Using Algorithms to Understand Transformers (and Using Transformers to Understand Algorithms)

Vatsal Sharan (USC)

USC University of Southern California

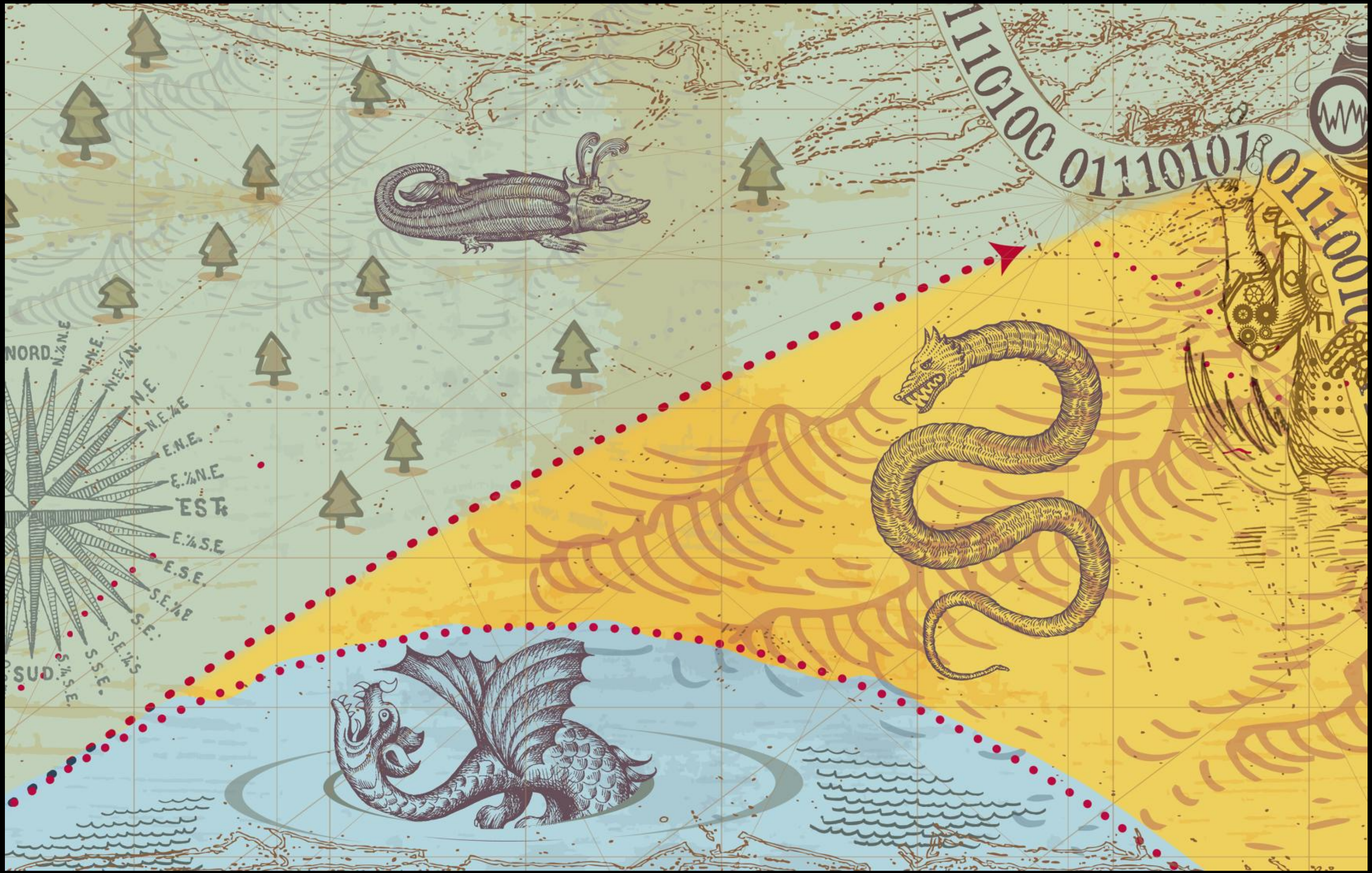
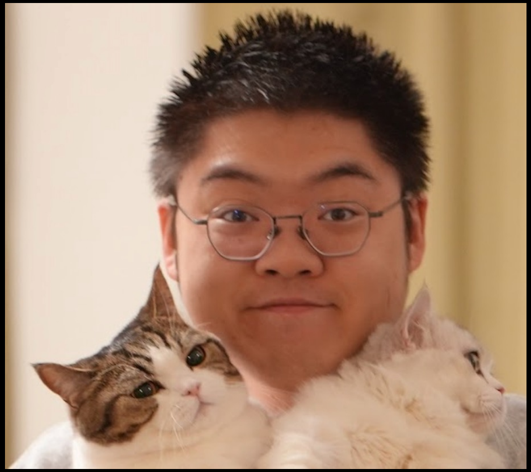


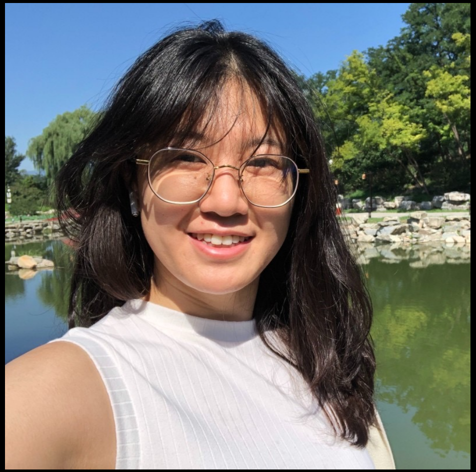
Image source: Simons program on “Computational Complexity of Statistical Inference”

- How can we use understanding of computational and information theoretic landscape to understand Transformers?
- How can we use Transformers to understand and discover algorithms and data structures?

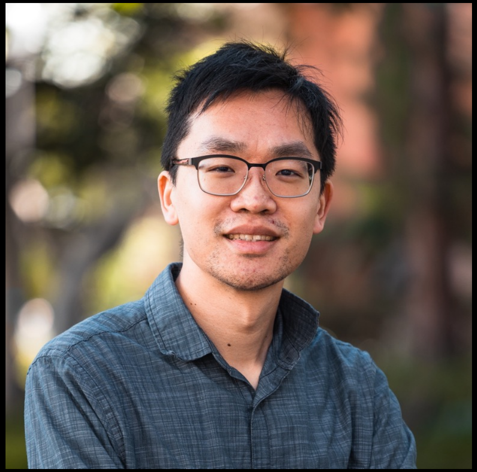
How do Transformers do linear regression?



Deqing Fu (USC)



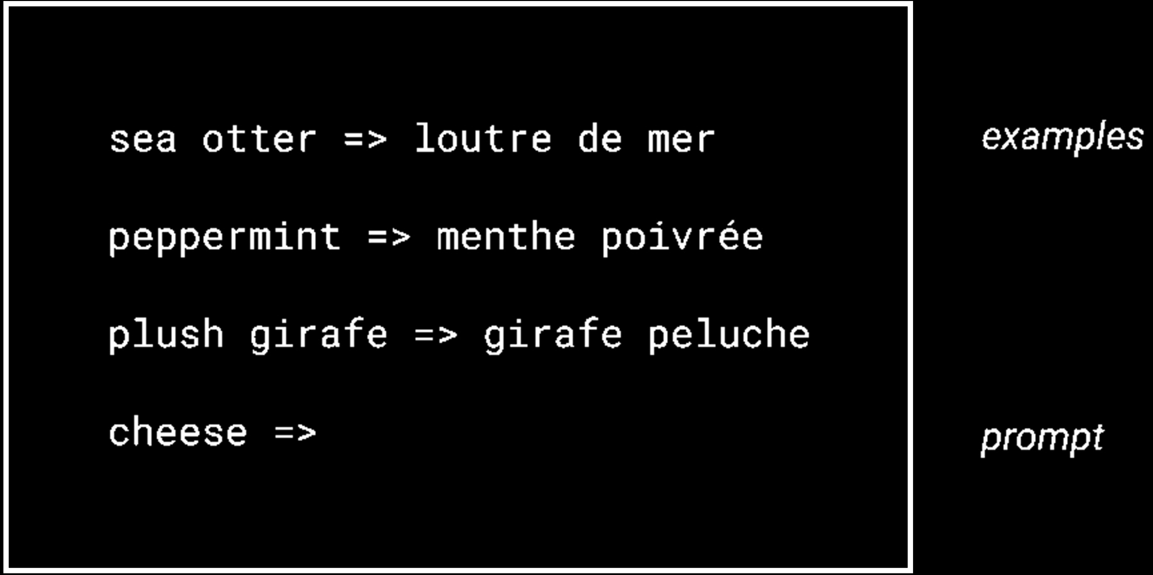
Tianqi Chen (USC)



Robin Jia (USC)

Transformers Learn Higher-Order Optimization Methods for In-Context Learning: A Study with Linear Models, Neurips 2024

Transformers excel at in-context learning



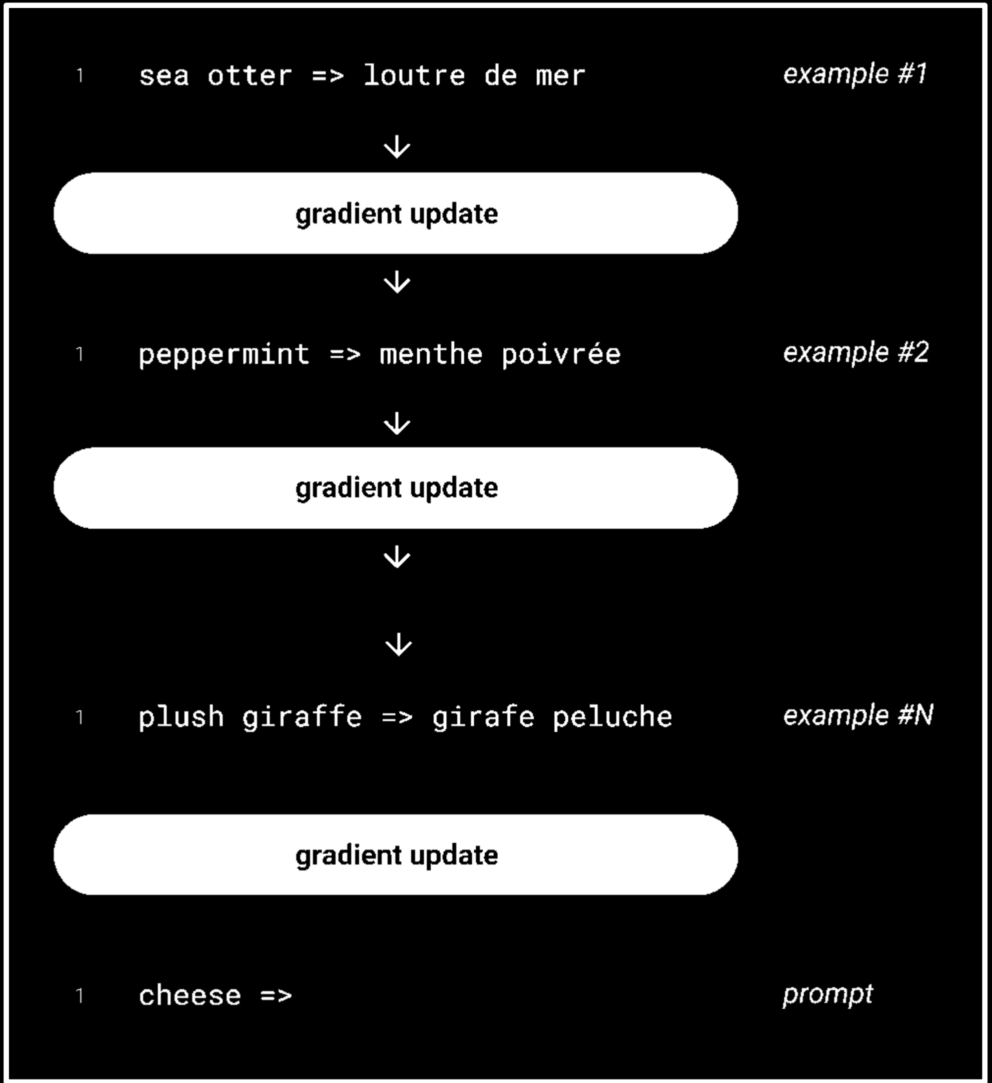
```
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

examples

prompt

In-context learning

vs.



