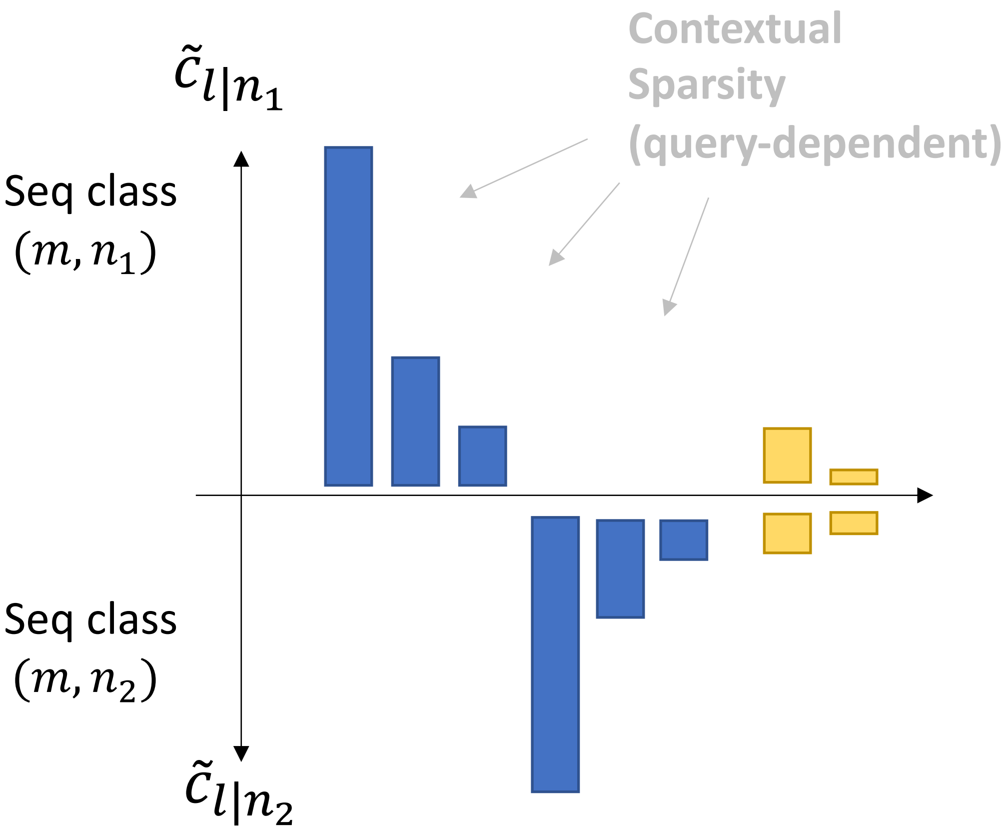
If l_0 is the dominant token: $r_{l_0/l|n}(0) > 0$ for all $l \neq l_0$ then

$$e^{2f^2_{nl_0}(0)B_n(t)} \leq \chi_{l_0}(t) \leq e^{2B_n(t)}$$

where $B_n(t) \geq 0$ monotonously increases, $B_n(0) = 0$

Overall Picture of the Training Dynamics

Attention frozen



Theorem 4 When $t \to +\infty$,

$$B_n(t) \sim \ln\left(C_0 + 2K\frac{\eta_Z}{\eta_Y}\ln^2\left(\frac{M\eta_Y t}{K}\right)\right)$$

Attention **scanning**:

When training starts, $B_n(t) = O(\ln t)$

Attention **snapping**:

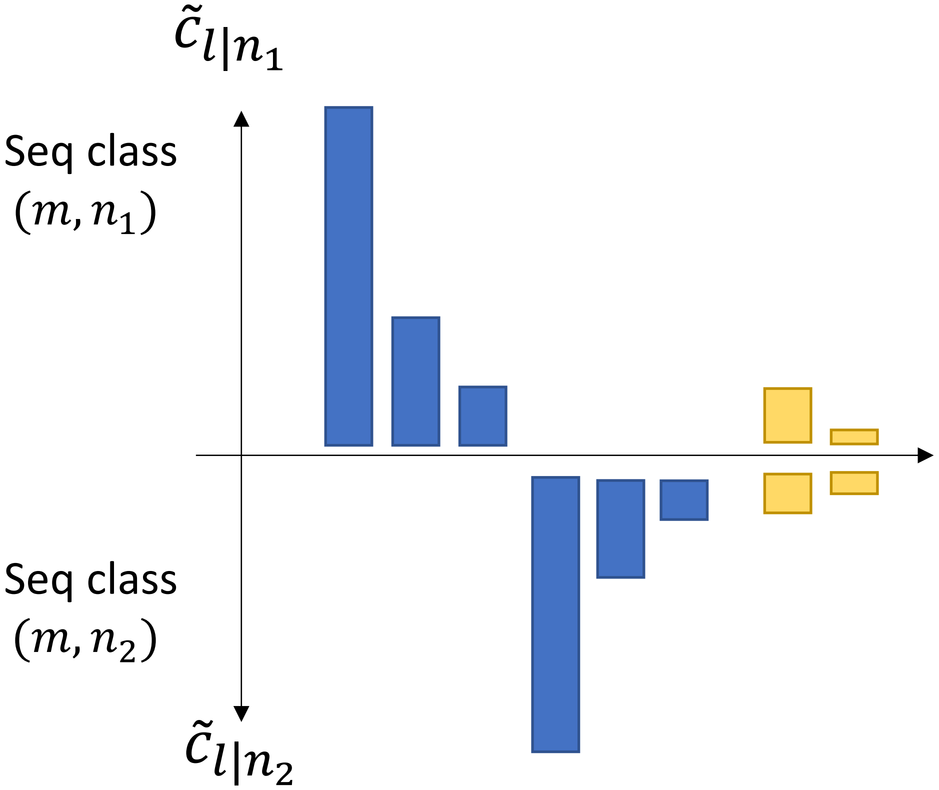
When $t \geq t_0 = O\left(\frac{2K \ln M}{\eta_Y}\right)$, $B_n(t) = O(\ln \ln t)$

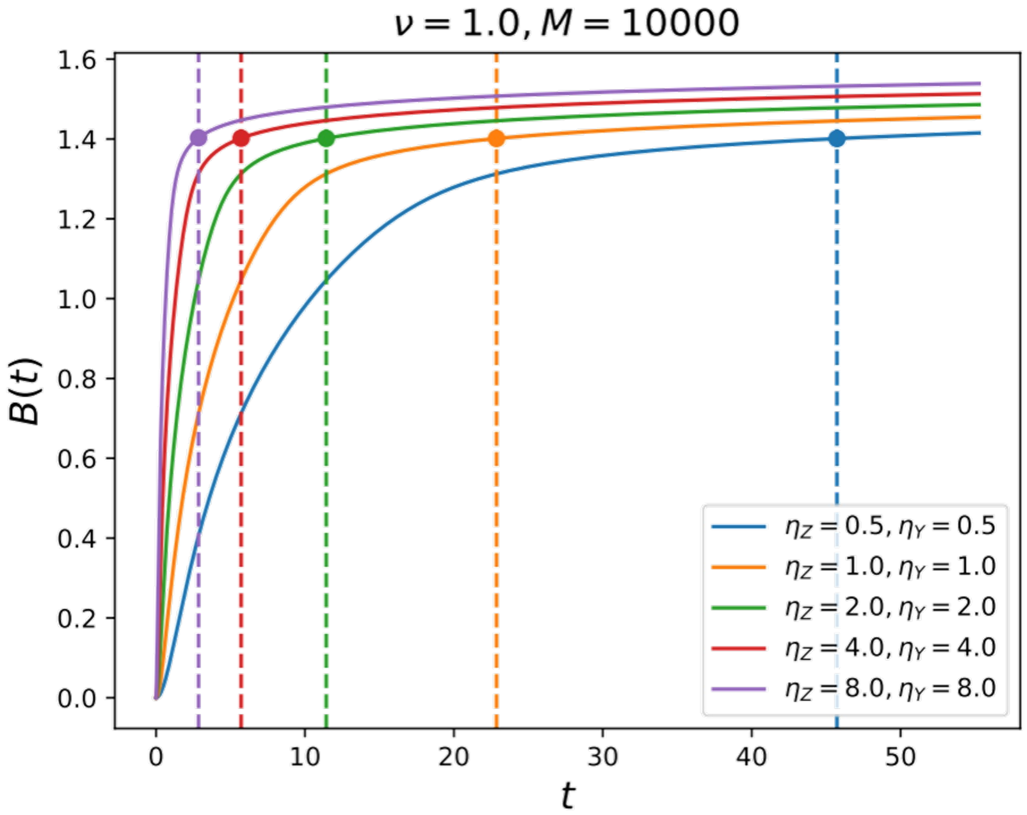
(1) η_Z and η_Y are large, $B_n(t)$ is large and attention is sparse

(2) Fixing η_Z, large η_Y leads to slightly small $B_n(t)$ and denser attention

Overall Picture of the Training Dynamics

<u>Attention frozen</u>





Larger learning rate η_z leads to faster phase transition

$$B_n(t) \sim \ln\left(C_0 + 2K\frac{\eta_Z}{\eta_Y}\ln^2\left(\frac{M\eta_Y t}{K}\right)\right)$$

Simple Real-world Experiments

WikiText2
(original parameterization)

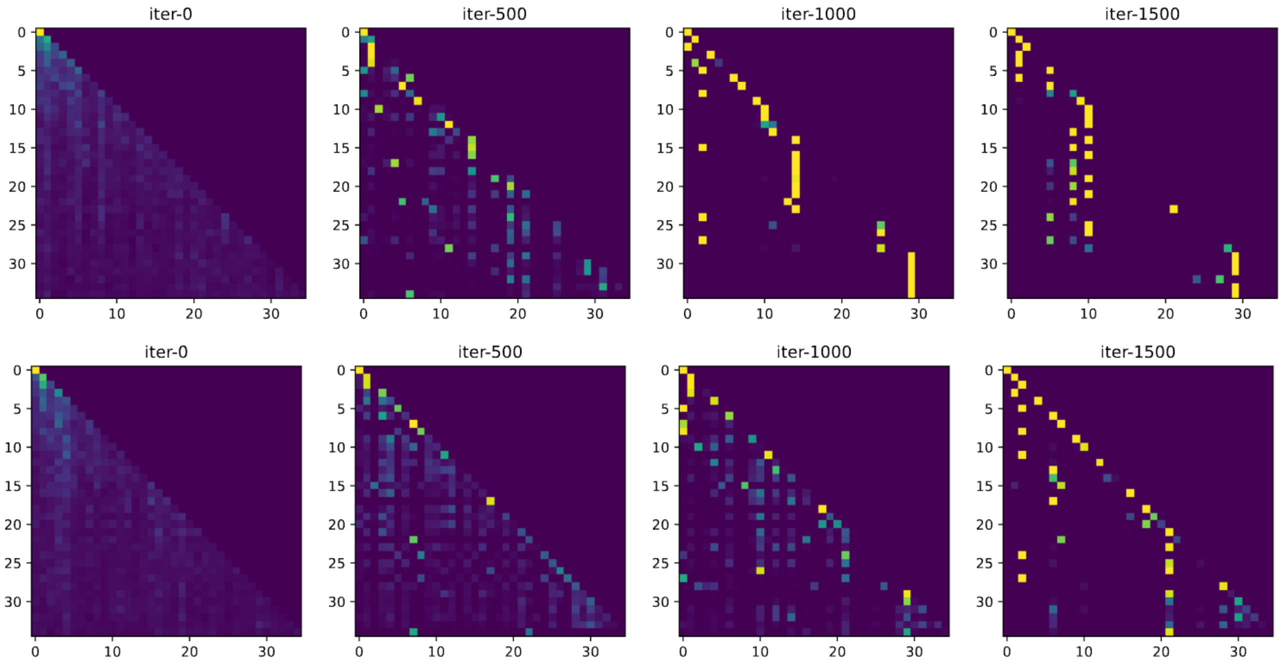


Figure 7: Attention patterns in the lowest self-attention layer for 1-layer (top) and 3-layer (bottom) Transformer trained on WikiText2 using SGD (learning rate is 5). Attention becomes sparse over training.

Further study of sparse attention
→ Deja Vu, H2O and StreamingLLM

[Z. Liu et al, *Deja vu: Contextual sparsity for efficient LLMs at inference time*, ICML'23 (oral)]
[Z. Zhang et al, *H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models*, NeurIPS'23]
[G. Xiao et al, *Efficient Streaming Language Models with Attention Sinks*, ICLR'24]

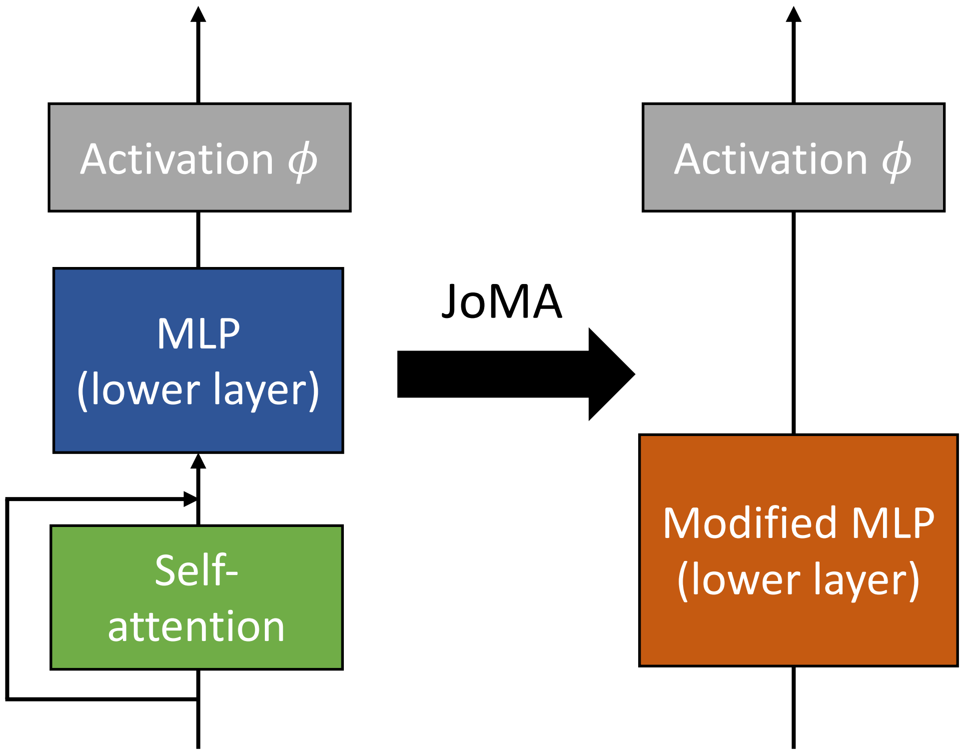
How to get rid of the assumptions?

- A few annoying assumptions in the analysis
  - No residual connections
  - No embedding vectors
  - The decoder needs to learn faster than the self-attention ($\eta_Y \gg \eta_Z$).
  - Single layer analysis

- How to get rid of them?

- New research work: **JoMA**

JoMA: JOint Dynamics of MLP/Attention layers

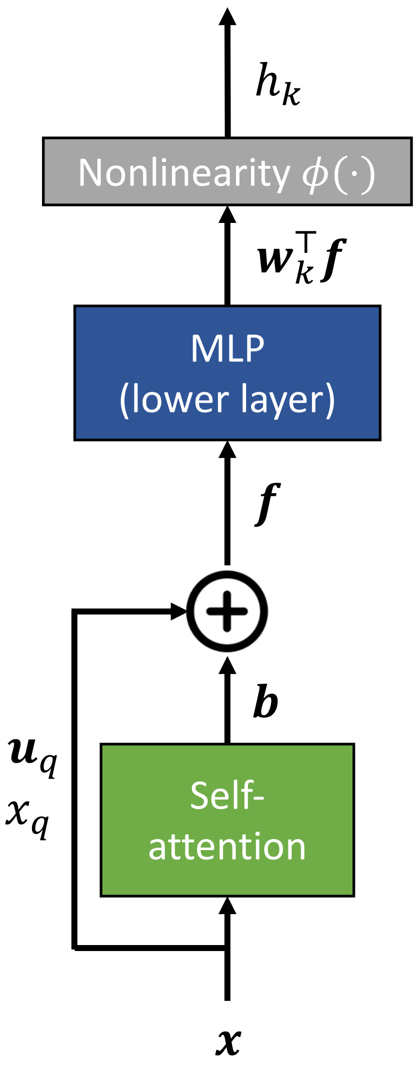


Main Contributions:

1. Find a joint dynamics that connects MLP with self-attention.
2. Understand self-attention behaviors for linear/nonlinear activations.
3. Explain how data hierarchy is learned in multi-layer Transformers.