```
sea otter => loutre de mer
```

example #1

gradient update

```
peppermint => menthe poivrée
```

example #2

gradient update

```
plush giraffe => girafe peluche
```

example #N

gradient update

```
cheese =>
```

prompt

Usual fine-tuning

Source: GPT3 paper, OpenAI

How do Transformers do in-context learning?

The case of linear models ($y_i = {w^*}^T x_i$):

$$x_1 = (3, 5), y_1 = 4$$

$$x_2 = (-2, 2), y_2 = 8$$

$$x_3 = (-7, -2), y_3 = 10$$

$$x_4 = (4, -1), y_4 = ?$$

References: Garg-Tsipras-Liang-Valiant 2022, Akyurek-Schuurmans-Andreas-Ma-Zhou 2022

A prevailing hypothesis: Transformers do in-context learning via gradient descent

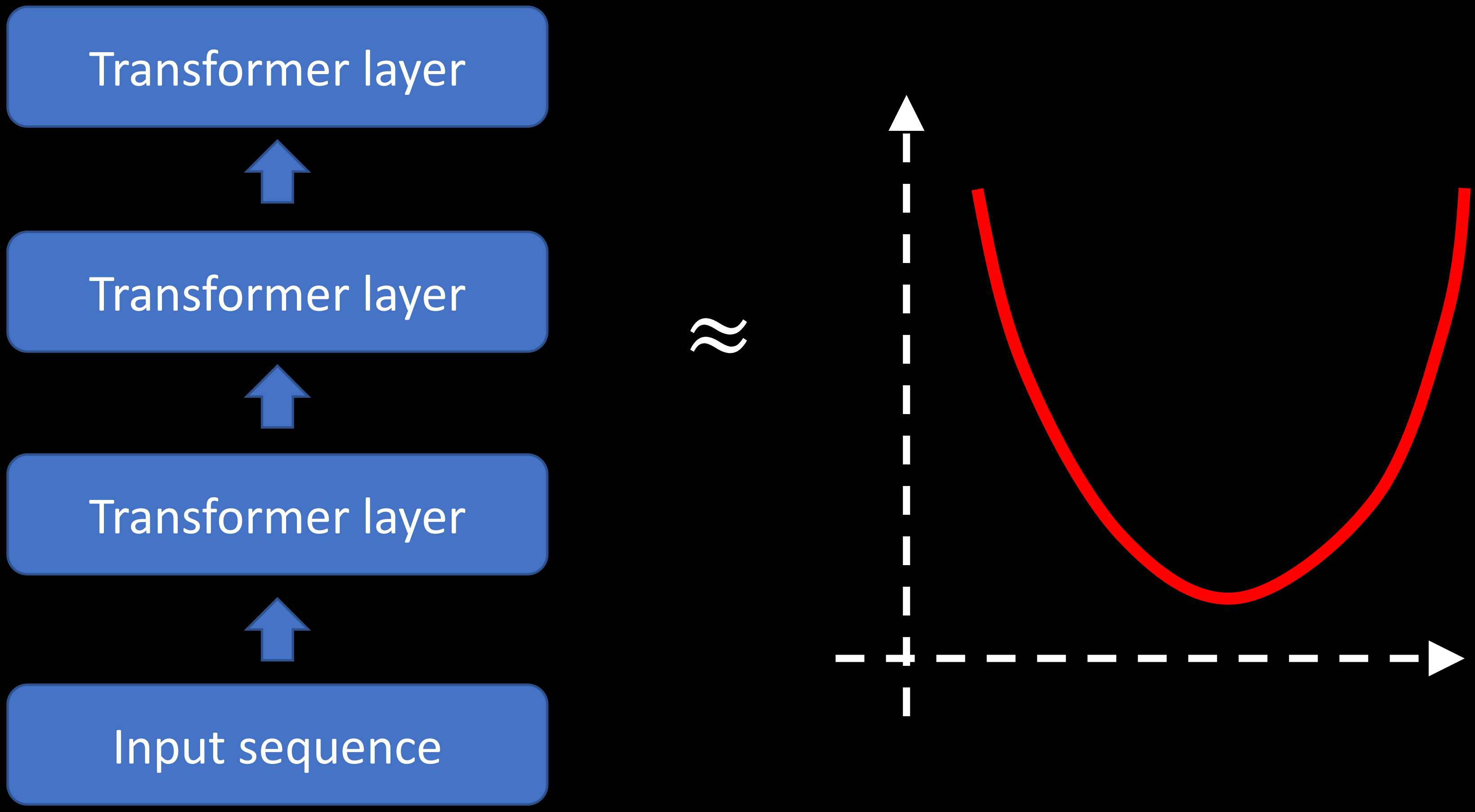
Linear models:

$x_1 = (3, 5), y_1 = 4$

$x_2 = (-2, 2), y_2 = 8$

$x_3 = (-7, -2), y_3 = 10$

$x_4 = (4, -1), y_4 = ?$



References: von Oswald et al. 2022, 2023, Ahn et al. 2023, Dai et al. 2023

A prevailing hypothesis: Transformers do in-context learning via gradient descent

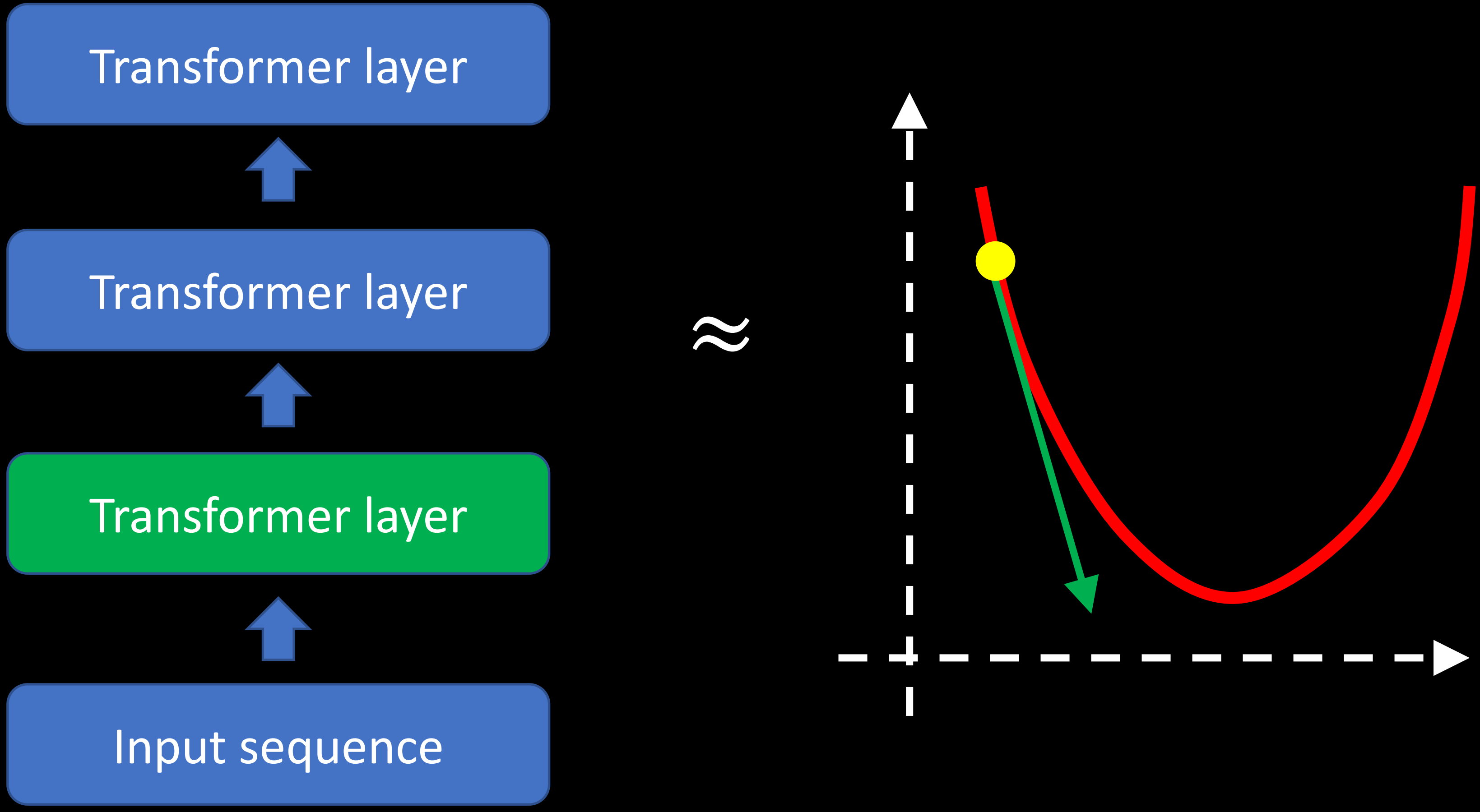
Linear models:

$x_1 = (3, 5), y_1 = 4$

$x_2 = (-2, 2), y_2 = 8$

$x_3 = (-7, -2), y_3 = 10$

$x_4 = (4, -1), y_4 = ?$



A prevailing hypothesis: Transformers do in-context learning via gradient descent

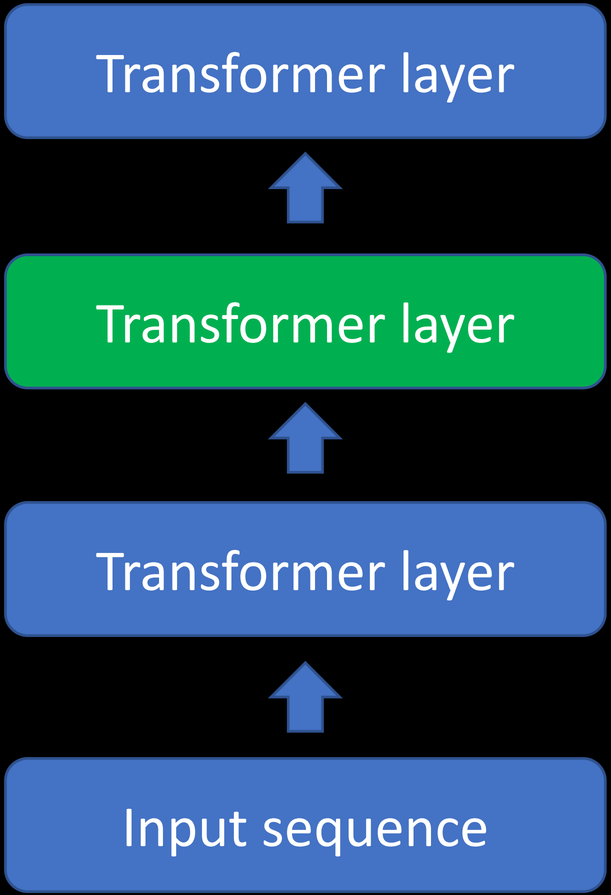
Linear models:

$x_1 = (3, 5), y_1 = 4$

$x_2 = (-2, 2), y_2 = 8$

$x_3 = (-7, -2), y_3 = 10$

$x_4 = (4, -1), y_4 = ?$



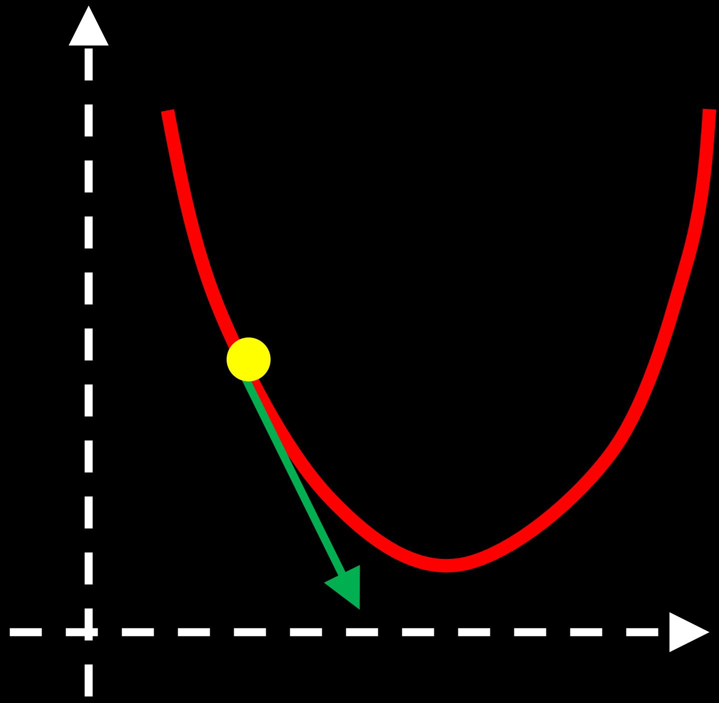
Transformer layer

Transformer layer

Transformer layer

Input sequence

$\approx$



A prevailing hypothesis: Transformers do in-context learning via gradient descent

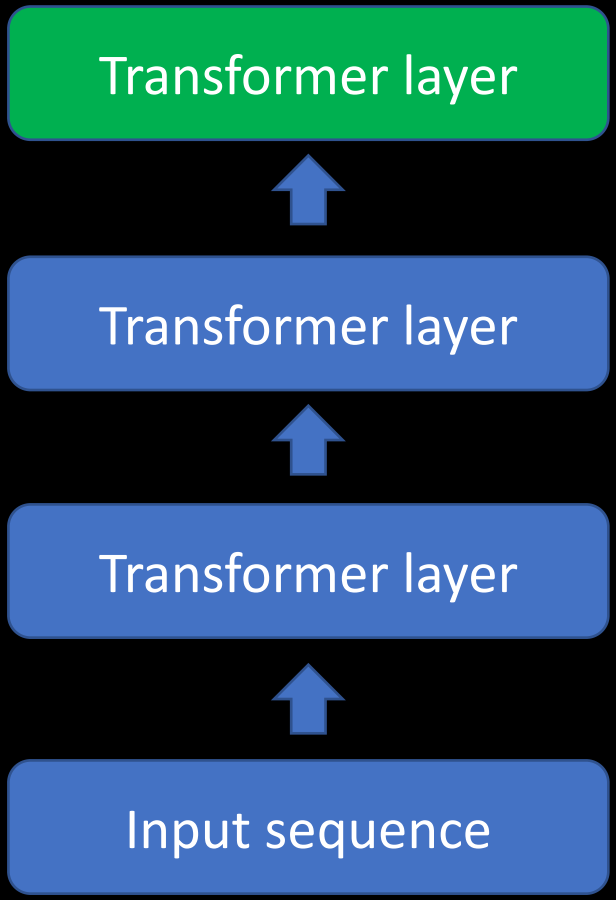
Linear models:

$x_1 = (3, 5), y_1 = 4$

$x_2 = (-2, 2), y_2 = 8$

$x_3 = (-7, -2), y_3 = 10$

$x_4 = (4, -1), y_4 = ?$



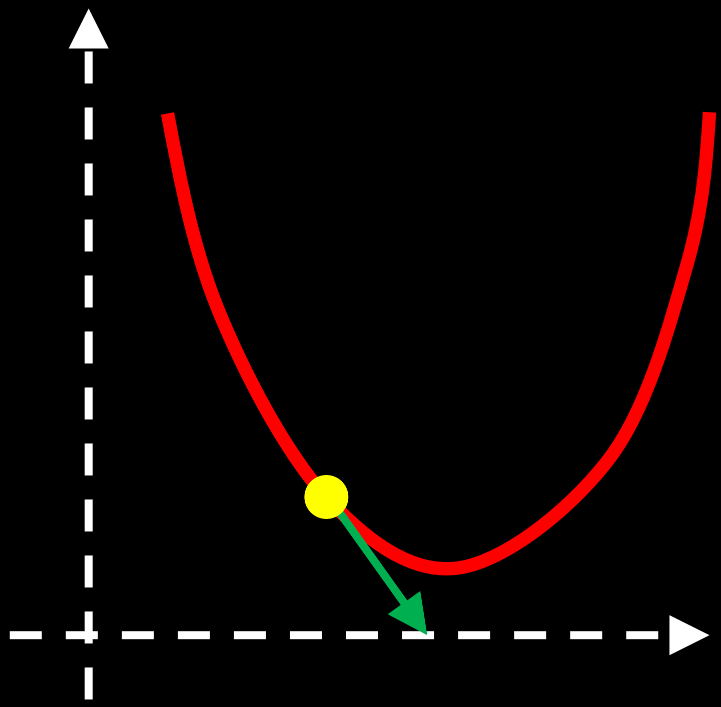
Transformer layer

Transformer layer

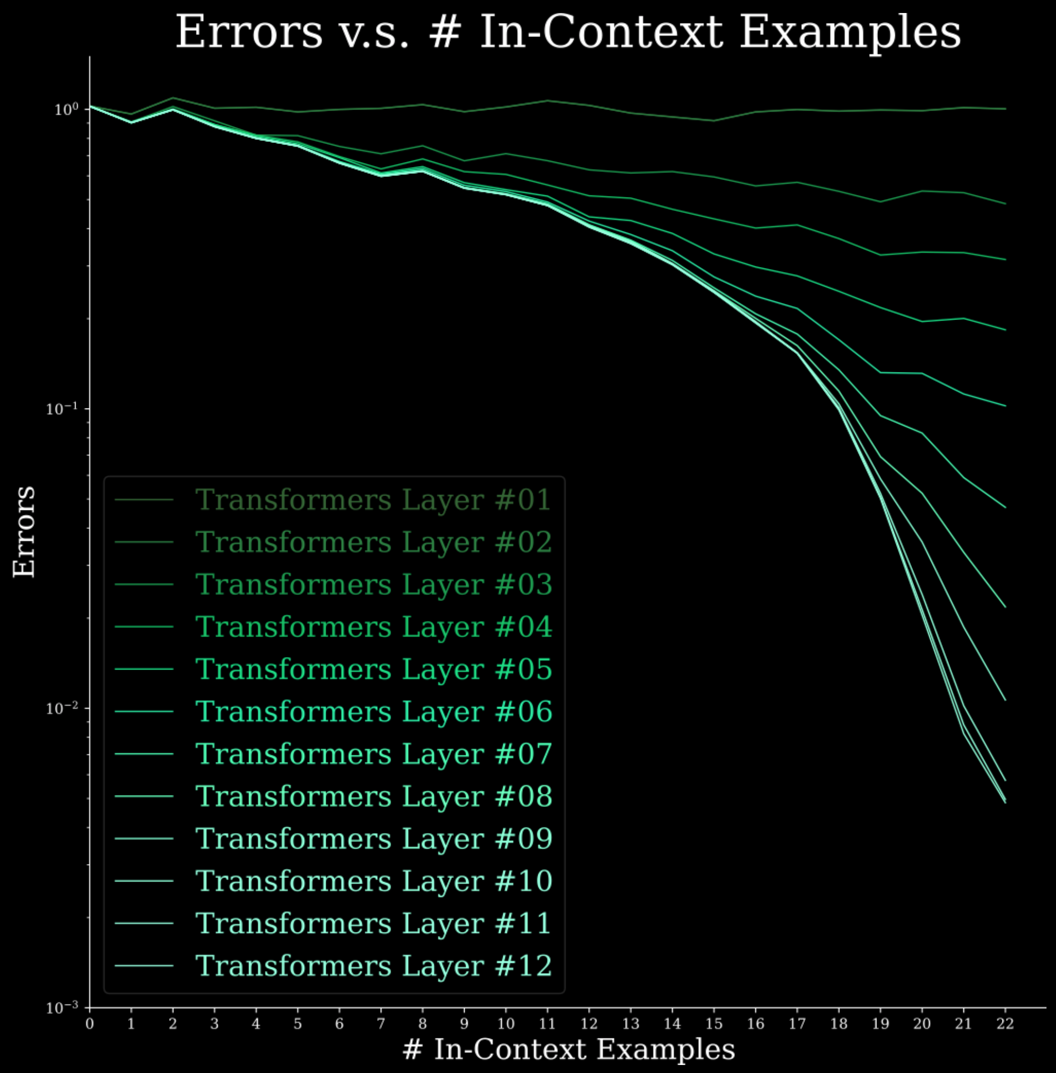
Transformer layer

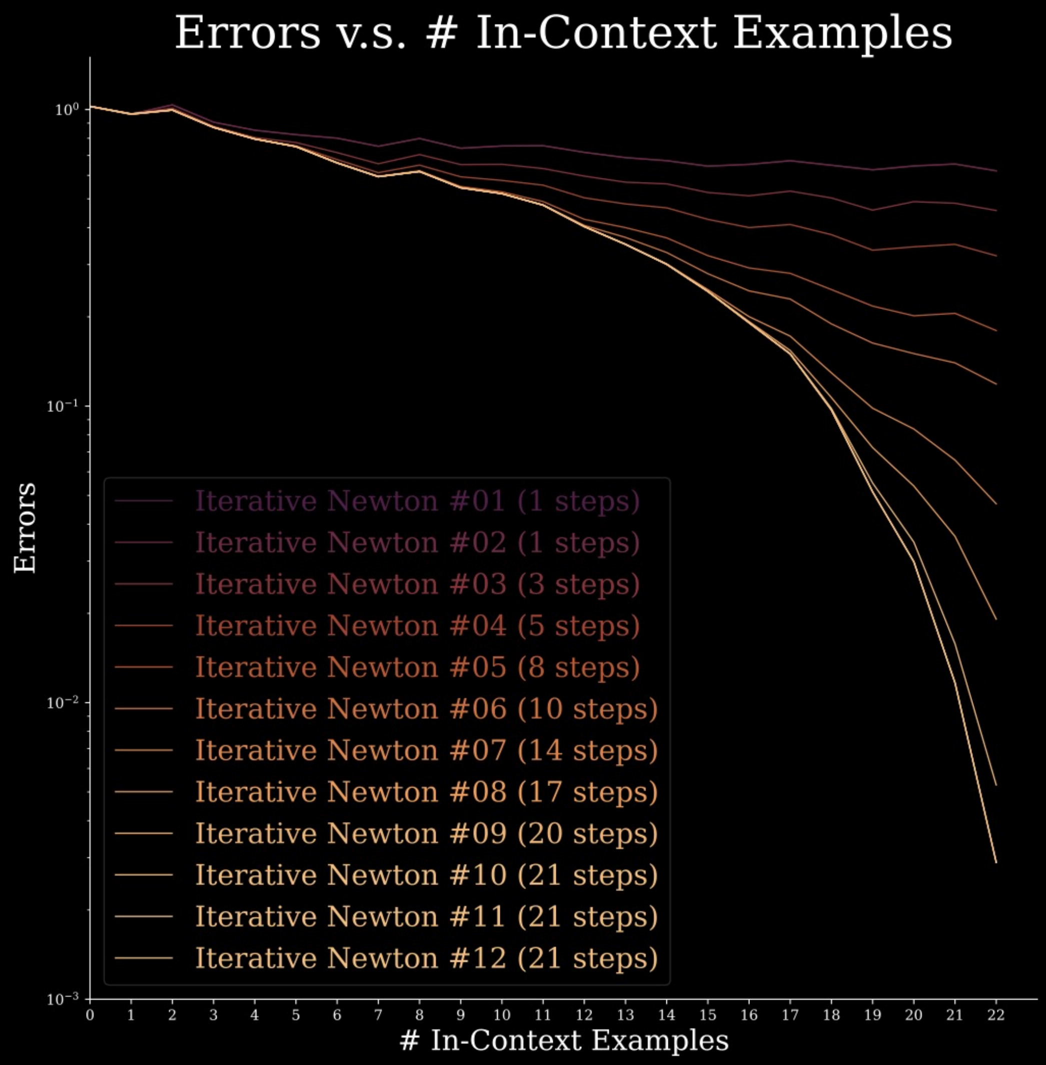
Input sequence

$\approx$



This work: Transformers do in-context learning via an iterative second-order method

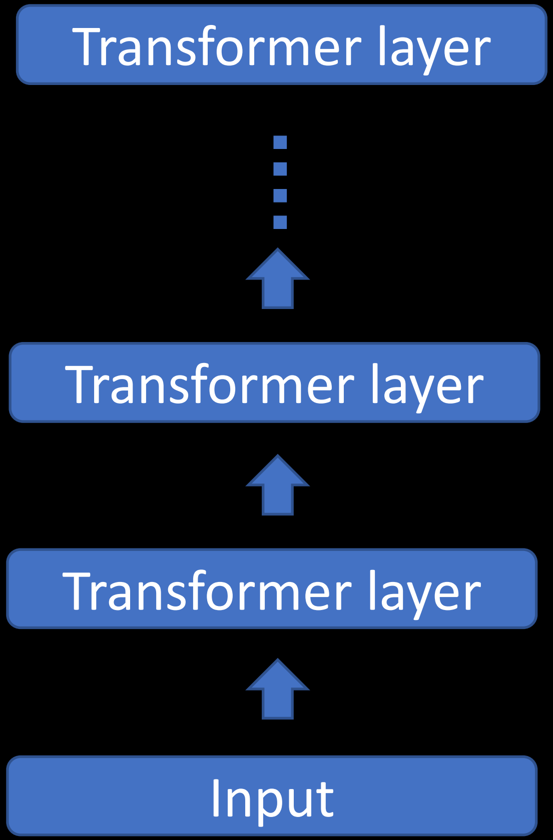




Techniques: “Applied theory”?

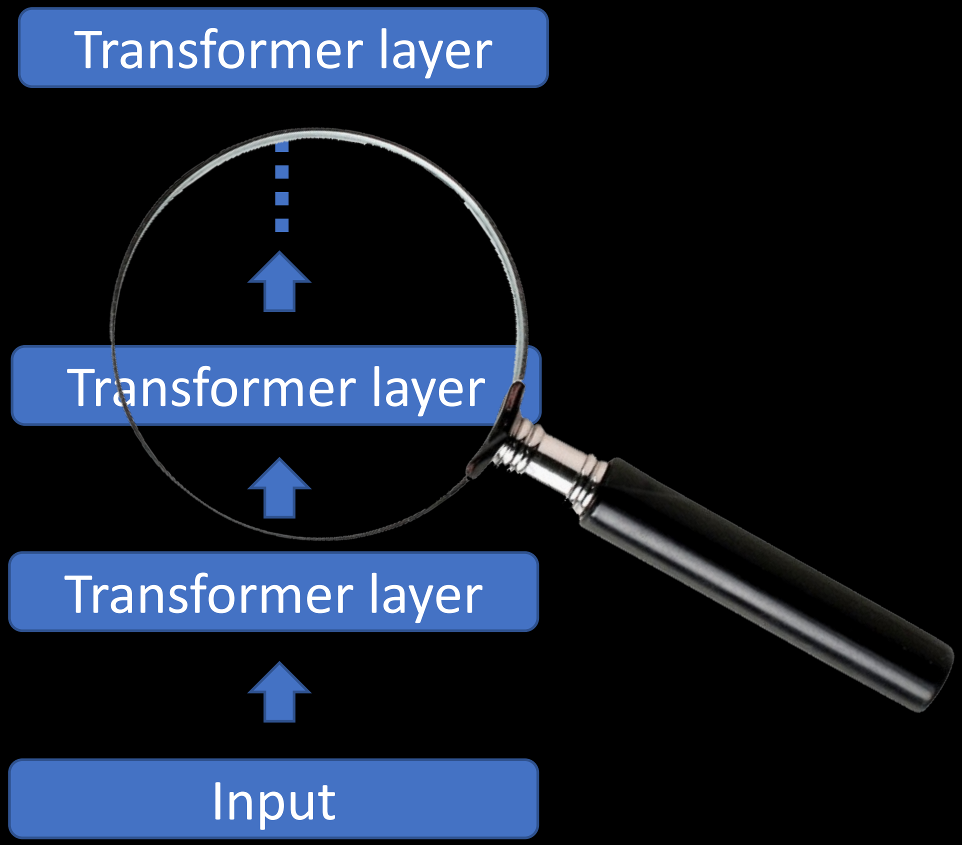
Techniques: “Applied theory”?

How should we understand how Transformers solve a problem?



Techniques: “Applied theory”?

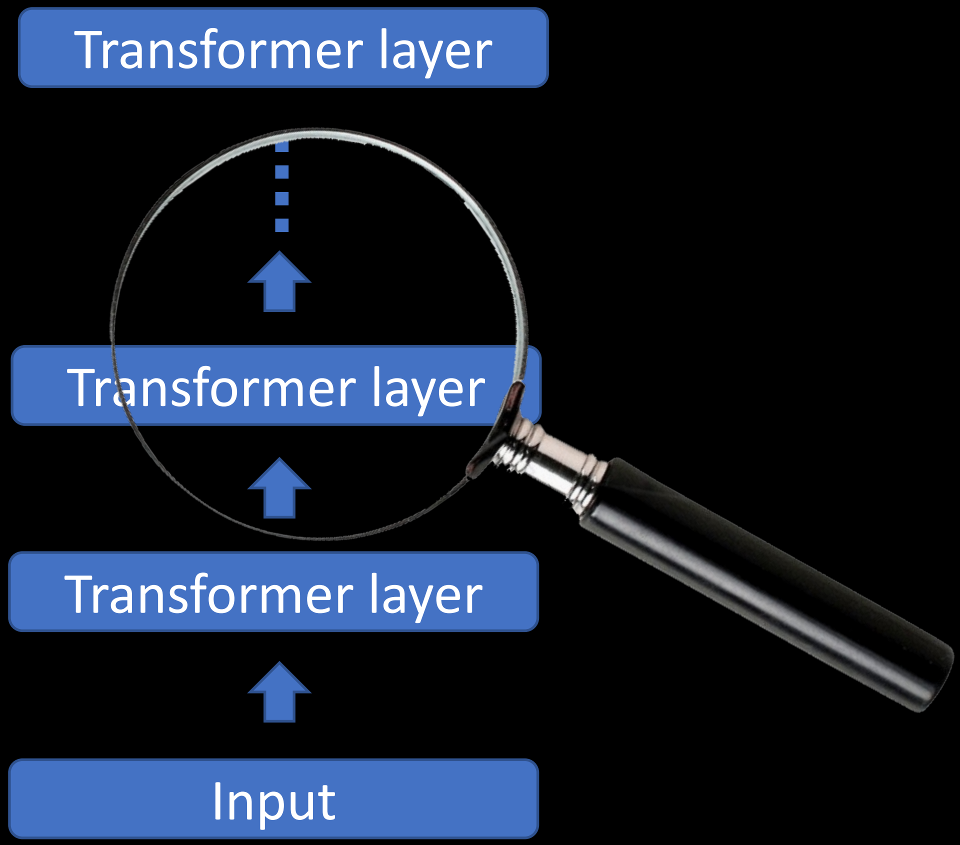
How should we understand how Transformers solve a problem?



Inspect weights to invert mechanism?

Techniques: “Applied theory”?

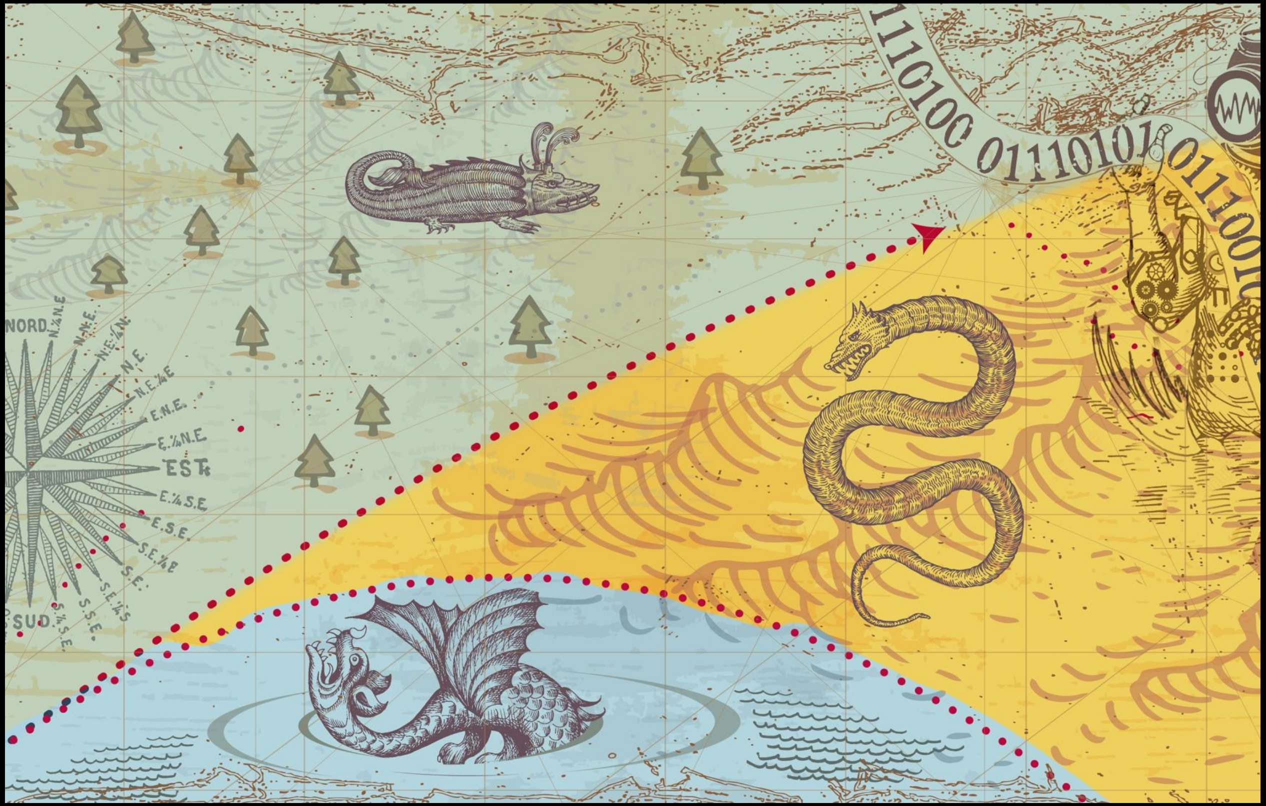
How should we understand how Transformers solve a problem?



Issue: Space of possible solutions can be too large and complex

Techniques: “Applied theory”?

How should we understand how Transformers solve a problem?



One Solution: Using understanding of information and computation can refine search

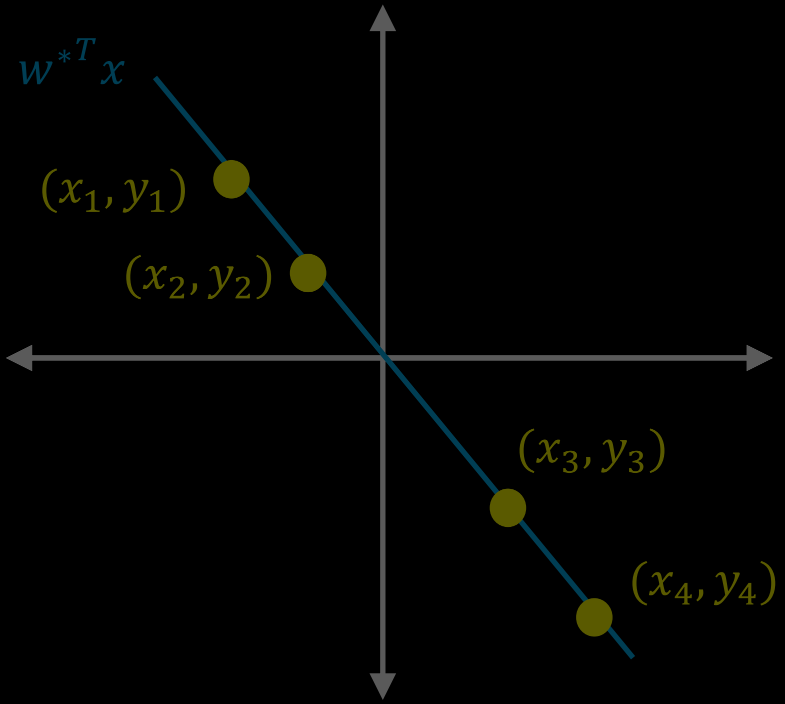
Techniques: “Applied theory”?

For linear regression:

- We know information-theoretic lower bounds on rates achievable by any first-order method
- We understand settings where gap between first and second-order methods is largest

Can we use this understanding, combined with empirical investigations, to uncover Transformer mechanisms?

Setup and algorithms

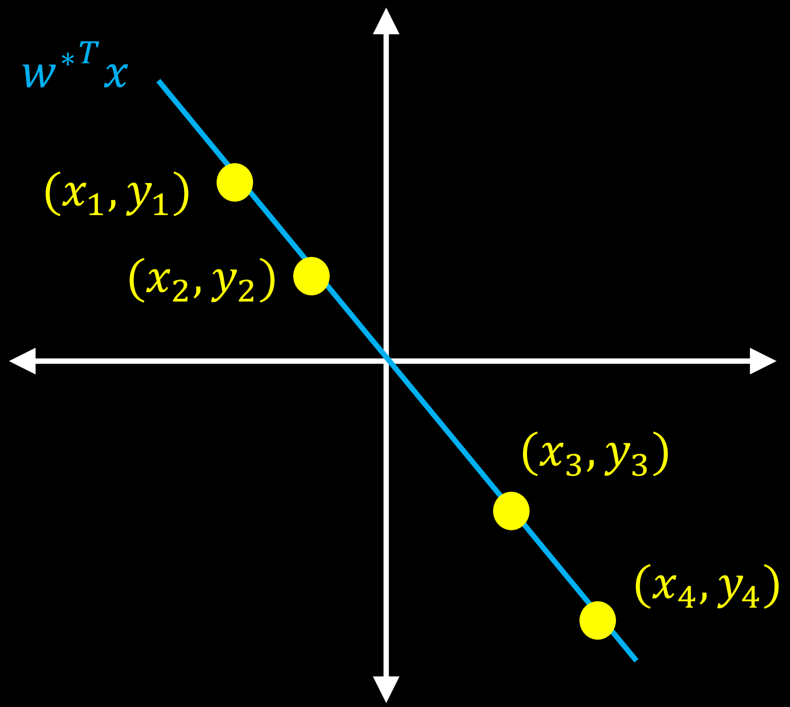


The Setup

Data distribution

For each sequence of n examples $\{x_i, y_i\}_{i=1}^{n}$

Sample $w^* \sim N(0, I)$
Sample data covariance Σ (for now, let $\Sigma = I$)
For each $i \in [n], x_i \sim N(0, \Sigma), y_i = {w^*}^T x_i$



Some algorithms for linear regression

For any time step t, let X be matrix of datapoints, y be vector of labels

Ordinary Least Squares: Minimum norm solution to sum of squares objective

$$w_{OLS} = (X^T X)^{\dagger} X^T y$$

Gradient descent on sum of squares objective:

$$w_{GD}^{(k+1)} = w_{GD}^{(k)} - \eta * (\text{Gradient at } w_{GD}^{(k)})$$

$O(\log(\frac{1}{\epsilon}))$ iterations to find ϵ accurate solution

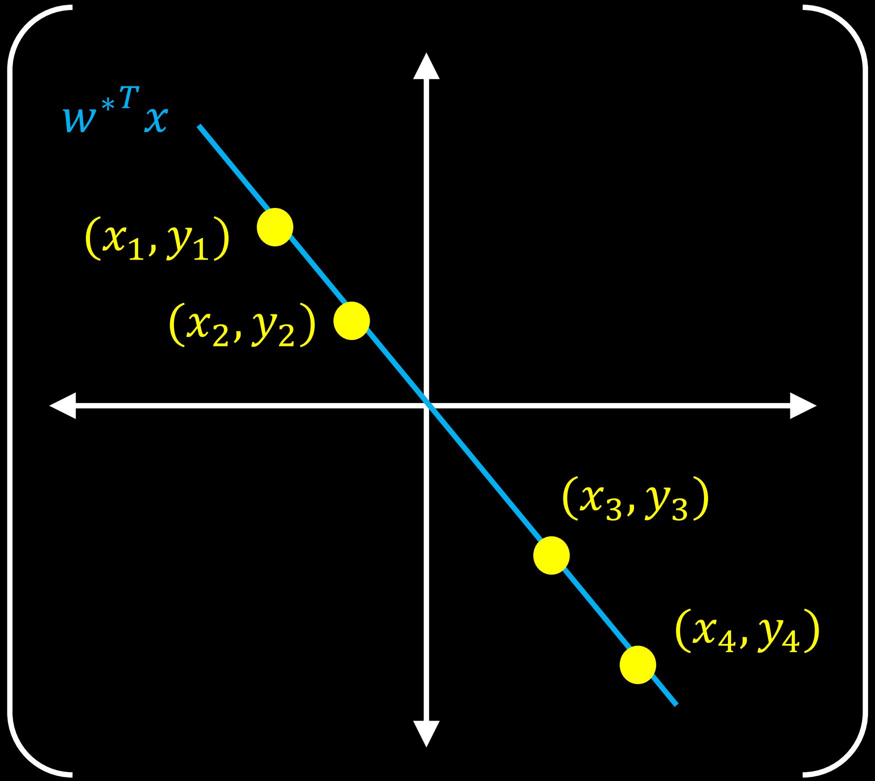
Iterative Newton's: Iterative 2nd order method to find inverse ($\approx$ matrix Taylor series)

Let $S = X^T X$

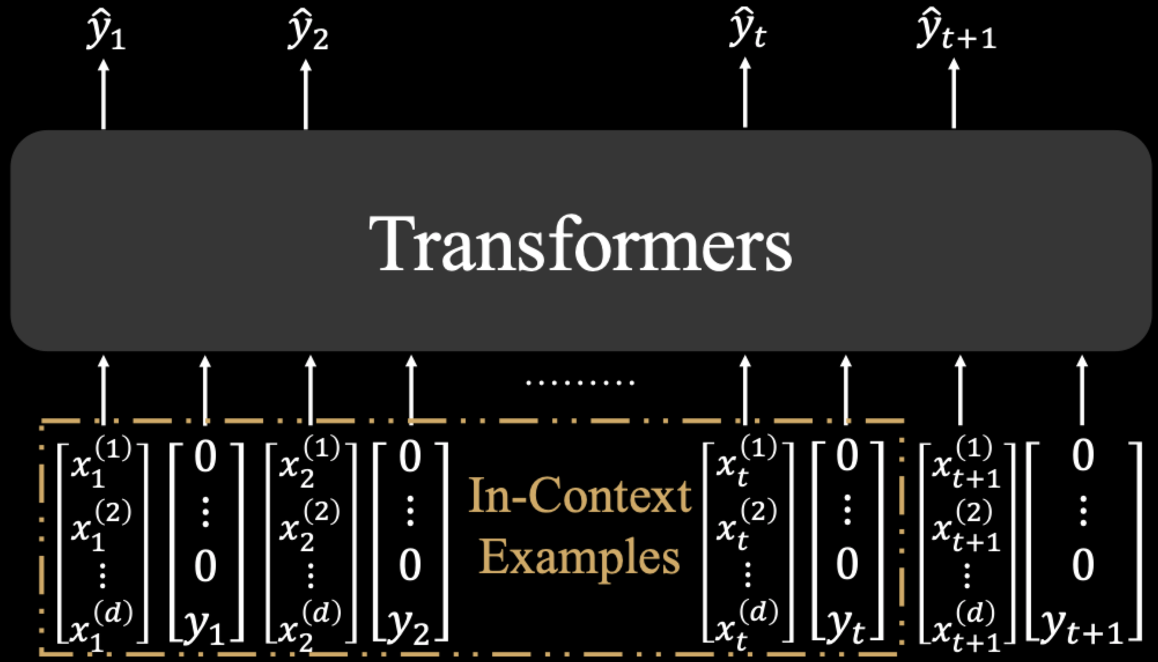
$O(\log\log(\frac{1}{\epsilon}))$ iterations to find ϵ accurate solution

$$M_0 = \alpha S, M_{k+1} = 2M_k - M_k S M_k$$
$$w_{Newton}^{(k)} = M_k X^T y$$

Transformers for linear regression

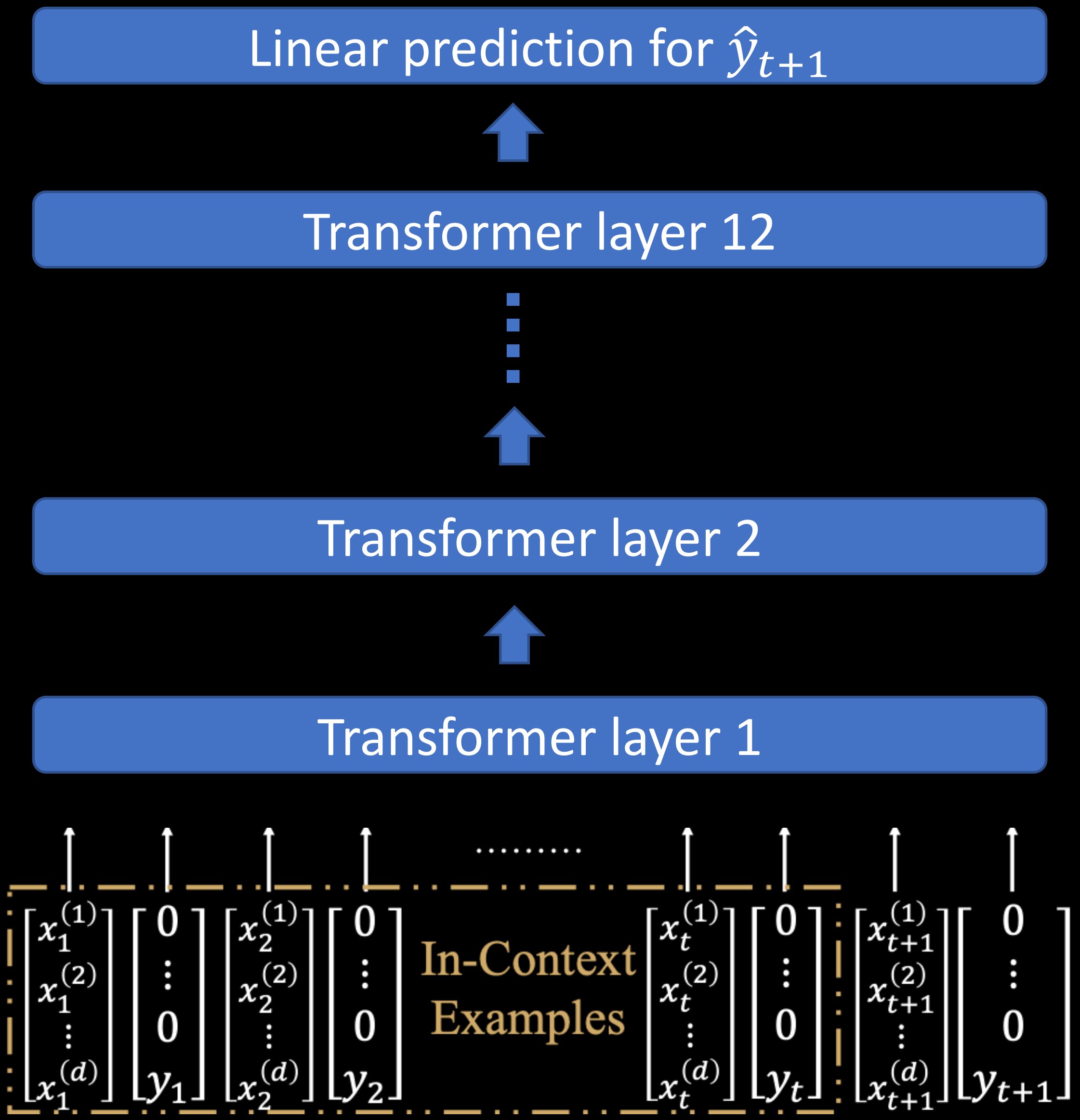


Train on T such instances

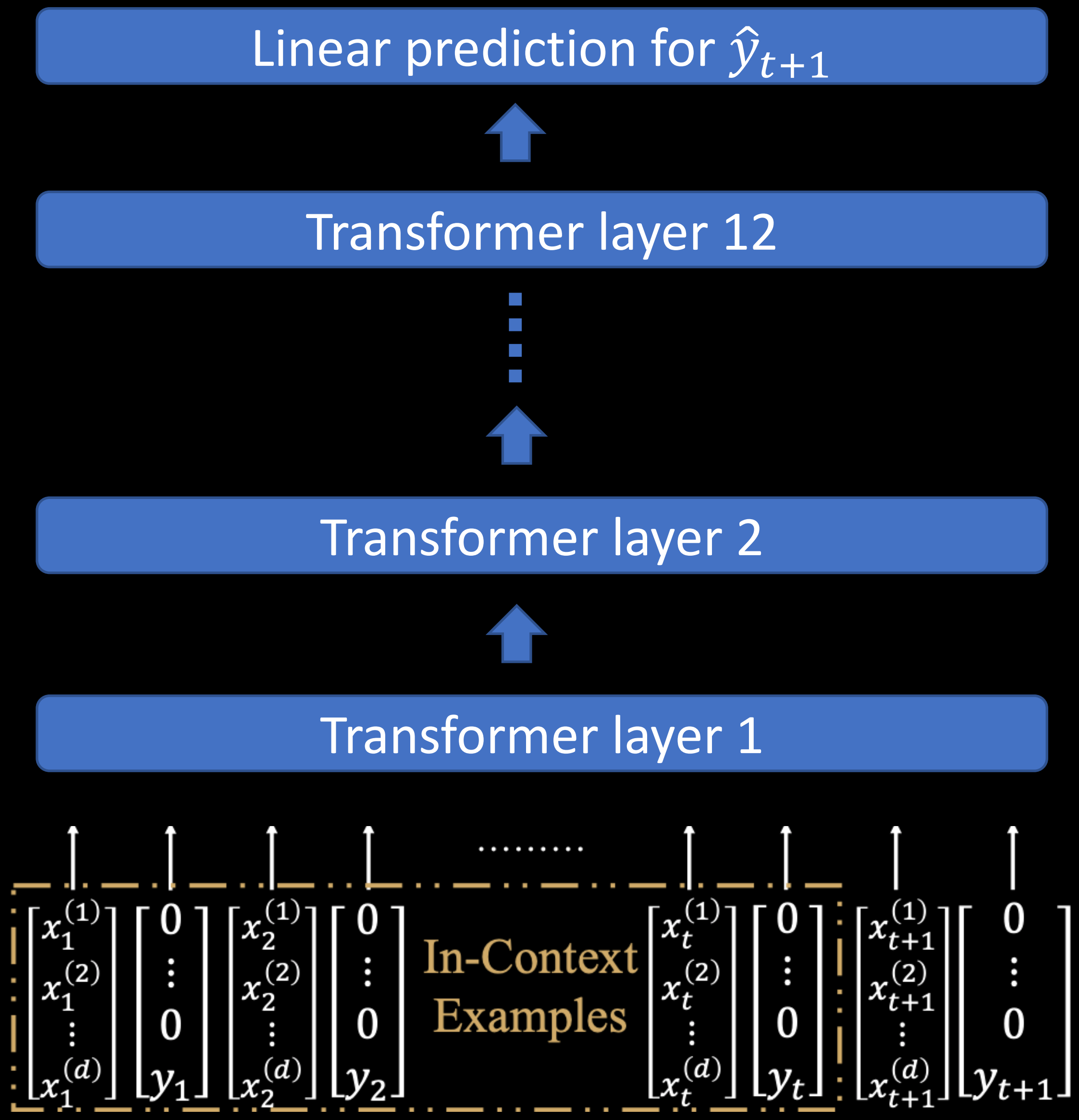


$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_t$
$\hat{y}_{t+1}$
Transformers
$\begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2^{(1)} \\ x_2^{(2)} \\ \vdots \\ x_2^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_2 \end{bmatrix} \cdots \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_t \end{bmatrix} \begin{bmatrix} x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \\ \vdots \\ x_{t+1}^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_{t+1} \end{bmatrix}$
In-Context Examples

Transformers for linear regression


Linear prediction for $\hat{y}_{t+1}$
Transformer layer 12
Transformer layer 2
Transformer layer 1
In-Context Examples
$\begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2^{(1)} \\ x_2^{(2)} \\ \vdots \\ x_2^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_2 \end{bmatrix} \cdots \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_t \end{bmatrix} \begin{bmatrix} x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \\ \vdots \\ x_{t+1}^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_{t+1} \end{bmatrix}$

Transformers as an iterative algorithm: probing layers



Linear prediction for $\hat{y}_{t+1}$

Transformer layer 12

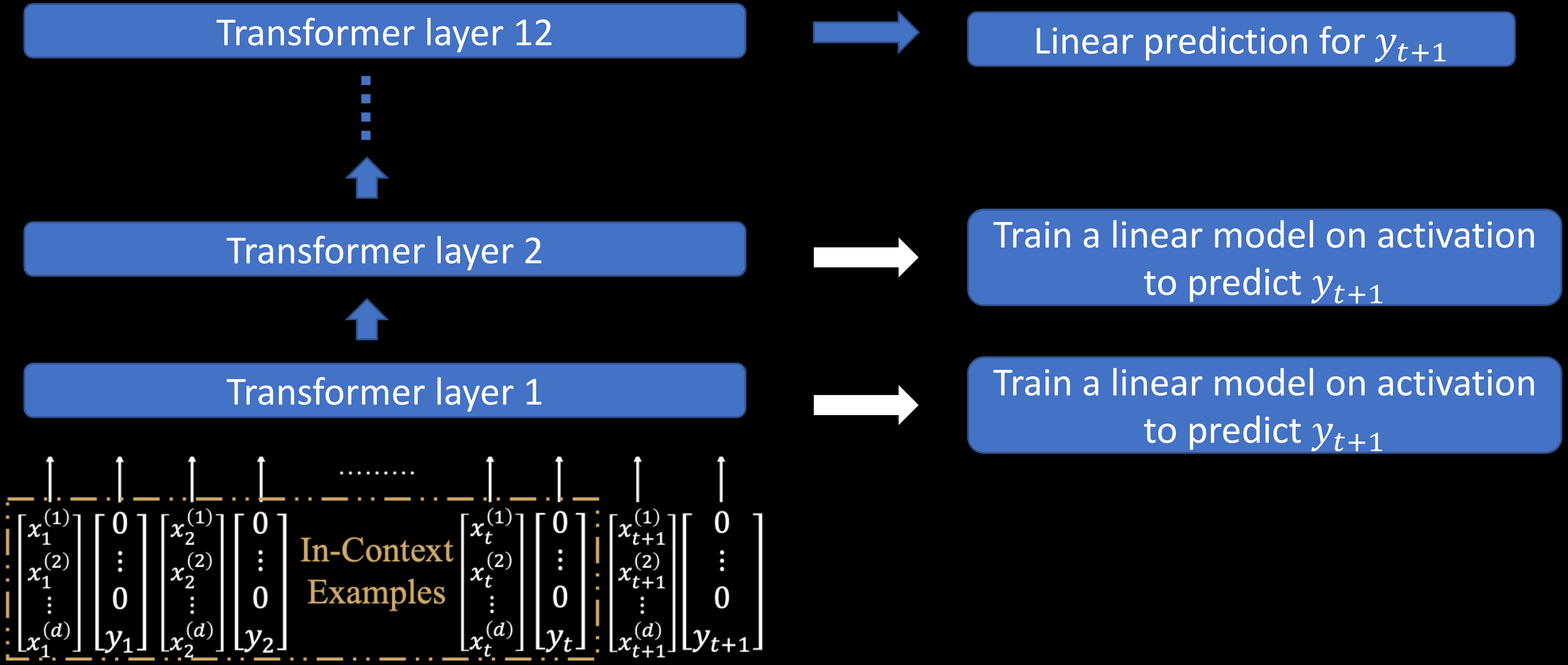
Transformer layer 2

Transformer layer 1

In-Context Examples

$$\begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2^{(1)} \\ x_2^{(2)} \\ \vdots \\ x_2^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_2 \end{bmatrix} \cdots \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_t \end{bmatrix} \begin{bmatrix} x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \\ \vdots \\ x_{t+1}^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_{t+1} \end{bmatrix}$$

Transformers as an iterative algorithm: probing layers



Transformer layer 12

Linear prediction for y_{t+1}

Transformer layer 2

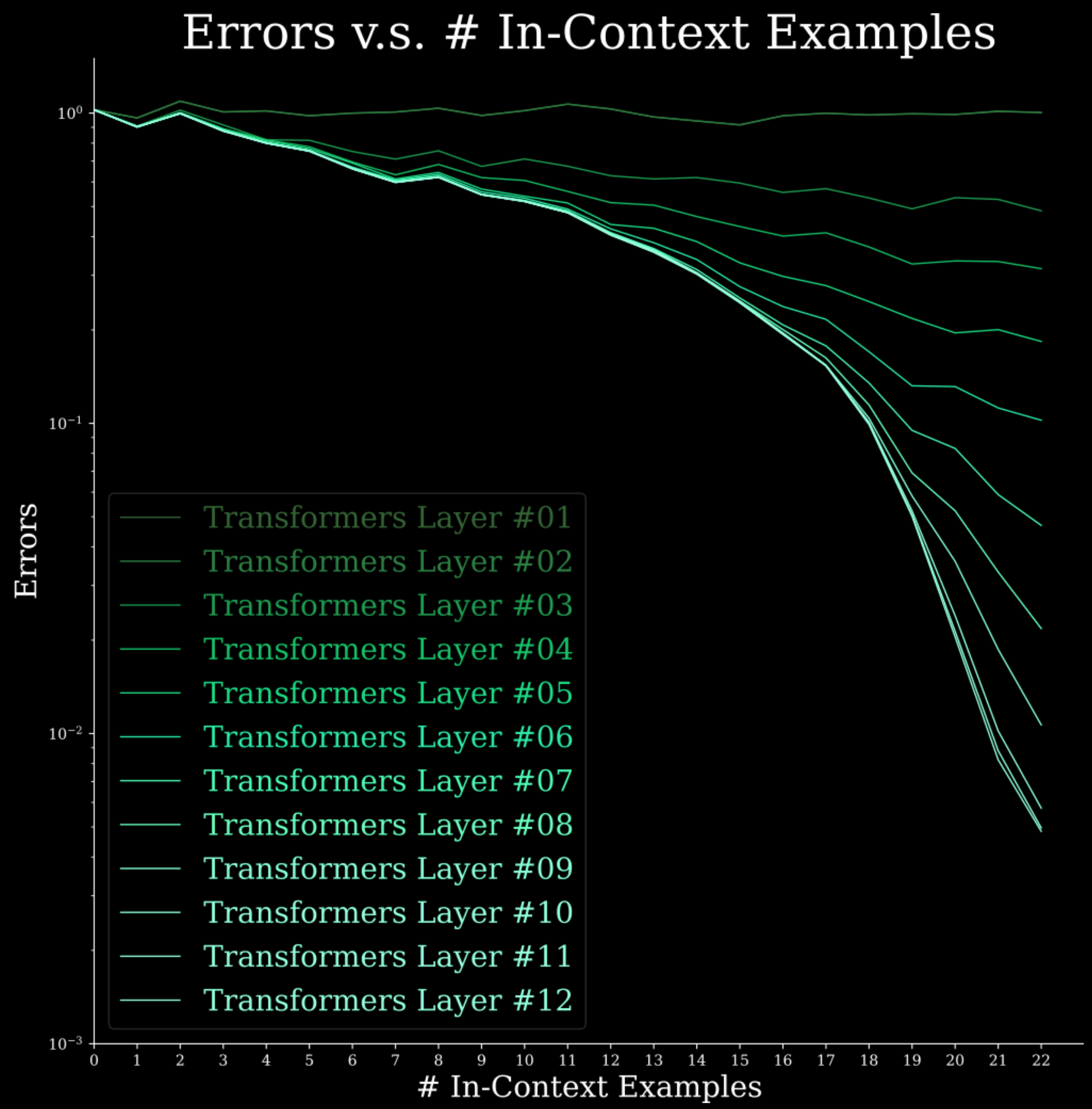
Train a linear model on activation to predict y_{t+1}

Transformer layer 1

Train a linear model on activation to predict y_{t+1}