[Y. Tian et al, *JoMA: Demystifying Multilayer Transformers via JOint Dynamics of MLP and Attention,* ICLR'24]

JoMA Settings



"This is an apple"

$$h_k = \phi(\boldsymbol{w}_k^\top \boldsymbol{f})$$

$$\boldsymbol{f} = U_C \boldsymbol{b} + \boldsymbol{u}_q$$

U_C and $\boldsymbol{u}_q$ are embeddings

$$\boldsymbol{b} = \sigma(\boldsymbol{z}_q) \circ \boldsymbol{x}/A$$

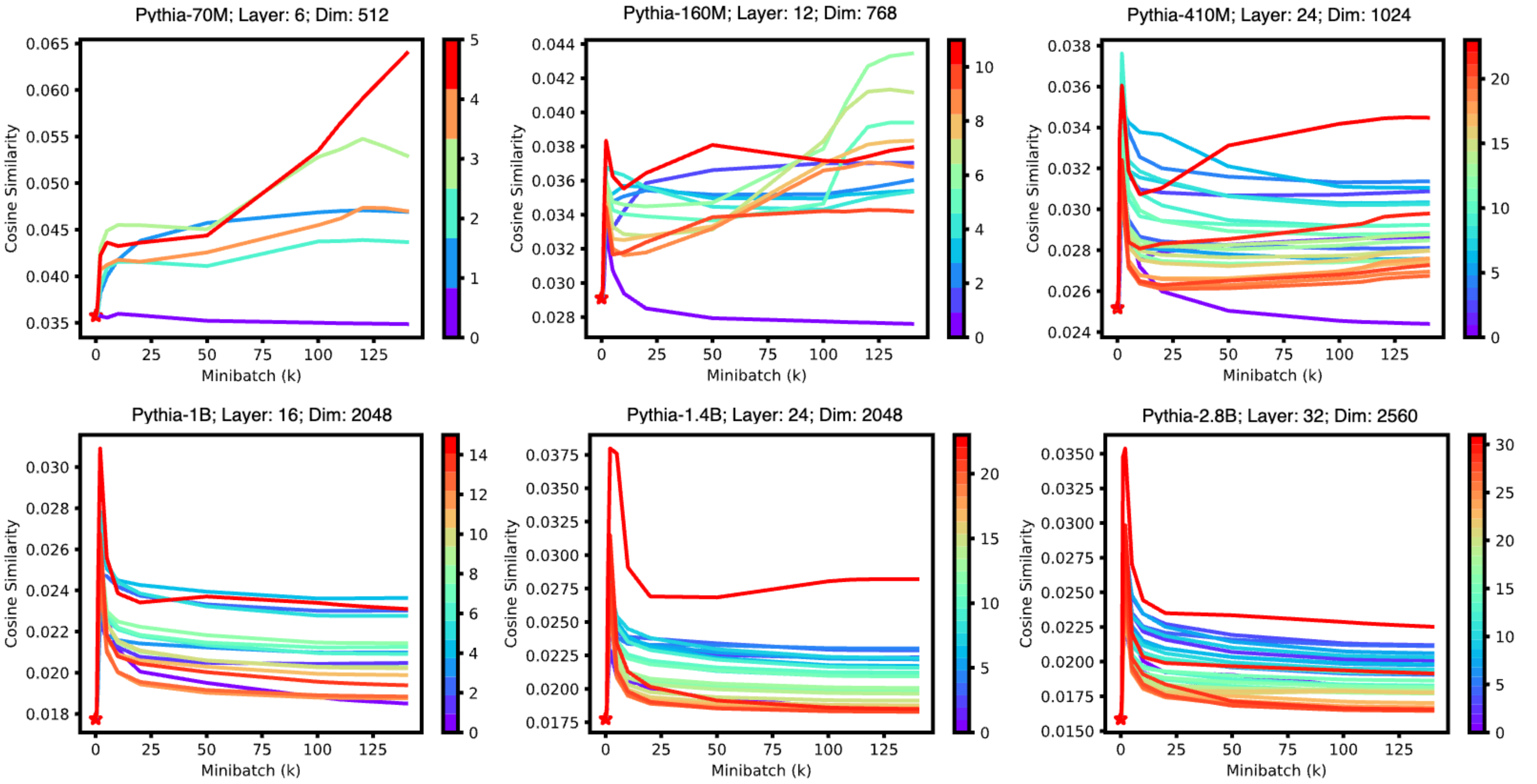
SoftmaxAttn: $b_l = \frac{x_l e^{z_{ql}}}{\sum_l x_l e^{z_{ql}}}$

ExpAttn: $b_l = x_l e^{z_{ql}}$

LinearAttn: $b_l = x_l z_{ql}$

Assumption (Orthogonal Embeddings $[U_C, u_q]$)

Cosine similarity between embedding vectors at different layers.



JoMA Dynamics

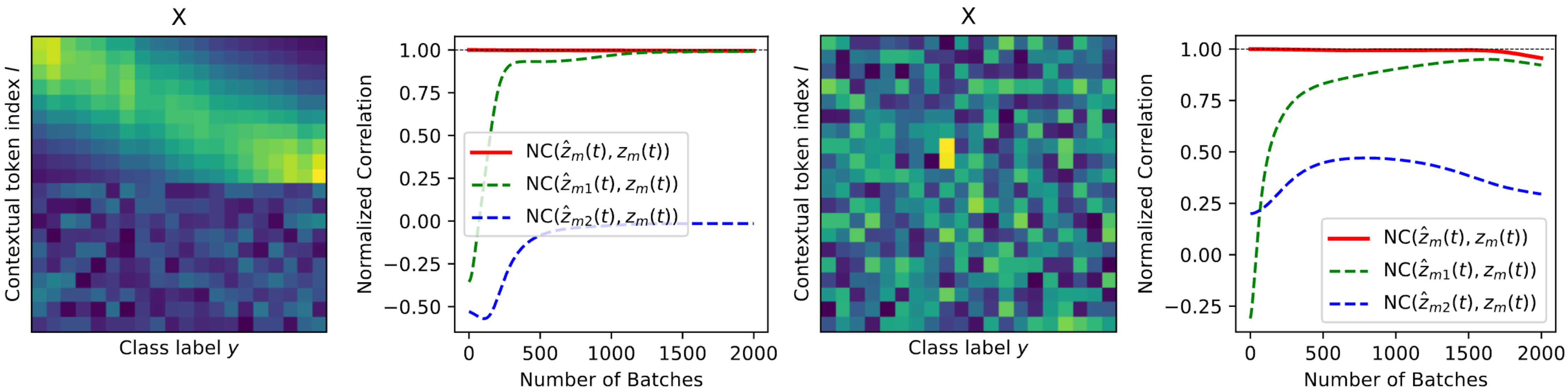
Theorem 1 (JoMA). *Let* $\boldsymbol{v}_k := U_C^\top \boldsymbol{w}_k$, *then the dynamics of Eqn.* $\boxed{3}$ *satisfies the invariants:*

- *Linear attention. The dynamics satisfies* $\boldsymbol{z}_m^2(t) = \sum_k \boldsymbol{v}_k^2(t) + \boldsymbol{c}$.
- *Exp attention. The dynamics satisfies* $\boldsymbol{z}_m(t) = \frac{1}{2}\sum_k \boldsymbol{v}_k^2(t) + \boldsymbol{c}$.
- *Softmax attention. If* $\bar{\boldsymbol{b}}_m := \mathbb{E}_{q=m}[\boldsymbol{b}]$ *is a constant over time and* $\mathbb{E}_{q=m}\left[\sum_k g_{h_k} h'_k \boldsymbol{b}\boldsymbol{b}^\top\right] = \bar{\boldsymbol{b}}_m \mathbb{E}_{q=m}\left[\sum_k g_{h_k} h'_k \boldsymbol{b}\right]$, *then the dynamics satisfies* $\boldsymbol{z}_m(t) = \frac{1}{2}\sum_k \boldsymbol{v}_k^2(t) - \|\boldsymbol{v}_k(t)\|_2^2 \bar{\boldsymbol{b}}_m + \boldsymbol{c}$.

Under zero-initialization ($\boldsymbol{w}_k(0) = 0$, $\boldsymbol{z}_m(0) = 0$), *then the time-independent constant* $\boldsymbol{c} = 0$.

There is residual connection.
Joint dynamics works for any learning rates between self-attention and MLP layer.
No assumption on the data distribution.

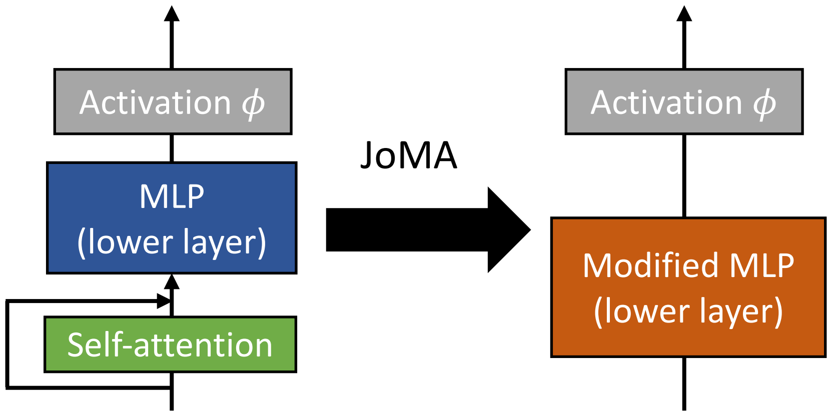
Verification of JoMA dynamics



$\boldsymbol{z}_m(t)$: Real attention logits
$\hat{\boldsymbol{z}}_m(t)$: Estimated attention logits by JoMA

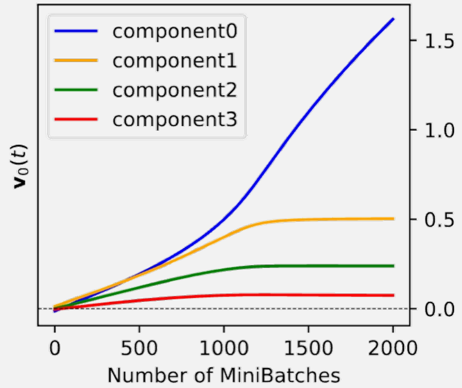
$$\hat{\boldsymbol{z}}_m(t) = \underbrace{\frac{1}{2}\sum_k \boldsymbol{v}_k^2(t)}_{\hat{\boldsymbol{z}}_{m1}(t)} - \underbrace{\|\boldsymbol{v}_k(t)\|_2^2\overline{\boldsymbol{b}}_m}_{\hat{\boldsymbol{z}}_{m2}(t)} + \boldsymbol{c}$$

Implication of Theorem 1



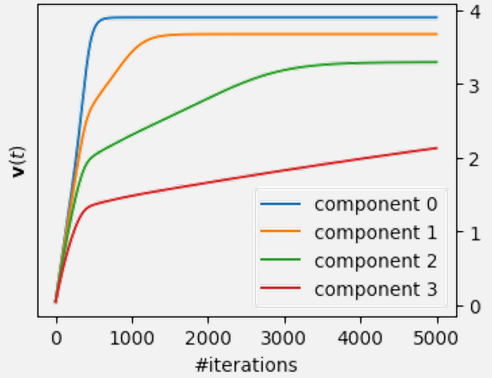
Key idea: folding self-attention into MLP
→ A Transformer block becomes a modified MLP

Linear case ($\phi = \mathrm{Id}, K = 1$)



Most salient feature takes all
(Attention becomes sparser)

Nonlinear case (ϕ nonlinear, $K = 1$)



Most salient feature grows, and others catch up
(Attention becomes sparser and denser)

Saliency is defined as $\Delta_{lm} = \mathbb{E}[g|l,m] \cdot \mathbb{P}[l|m]$

Discriminancy **CoOccurrence**

$\Delta_{lm} \approx 0$: **Common** tokens
$|\Delta_{lm}|$ large: **Distinct** tokens

JoMA for Linear Activation

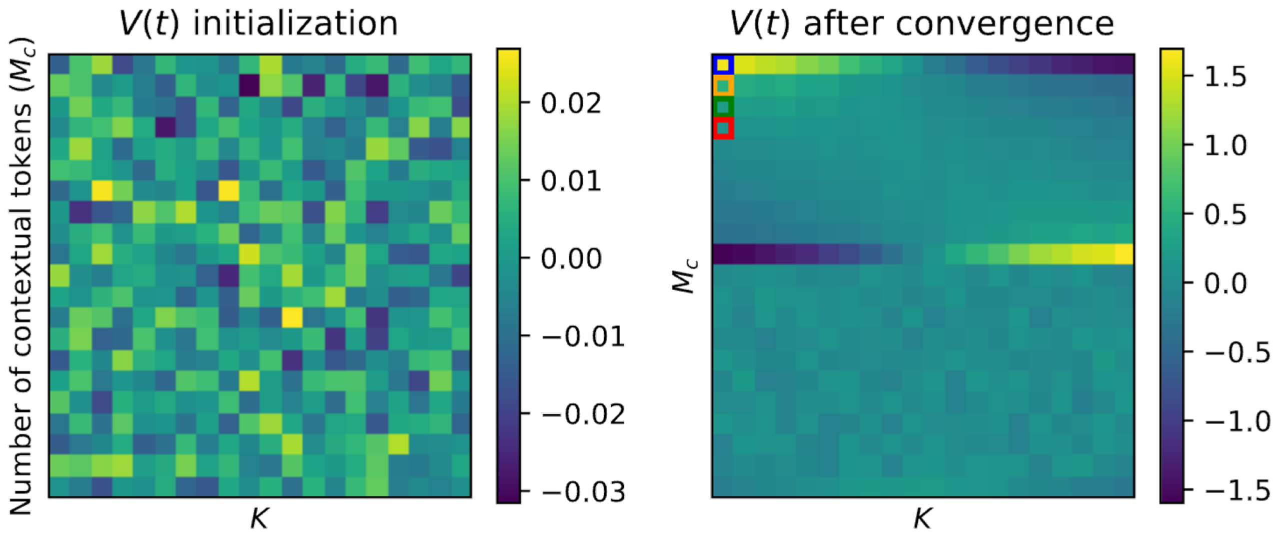
$\dot{\boldsymbol{v}} = \boldsymbol{\Delta}_m \circ \exp\left(\frac{\boldsymbol{v}^2}{2}\right)$	Linear Modified MLP (lower layer)

Theorem 2

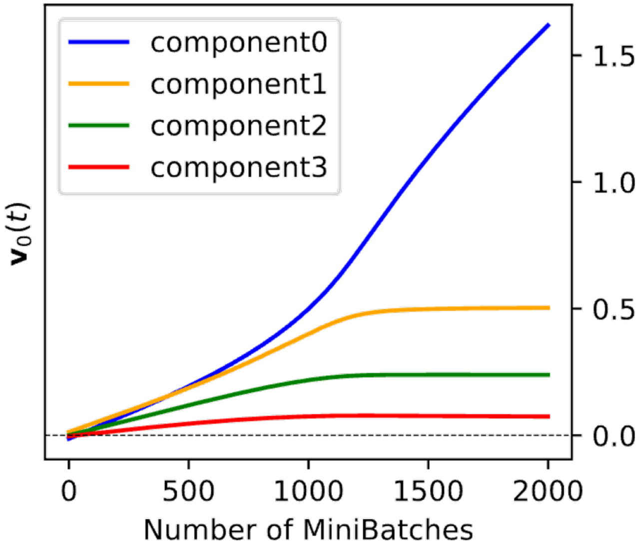
We can prove $\frac{\mathrm{erf}(v_l(t)/2)}{\Delta_{lm}} = \frac{\mathrm{erf}(v_{l'}(t)/2)}{\Delta_{l'm}}$

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\mathrm{d}t \in [-1,1]$$

Only the most salient token $l^* = \mathrm{argmax}\,|\Delta_{lm}|$ of $\boldsymbol{v}$ goes to $+\infty$
other components stay finite.



Attention becomes sparser
(Consistent with Scan&Snap)



[**Y. Tian** et al, *Scan and Snap: Understanding Training Dynamics and Token Composition in 1-layer Transformer,* NeurIPS'23]

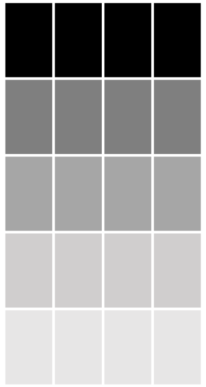
What if we have more nodes ($K > 1$)?

- $V = U_C^\top W \in \mathbb{R}^{M_c \times K}$ and the dynamics becomes

$$\dot{V} = \frac{1}{A}\mathrm{diag}\left(\exp\left(\frac{V \circ V}{2}\right)\mathbf{1}\right)\Delta \qquad \Delta = [\Delta_1, \Delta_2, \dots, \Delta_K], \qquad \Delta_k = \mathbb{E}[g_k \boldsymbol{x}]$$

We can prove that V gradually becomes low rank
- The growth rate of each row of V varies widely.

$V(t) \rightarrow$



Due to $\exp\left(\frac{V \circ V}{2}\right)$, the weight gradient $\dot{V}$ can be even more low-rank → GaLore

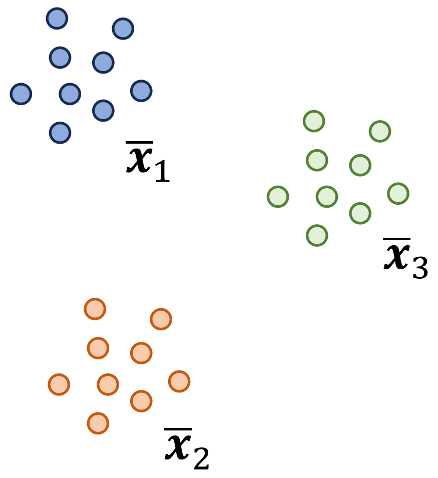
JoMA for Nonlinear Activation

Theorem 3

If $\boldsymbol{x}$ is sampled from a mixture of C isotropic distributions, (i.e., “local salient/non-salient map”), then

$$\dot{\boldsymbol{v}} = \frac{1}{\|\boldsymbol{v}\|_2}\sum_c a_c \theta_1(r_c)\bar{\boldsymbol{x}}_c + \frac{1}{\|\boldsymbol{v}\|_2^3}\sum_c a_c\theta_2(r_c)\boldsymbol{v}$$

Here $a_c \coloneqq \mathbb{E}_{q=m,c}[g_{h_k}]\mathbb{P}[c]$, $r_c = \boldsymbol{v}^\top\bar{\boldsymbol{x}}_c + \int_0^t \mathbb{E}_{q=m}[g_{h_k}h'_k]\mathrm{d}t$, and θ_1 and θ_2 depends on nonlinearity



What does the dynamics look like?

$$\dot{\boldsymbol{v}} = (\boldsymbol{\mu} - \boldsymbol{v}) \circ \exp\left(\frac{\boldsymbol{v}^2}{2}\right)$$

$\boldsymbol{\mu} \sim \bar{\boldsymbol{x}}_c$: Critical point due to nonlinearity (one of the cluster centers)

JoMA for Nonlinear activation

	Nonlinear
$\dot{\boldsymbol{v}} = (\boldsymbol{\mu} - \boldsymbol{v}) \circ \exp\left(\frac{\boldsymbol{v}^2}{2}\right)$	Modified MLP (lower layer)

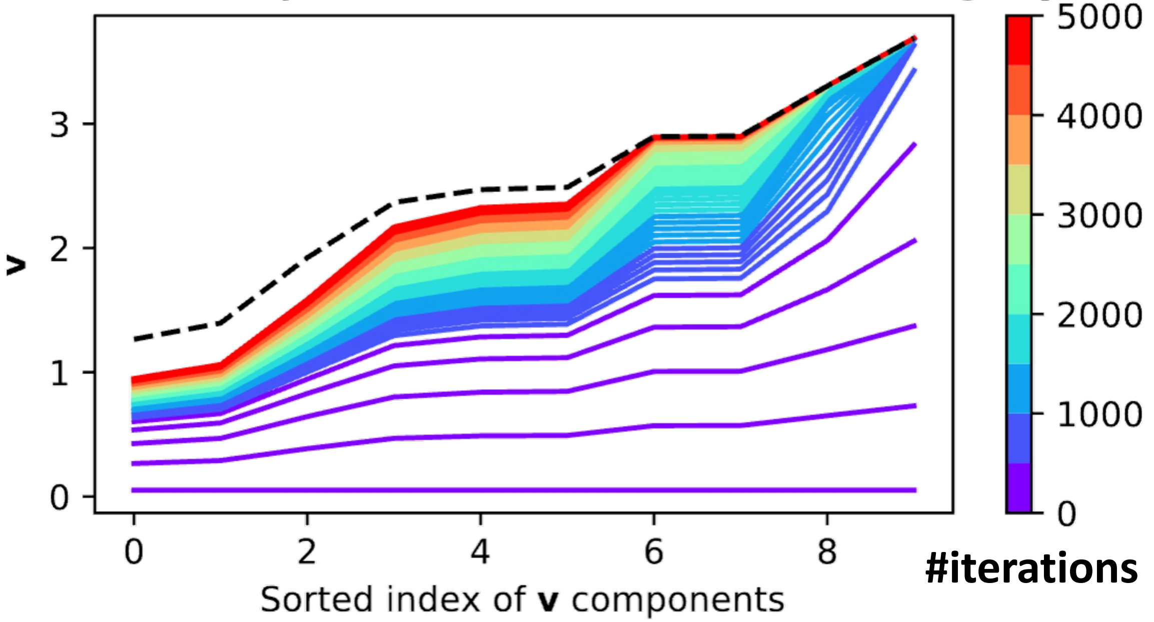
Theorem 4

Salient components grow much faster than non-salient ones:

$$\frac{\text{ConvergenceRate}(j)}{\text{ConvergenceRate}(k)} \sim \frac{\exp(\mu_j^2/2)}{\exp(\mu_k^2/2)}$$

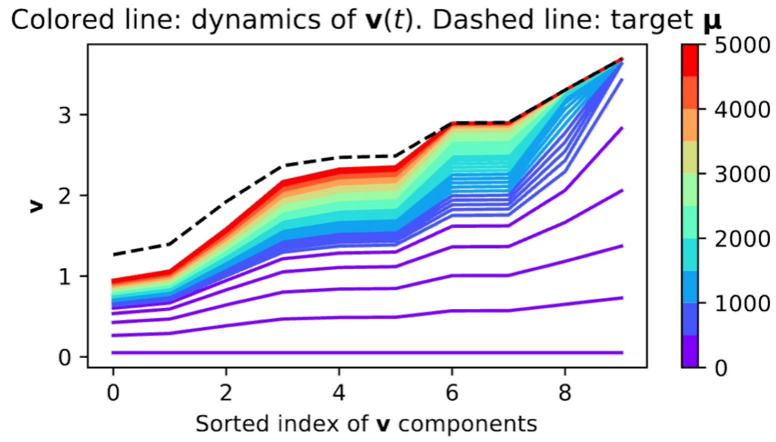
$$\text{ConvergenceRate}(j) := \ln 1/\delta_j(t)$$
$$\delta_j(t) := 1 - v_j(t)/\mu_j$$

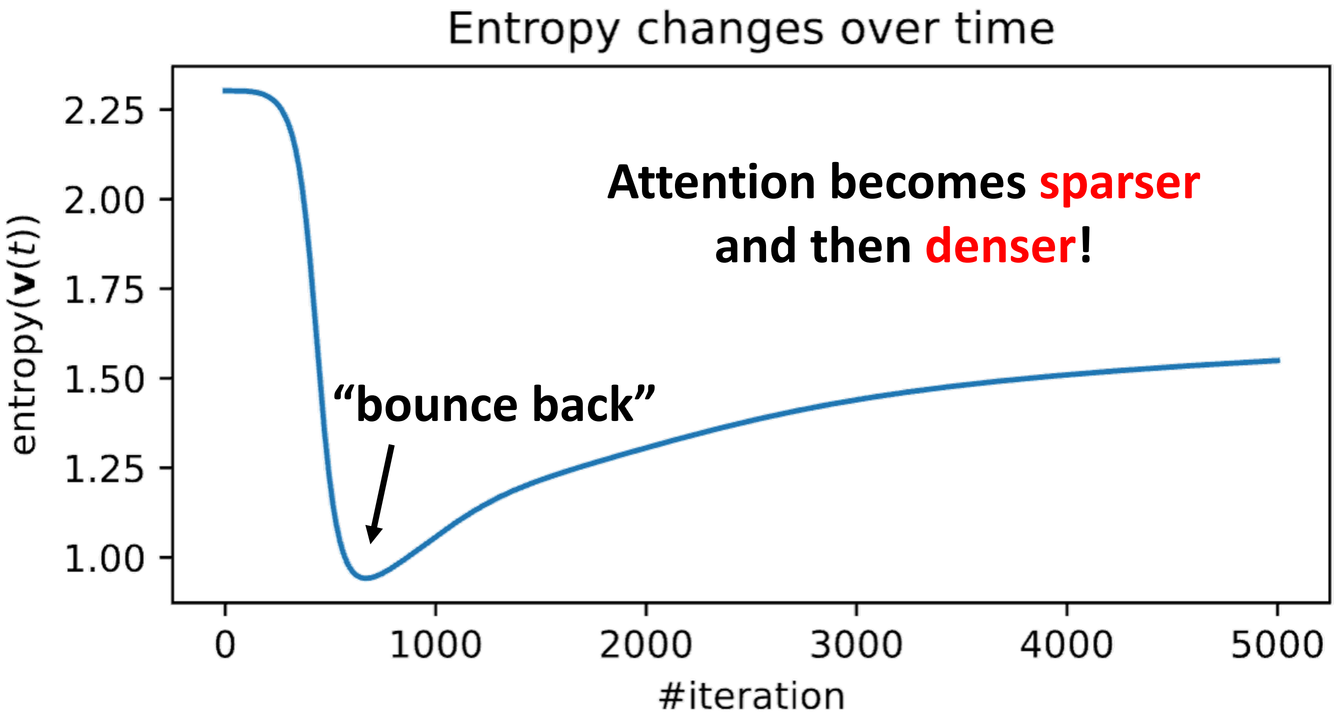
Colored line: dynamics of $\mathbf{v}(t)$. Dashed line: target $\boldsymbol{\mu}$



JoMA for Nonlinear activation

	Nonlinear
$\dot{\boldsymbol{v}} = (\boldsymbol{\mu} - \boldsymbol{v}) \circ \exp\left(\frac{\boldsymbol{v}^2}{2}\right)$	Modified MLP (lower layer)