$$\begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2^{(1)} \\ x_2^{(2)} \\ \vdots \\ x_2^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_2 \end{bmatrix} \cdots \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_t \end{bmatrix} \begin{bmatrix} x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \\ \vdots \\ x_{t+1}^{(d)} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_{t+1} \end{bmatrix}$$

Transformers as an iterative algorithm: probing layers



Metric: Similarity of errors

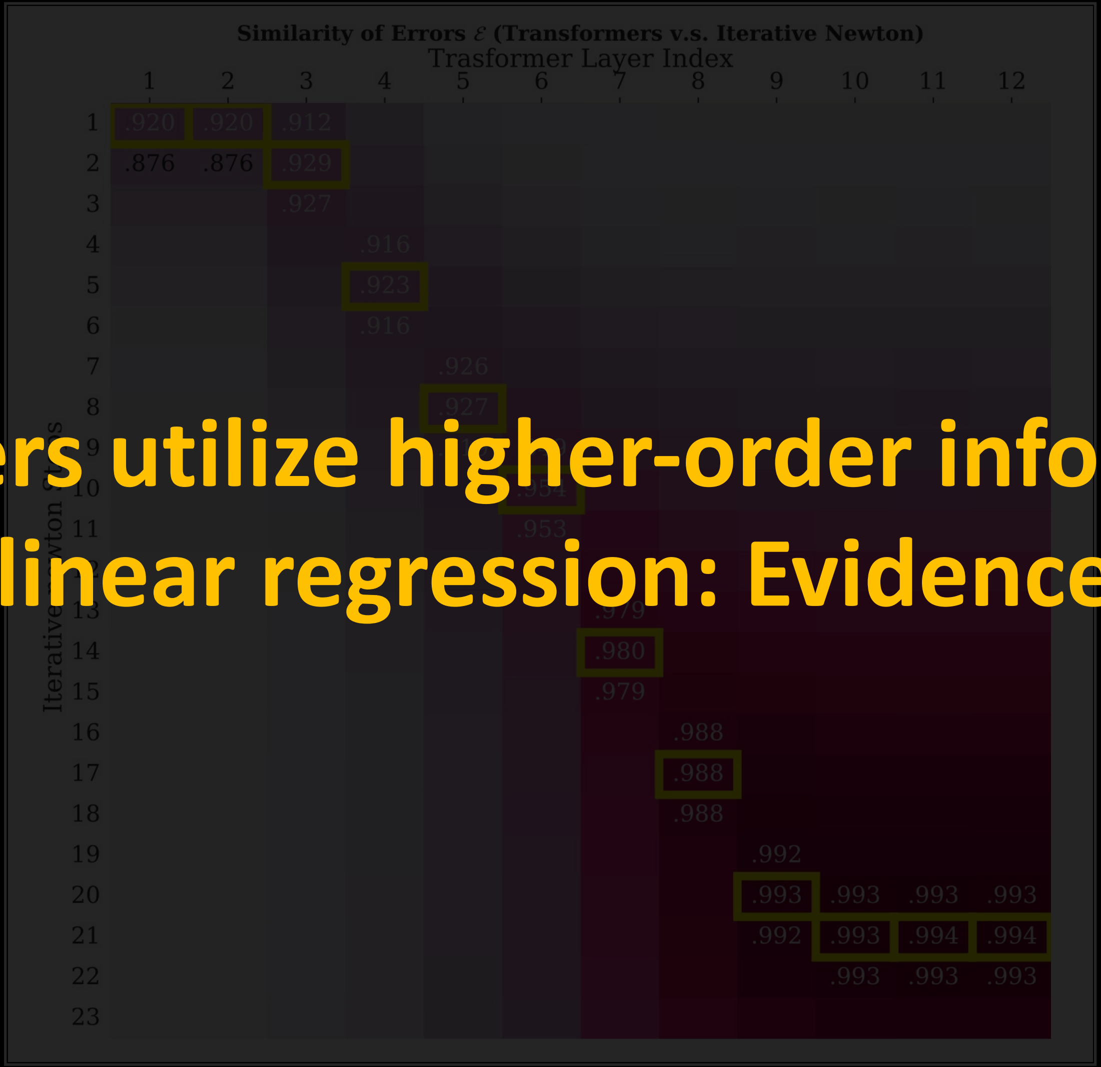
$$x_1,\ y_1,\ x_2,\ y_2,\ x_3,\ y_3,\quad \ldots,\ x_n,\ y_n,$$

	$x_1,\ y_1,\ x_2,\ y_2,\ x_3,\ y_3,\quad \ldots,\ x_n,\ y_n,$
Algorithm A	$y_1^A,\quad y_2^A,\quad y_3^A,\quad \ldots,\ y_n^A,$
Algorithm B	$y_1^B,\quad y_2^B,\quad y_3^B,\quad \ldots,\ y_n^B,$
Algorithm A residuals	$(y_1 - y_1^A),\ (y_2 - y_2^A),\ (y_3 - y_3^A),\ \ldots,\ (y_n - y_n^A),$
Algorithm B residuals	$(y_1 - y_1^B),\ (y_2 - y_2^B,\ \ (y_3 - y_3^B),\ \ldots,\ (y_n - y_n^B),$

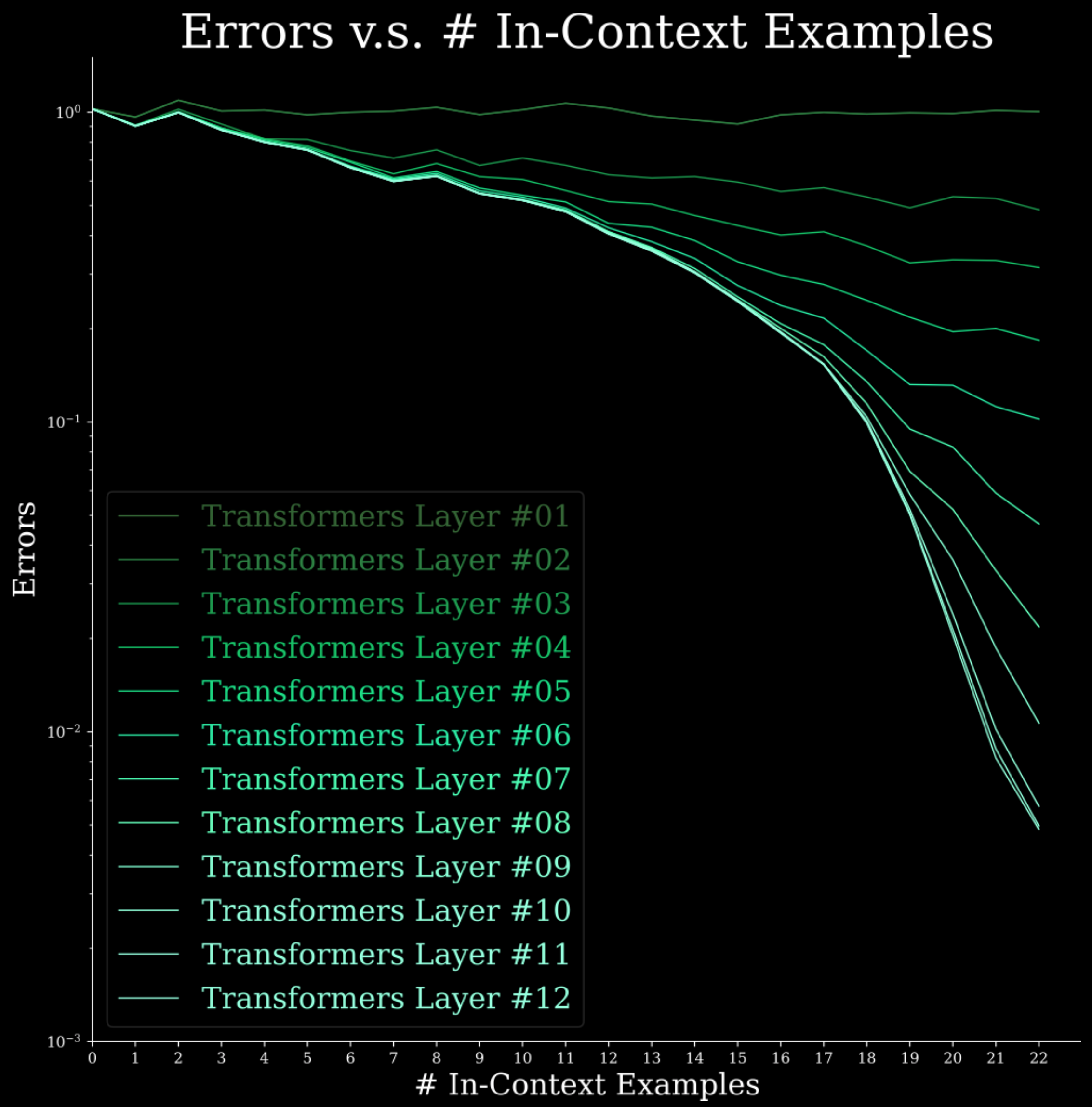
Similarity of errors on $\{x_i, y_i\}_{i=1}^{n}$ between Algorithm A, Algorithm B
= Cosine similarity between residuals of A , B

Overall similarity of errors (Algorithm A, Algorithm B)
= $\mathbb{E}_{\{x_i, y_i\}}$ [Cosine similarity between residuals of A , B]

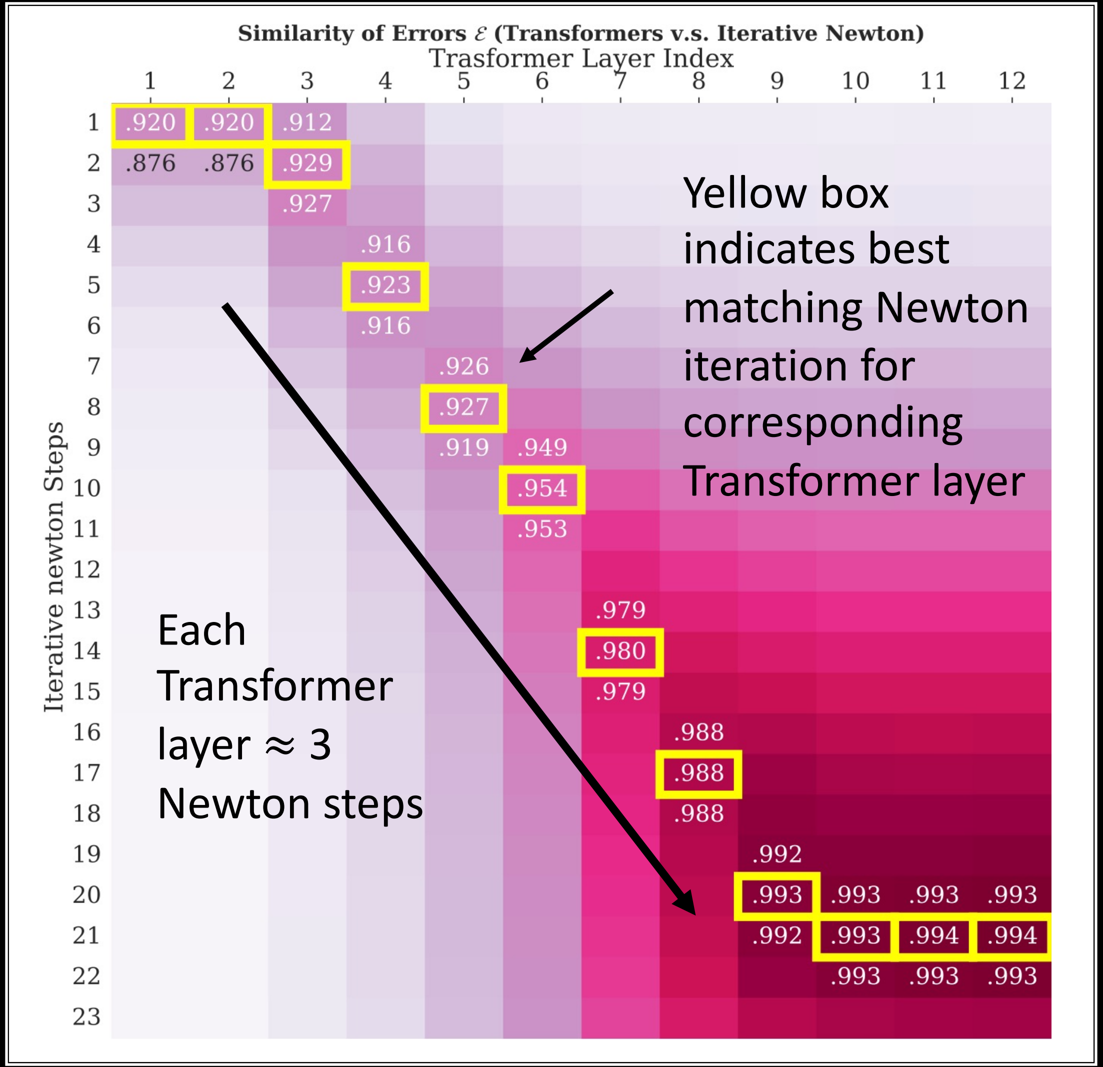
Transformers utilize higher-order information for linear regression: Evidence



Claim 1: Transformers improve across layers



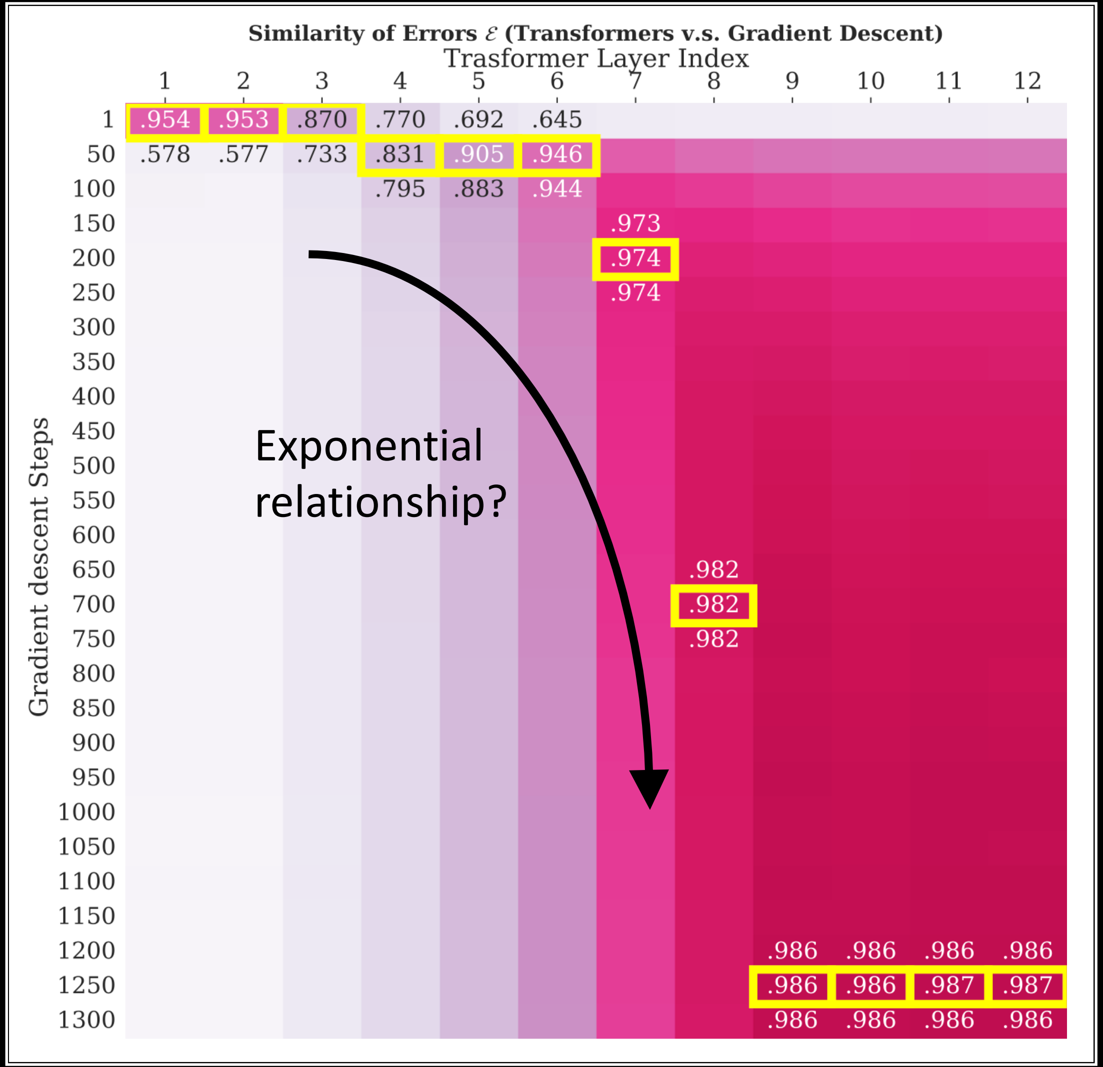
Claim 2: Transformers are more similar to Iterative Newton than to GD



Yellow box indicates best matching Newton iteration for corresponding Transformer layer

Each Transformer layer ≈ 3 Newton steps

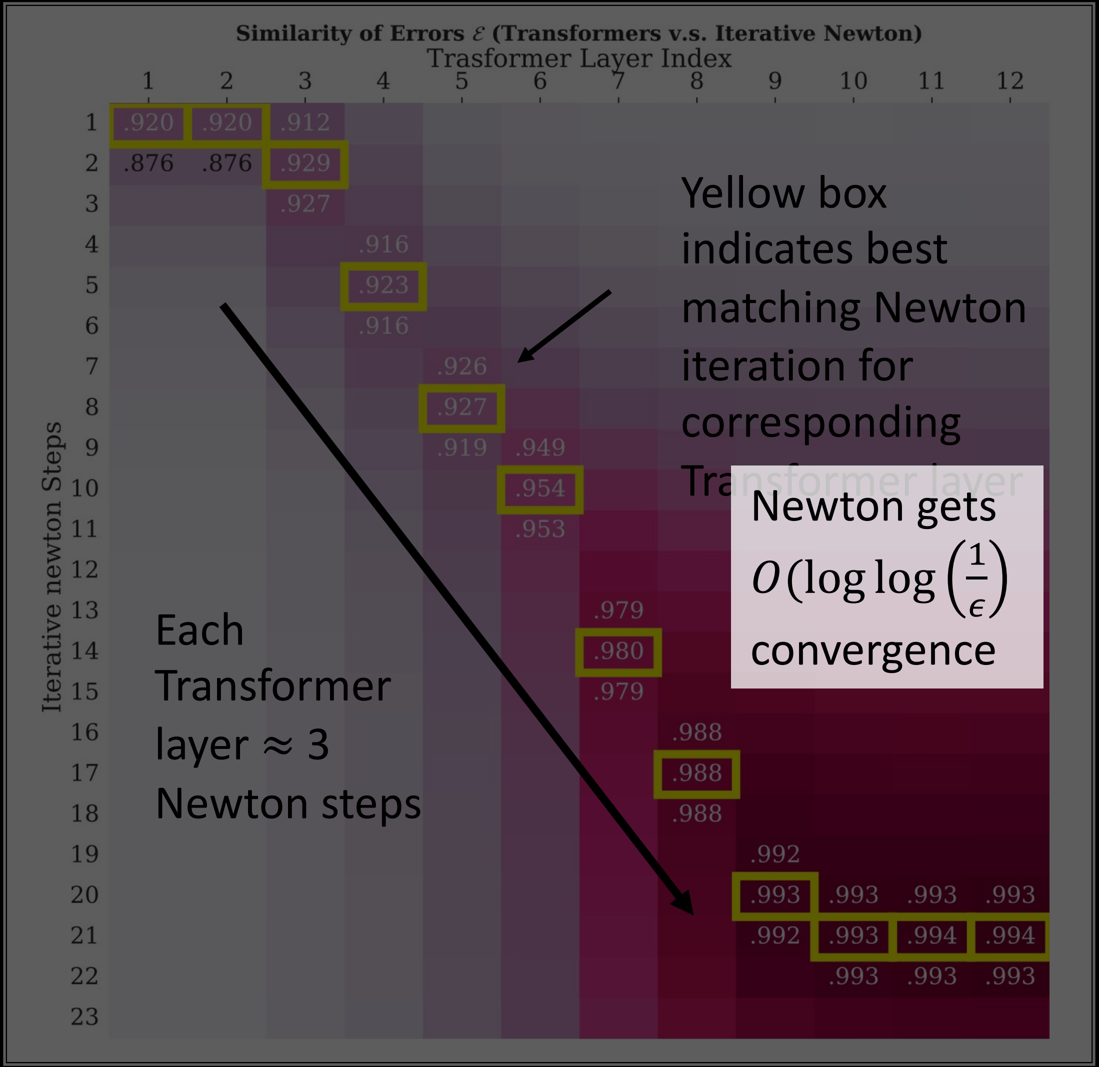
Transformers vs Newton



Exponential relationship?

Transformers vs Gradient Descent

Claim 2: Transformers are more similar to Iterative Newton than to GD

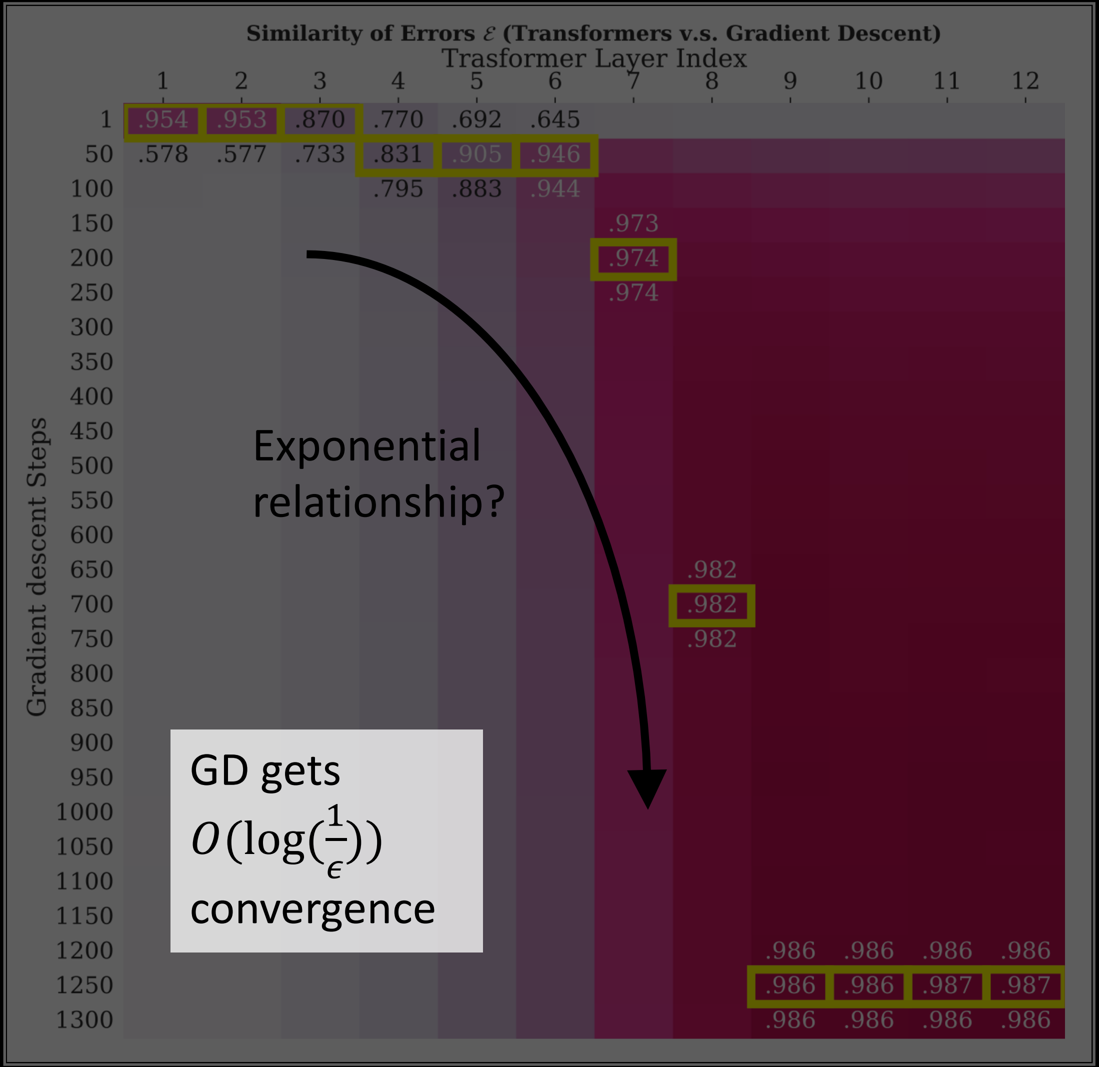


Yellow box indicates best matching Newton iteration for corresponding Transformer layer

Newton gets $O(\log\log\left(\frac{1}{\epsilon}\right)$ convergence

Each Transformer layer ≈ 3 Newton steps

Transformers vs Newton

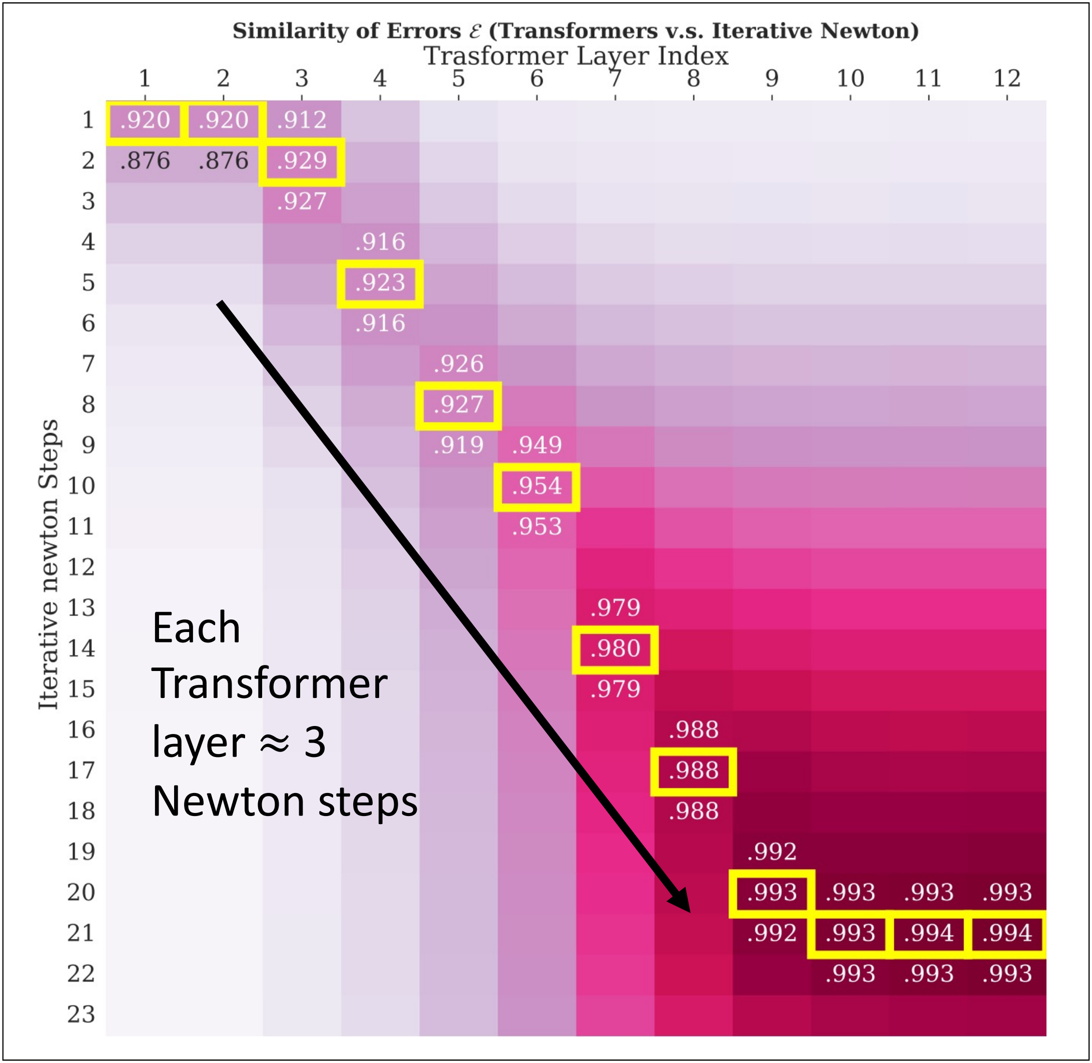


Exponential relationship?

GD gets $O(\log(\frac{1}{\epsilon}))$ convergence

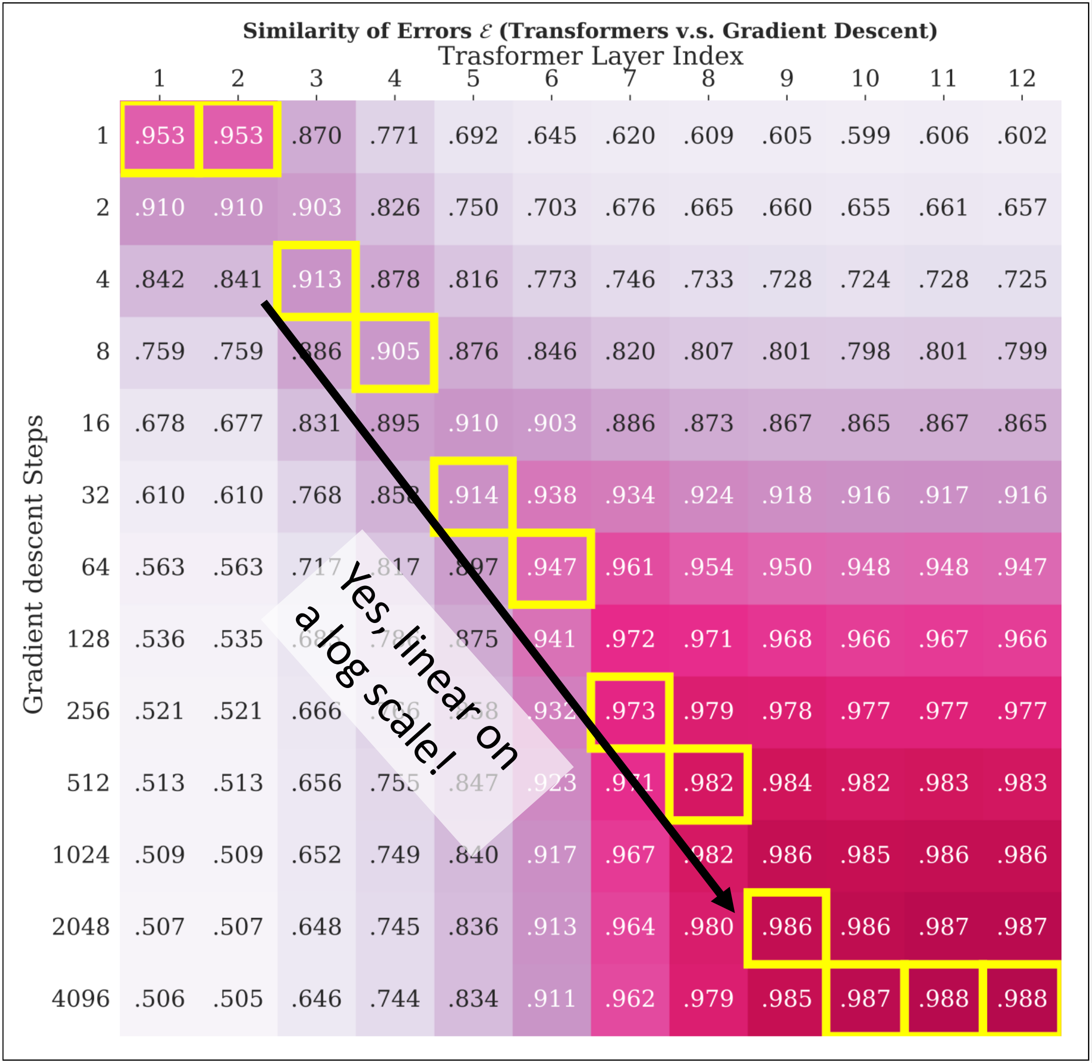
Transformers vs Gradient Descent

Claim 2: Transformers are more similar to Iterative Newton than to GD



Iterative newton Steps	1	2	3	4	5	6	7	8	9	10	11	12
1	.920	.920	.912									
2	.876	.876	.929									
3			.927									
4				.916								
5				.923								
6				.916								
7					.926							
8					.927							
9					.919	.949						
10						.954						
11						.953						
12												
13							.979					
14							.980					
15							.979					
16								.988				
17								.988				
18								.988				
19									.992			
20									.993	.993	.993	.993
21									.992	.993	.994	.994
22										.993	.993	.993
23												

Each Transformer layer ≈ 3 Newton steps



Gradient descent Steps	1	2	3	4	5	6	7	8	9	10	11	12
1	.953	.953	.870	.771	.692	.645	.620	.609	.605	.599	.606	.602
2	.910	.910	.903	.826	.750	.703	.676	.665	.660	.655	.661	.657
4	.842	.841	.913	.878	.816	.773	.746	.733	.728	.724	.728	.725
8	.759	.759	.886	.905	.876	.846	.820	.807	.801	.798	.801	.799
16	.678	.677	.831	.895	.910	.903	.886	.873	.867	.865	.867	.865
32	.610	.610	.768	.858	.914	.938	.934	.924	.918	.916	.917	.916
64	.563	.563	.717	.817	.897	.947	.961	.954	.950	.948	.948	.947