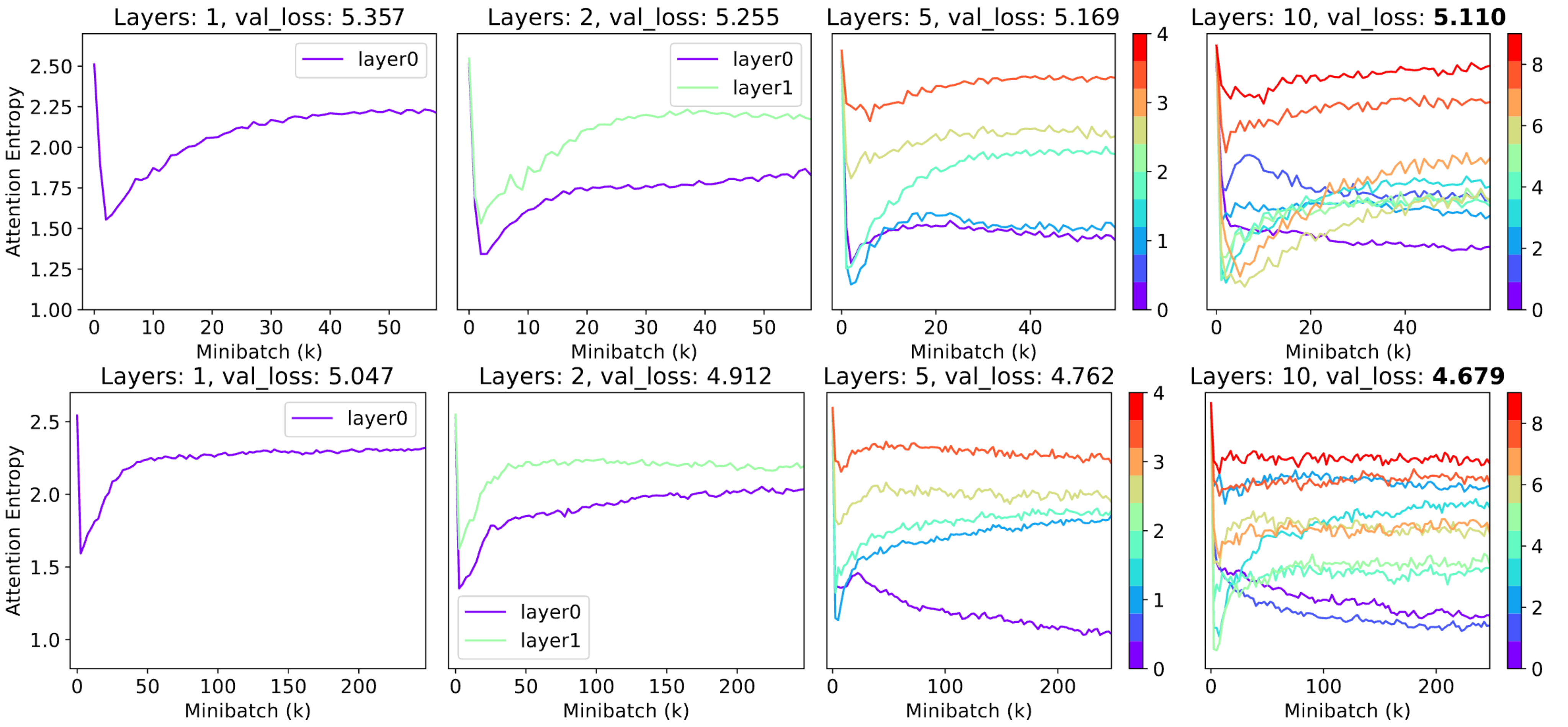




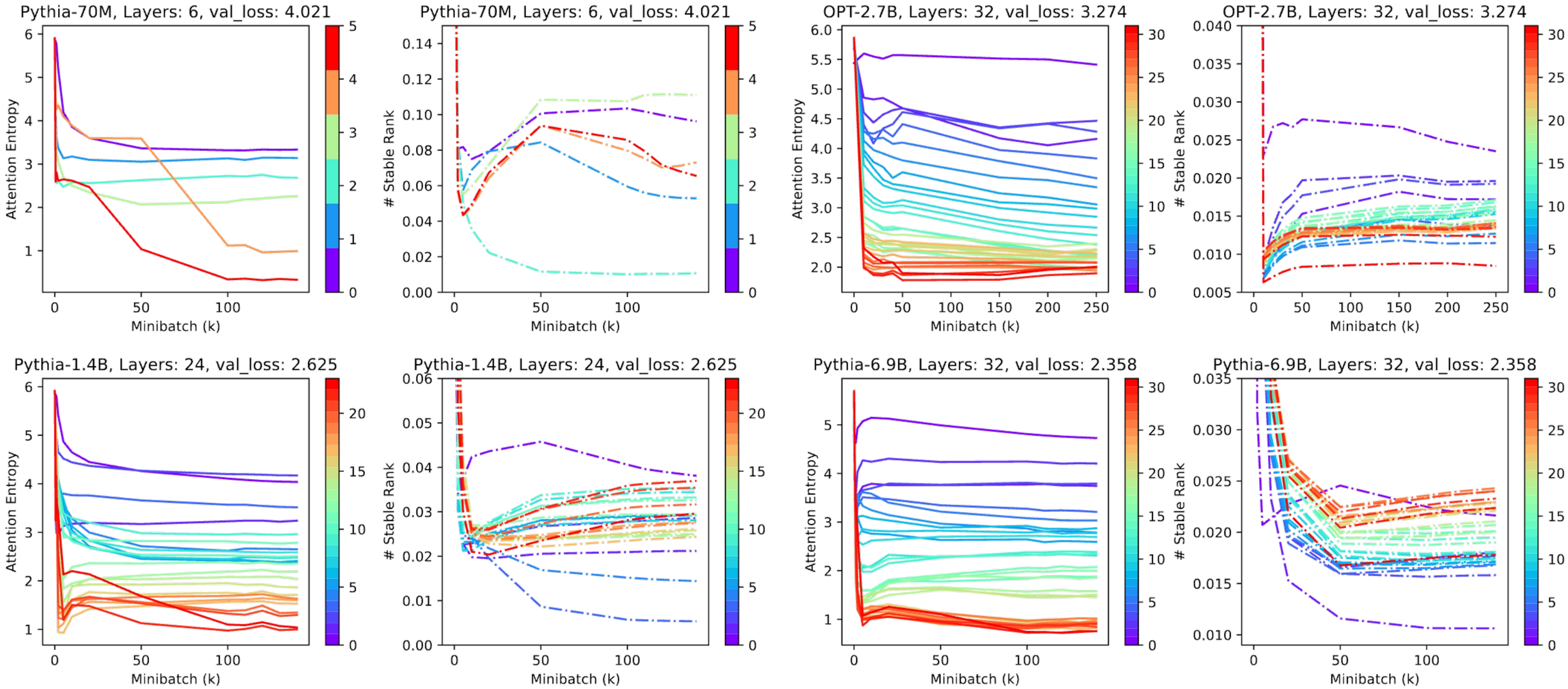
Real-world Experiments

Wikitext2

Wikitext103



Real-world Experiments



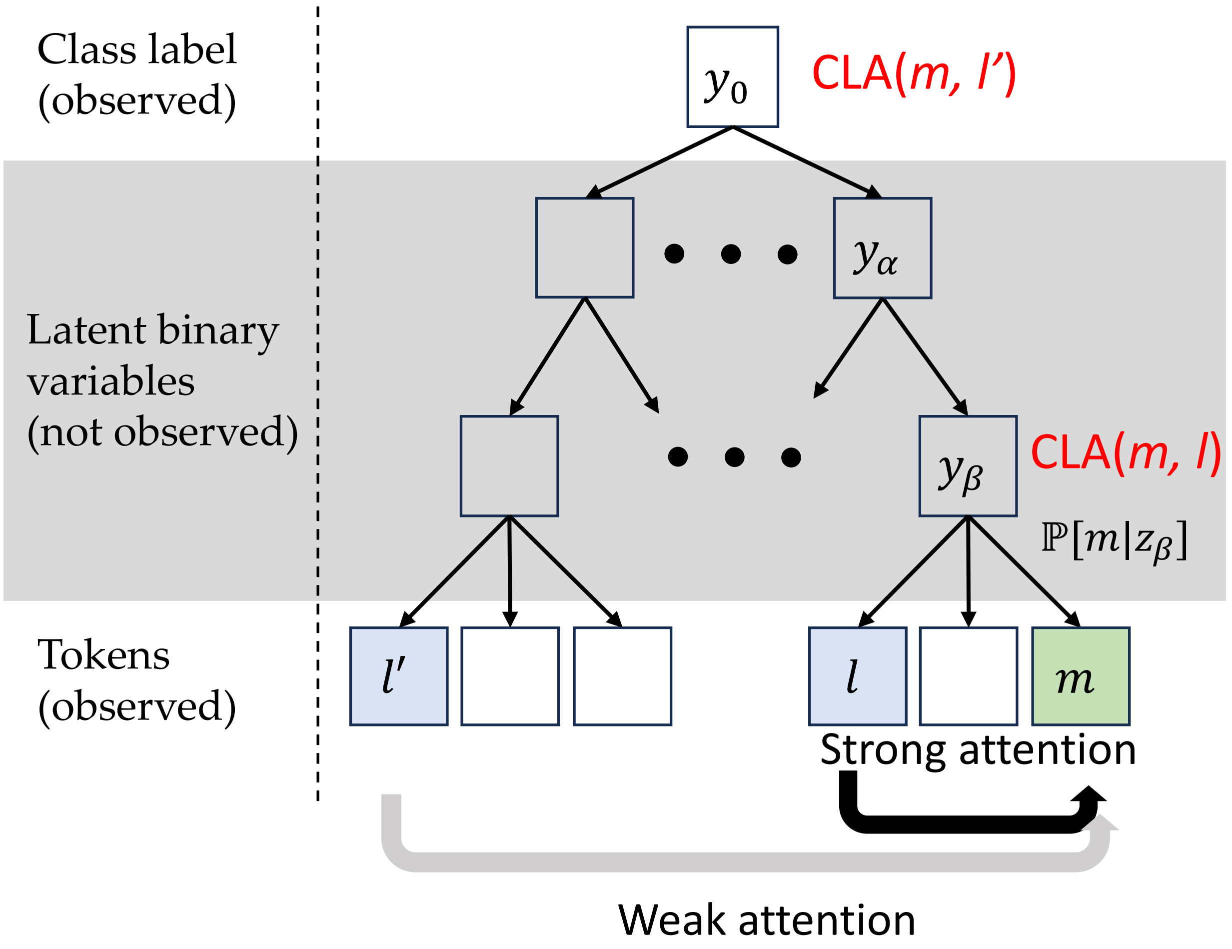
Stable Rank of the lower layer of MLP shows the “bouncing back” effects as well.

Why is this "bouncing back" property useful?

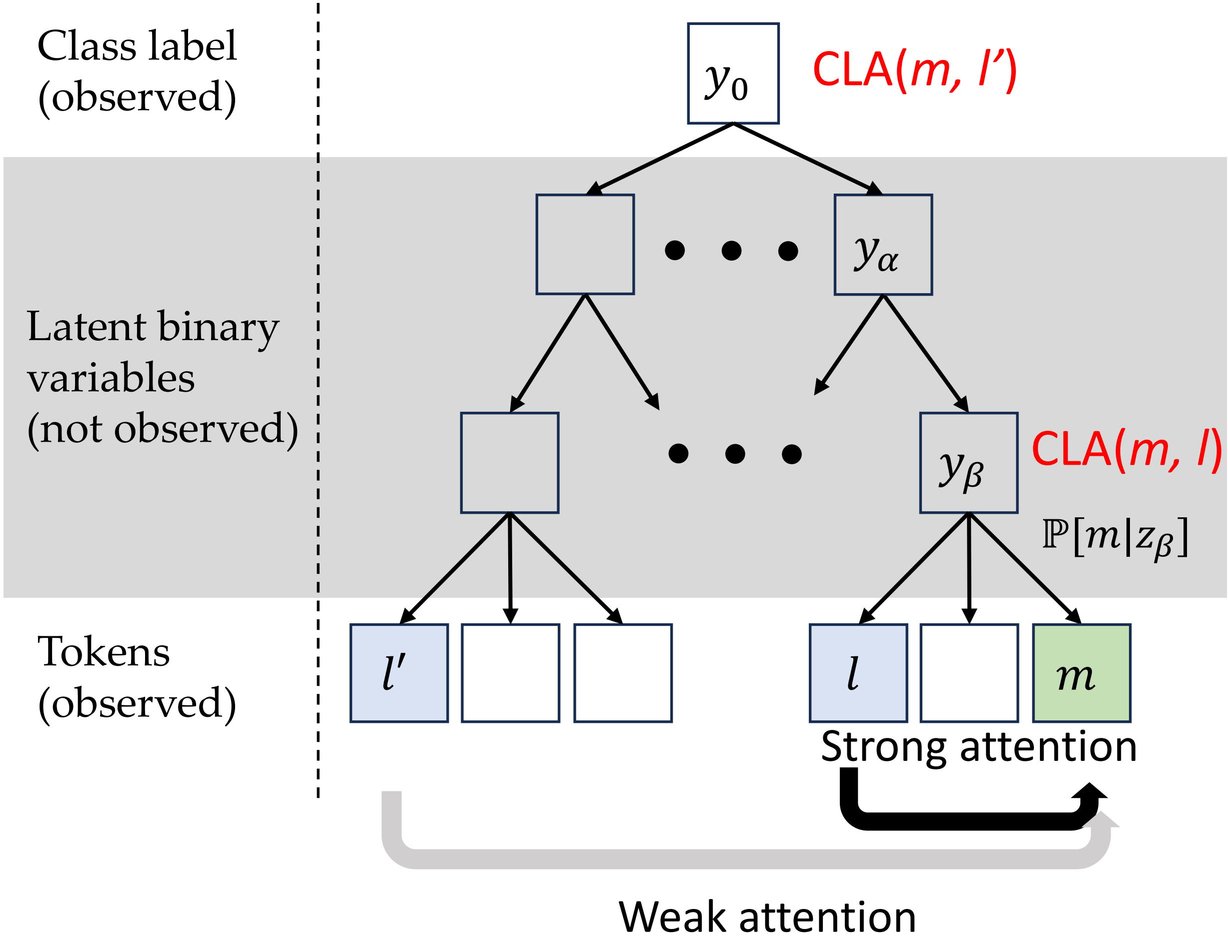
It seems that it only slows down the training??

Not useful in 1-layer, but useful in multiple Transformer layers!

Data Hierarchy & Multilayer Transformer



Data Hierarchy & Multilayer Transformer



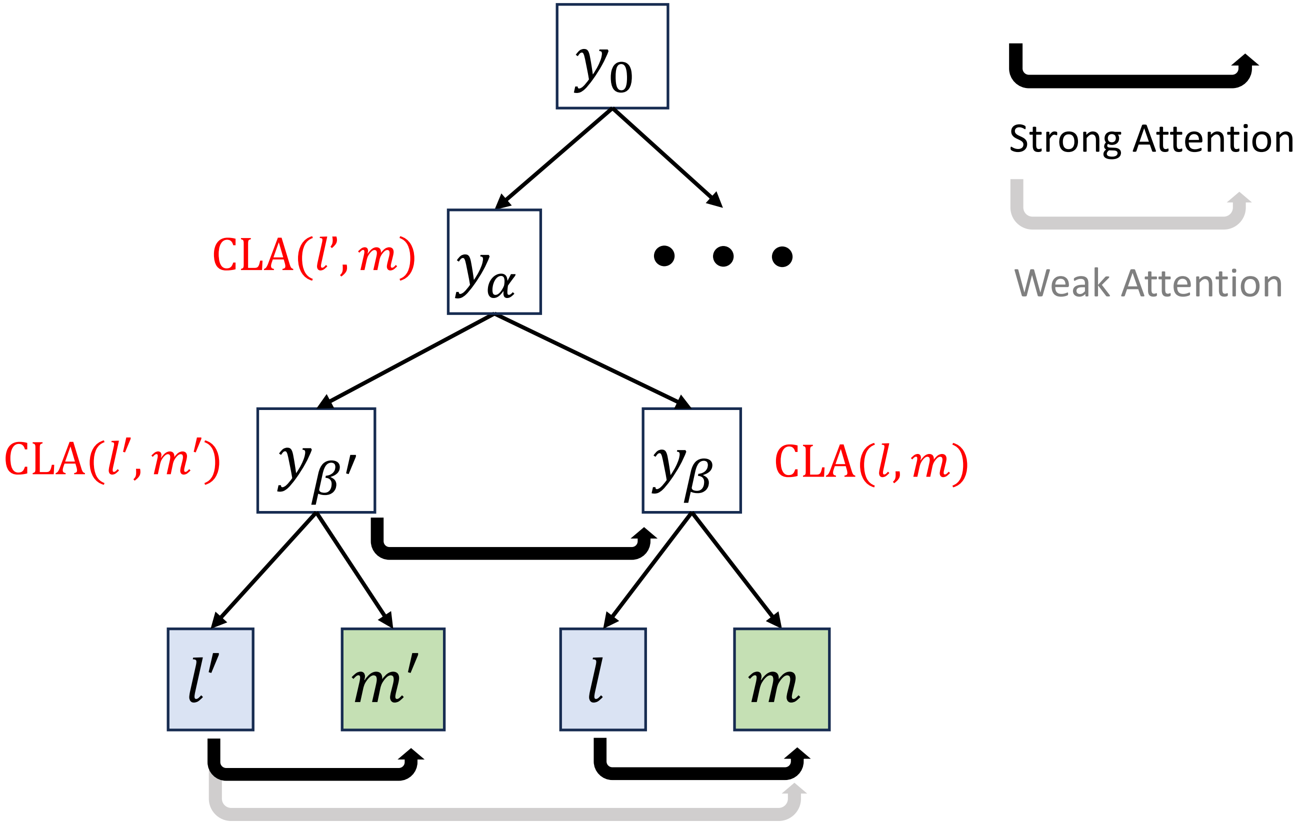
Theorem 5

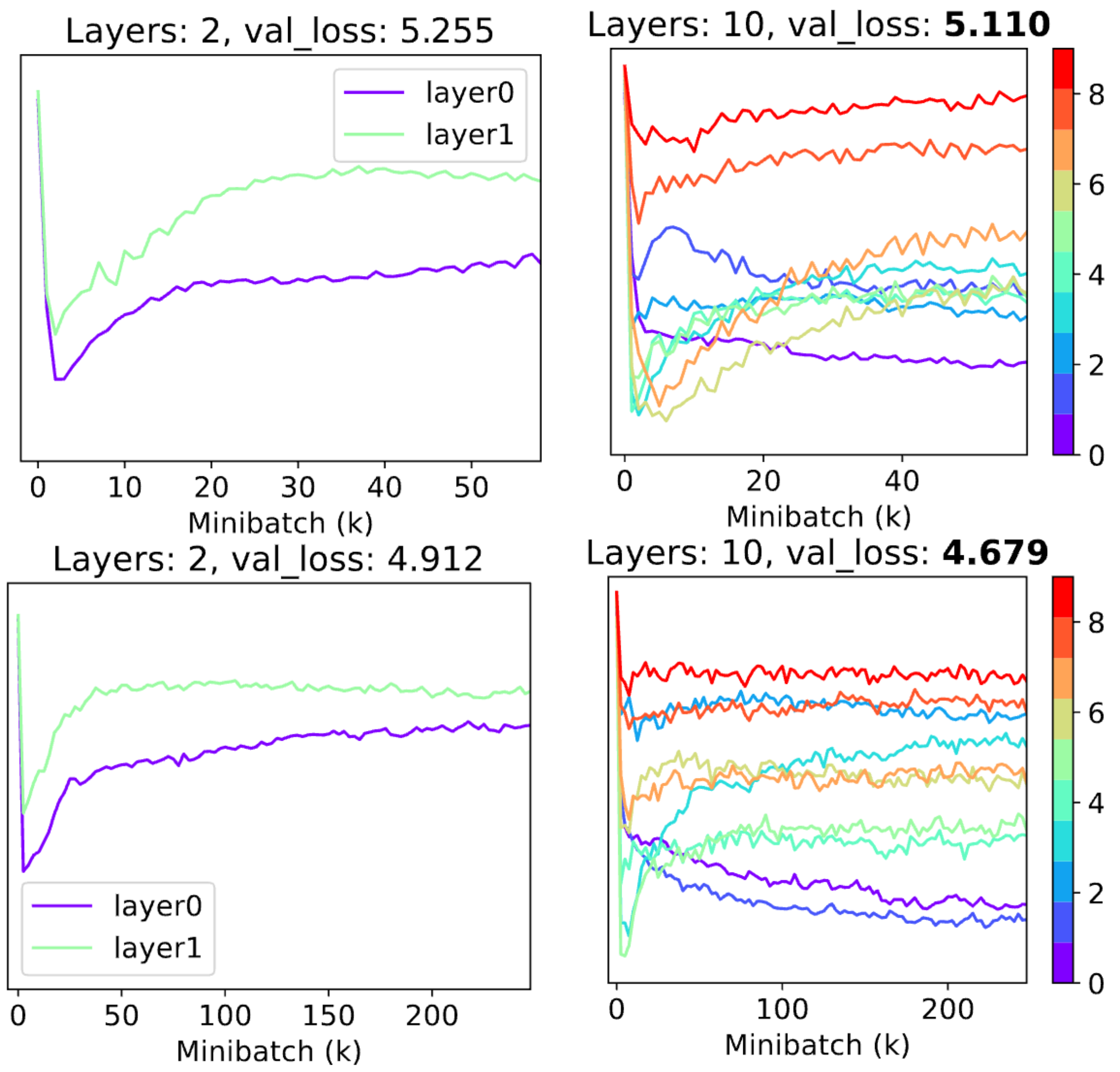
$$\mathbb{P}[l|m] \approx 1 - \frac{H}{L}$$