128	.536	.535	.685	.786	.875	.941	.972	.971	.968	.966	.967	.966
256	.521	.521	.666	.766	.858	.932	.973	.979	.978	.977	.977	.977
512	.513	.513	.656	.755	.847	.923	.971	.982	.984	.982	.983	.983
1024	.509	.509	.652	.749	.840	.917	.967	.982	.986	.985	.986	.986
2048	.507	.507	.648	.745	.836	.913	.964	.980	.986	.986	.987	.987
4096	.506	.505	.646	.744	.834	.911	.962	.979	.985	.987	.988	.988

Yes, linear on a log scale!

Transformers vs Newton

Transformers vs Gradient Descent

Claim 3: Transformers are still able to match Newton on harder distributions

What is a setting where the gap between 1st and 2nd order methods is especially large?

On **ill-conditioned instances**, gradient descent (or its variants) get $poly(\kappa)$ dependence on the condition number of the linear system κ, 2nd order methods get $polylog(\kappa)$ dependence.

Claim 3: Transformers are still able to match Newton on harder distributions

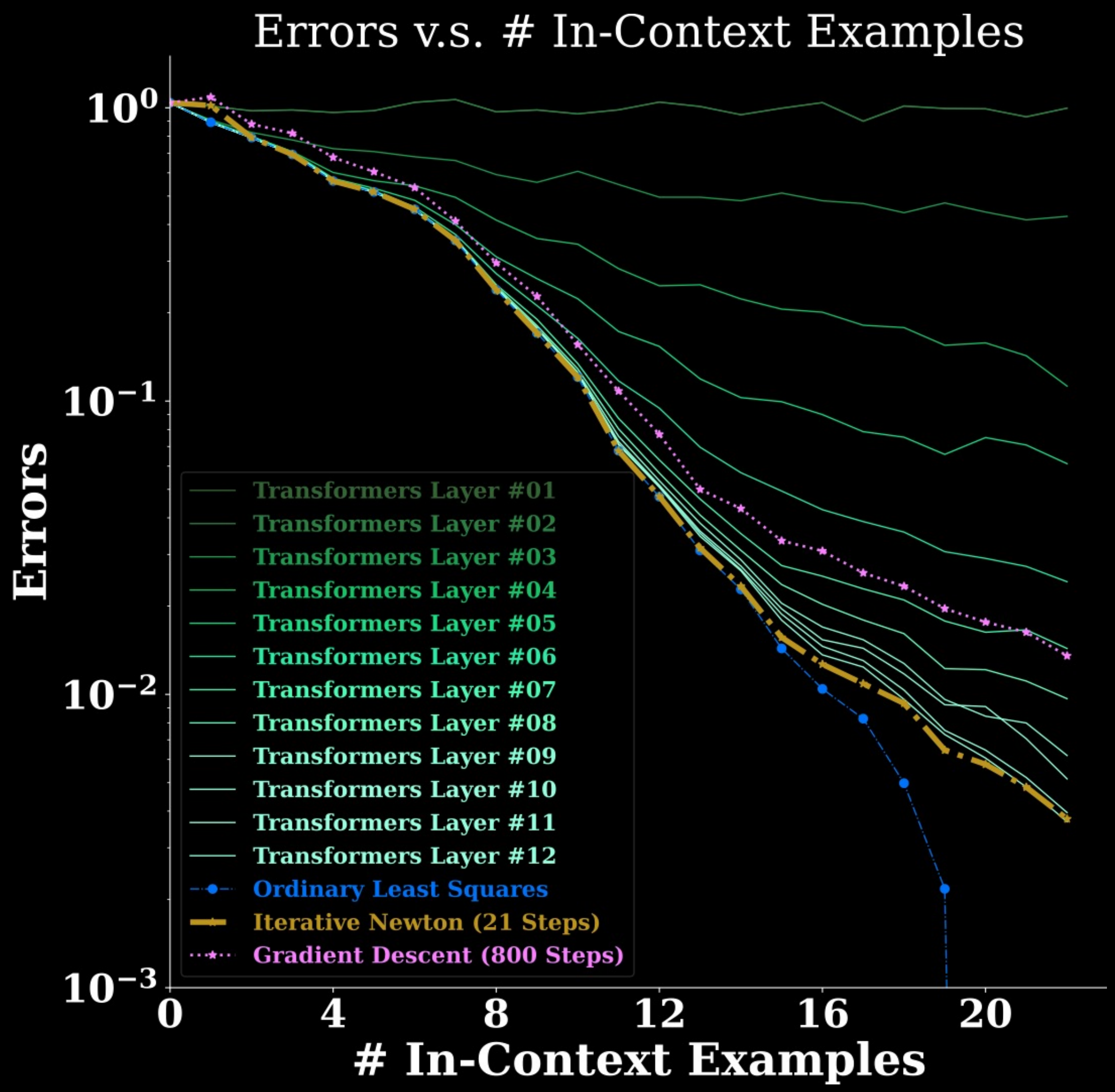
What is a setting where the gap between 1st and 2nd order methods is especially large?

On **ill-conditioned instances**, gradient descent (or its variants) get $poly(\kappa)$ dependence on the condition number of the linear system κ, 2nd order methods get $polylog(\kappa)$ dependence.

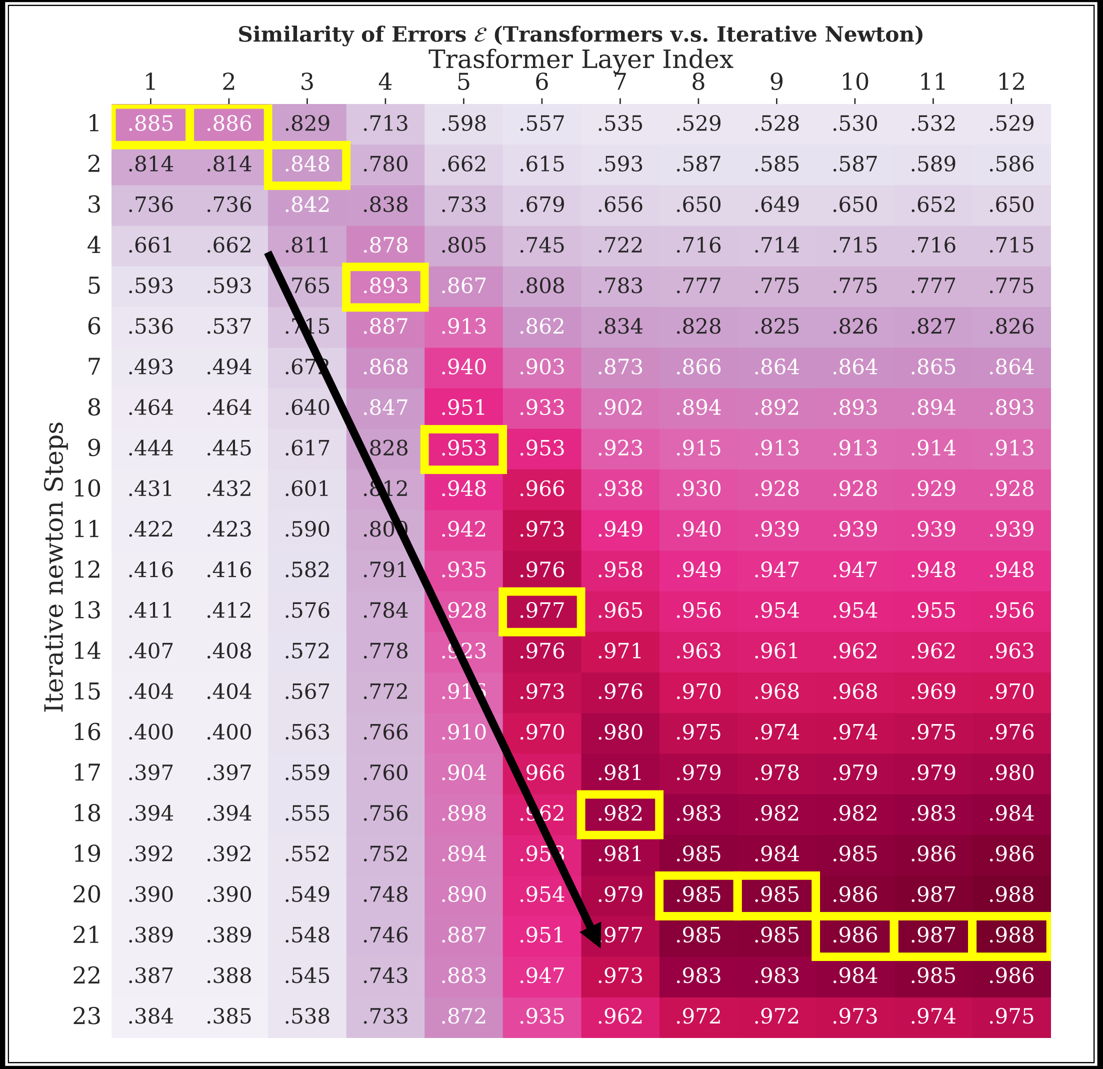
Conjecture (Sharan-Sidford-Valiant'19): No first-order (linear memory method) can avoid a $poly(\kappa)$ dependence on κ in general.

Hard distribution: Sample Σ with $d/2$ eigenvalues at 100, $d/2$ eigenvalues at 1, uniformly random eigenbasis.

Claim 3: Transformers are still able to match Newton on ill-conditioned data



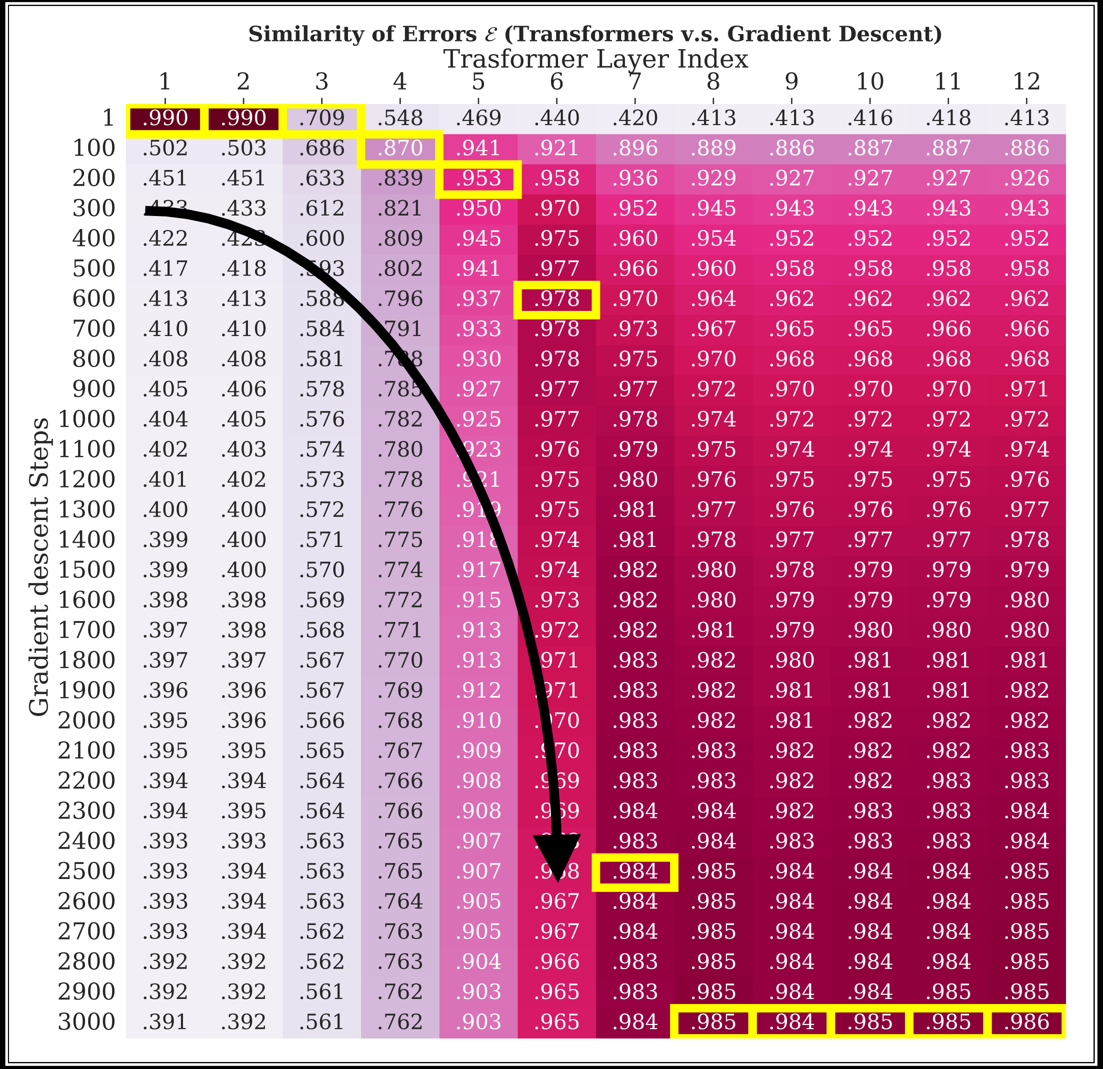
Claim 3: Transformers are still able to match Newton on ill-conditioned data



Similarity of Errors $\mathcal{E}$ (Transformers v.s. Iterative Newton)

Iterative newton Steps \ Trasformer Layer Index	1	2	3	4	5	6	7	8	9	10	11	12
1	.885	.886	.829	.713	.598	.557	.535	.529	.528	.530	.532	.529
2	.814	.814	.848	.780	.662	.615	.593	.587	.585	.587	.589	.586
3	.736	.736	.842	.838	.733	.679	.656	.650	.649	.650	.652	.650
4	.661	.662	.811	.878	.805	.745	.722	.716	.714	.715	.716	.715
5	.593	.593	.765	.893	.867	.808	.783	.777	.775	.775	.777	.775
6	.536	.537	.715	.887	.913	.862	.834	.828	.825	.826	.827	.826
7	.493	.494	.672	.868	.940	.903	.873	.866	.864	.864	.865	.864
8	.464	.464	.640	.847	.951	.933	.902	.894	.892	.893	.894	.893
9	.444	.445	.617	.828	.953	.953	.923	.915	.913	.913	.914	.913
10	.431	.432	.601	.812	.948	.966	.938	.930	.928	.928	.929	.928
11	.422	.423	.590	.809	.942	.973	.949	.940	.939	.939	.939	.939
12	.416	.416	.582	.791	.935	.976	.958	.949	.947	.947	.948	.948
13	.411	.412	.576	.784	.928	.977	.965	.956	.954	.954	.955	.956
14	.407	.408	.572	.778	.923	.976	.971	.963	.961	.962	.962	.963
15	.404	.404	.567	.772	.916	.973	.976	.970	.968	.968	.969	.970
16	.400	.400	.563	.766	.910	.970	.980	.975	.974	.974	.975	.976
17	.397	.397	.559	.760	.904	.966	.981	.979	.978	.979	.979	.980
18	.394	.394	.555	.756	.898	.962	.982	.983	.982	.982	.983	.984
19	.392	.392	.552	.752	.894	.953	.981	.985	.984	.985	.986	.986
20	.390	.390	.549	.748	.890	.954	.979	.985	.985	.986	.987	.988
21	.389	.389	.548	.746	.887	.951	.977	.985	.985	.986	.987	.988
22	.387	.388	.545	.743	.883	.947	.973	.983	.983	.984	.985	.986