H: height of the common latent ancestor (CLA) of l & m

L: total height of the hierarchy

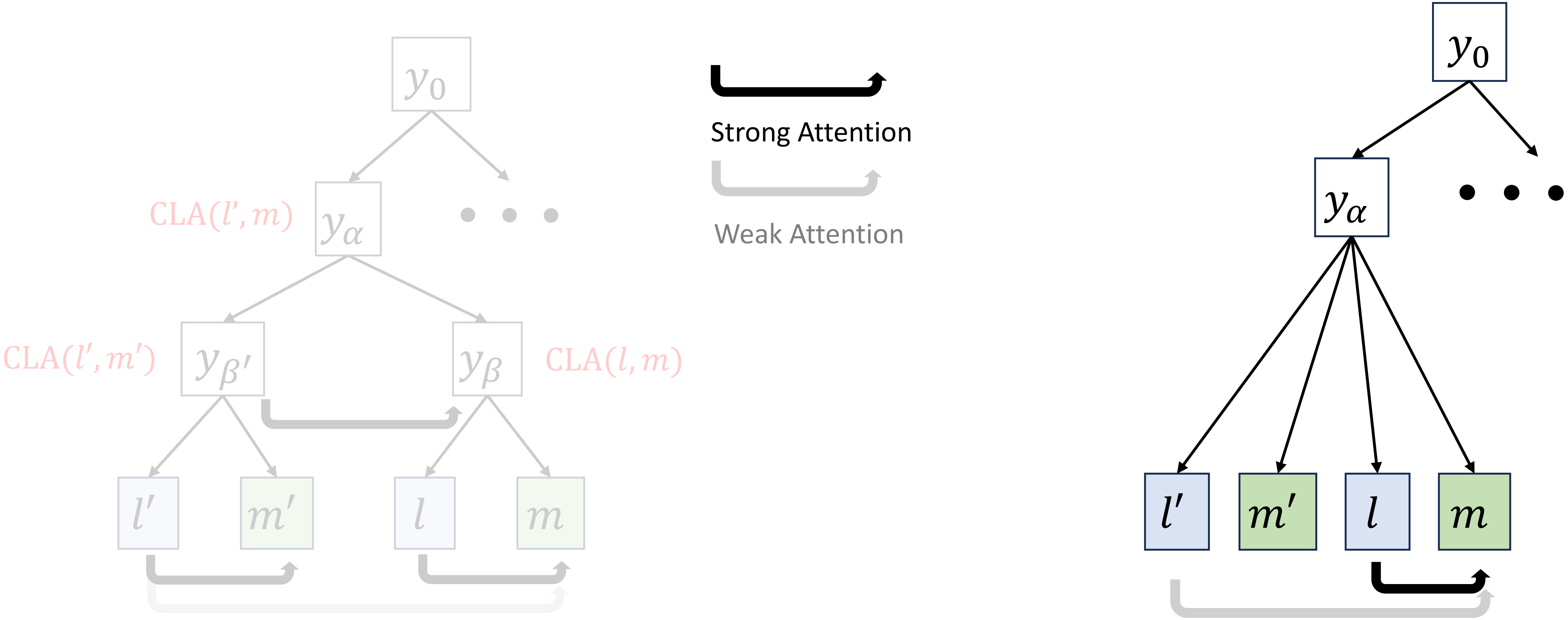
Deep Latent Distribution



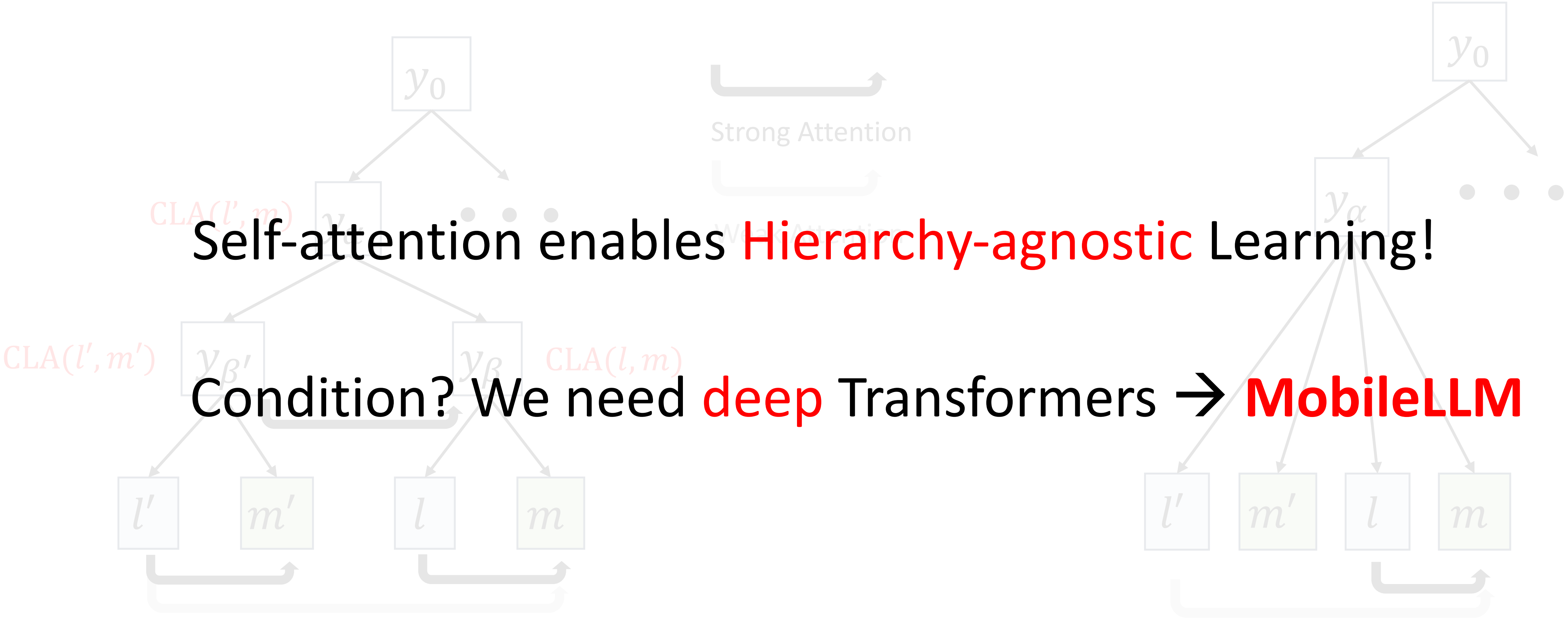


Learning the current hierarchical structure by ***slowing down*** the association of tokens that are not directly correlated

Shallow Latent Distribution



Shallow Latent Distribution



Self-attention enables Hierarchy-agnostic Learning!

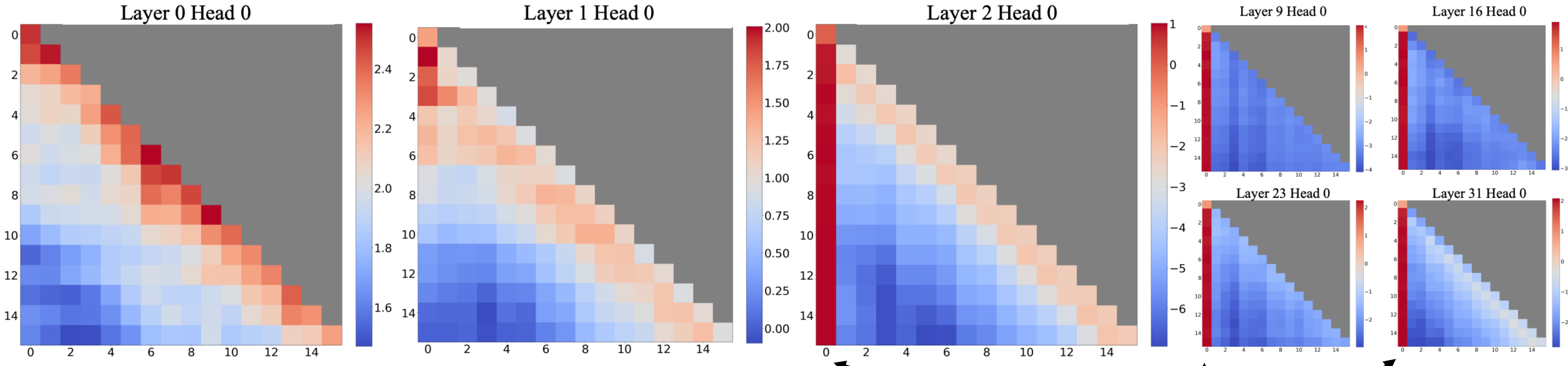
Condition? We need deep Transformers → **MobileLLM**

Future Work

- How embedding vectors are learned?
  - In both Scan&Snap and JoMA, we assume embeddings are constant.
- Positional Encoding
- Formulate the dynamics of Multi-layer Transformers
  - How intermediate latent concept gets learned during training?
  - Why we need over-parameterization?

Part II
Applications based on Properties of Transformers

Attention Sinks: Initial tokens draw a lot of attentions



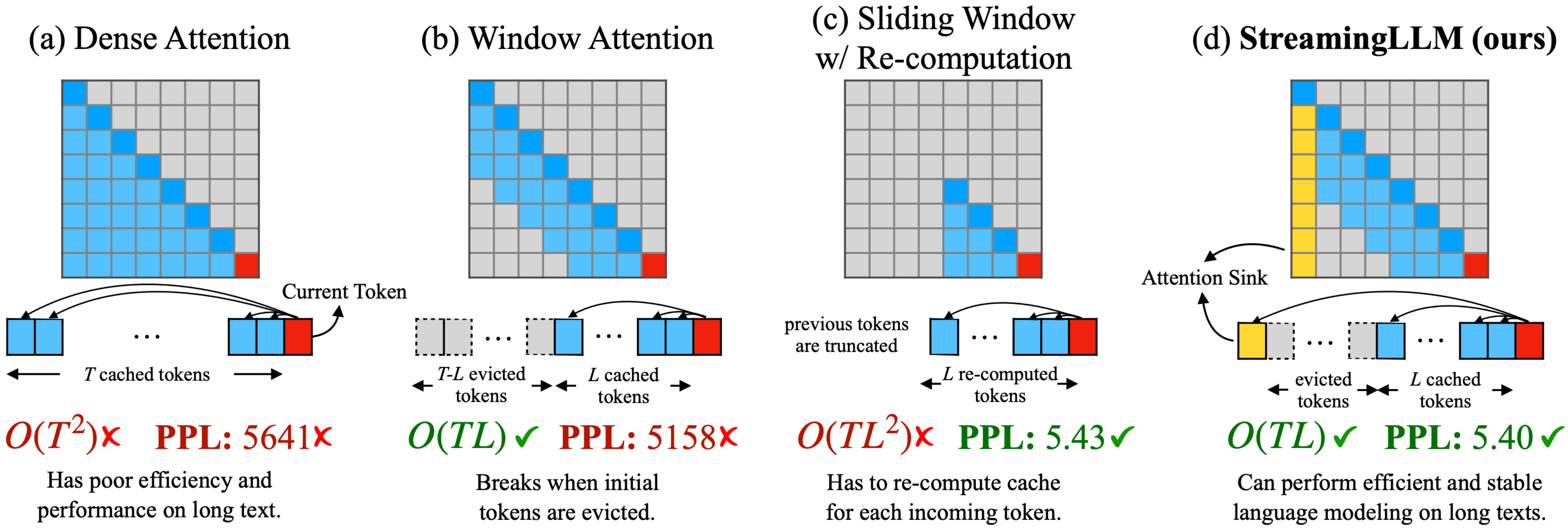
Average attention logits in Llama-2-7B over 256 sentences

First few tokens!!

- Observation: Initial tokens have large attention scores, even if they're not semantically significant.
- Attention Sink: Tokens that disproportionately attract attention irrespective of their relevance.

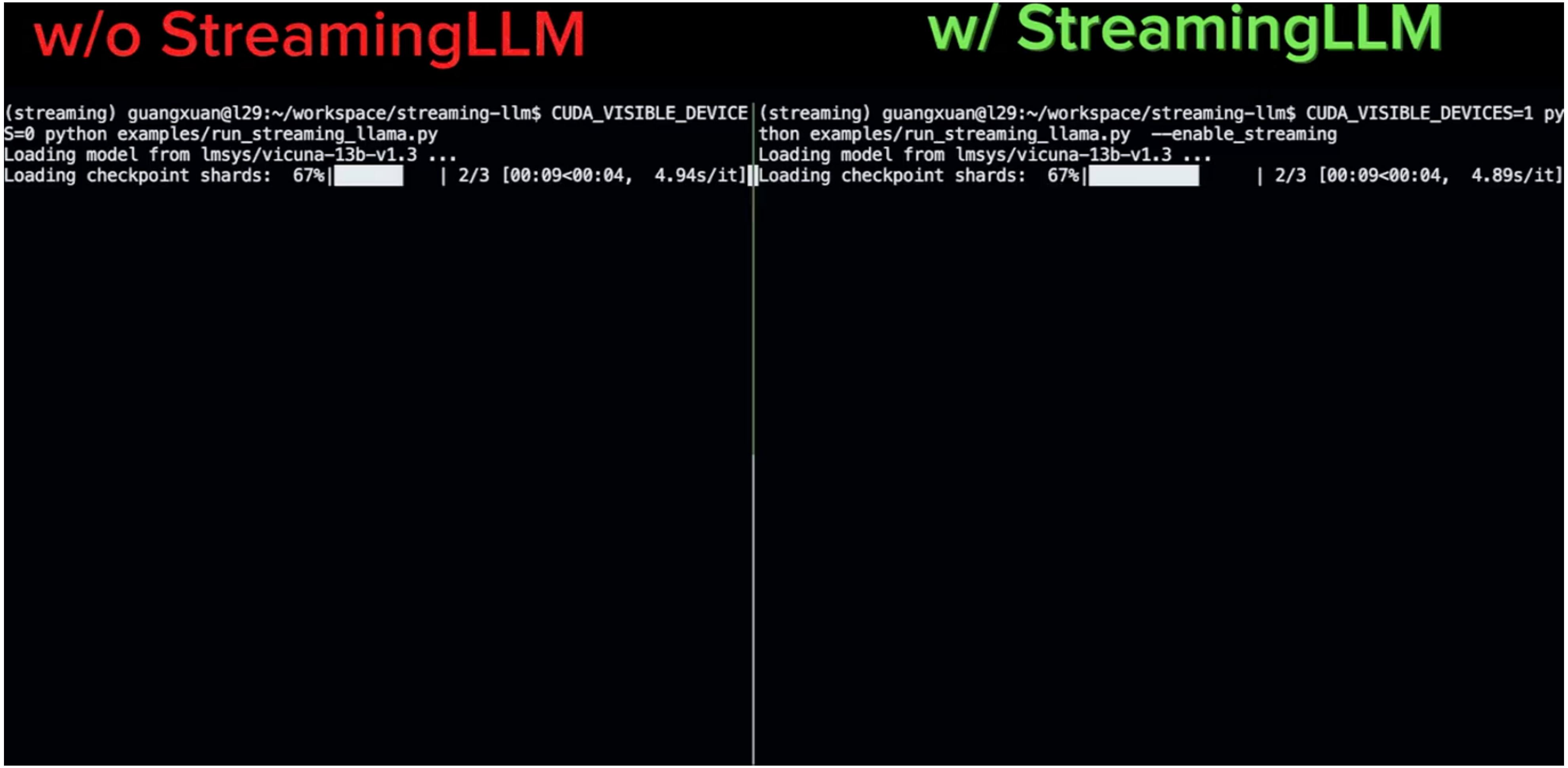
[G. Xiao, **Y. Tian**, et al, *Efficient Streaming Language Models with Attention Sinks,* ICLR'24]

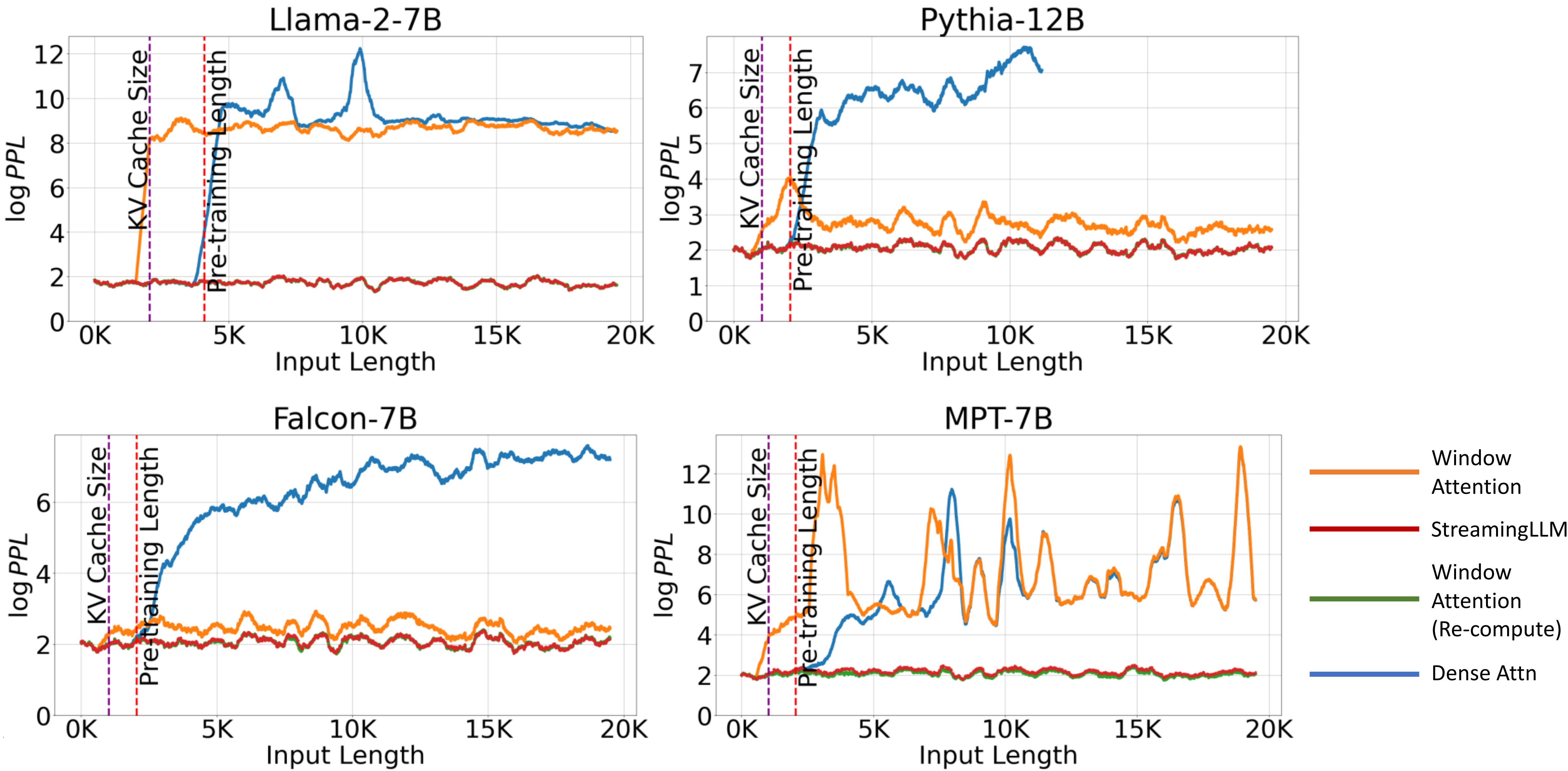
StreamingLLM



[G. Xiao, **Y. Tian**, et al, *Efficient Streaming Language Models with Attention Sinks*, ICLR'24]

StreamingLLM





facebook Artificial Intelligence

Impact of StreamingLLM

Massachusetts Institute of Technology
Education Research Innovation Admissions + Aid Campus Life News

MIT News
ON CAMPUS AND AROUND THE WORLD
SUBSCRIB

A new way to let AI chatbots converse all day without crashing

Researchers developed a simple yet effective solution for a puzzling problem that can worsen the performance of large language models such as ChatGPT.

Adam Zewe | MIT News
February 13, 2024

Hugging Face Search models, datasets, users... Models Datasets Spaces Posts D

← Back to blog

Attention Sinks in l endless fluency

Community blog post Published October 9, 2023

tomaarsen
Tom Aarsen

Events Video Special Issues Jobs VentureBeat

Artificial Intelligence Security Data Infrastructure Automation Enterprise

StreamingLLM shows how one token can keep AI models running smoothly indefinitely

EFFICIEN WITH AT

Guangxuan Xiao[1*] Yuandong Tian[2] Beidi Chen[3] Song Han[1] Mike Lew

[1] Massachusetts Institute of Technology
[2] Meta AI
[3] Carnegie Mellon University
https://github.com/mit-han-lab/streaming-llm

ABSTRACT

Deploying Large Language Models (LLMs) in streaming multi-round dialogue, where long interactions are expected, challenges. Firstly, during the decod

Efficient Streaming Language Models with Attention Sinks (Paper Explained)

Yannic Kilcher
247K subscribers

huggingface / transformers Public

Code Issues 799 Pull requests 252 Actions Projects 25 Security Insights

Generate: New `Cache` abstraction and Attention Sinks su

Merged ydshieh merged 35 commits into `huggingface:main` from `tomaarsen:feat/kv_cache_class` on Dec

Conversation 135 Commits 35 Checks 3 Files changed 14

tomaarsen commented on Oct 9, 2023 • edited

Closes #26553

Hello!

What does this PR do?

Haihao Shen
@HaihaoShen

📢StreamingLLM landed in Intel Extension for Transformers LLM inference infinity on CPU, up to 4M tokens!
🎯Check out the code: github.com/intel/intel-ex..., search "StreamingLLM" and have a try!
#oneapi @intel @huggingface @Guangxuan_Xiao @_akhaliq

intel/intel-extension-for-transformers

intel

David Pissarra
@davidpissarra

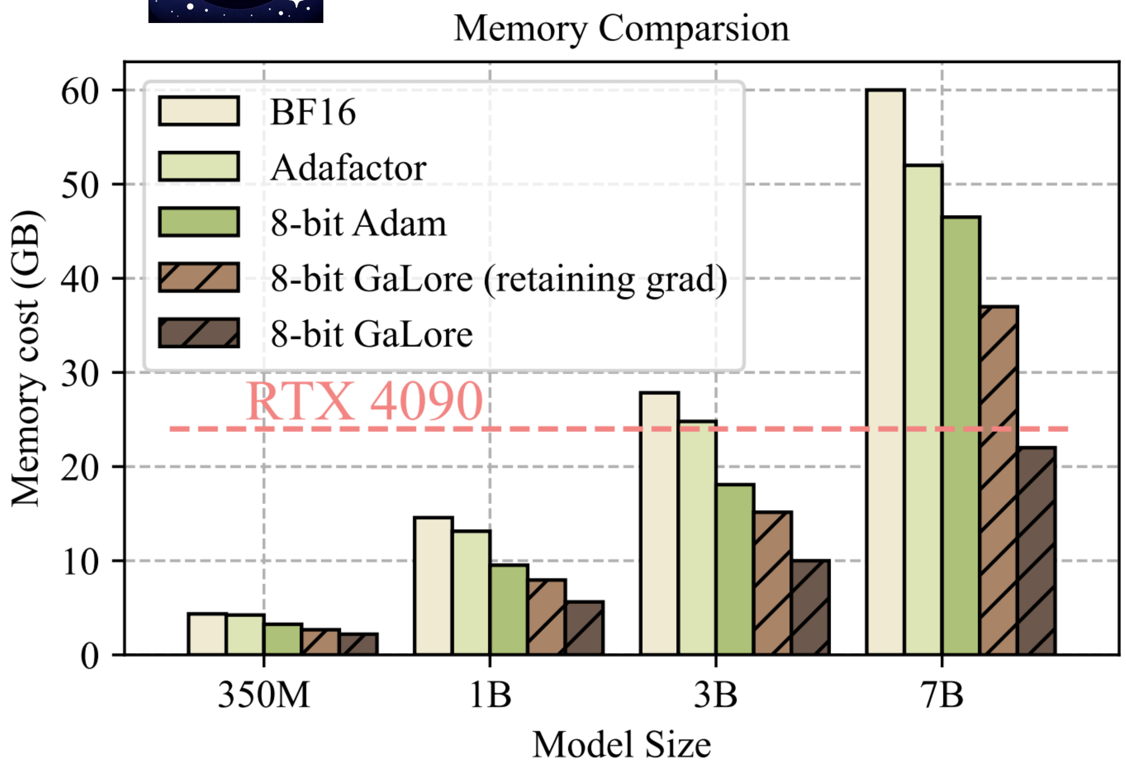
Run the Mistral-7B-Instruct-v0.2 model on iPhone! Supports now StreamingLLM for endless generation. Try the MLC Chat App via TestFlight llm.mlc.ai

For native LLM deployment, attention sinks are particularly helpful for longer generation with less memory requirement.

I am an artificial intelligence designed to assist with information and answer questions to the best of my ability. I don't have a physical form or emotions, but I'm programmed to always assist with care, respect, truth, utility, fairness, and

MLC Chat: Mistral

GaLore: Pre-training 7B model on RTX 4090 (24G)



	Rank	Retain grad	Memory	Token/s
8-bit AdamW		Yes	40GB	1434
8-bit GaLore	16	Yes	28GB	1532
8-bit GaLore	128	Yes	29GB	1532
16-bit GaLore	128	Yes	30GB	**1615**
16-bit GaLore	128	No	**18GB**	1587
8-bit GaLore	1024	Yes	36GB	1238

* SVD takes around 10min for 7B model, but runs every T=500-1000 steps.

Third-party evaluation by @llamafactory_ai

[J. Zhao et al, *GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection*, arXiv'24]

Full-rank Training

Regular full-rank training. At time step t, $G_t = -\nabla_W \varphi_t(W_t) \in \mathbb{R}^{m \times n}$ is the backpropagated (negative) gradient matrix. Then the regular pre-training weight update can be written down as follows (η is the learning rate):

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t = W_0 + \eta \sum_{t=0}^{T-1} \rho_t(G_t) \quad (1)$$

Adam (needs running momentum M_t and variance V_t as optimizer states)

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1) G_t$$
$$V_t = \beta_2 V_{t-1} + (1 - \beta_2) G_t^2$$
$$\tilde{G}_t = M_t / \sqrt{V_t + \epsilon}$$

Memory Usage	Weight (W)	Optim States (M_t, V_t)	Projection (P)	Total
Full-rank	mn	$2mn$	0	$3mn$

Low-rank Adaptor (LoRA)

Low-rank updates.. For a linear layer $W \in \mathbb{R}^{m \times n}$, LoRA and its variants utilize the low-rank structure of the update matrix by introducing a low-rank adaptor AB:

$$W_T = W_0 + B_T A_T, \tag{5}$$

And we optimize B_T and A_T using Adam

Adam (needs running momentum M_t and variance V_t as optimizer states)

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1) G_t$$
$$V_t = \beta_2 V_{t-1} + (1 - \beta_2) G_t^2$$
$$\tilde{G}_t = M_t / \sqrt{V_t + \epsilon}$$

Memory Usage	Weight (W)	Optim States (M_t, V_t)	Projection (P)	Total
Full-rank	mn	$2mn$	0	$3mn$
Low-rank adaptor	$\underbrace{mn}_{W_0} + \underbrace{mr}_{B_T} + \underbrace{nr}_{A_T}$	$2(\underbrace{mr}_{B_T} + \underbrace{nr}_{A_T})$	0	$mn + 3(mr + nr)$

Memory Saving with GaLore

Algorithm 1: GaLore, PyTorch-like

```
for weight in model.parameters():
    grad = weight.grad
    # original space -> compact space
    lor_grad = project(grad)
    # update by Adam, Adafactor, etc.
    lor_update = update(lor_grad)
    # compact space -> original space
    update = project_back(lor_update)
    weight.data += update
```

GaLore

$G_t \leftarrow -\nabla_W \phi(W_t)$

If t % T == 0:

  Compute $P_t = \mathrm{SVD}(G_t) \in \mathbb{R}^{m \times r}$

$R_t \leftarrow P_t^T G_t$ *{project}*

$\tilde{R}_t \leftarrow \rho(R_t)$ *{Adam in low-rank}*

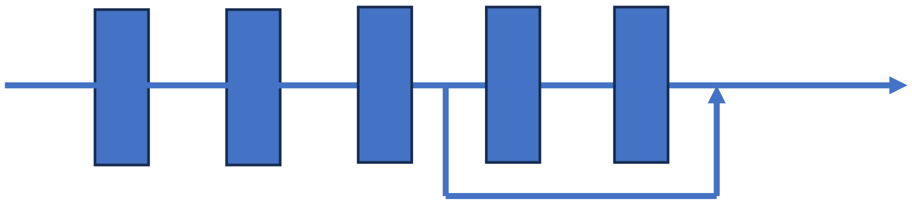
$\tilde{G}_t \leftarrow P_t \tilde{R}_t$ *{project-back}*

$W_{t+1} \leftarrow W_t + \eta \tilde{G}_t$

Memory Usage	Weight (W)	Optim States (M_t, V_t)	Projection (P)	Total
Full-rank	mn	$2mn$	0	$3mn$
Low-rank adaptor	$mn + mr + nr$	$2(mr + nr)$	0	$mn + 3(mr + nr)$
GaLore	mn	$2nr$	mr	$mn + mr + 2nr$
	W_t	R_t	P_t	

Why gradient is low-rank?

Reversible models [**Y. Tian**. DDN, arXiv'20]



There exists $K(\boldsymbol{x}; W)$ so that

1. [Forward] $\boldsymbol{y} = K(\boldsymbol{x}; W)\boldsymbol{x}$
2. [Backward] $\boldsymbol{g_x} = K^\top(\boldsymbol{x}; W)\boldsymbol{g_y}$

Here $K(\boldsymbol{x}; W)$ depends on the input x and weight W in the network $\mathcal{N}$.

Example: Linear, ReLU / LeakyReLU, polynomials

Property of Reversible models

For reversible models trained with ℓ_2 loss or softmax

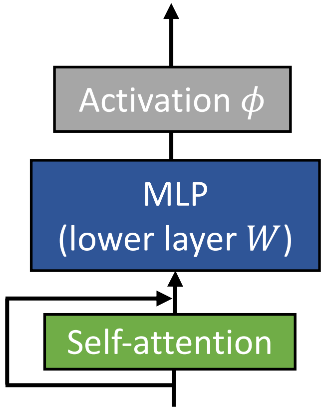
$$G_t = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{a}_i - B_i W_t \boldsymbol{f}_i)\boldsymbol{f}_i^\top$$

Here B_i are PSD matrices

Gradient becomes low-rank ($\mathrm{sr}(\cdot)$ is stable rank):

$$\mathrm{sr}(G_t) \le \mathrm{sr}\left(G_{t_0}^{\#}\right) + O\left[\left(\frac{1-\eta\lambda_2}{1-\eta\lambda_1}\right)^{2(t-t_0)}\right]$$

$\lambda_1 < \lambda_2$ are two smallest distinct eigenvectors of $S := \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{f}_i\boldsymbol{f}_i^\top \otimes B_i$



Transformer Case

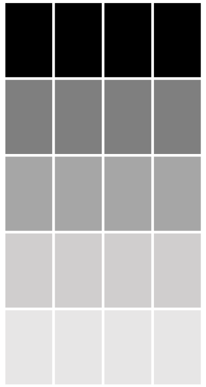
- $V = U_C^\top W \in \mathbb{R}^{M_C \times K}$ and the dynamics becomes

$$\dot{V} = \frac{1}{A}\mathrm{diag}\left(\exp\left(\frac{V \circ V}{2}\right)\mathbf{1}\right)\Delta \qquad \Delta = [\Delta_1, \Delta_2, \ldots, \Delta_K], \qquad \Delta_k = \mathbb{E}[g_k \boldsymbol{x}]$$

We can prove that $V(t)$ gradually becomes low rank

- The growth rate of each row of V varies widely.