23	.384	.385	.538	.733	.872	.935	.962	.972	.972	.973	.974	.975

Transformers vs Newton



Similarity of Errors $\mathcal{E}$ (Transformers v.s. Gradient Descent)

Gradient descent Steps \ Trasformer Layer Index	1	2	3	4	5	6	7	8	9	10	11	12
1	.990	.990	.709	.548	.469	.440	.420	.413	.413	.416	.418	.413
100	.502	.503	.686	.870	.941	.921	.896	.889	.886	.887	.887	.886
200	.451	.451	.633	.839	.953	.958	.936	.929	.927	.927	.927	.926
300	.433	.433	.612	.821	.950	.970	.952	.945	.943	.943	.943	.943
400	.422	.423	.600	.809	.945	.975	.960	.954	.952	.952	.952	.952
500	.417	.418	.593	.802	.941	.977	.966	.960	.958	.958	.958	.958
600	.413	.413	.588	.796	.937	.978	.970	.964	.962	.962	.962	.962
700	.410	.410	.584	.791	.933	.978	.973	.967	.965	.965	.966	.966
800	.408	.408	.581	.788	.930	.978	.975	.970	.968	.968	.968	.968
900	.405	.406	.578	.785	.927	.977	.977	.972	.970	.970	.970	.971
1000	.404	.405	.576	.782	.925	.977	.978	.974	.972	.972	.972	.972
1100	.402	.403	.574	.780	.923	.976	.979	.975	.974	.974	.974	.974
1200	.401	.402	.573	.778	.921	.975	.980	.976	.975	.975	.975	.976
1300	.400	.400	.572	.776	.919	.975	.981	.977	.976	.976	.976	.977
1400	.399	.400	.571	.775	.918	.974	.981	.978	.977	.977	.977	.978
1500	.399	.400	.570	.774	.917	.974	.982	.980	.978	.979	.979	.979
1600	.398	.398	.569	.772	.915	.973	.982	.980	.979	.979	.979	.980
1700	.397	.398	.568	.771	.913	.972	.982	.981	.979	.980	.980	.980
1800	.397	.397	.567	.770	.913	.971	.983	.982	.980	.981	.981	.981
1900	.396	.396	.567	.769	.912	.971	.983	.982	.981	.981	.981	.982
2000	.395	.396	.566	.768	.910	.970	.983	.982	.981	.982	.982	.982
2100	.395	.395	.565	.767	.909	.970	.983	.983	.982	.982	.982	.983
2200	.394	.394	.564	.766	.908	.969	.983	.983	.982	.982	.983	.983
2300	.394	.395	.564	.766	.908	.969	.984	.984	.982	.983	.983	.984
2400	.393	.393	.563	.765	.907	.968	.983	.984	.983	.983	.983	.984
2500	.393	.394	.563	.765	.907	.968	.984	.985	.984	.984	.984	.985
2600	.393	.394	.563	.764	.905	.967	.984	.985	.984	.984	.984	.985
2700	.393	.394	.562	.763	.905	.967	.984	.985	.984	.984	.984	.985
2800	.392	.392	.562	.763	.904	.966	.983	.985	.984	.984	.984	.985
2900	.392	.392	.561	.762	.903	.965	.983	.985	.984	.984	.985	.985
3000	.391	.392	.561	.762	.903	.965	.984	.985	.984	.985	.985	.986

Transformers vs Gradient Descent

Theoretical justification

Can Transformers efficiently implement Iterative Newton's?

Informal Theorem:
Transformers can match predictions of k steps of Iterative Newton's with $(k+8)$ layers, $O(d)$ hidden units per layer.

Construction uses ideas from Akyurek-Schuurmans-Andreas-Ma-Zhou'2022,
and is similar to a matrix inverse construction by Giannou-Rajput-Sohn-Lee-Lee-Papailiopoulos'2023

Some more related work

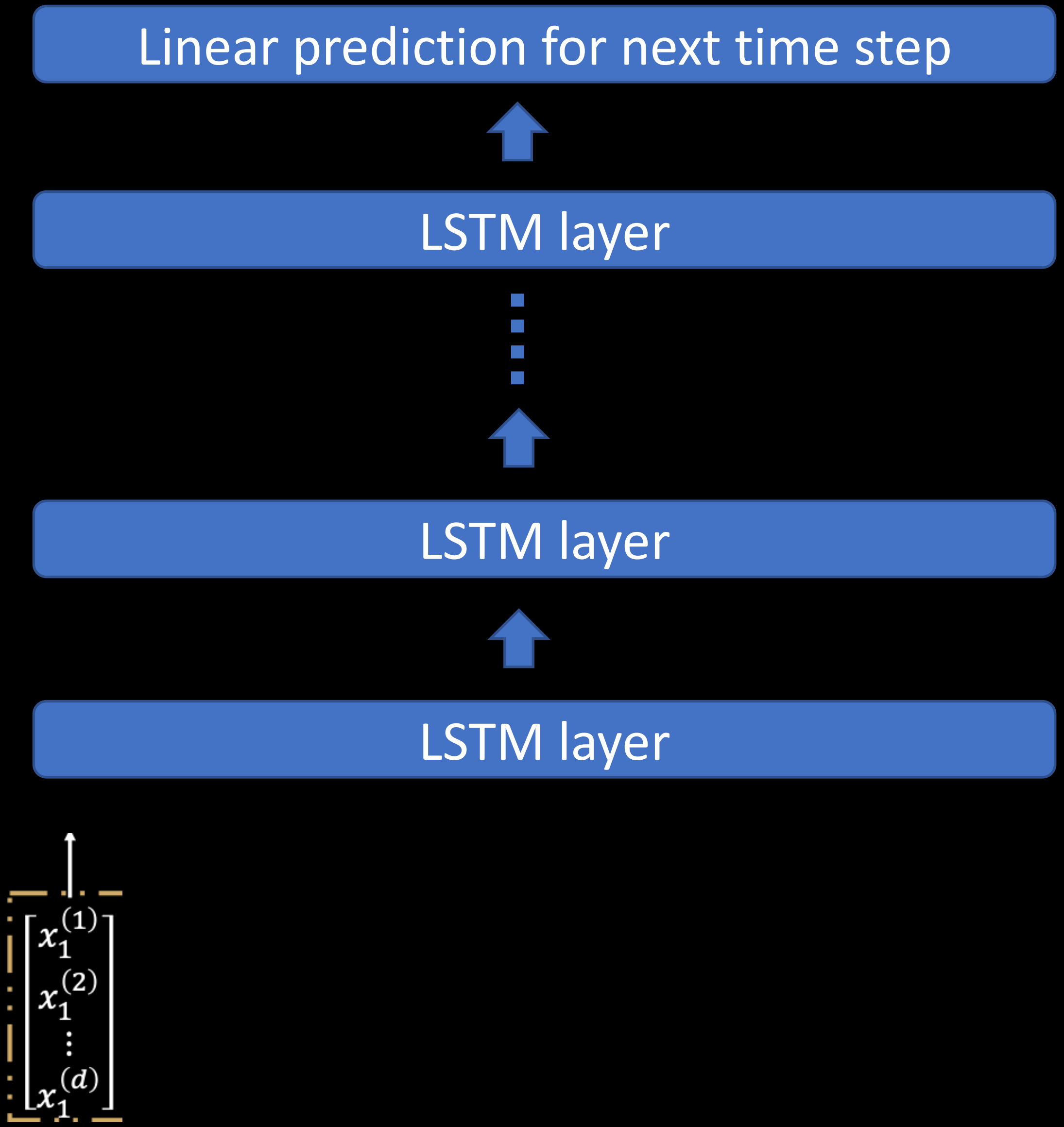
Ahn-Cheng-Daneshmand-Sra’2023, Zhang-Frei-Bartlett’2023 & Mahankali-Hashimoto-Ma’2024 analyze dynamics of trained one-layer Transformers

Vladymyrov-von Oswald-Sandler-Ge’2024 show that a second-order variant of GD can mimic Iterative Newton by implicitly approximating inverse

Giannou-Yang-Wang-Papailiopoulos-Lee’2024 show that Transformers can do Iterative Newton beyond linear regression

What makes Transformers suitable for utilizing 2nd order information?

LSTMs for linear regression



Linear prediction for next time step

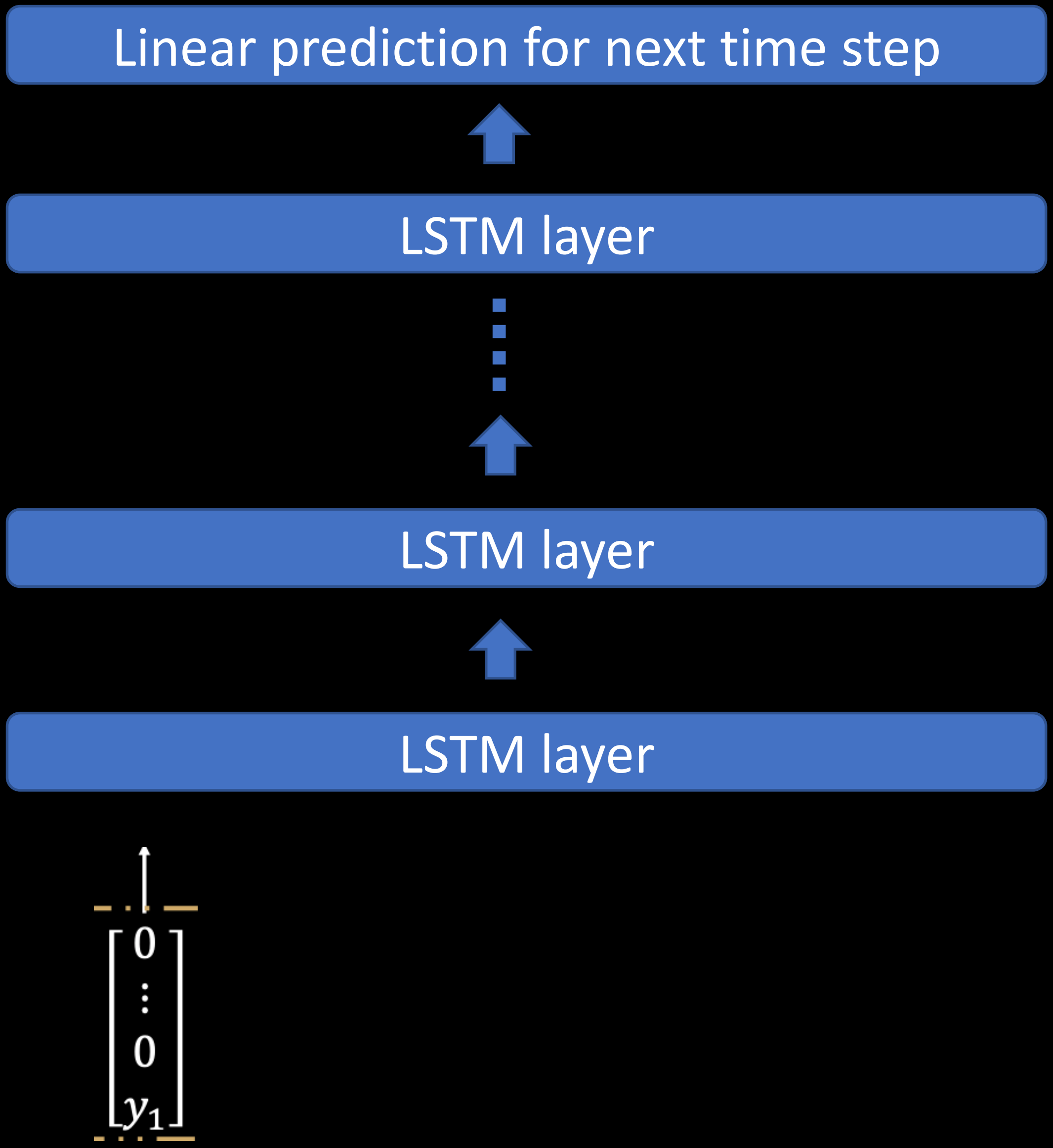
LSTM layer

LSTM layer

LSTM layer

$$\begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(d)} \end{bmatrix}$$

LSTMs for linear regression



Linear prediction for next time step

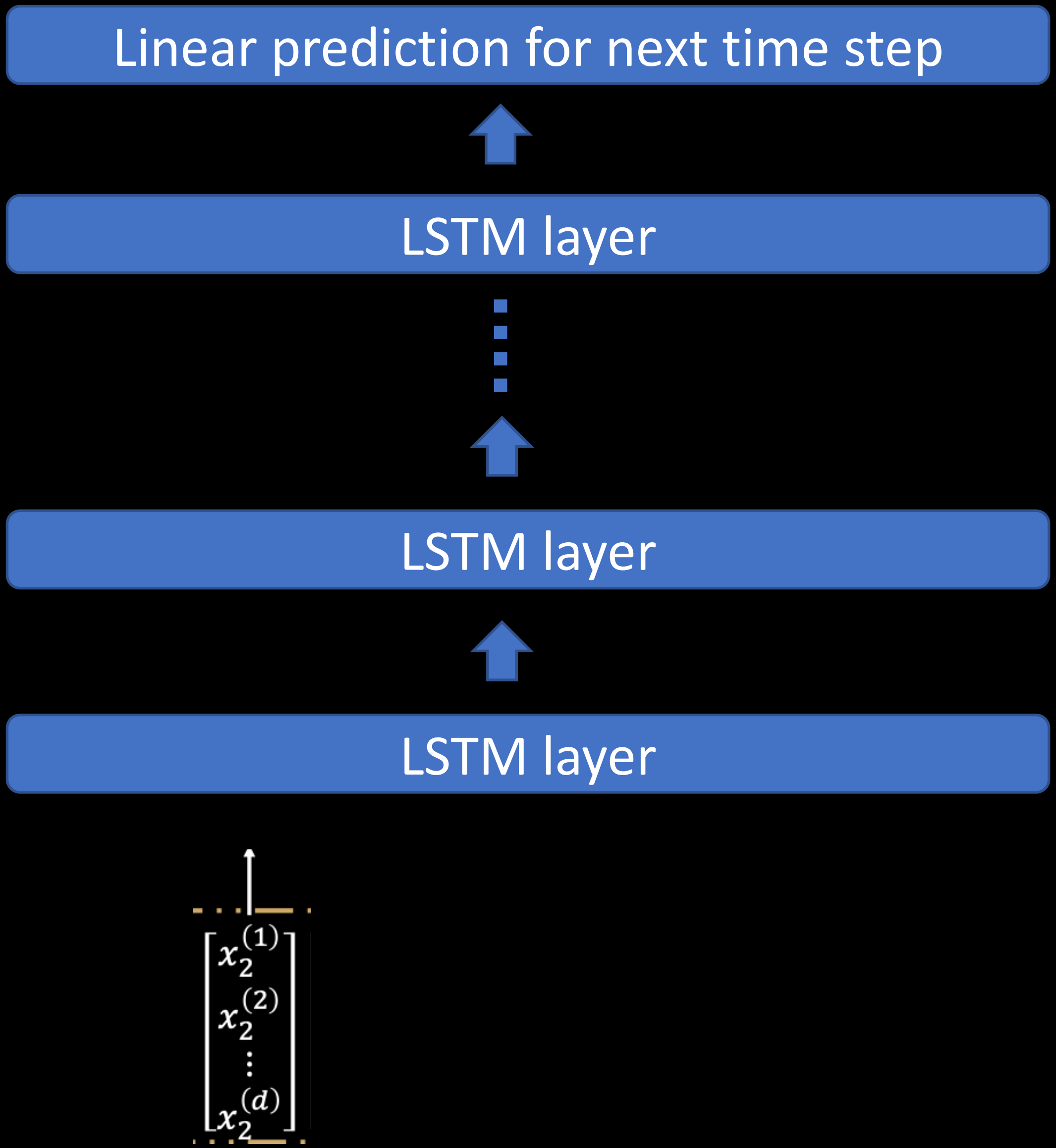
LSTM layer

LSTM layer

LSTM layer

$$\begin{bmatrix}0\\ \vdots\\ 0\\ y_1\end{bmatrix}$$

LSTMs for linear regression



Linear prediction for next time step

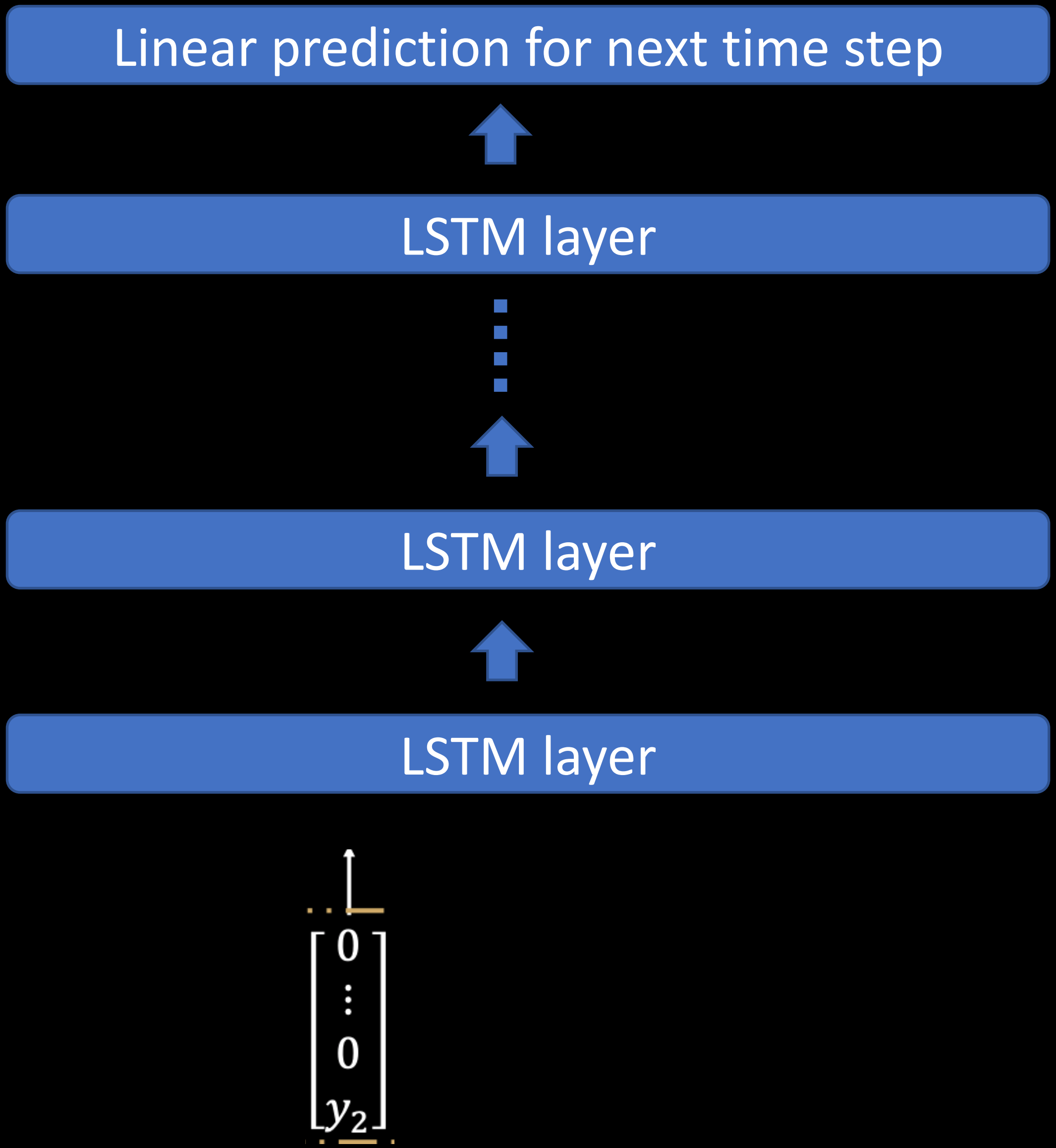
LSTM layer

LSTM layer

LSTM layer

$$\begin{bmatrix} x_2^{(1)} \\ x_2^{(2)} \\ \vdots \\ x_2^{(d)} \end{bmatrix}$$

LSTMs for linear regression



Linear prediction for next time step

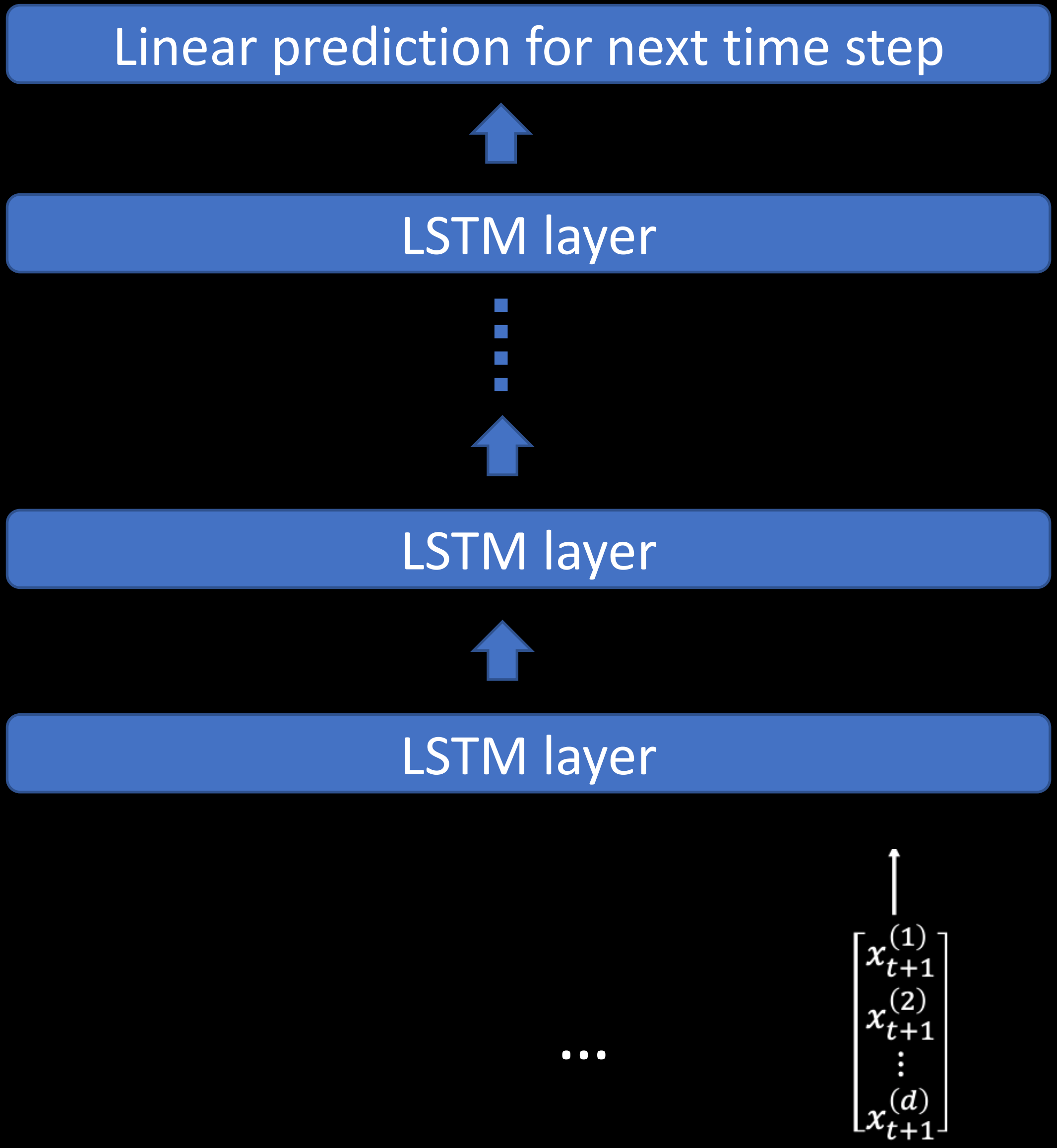
LSTM layer

LSTM layer

LSTM layer

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_2 \end{bmatrix}$$

LSTMs for linear regression



Linear prediction for next time step

LSTM layer

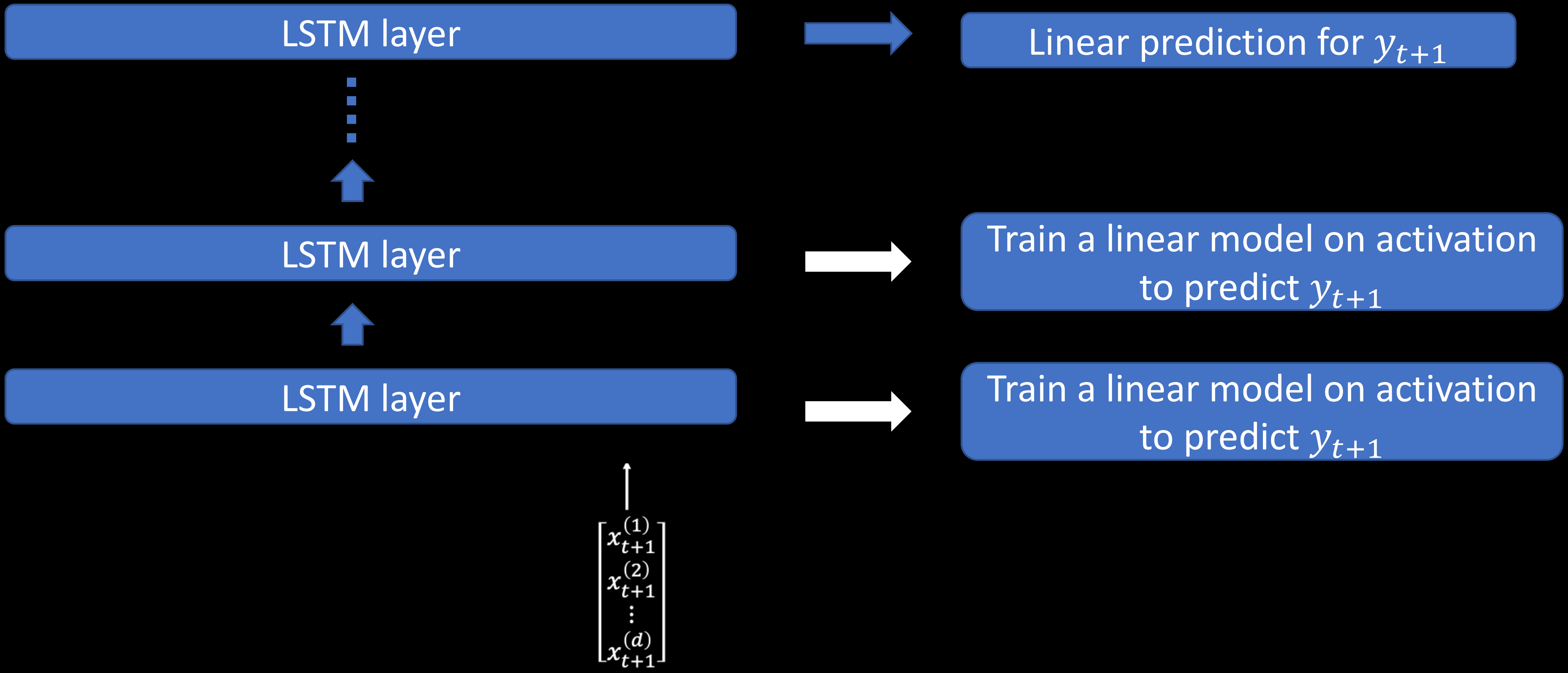
LSTM layer

LSTM layer

…

$$\begin{bmatrix} x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \\ \vdots \\ x_{t+1}^{(d)} \end{bmatrix}$$

LSTMs as an iterative algorithm: probing layers



LSTM layer

Linear prediction for y_{t+1}

LSTM layer

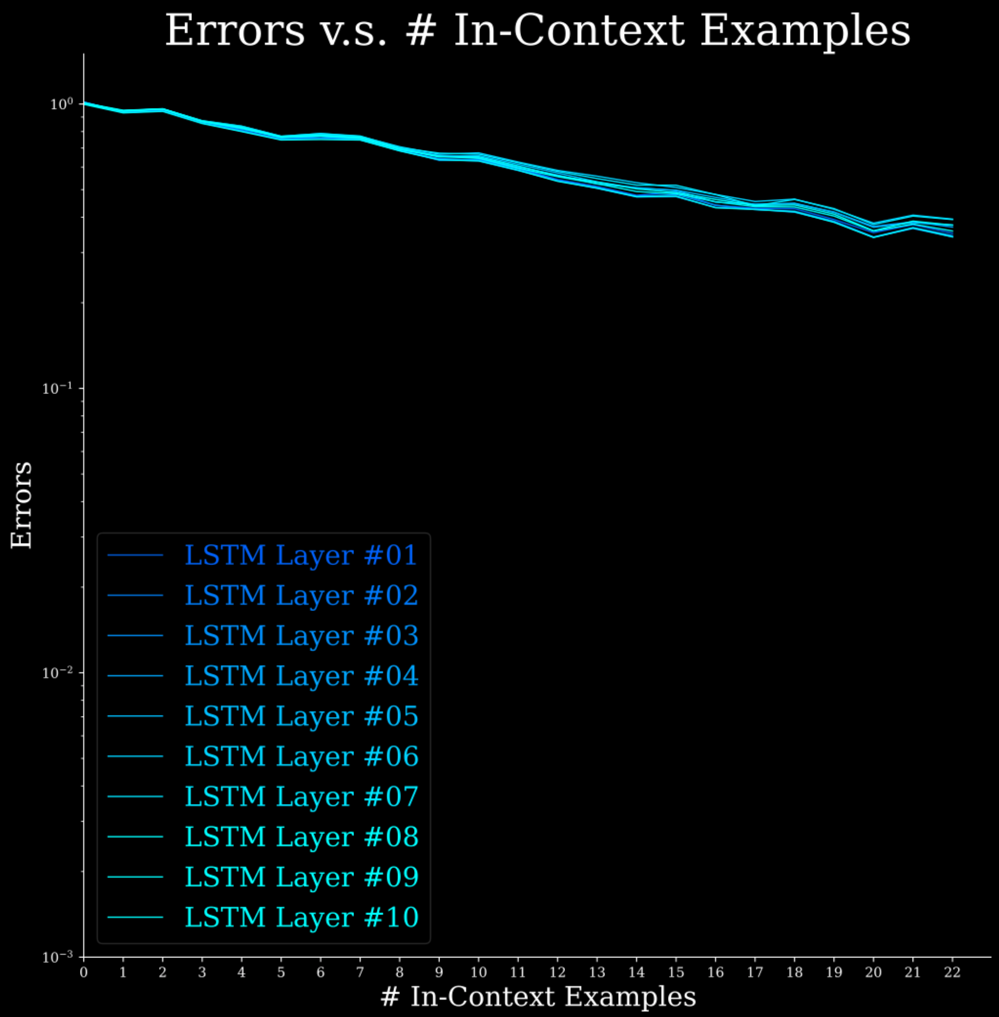
Train a linear model on activation to predict y_{t+1}

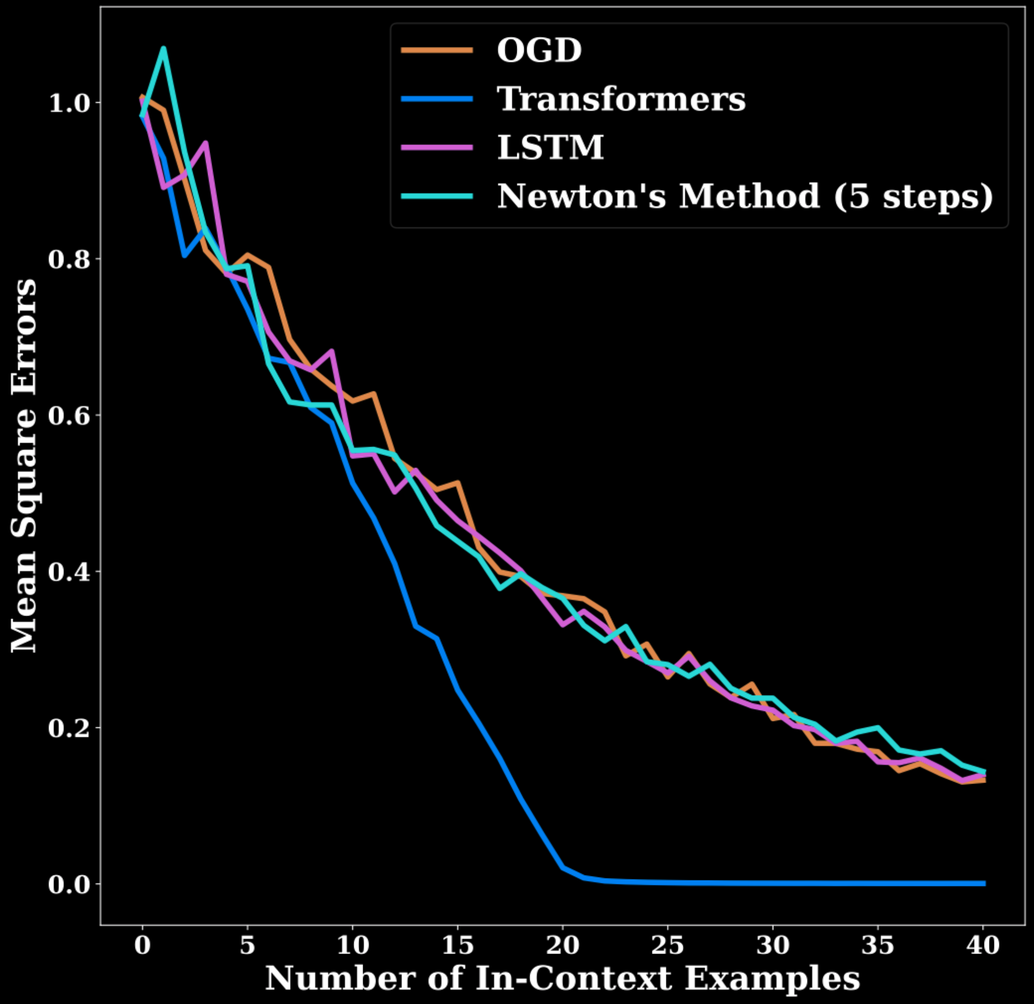
LSTM layer

Train a linear model on activation to predict y_{t+1}

$$\begin{bmatrix} x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \\ \vdots \\ x_{t+1}^{(d)} \end{bmatrix}$$

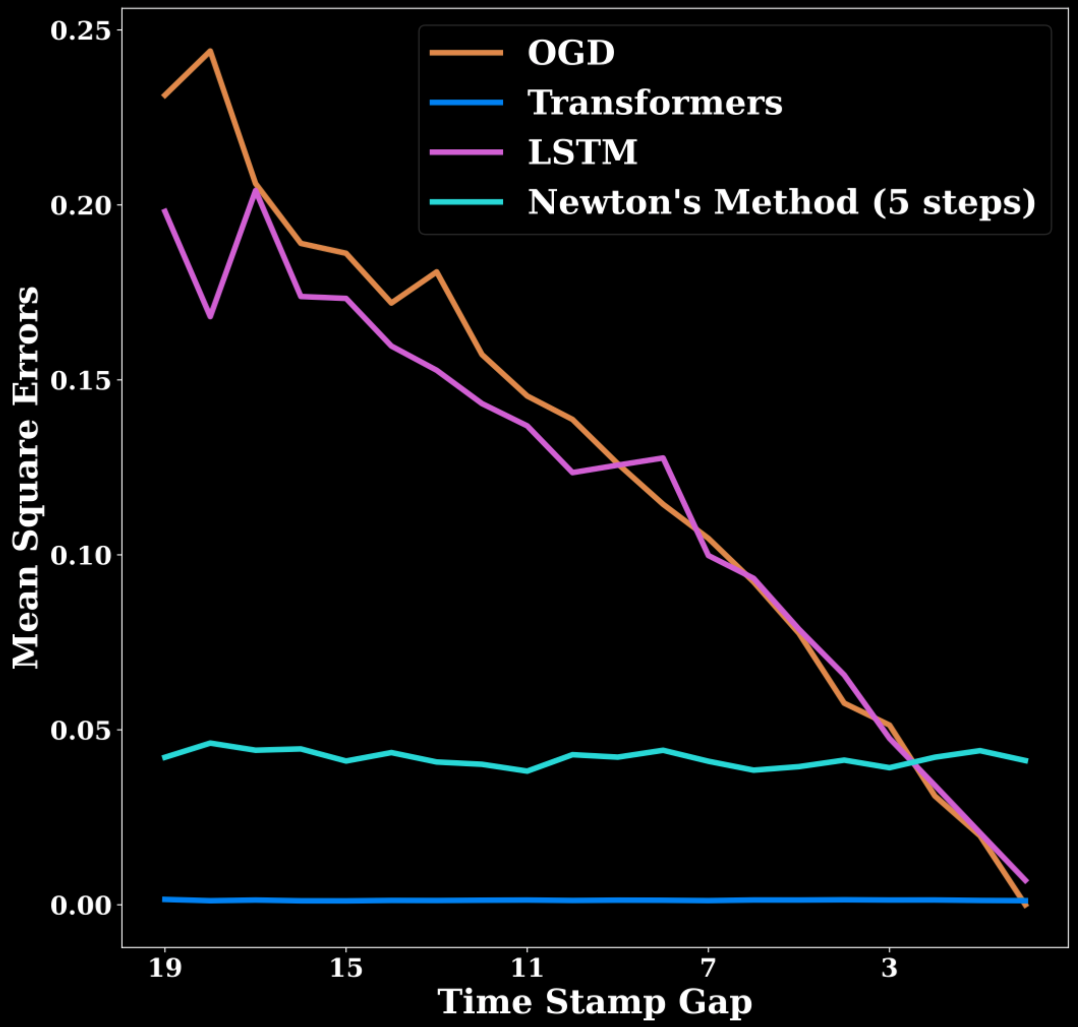
What do LSTMs implement?





LSTMs seem similar to online gradient descent

Like OGD, LSTMs ‘forget’ previous examples



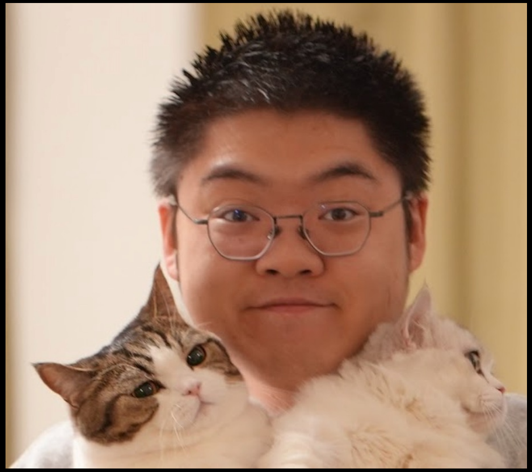
Error when input from t time steps ago is given as query point

Hypothesis: The additional memory available to Transformers (since they have access to entire past sequence) versus recurrent architectures enables it to learn more efficient algorithm

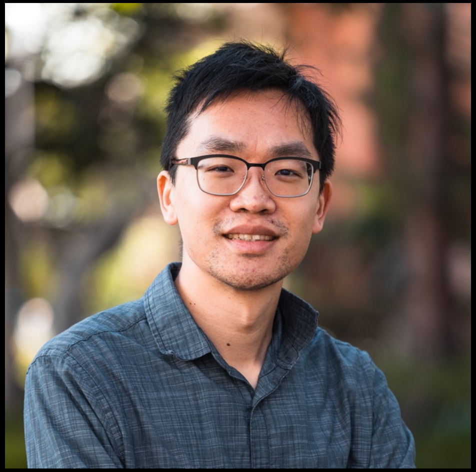
Recent line of theoretical work suggests that the available memory determines the best possible convergence rate, is gap between architectures an instantiation of this?

What is the role of pre-training?
How do LLMs add?

Tianyi Zhou (USC)



Deqing Fu (USC)



Robin Jia (USC)

Pre-trained LLMs Use Fourier Features to Compute Addition,
Neurips 2024

How do pre-trained Transformers do addition?

Fine-tune GPT-2XL on addition dataset:

- *What is the sum of 15 and 93? 108*
- *What is the sum of 24 and 171? 195*

…

How do pre-trained Transformers do addition?

Fine-tune GPT-2XL on addition dataset:

- *What is the sum of 15 and 93? 108*
- *What is the sum of 24 and 171? 195*

…

Each number is its own token

How do pre-trained Transformers do addition?

Fine-tune GPT-2XL on addition dataset:

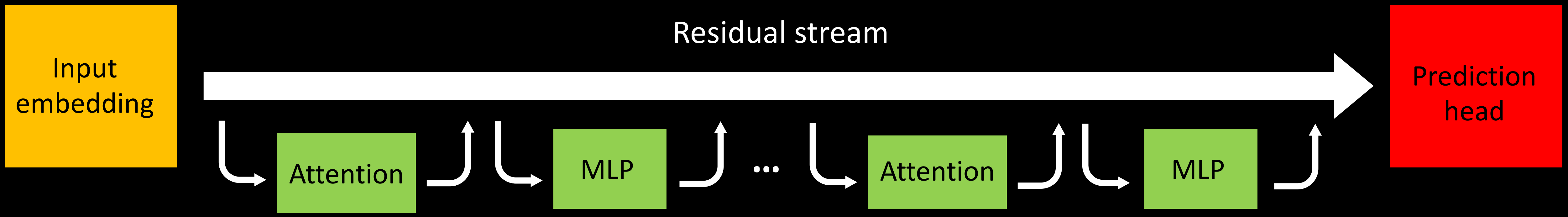
- *What is the sum of 15 and 93? 108*
- *What is the sum of 24 and 171? 195*

…

Model gets $\approx 100\%$ test accuracy.

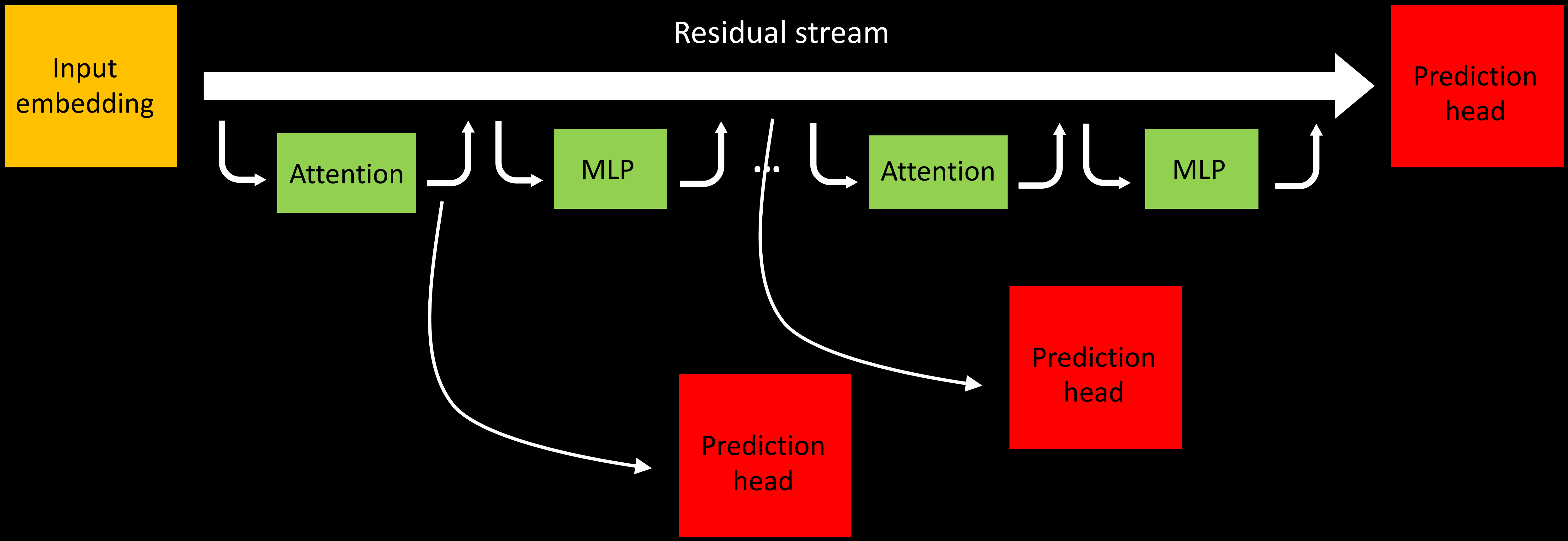
What mechanisms does the model use?

Understanding mechanisms: Logit Lens



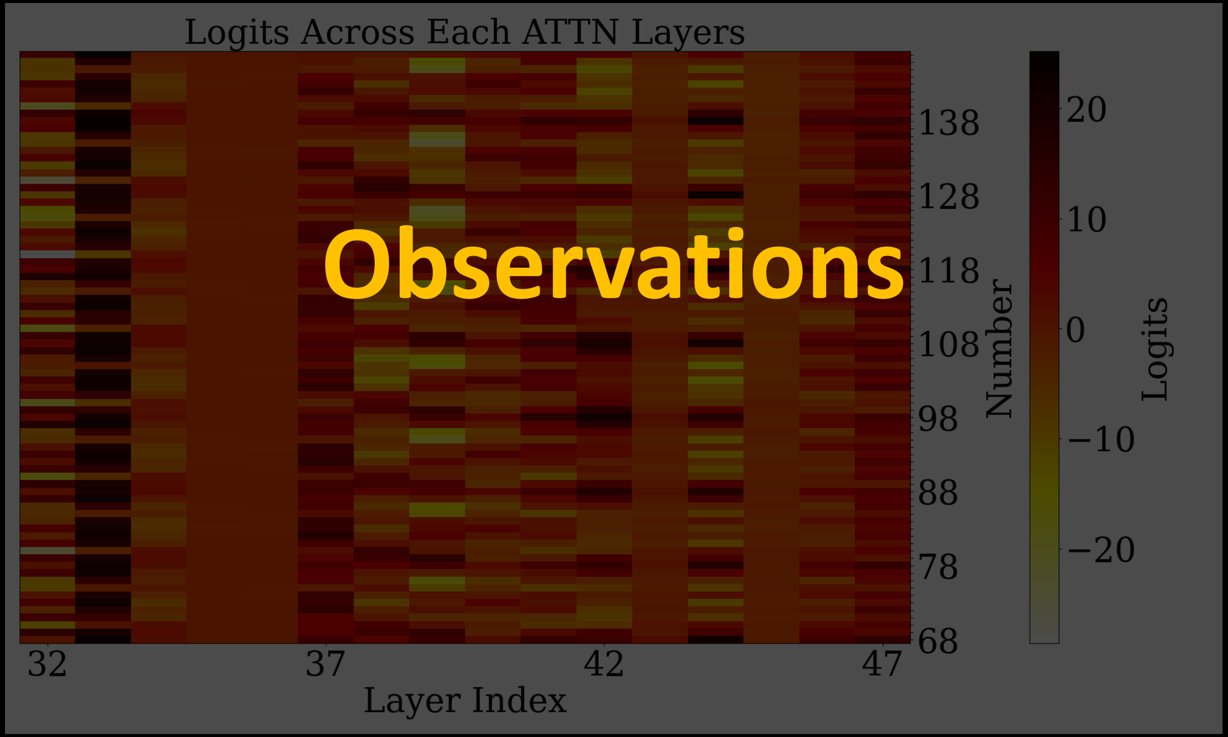
Each Attention/MLP component makes additive contribution to residual stream

Understanding mechanisms: Logit Lens

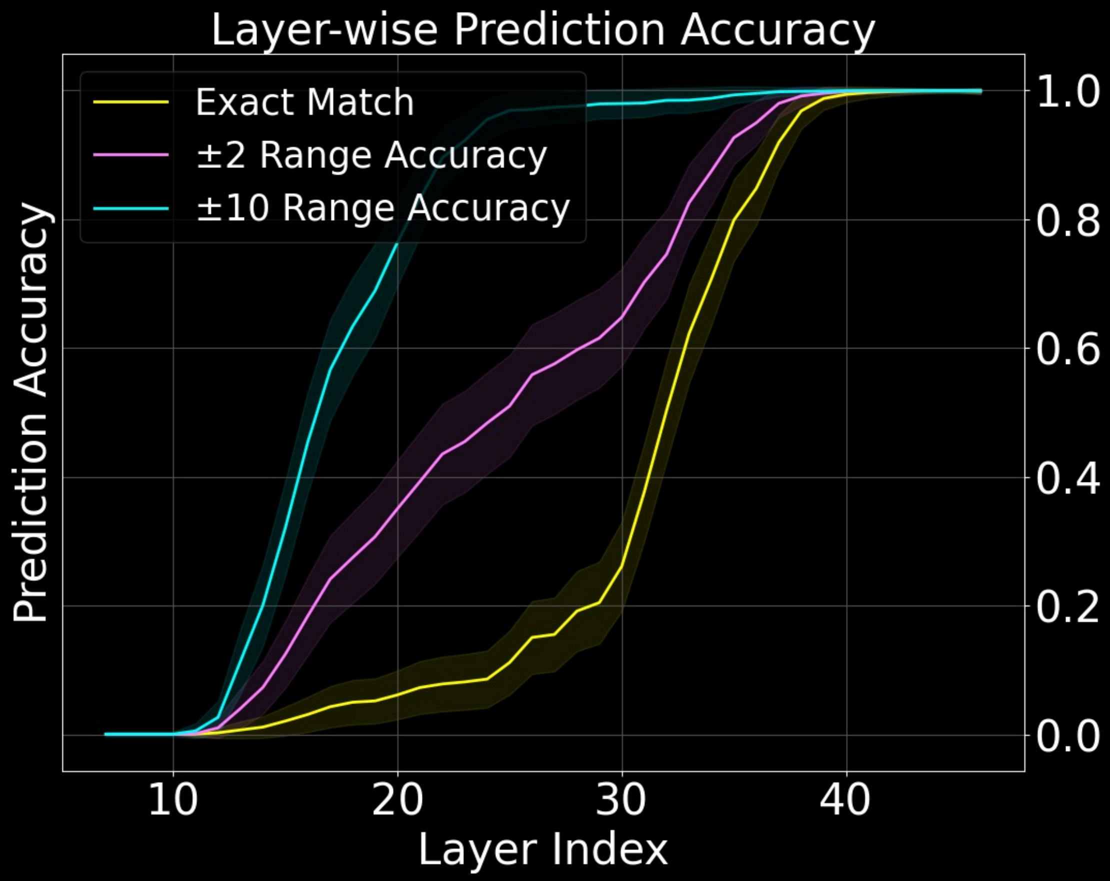


Can use prediction head to understand predictions at any stage

Observations