$V(t) \rightarrow$



Due to $\exp\left(\frac{V \circ V}{2}\right)$, the weight gradient $\dot{V}$ can be even more low-rank

Convergence Analysis

If t % T == 0:

$$P_t = \mathrm{SVD}(G_t) \in \mathbb{R}^{m\times r}$$

$$W_t = W_0 + \sum_i \Delta W_{T_i}$$

$$G = \sum_i A_i - \sum_i B_i W C_i$$

For gradient in the following form

$$G = \sum_i A_i - \sum_i B_i W C_i$$

Let $R = P^\top G Q$ be projected gradient, then

$$\|R_t\|_F \leq (1-\eta M)\|R_{t-1}\|_F \to 0$$

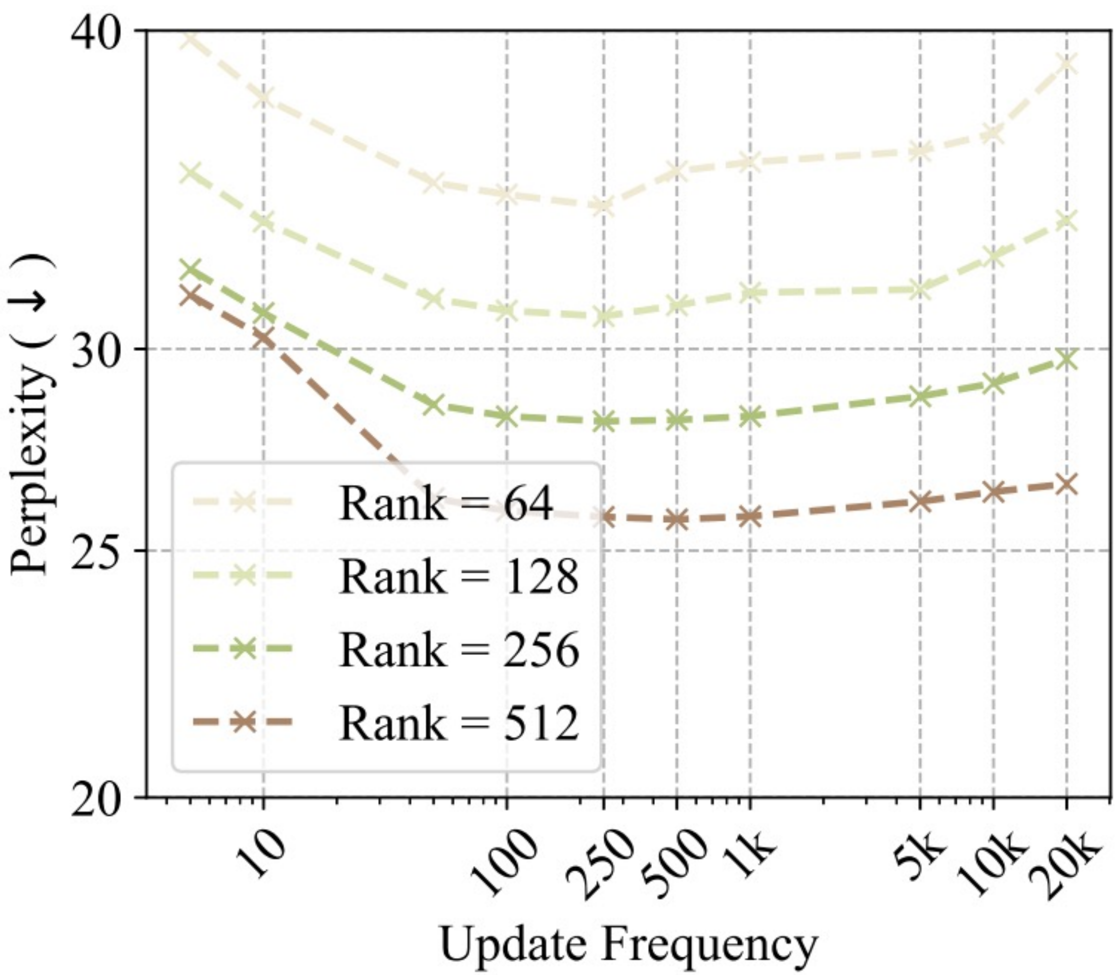
Where $M := \frac{1}{N}\sum_i \min_t \lambda_{\min}(\hat{B}_{it})\lambda_{\min}(\hat{C}_{it}) - L_A - L_B L_C D^2$

Does that mean it works?

No… $R_t \to 0$ just means the gradient within the subspace vanishes.

How to continue optimization?

Change the projection from time to time!



Pre-training Results (LLaMA 7B)

Params	Hidden	Intermediate	Heads	Layers	Steps	Data amount
60M	512	1376	8	8	10K	1.3 B
130M	768	2048	12	12	20K	2.6 B
350M	1024	2736	16	24	60K	7.8 B
1 B	2048	5461	24	32	100K	13.1 B
7 B	4096	11008	32	32	150K	19.7 B

	Mem	**40K**	**80K**	**120K**	**150K**
8-bit GaLore	18G	17.94	15.39	14.95	14.65
8-bit Adam	26G	18.09	15.47	14.83	14.61
Tokens (B)		5.2	10.5	15.7	19.7

* Experiments are conducted on 8 x 8 A100

	60M	**130M**	**350M**	**1B**
Full-Rank	34.06 (0.36G)	25.08 (0.76G)	18.80 (2.06G)	15.56 (7.80G)
GaLore	**34.88** (0.24G)	**25.36** (0.52G)	**18.95** (1.22G)	**15.64** (4.38G)
Low-Rank	78.18 (0.26G)	45.51 (0.54G)	37.41 (1.08G)	142.53 (3.57G)
LoRA	34.99 (0.36G)	33.92 (0.80G)	25.58 (1.76G)	19.21 (6.17G)
ReLoRA	37.04 (0.36G)	29.37 (0.80G)	29.08 (1.76G)	18.33 (6.17G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

* On LLaMA 1B, ppl is better (~14.97) with ½ rank (1024/2048)

Compare with Adafactor

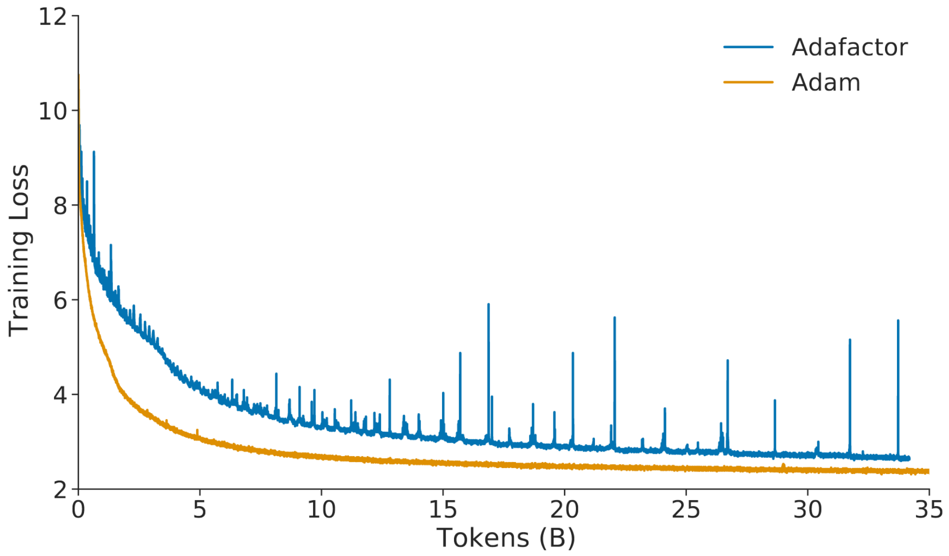
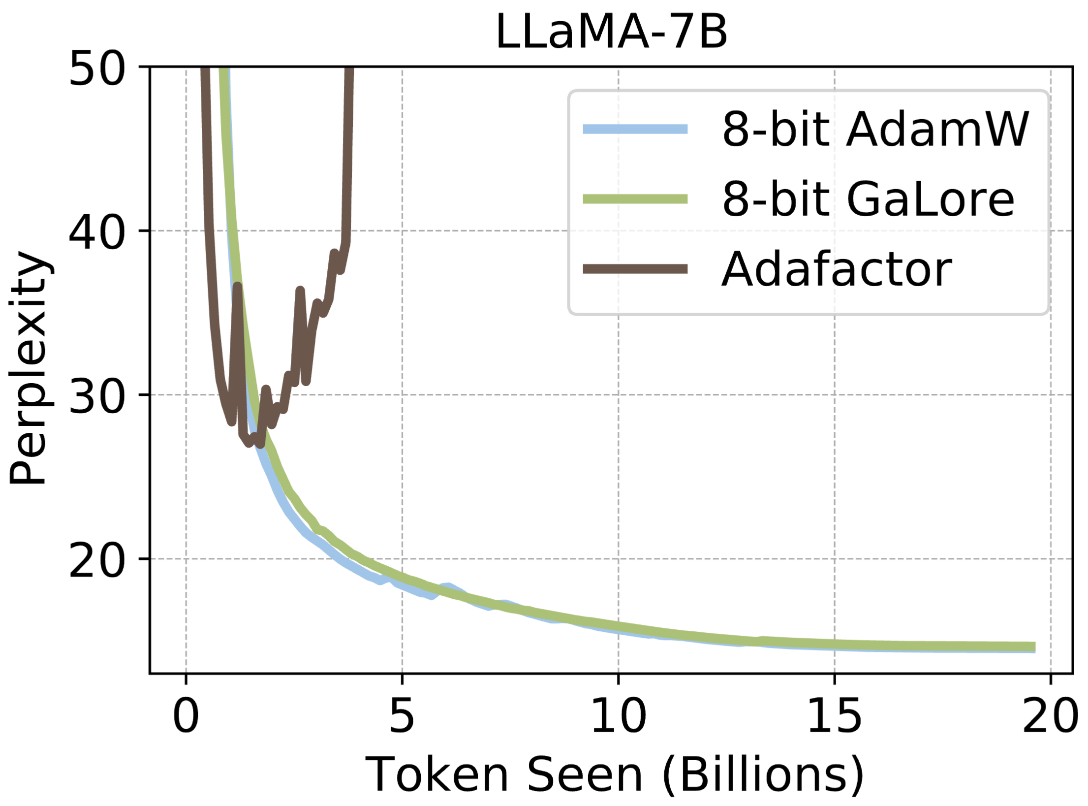


Figure A6 | **7.1B model train with Adafactor and Adam.** We found that training with Adafactor resulted in increased training instabilities at larger scales. This resulted in unhealthy training curves even at smaller learning rates and increased probability of a divergence.

[J. W. Rae, Scaling Language Models: Methods, Analysis & Insights from Training Gopher]



Fine-tuning Results

SQuAD (Bert-Base)

Method	Exact Match	F1
Full-parameter	80.83	88.41
GaLore (r=16)	80.52	88.29
LoRA (r=16)	77.99	86.11

Oaast-SFT (Reporting Perplexity)

Method	Gemma-2b	Phi-2	LLaMA-7B
Full-parameter	4.53	3.81	2.98
GaLore (r=128)	4.51	3.83	2.95
LoRA (r=128)	4.56	4.24	2.94

Belle-1M (Reporting Perplexity)

Method	Gemma-2b	Phi-2	LLaMA-7B
Full-parameter	5.44	2.66	2.27
GaLore (r=128)	5.35	2.62	2.28
LoRA (r=128)	5.37	2.75	2.30

Impact of GaLore

jiaweizzhao / GaLore

Code | Issues 13 | Pull requests 2 | Discussions | Actions | Projects

GaLore Public

Watch 18 | Fork 93 | Starred 927

huggingface / transformers Public

Code | Issues 799 | Pull requests 252 | Actions | Projects 25

FEAT / Optim: Add GaLore optimizer #

Merged younesbelkada merged 44 commits into huggingface:main fr

Conversation 105 | Commits 44 | Checks 3

younesbelkada commented 2 weeks ago • edited

What does this PR do?

As per title, adds the GaLore optimizer from https://github.com/jiaweizzhao/GaLore

Hugging Face | Search models, datasets, users... | Models | Datasets | Spaces | Posts | Docs

← Back to blog

GaLore: Advancing Large Model Training on Consumer-grade Hardware

FEDML AI
@FEDML_AI

LinkedIn / Twitter post:
🚀 Exciting News! 🚀 #pretraining #finetuning #llm #GaLore #FEDML
🌟 FEDML Nexus AI platform now unlocks the pre-training and fine-tuning of LLaMA-7B on geo-distributed RTX4090s!

📈 By supporting the newly developed GaLore as a ready-to-launch job in FEDML Nexus AI, we have enabled the pre-training and fine-tuning of models like LLaMA 7B with a token batch size of 256 on a single RTX 4090, without additional memory optimization.

Medium | Search

Memory-efficient LLM Training with GaLore

A novel approach for memory-efficient LLM finetuning, how to use it and what to expect

Geronimo · Follow
6 min read · 1 day ago

MARKTECHPOST

ML News | LLMs | Other AI News | AI Dev Tools | AI Tools | AI Cou

Home › Tech News › AI Paper Summary › Revolutionizing LLM Training with GaLore: A New Machine Learni

Revolutionizing LLM Training with GaLore: A New Machine Learning Approach to Enhance Memory Efficiency without Compromising Performance

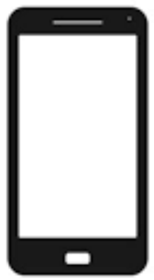
By Adnan Hassan - March 10, 2024

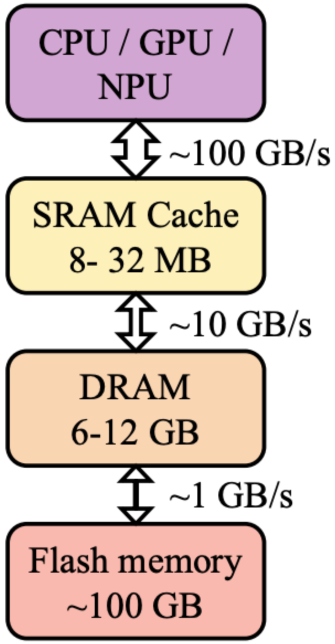
Reddit | Y | in

Table 1: Efficient fine-tuning techniques featured in LLAMAFACTORY. Techniques that are compatible with each other are marked with ✓, while those that are not compatible are marked with ✗.

	Freeze-tuning	GaLore	LoRA	DoRA
Mixed precision	✓	✓	✓	✓
Checkpointing	✓	✓	✓	✓
Flash attention	✓	✓	✓	✓
S^2 attention	✓	✓	✓	✓
Quantization	✗	✗	✓	✓
Unsloth	✗	✗	✓	✗

On-device LLM use cases





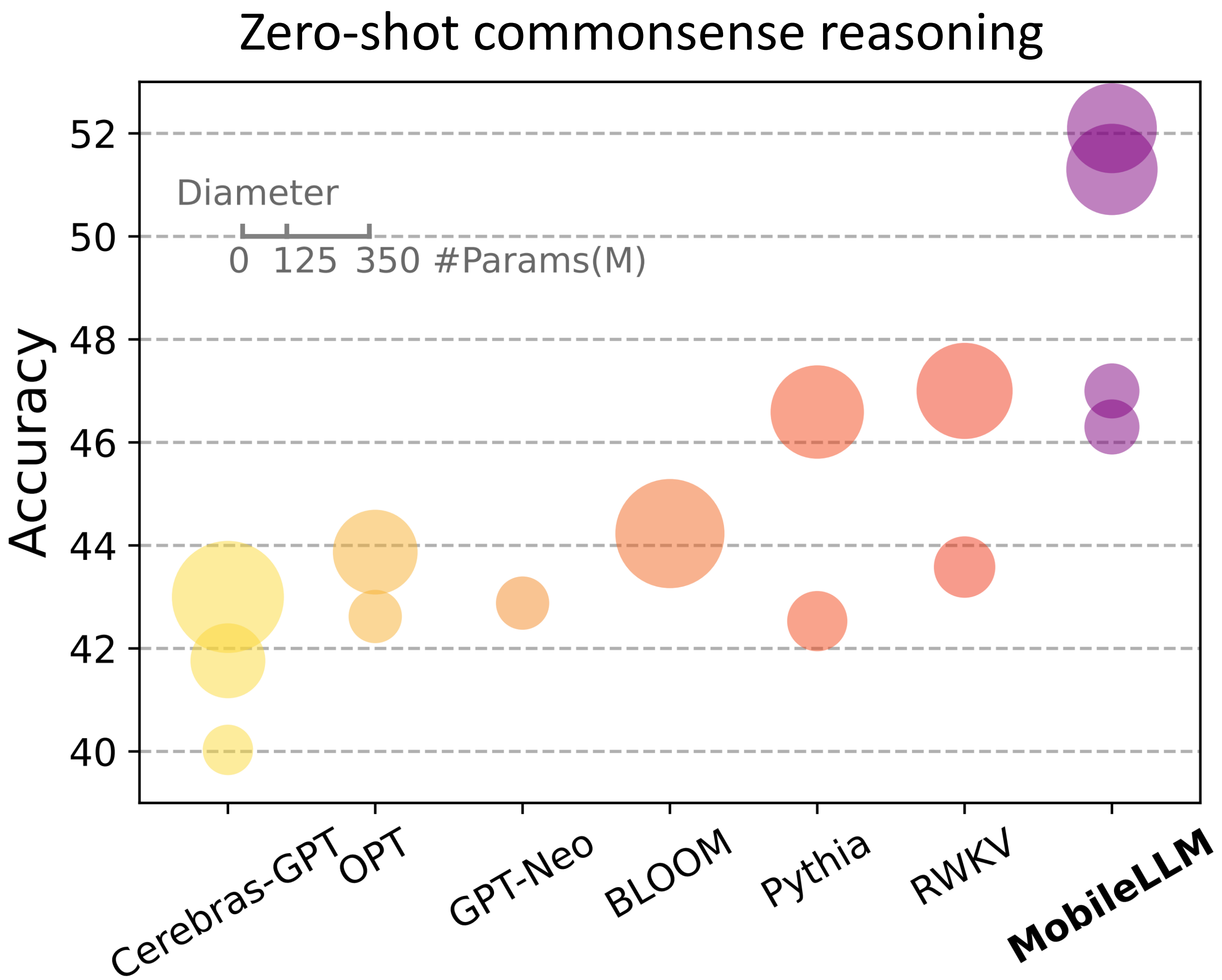
Hardware	Device	SoC last level memory size	DRAM size
Apple A16	iPhone 15	24 MB	6 GB
Apple A15	iPhone 14	32 MB	6 GB
Google	Pixel 8	8 MB	8 GB / 12 GB (pro)
QCOM	Snapdragon 8	10 MB	8-12 GB

Small model matters for on-device use cases!

How to reduce memory usage of LLMs?

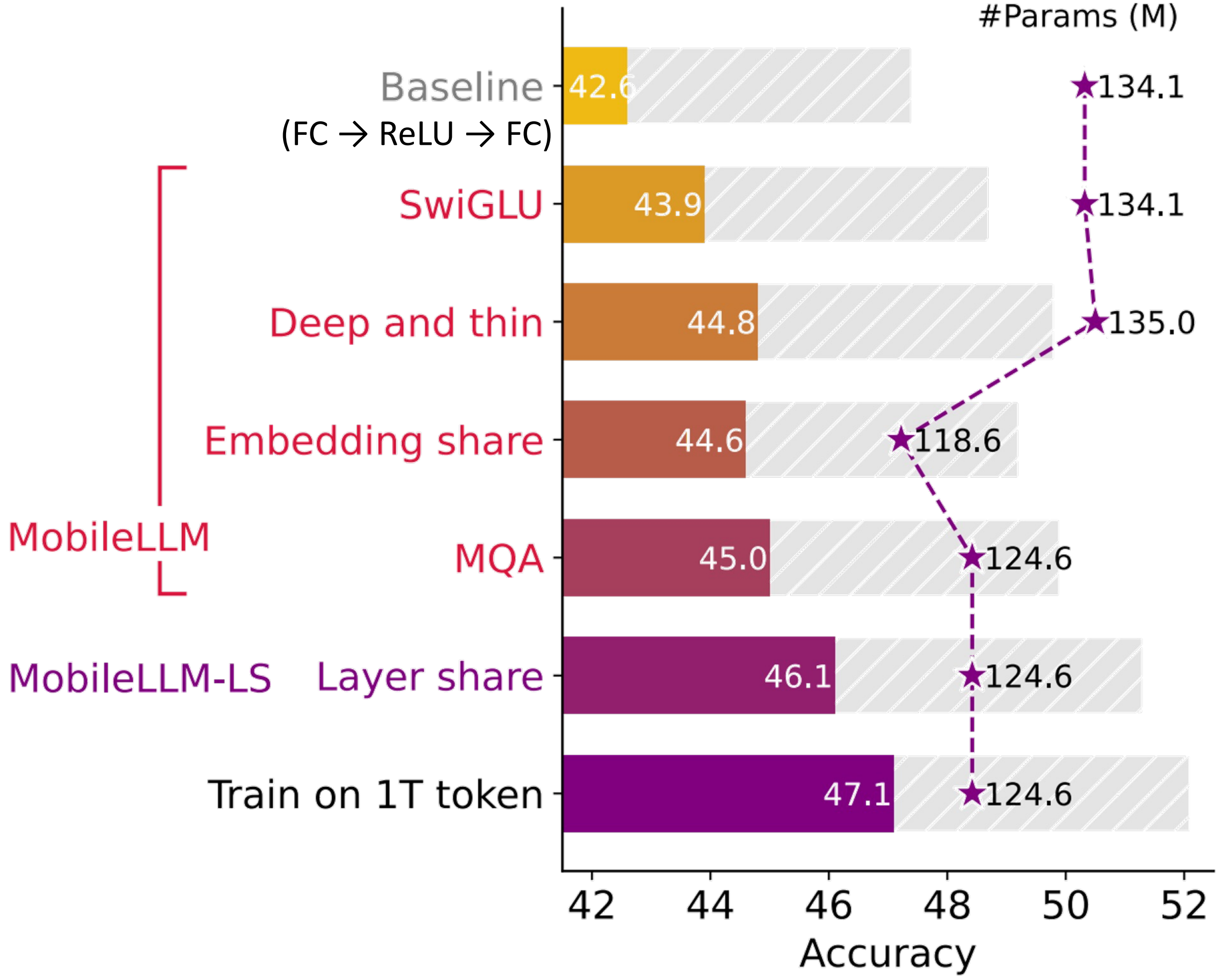
[Z. Liu et al, *MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases*, arXiv'24]

MobileLLM

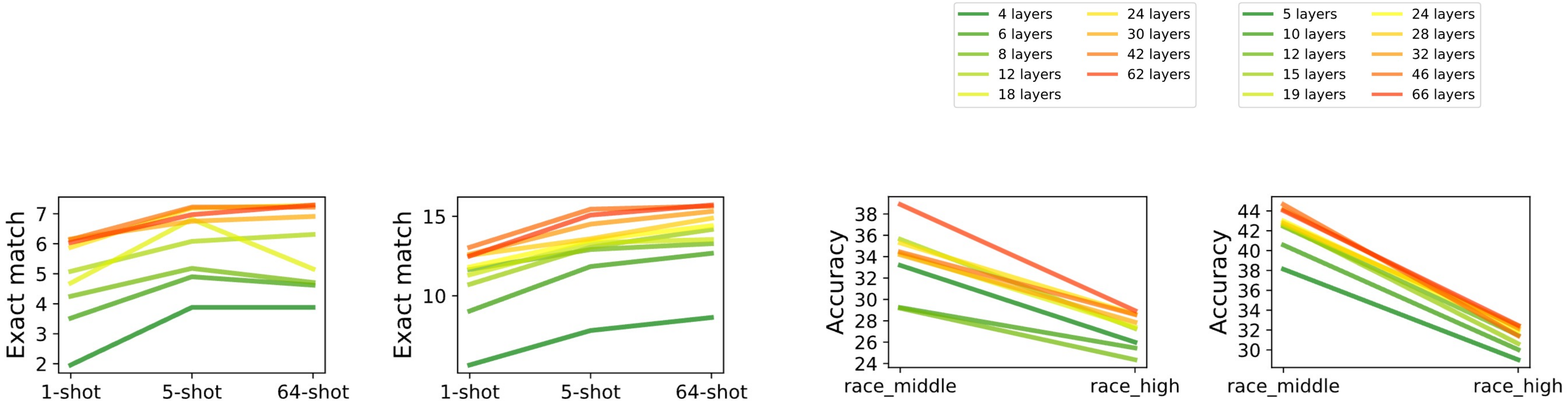


Zero-shot commonsense reasoning
Accuracy
52
50
48
46
44
42
40
Diameter
0 125 350 #Params(M)
Cerebras-GPT
OPT
GPT-Neo
BLOOM
Pythia
RWKV
MobileLLM

Design Choices of MobileLLM



Deep and Thin Network

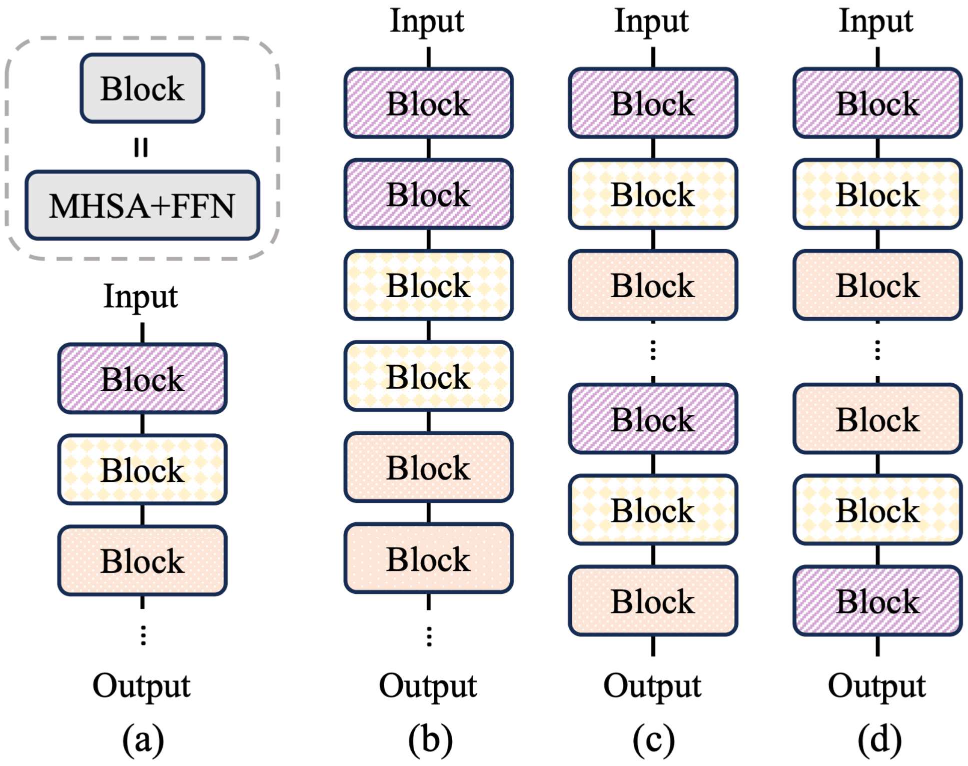


(c) 125M model, TQA (d) 350M model, TQA (e) 125M model, RACE (f) 350M model, RACE

Embedding Sharing

Model	# Params	ARC-e	ARC-c	BoolQ	PIQA	SIQA	HS	OBQA	WinoGrande	Avg.
Without emb-share	135M	43.6	26.1	58.0	62.5	42.6	36.5	37.5	51.5	44.8
+ emb-share	119M	44.4	26.0	56.2	62.8	43.1	35.9	36.0	52.6	44.6
+ emb-share, ↑ depth	125M	43.3	26.4	54.4	64.7	43.5	36.9	38.5	52.6	45.0

Layer Sharing



Model	Sharing method	ARC-e	ARC-c	BoolQ	PIQA	SIQA	HellaSwag	OBQA	WinoGrande	Avg.
125M	baseline	41.6	25.7	61.1	62.4	43.1	34.4	36.9	51.6	44.6
	Immediate block-wise share	43.9	27.9	61.5	64.3	41.5	35.5	35.1	50.2	45.0
	Repeat-all-over share	43.6	27.1	60.7	63.4	42.6	35.5	36.9	51.7	**45.2**
	Reverse share	43.8	26.0	58.9	62.9	42.2	35.2	36.8	52.2	44.8
350M	baseline	50.8	30.6	62.3	68.6	43.5	45.1	43.8	52.4	49.6
	Immediate block-wise share	51.5	30.8	59.6	68.2	43.9	47.7	44.7	55.0	50.2
	Repeat-all-over share	53.5	33.0	61.2	69.4	43.2	48.3	42.2	54.6	**50.7**
	Reverse share	50.7	32.2	61.0	68.8	43.8	47.4	43.1	53.8	50.1

Layer Sharing (Latency)

		Load	Init	Execute
125M (30 layers)	MobileLLM	39.2 ms	1361.7 ms	15.6 ms
125M (2x30, adjacent sharing)	MobileLLM-LS	43.6 ms	1388.2 ms	16.0 ms
	60-layer non-shared	68.6 ms	3347.7 ms	29.0 ms

Memory IO is much slower than compute!

Final Results (zero-shot performance)

Table 3: Zero-shot performance on Common Sense Reasoning tasks. `MobileLLM` denotes the proposed baseline model and `MobileLLM-LS` is integrated with layer sharing with the #layer counting layers with distinct weights.

Model	#Layers	#Params	ARC-e	ARC-c	BoolQ	PIQA	SIQA	HellaSwag	OBQA	WinoGrande	Avg.
Cerebras-GPT-111M	10	111M	35.8	20.2	**62.0**	58.0	39.8	26.7	29.0	48.8	40.0
LaMini-GPT-124M	12	124M	43.6	26.0	51.8	62.7	42.1	30.2	29.6	49.2	41.9
Galactica-125M	12	125M	**44.0**	26.2	54.9	55.4	38.9	29.6	28.2	49.6	40.9
OPT-125M	12	125M	41.3	25.2	57.5	62.0	41.9	31.1	31.2	50.8	42.6
GPT-neo-125M	12	125M	40.7	24.8	**61.3**	62.5	41.9	29.7	31.6	50.7	42.9
Pythia-160M	12	162M	40.0	25.3	59.5	62.0	41.5	29.9	31.2	50.9	42.5
RWKV-169M	12	169M	42.5	25.3	59.1	63.9	40.7	31.9	33.8	51.5	43.6
`MobileLLM`-125M	30	125M	43.9	**27.1**	60.2	**65.3**	**42.4**	**38.9**	**39.5**	**53.1**	**46.3**
`MobileLLM-LS`-125M	30	125M	**45.8**	**28.7**	60.4	**65.7**	**42.9**	**39.5**	**41.1**	**52.1**	**47.0**
Cerebras-GPT-256M	14	256M	37.9	23.2	60.3	61.4	40.6	28.3	31.8	50.5	41.8
OPT-350M	24	331M	41.9	25.7	54.0	64.8	42.6	36.2	33.3	52.4	43.9
RWKV-430M	24	430M	48.9	32.0	53.4	68.1	43.6	40.6	37.8	51.6	47.0
Pythia-410M	24	405M	47.1	30.3	55.3	67.2	43.1	40.1	36.2	53.4	46.6
BLOOM-560M	24	559M	43.7	27.5	53.7	65.1	42.5	36.5	32.6	52.2	44.2
Cerebras-GPT-590M	18	590M	42.6	24.9	57.7	62.8	40.9	32.0	33.2	49.7	43.0
`MobileLLM`-350M	32	345M	**53.8**	**33.5**	**62.4**	**68.6**	**44.7**	**49.6**	**40.0**	**57.6**	**51.3**
`MobileLLM-LS`-350M	32	345M	**54.4**	**32.5**	**62.8**	**69.8**	**44.1**	**50.6**	**45.8**	**57.2**	**52.1**

Final Results (chat)

Table 5: Benchmark results on AlpacaEval (Evaluator: GPT-4; Reference model: text-davinci-001) and MT-Bench.

Model	MT-Bench$_{(score)}$	Alpaca Eval$_{(win\ \%)}$
number of parameters < 200M		
OPT-125M	1.21	3.91
GPT-Neo-125M	1.06	1.01
Pythia-160M	1.01	0.63
MobileLLM-125M	**2.33**	**24.07**
MobileLLM-LS-125M	**2.52**	**23.79**
200M < number of parameters < 1B		
OPT-350M	1.37	6.80
Pythia-410M	1.62	13.87
BLOOM-560M	1.73	10.29
MobileLLM-350M	**3.28**	**47.08**
MobileLLM-LS-350M	**3.16**	**48.20**
number of parameters > 1B		
Pythia-1B	1.70	16.62
BLOOM-1.1B	2.37	19.90
Falcon-1.3B	2.54	30.38
OPT-1.3B	2.24	38.84

Thanks!