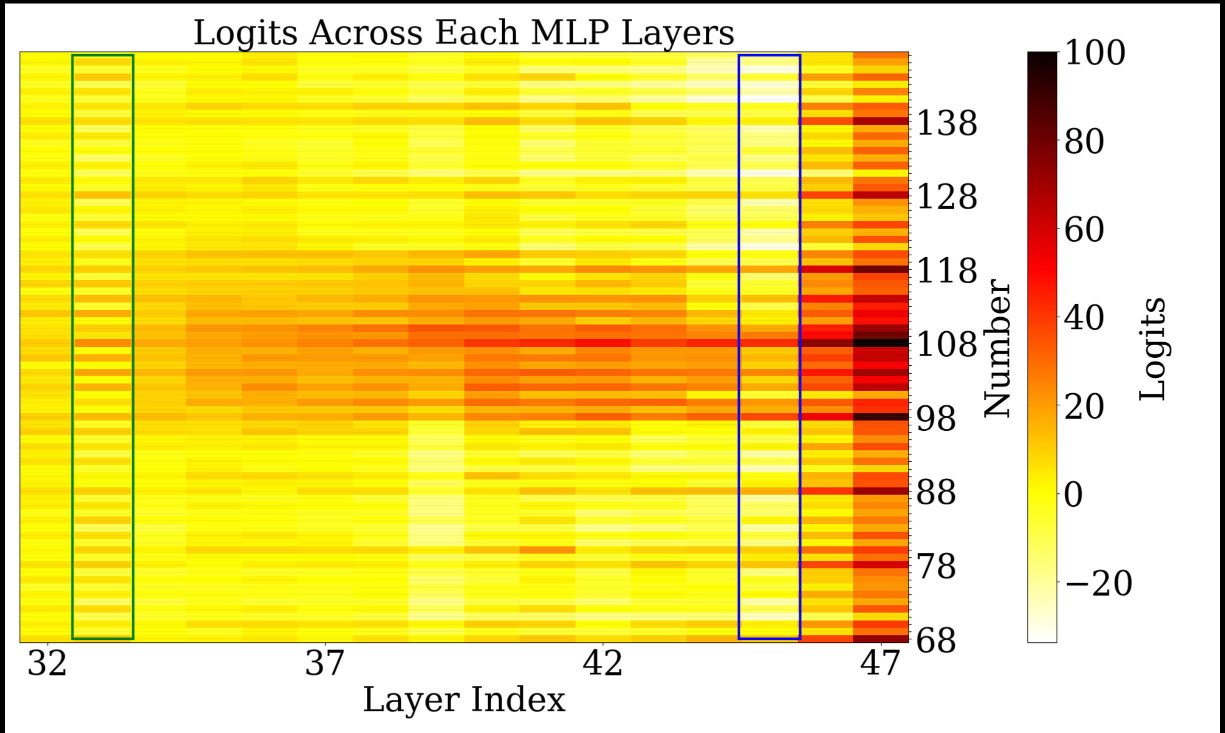
Model improves across layers

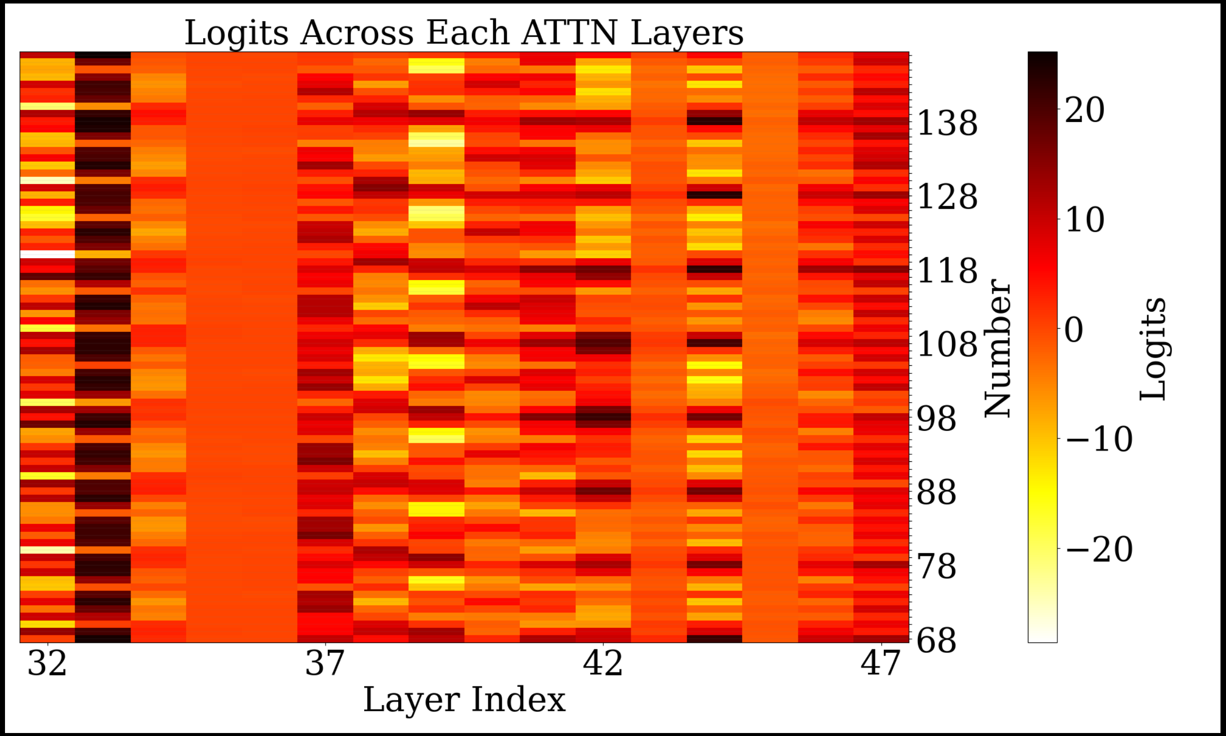


Model finds answer within a ±2 and ±10 range early on, and finds exact match later

Examining the contribution of each MLP & Attention layer

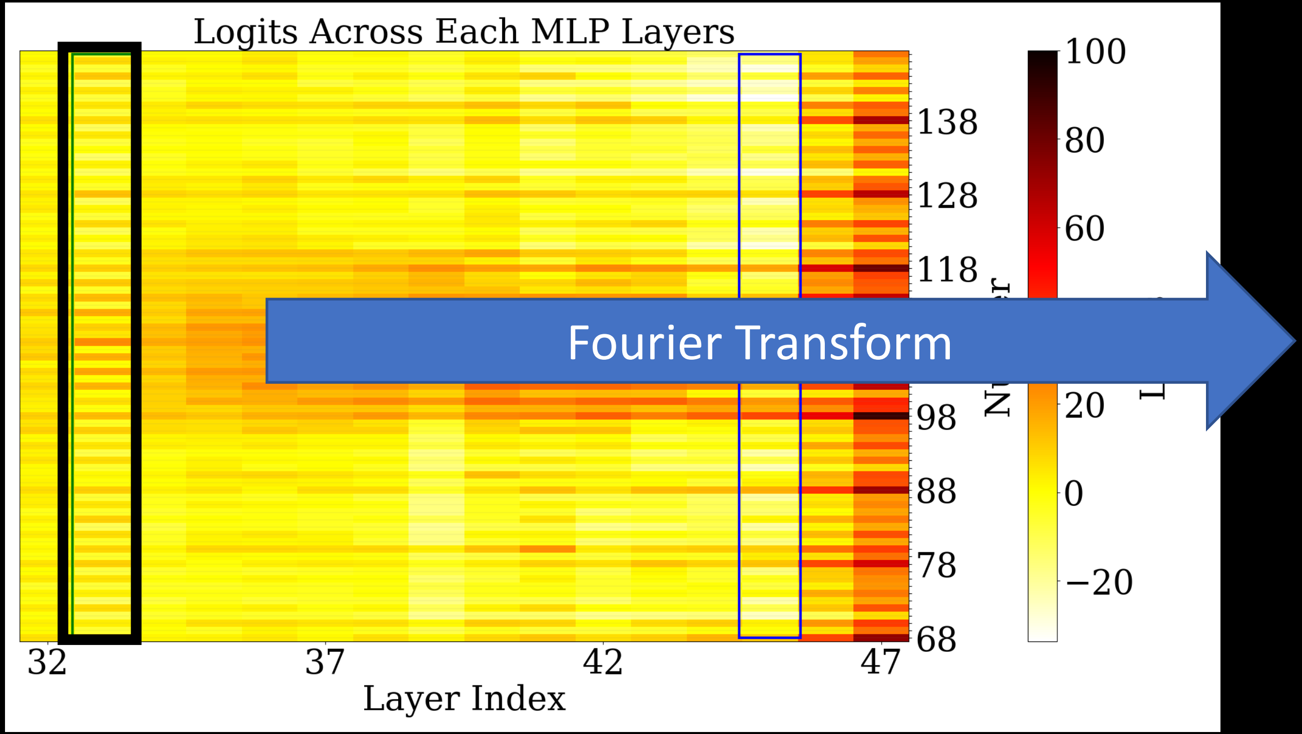
Input: What is the sum of 15 and 93?

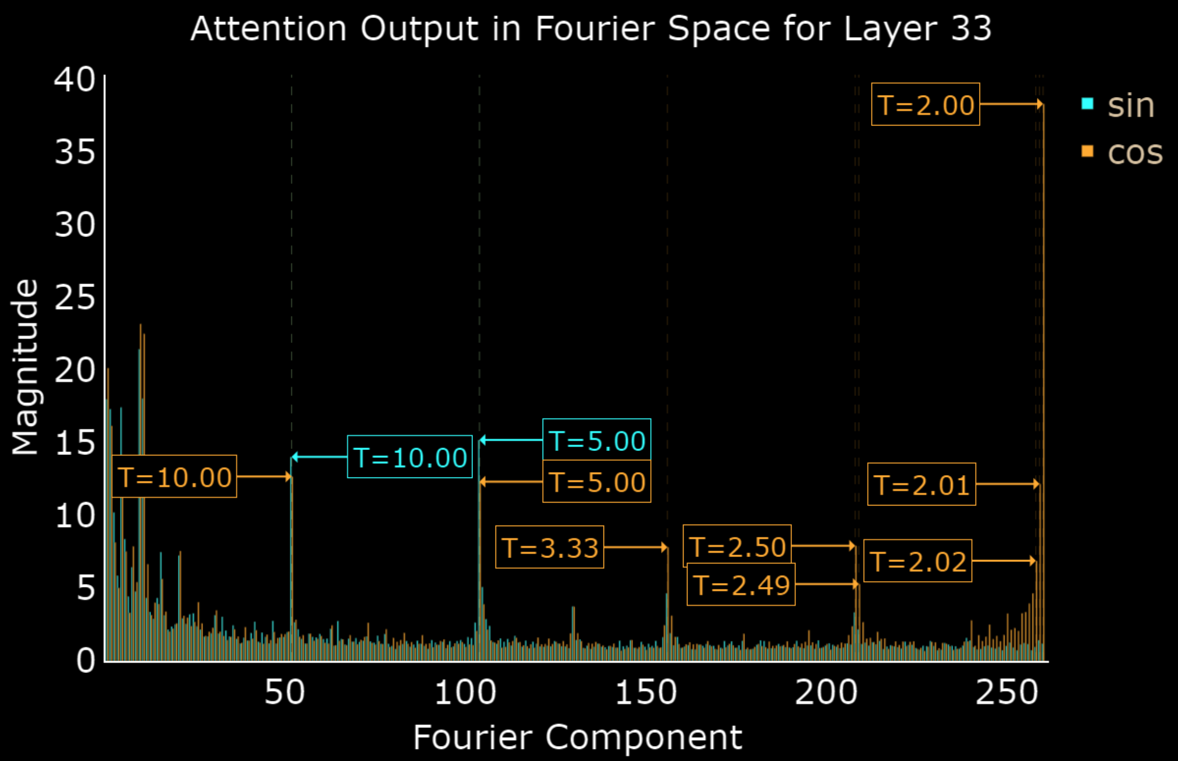




Examining the contribution of each MLP & Attention layer

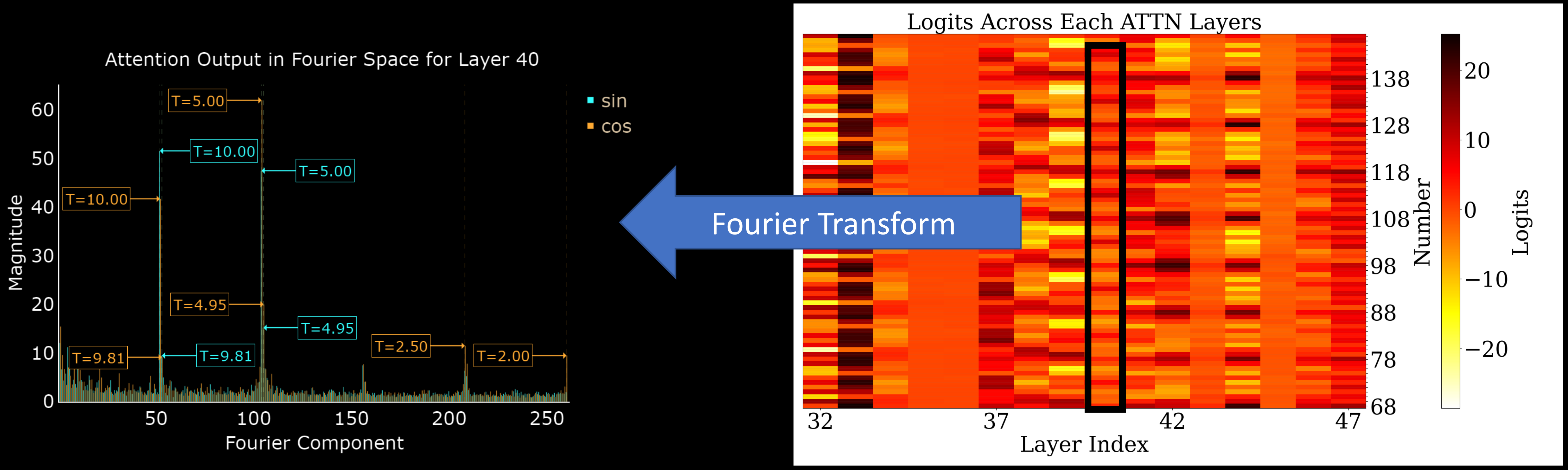
Input: What is the sum of 15 and 93?





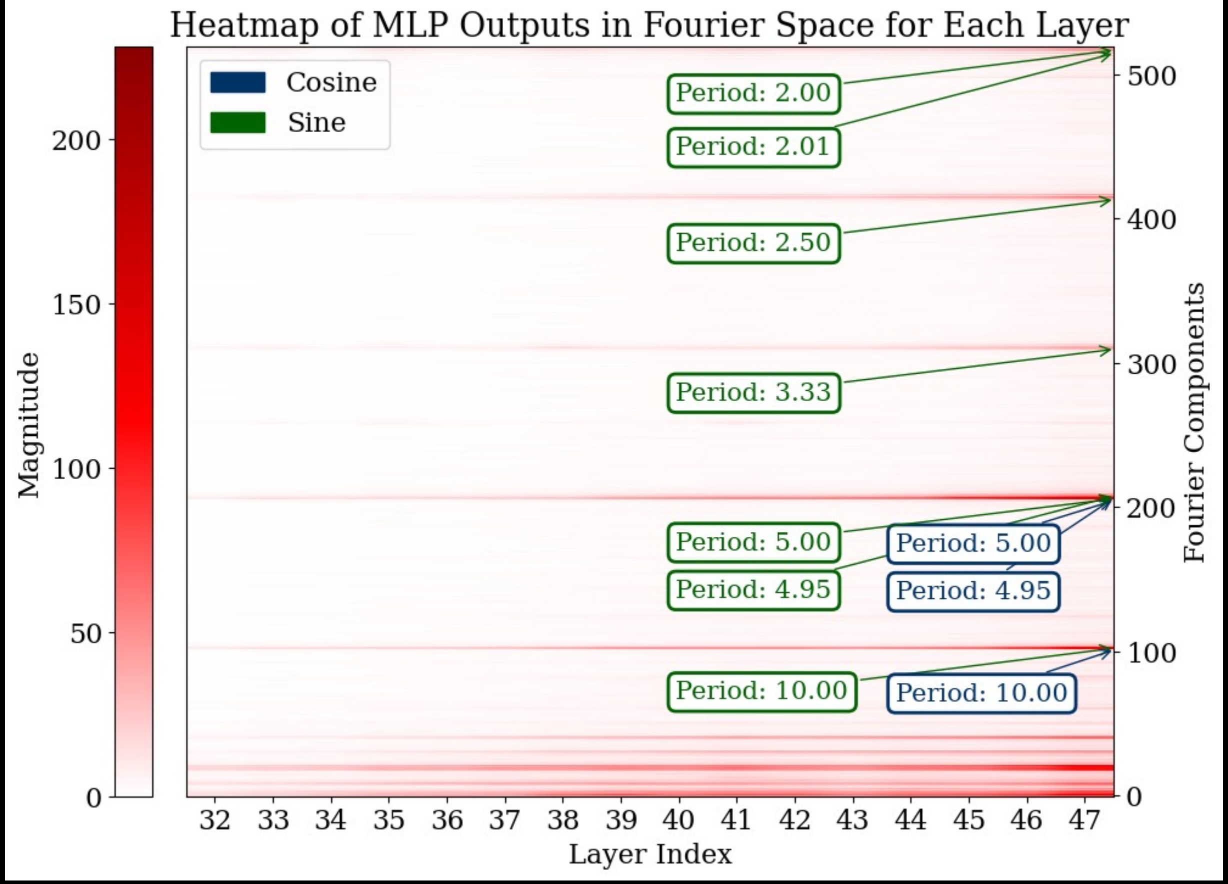
Examining the contribution of each MLP & Attention layer

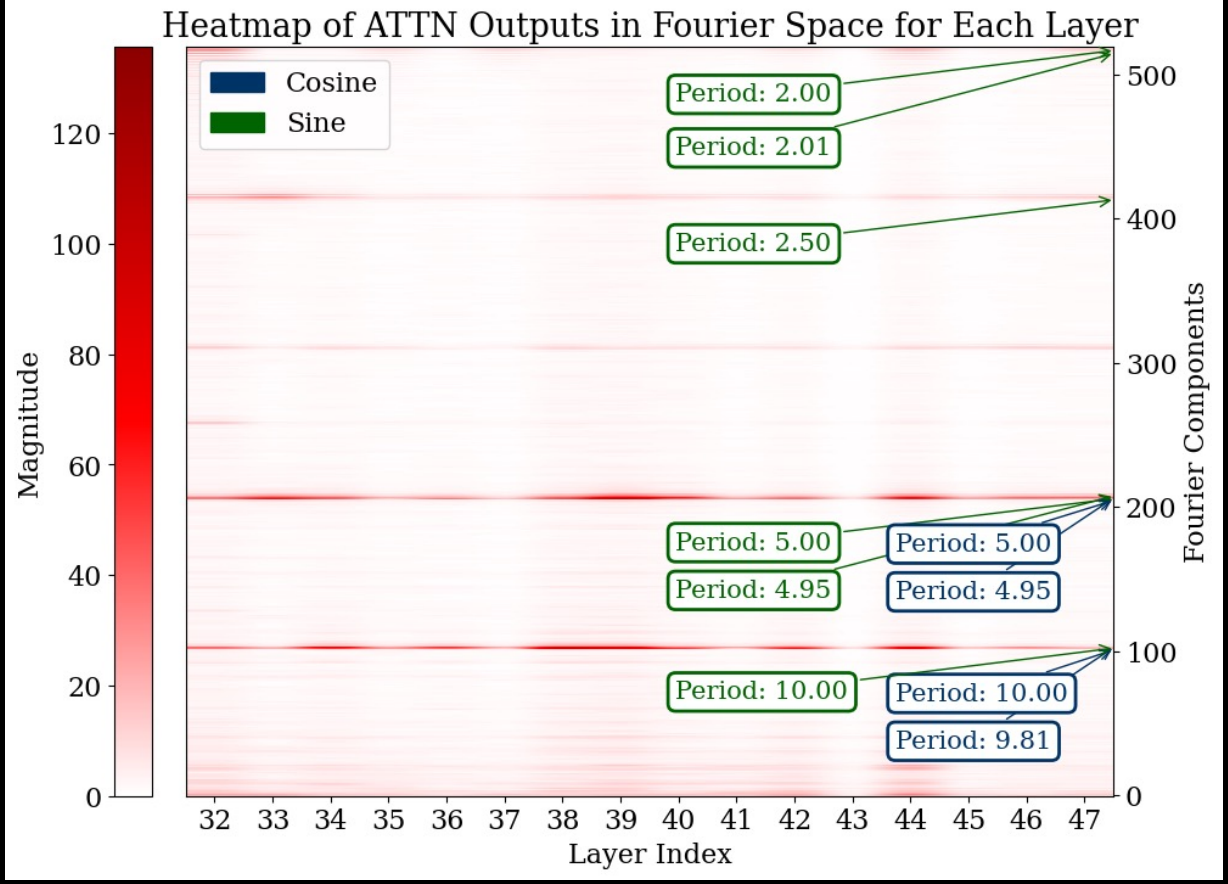
Input: What is the sum of 15 and 93?



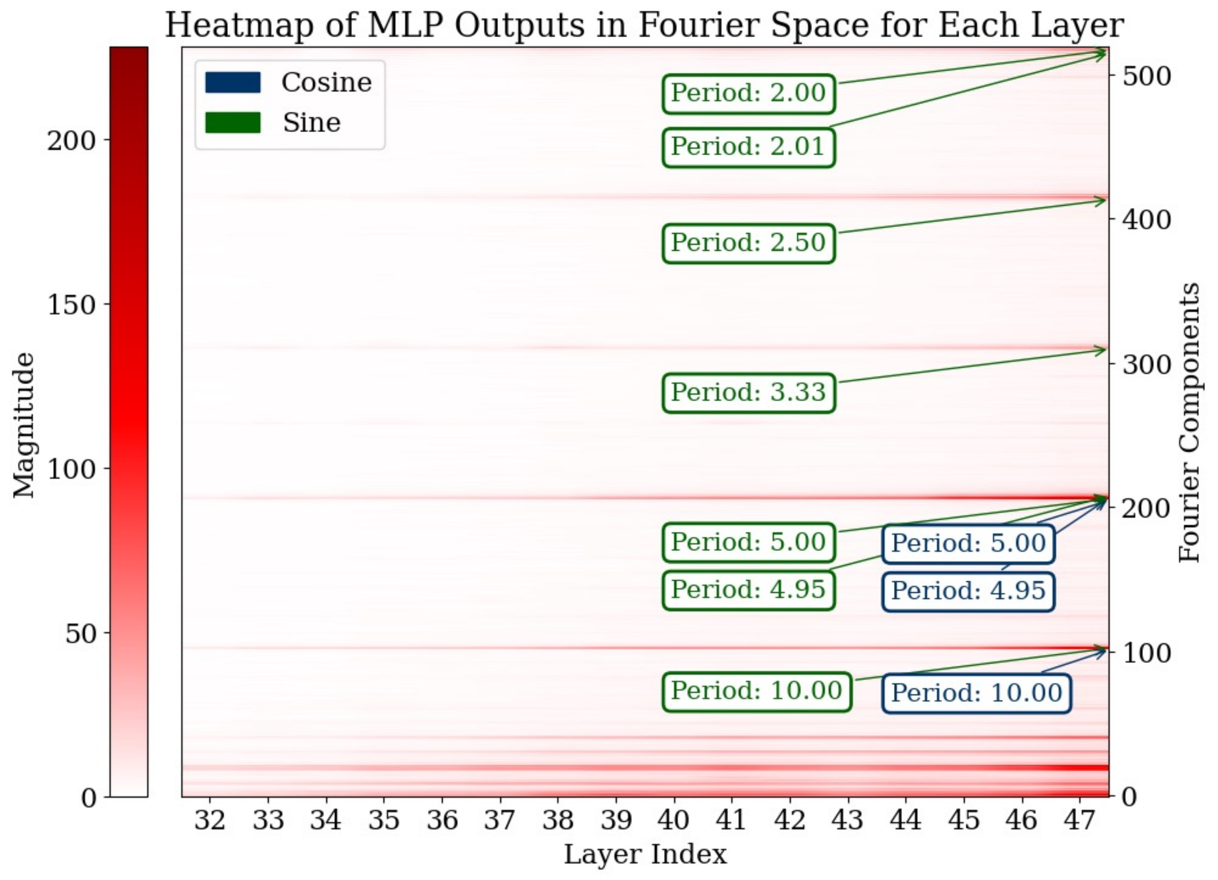
Fourier Transform

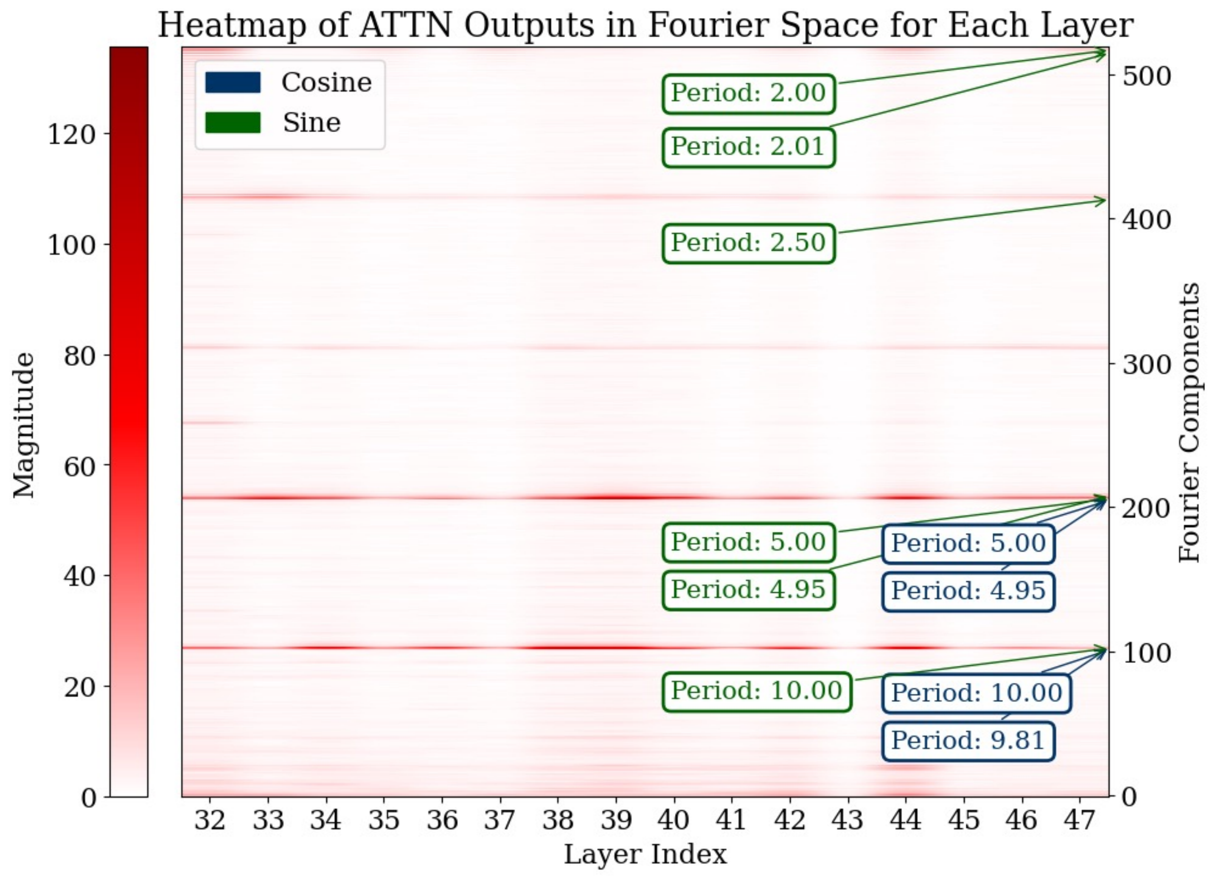
On average across all examples, logits are sparse in Fourier space





Fourier features: Sparse representations in Fourier space



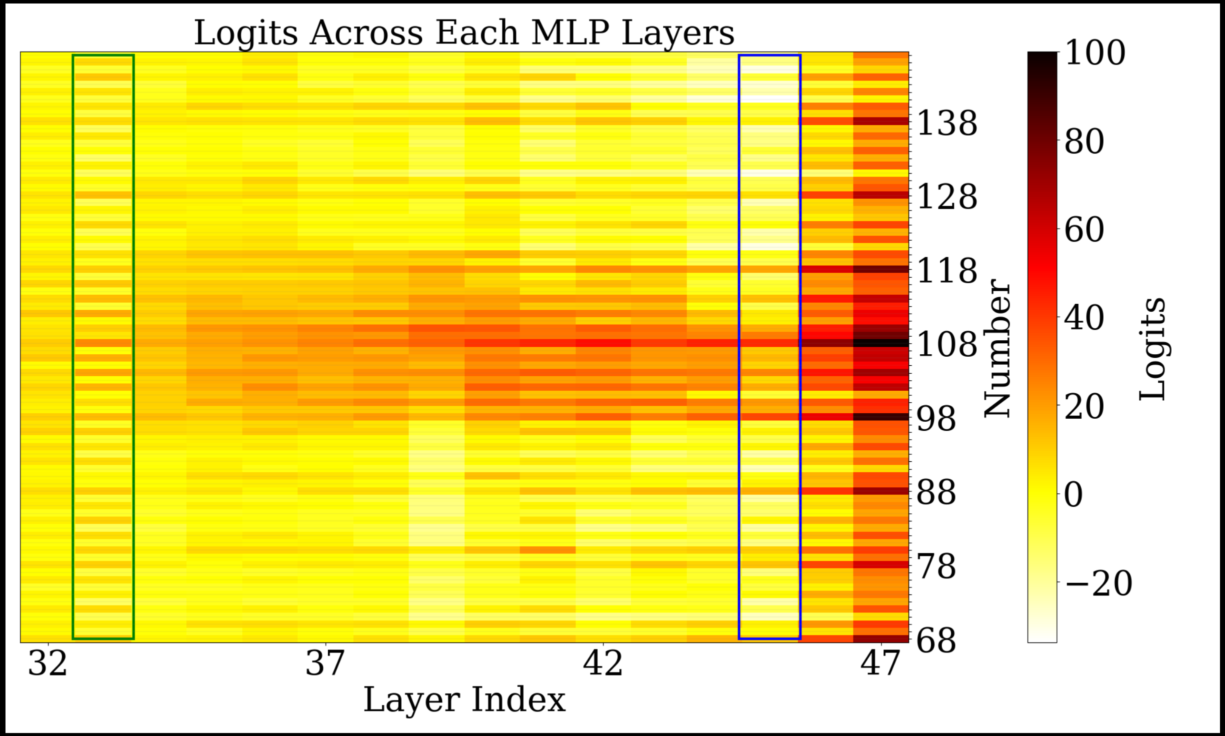


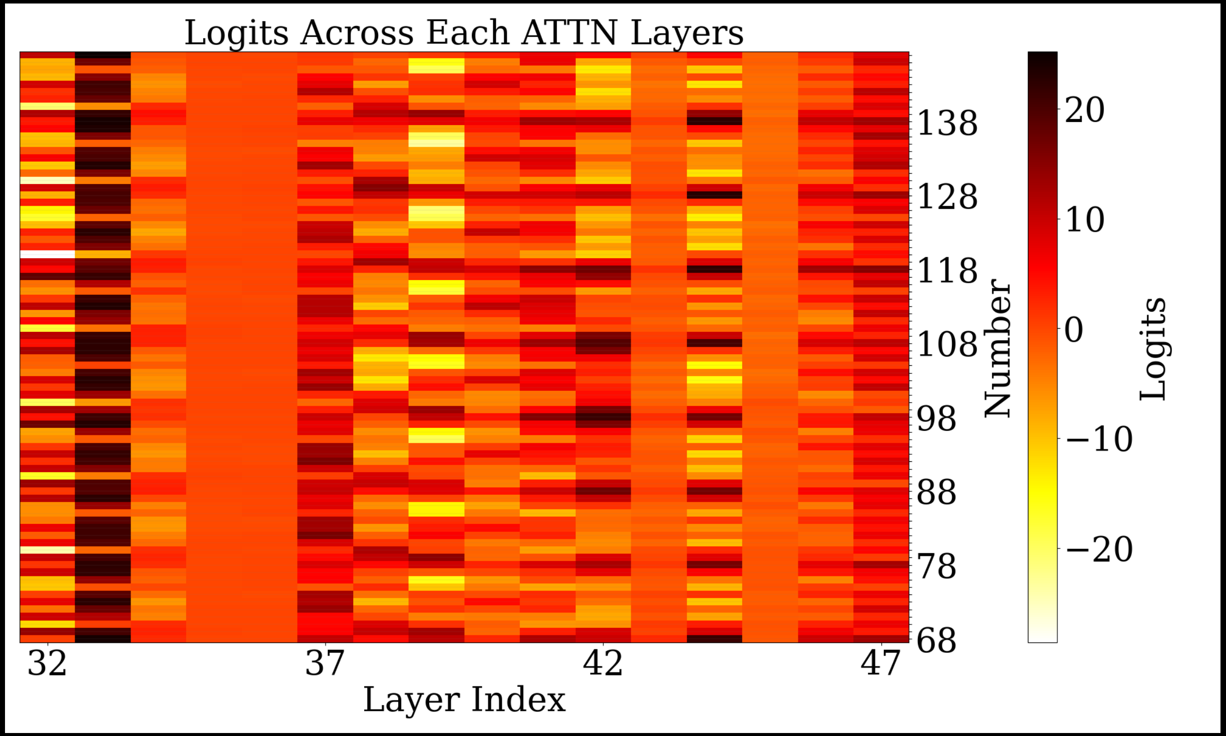
Low frequency components **approximate** magnitude of answer

High frequency components do **classification**: compute sum modulo p for $p \in \{2, 5, 10, etc.\}$

Fourier features: Sparse representations in Fourier space

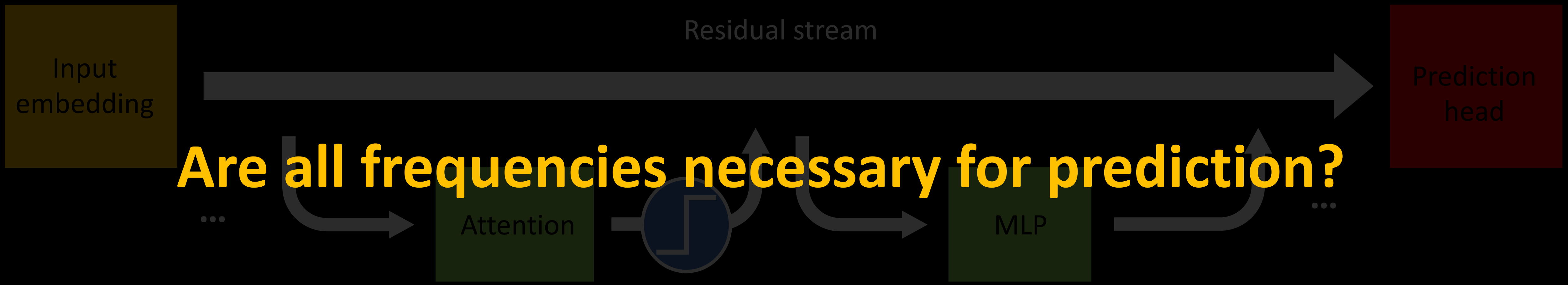
Input: What is the sum of 15 and 93?





Low frequency components **approximate** magnitude of answer

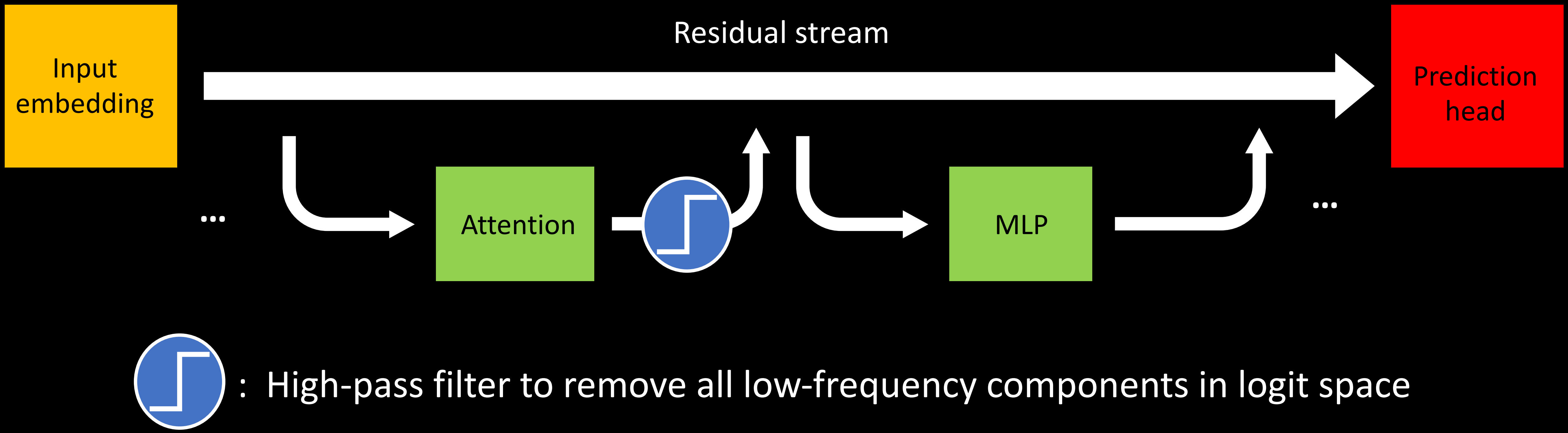
High frequency components do **classification**: compute sum modulo p for p $\in \{2,5,10, etc.\}$



Are all frequencies necessary for prediction?

Do Attention and MLP layers have similar roles?

Applying filters to understand role of components



Input embedding

Residual stream

Prediction head

...

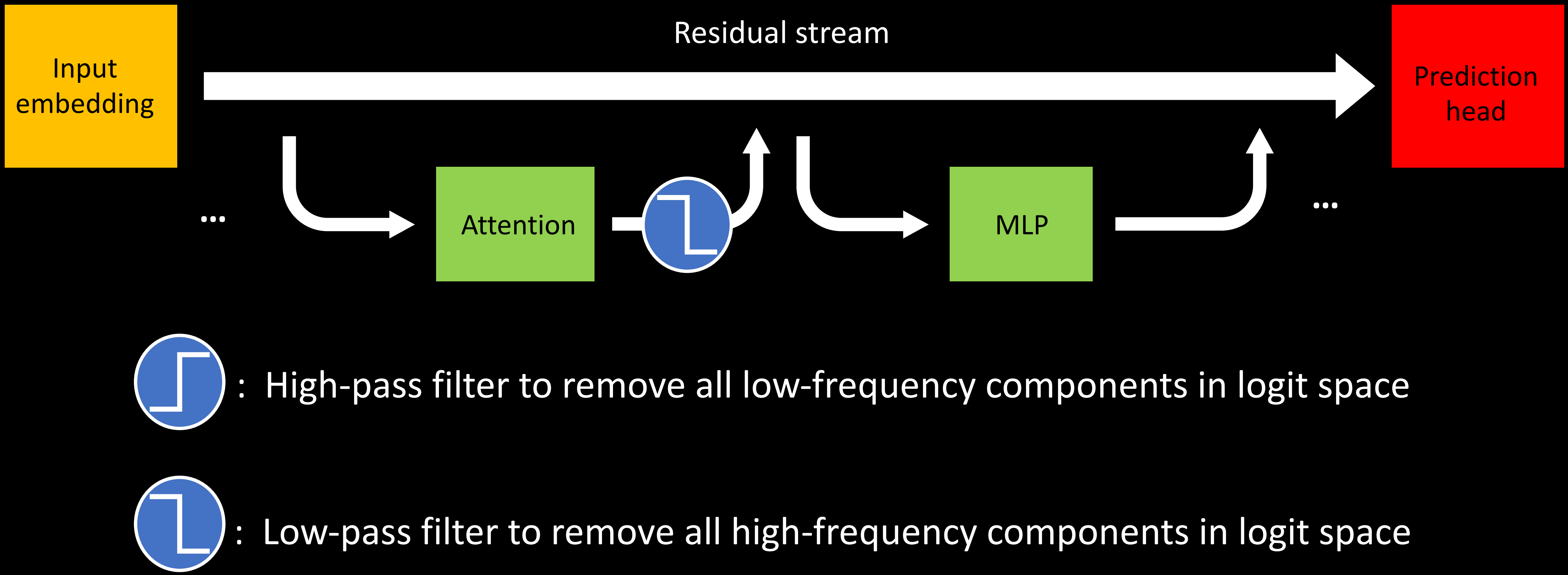
Attention

MLP

...

: High-pass filter to remove all low-frequency components in logit space

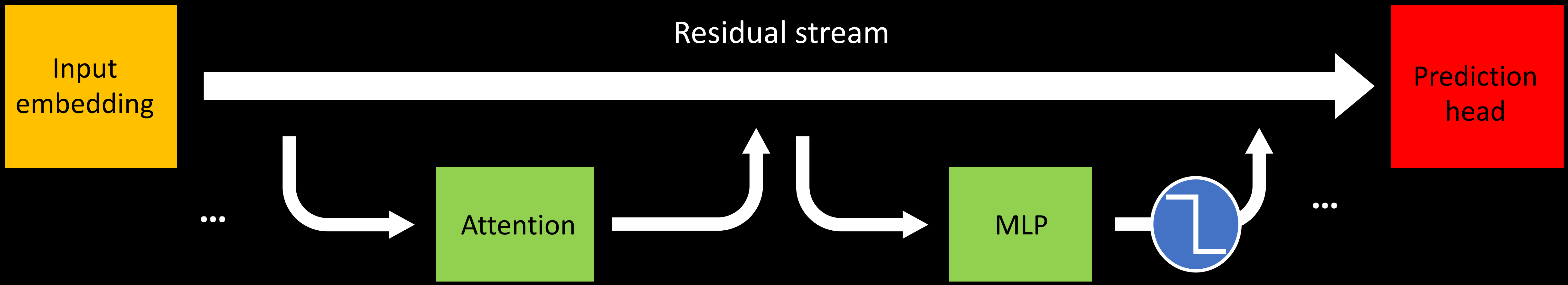
Applying filters to understand role of components



: High-pass filter to remove all low-frequency components in logit space

: Low-pass filter to remove all high-frequency components in logit space

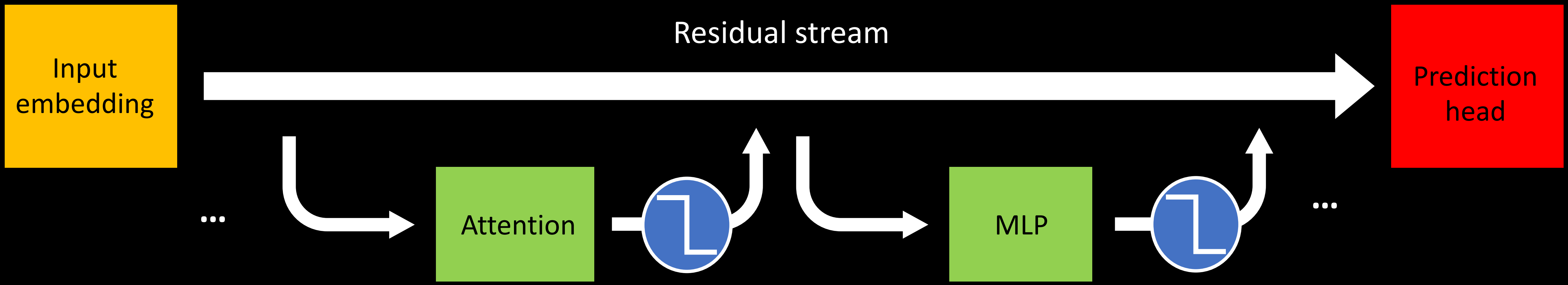
Applying filters to understand role of components



: High-pass filter to remove all low-frequency components in logit space

: Low-pass filter to remove all high-frequency components in logit space

Applying filters to understand role of components



: High-pass filter to remove all low-frequency components in logit space

: Low-pass filter to remove all high-frequency components in logit space

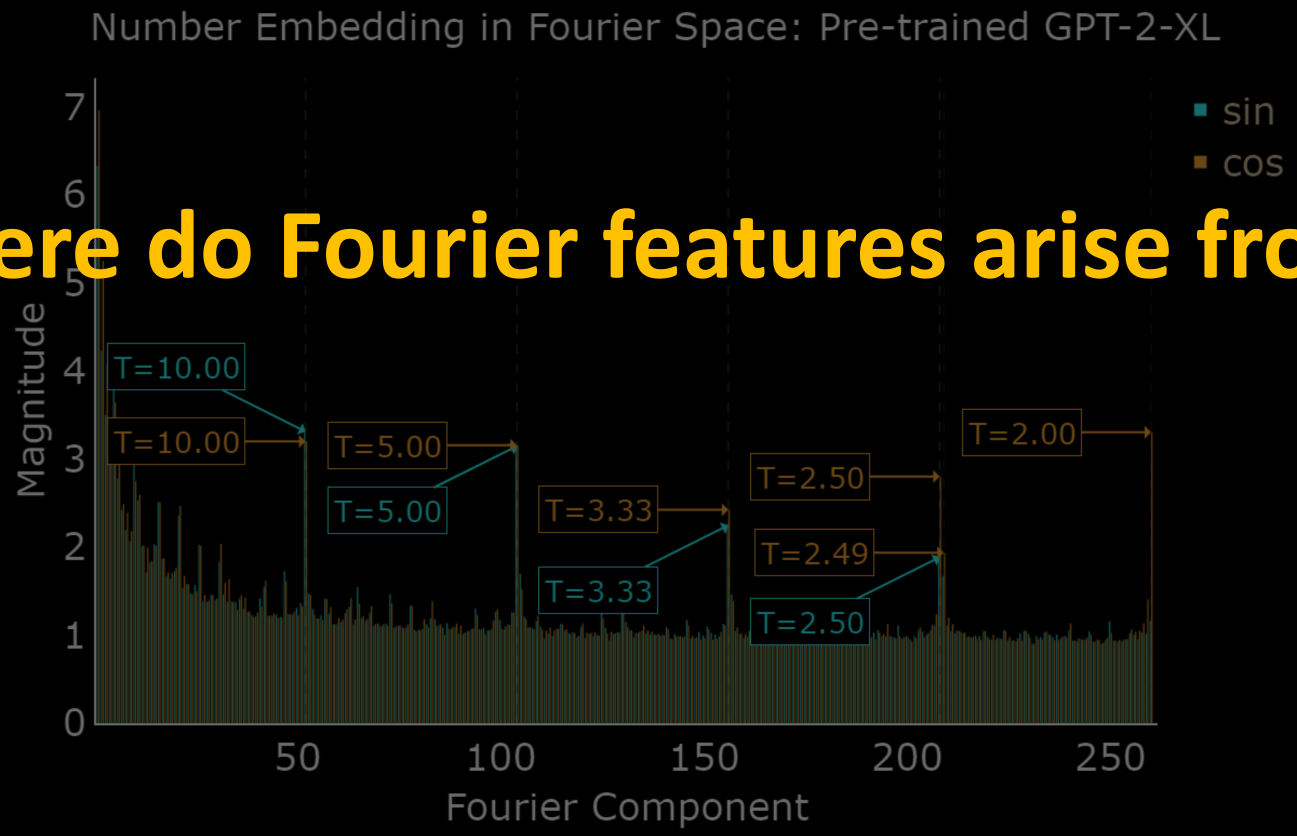
Applying filters to understand role of components

Module	Fourier Component Removed	Accuracy
None	No filtering	99.74%
Attn & MLP	Low frequency	5.94%
Attn	Low frequency	**99.12%**
MLP	Low frequency	35.89%
Attn & MLP	High frequency	27.08%
Attn	High frequency	78.36%
MLP	High frequency	**98.10%**

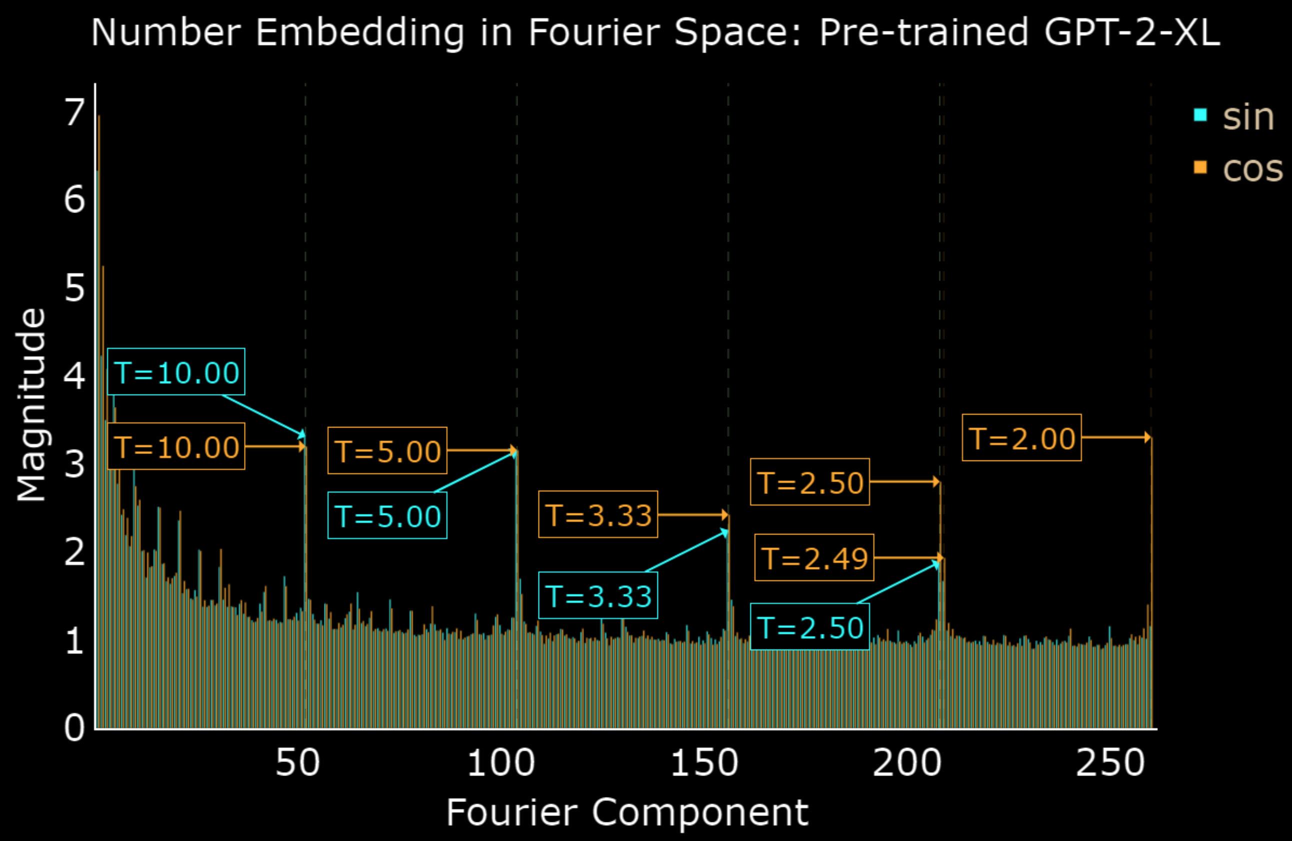
MLP: mainly low-frequency, Attn: mainly high-frequency

Module	Fourier Component Removed	Accuracy
None	No filtering	99.74%
Attn & MLP	Low frequency	5.94%
Attn	Low frequency	**99.12%**
MLP	Low frequency	35.89%
Attn & MLP	High frequency	27.08%
Attn	High frequency	78.36%
MLP	High frequency	**98.10%**

Where do Fourier features arise from?

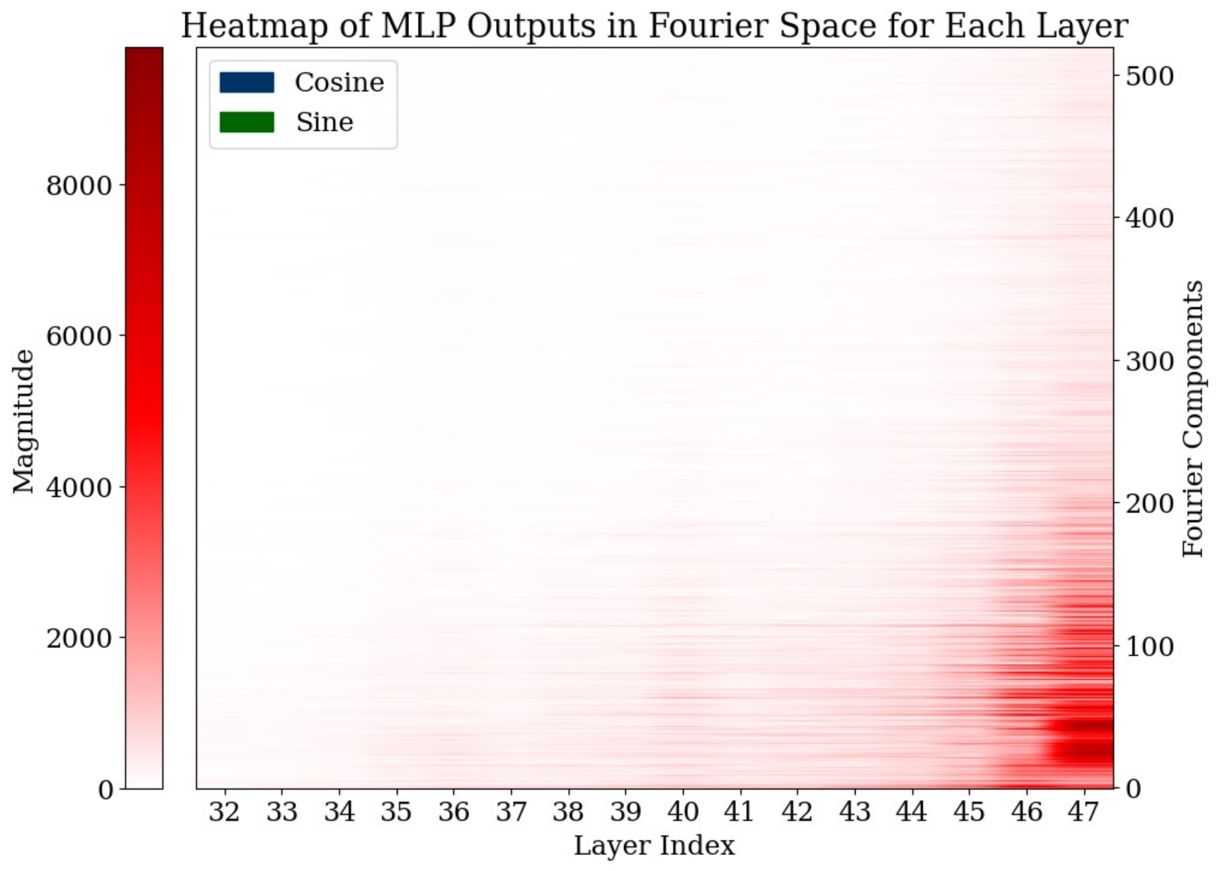


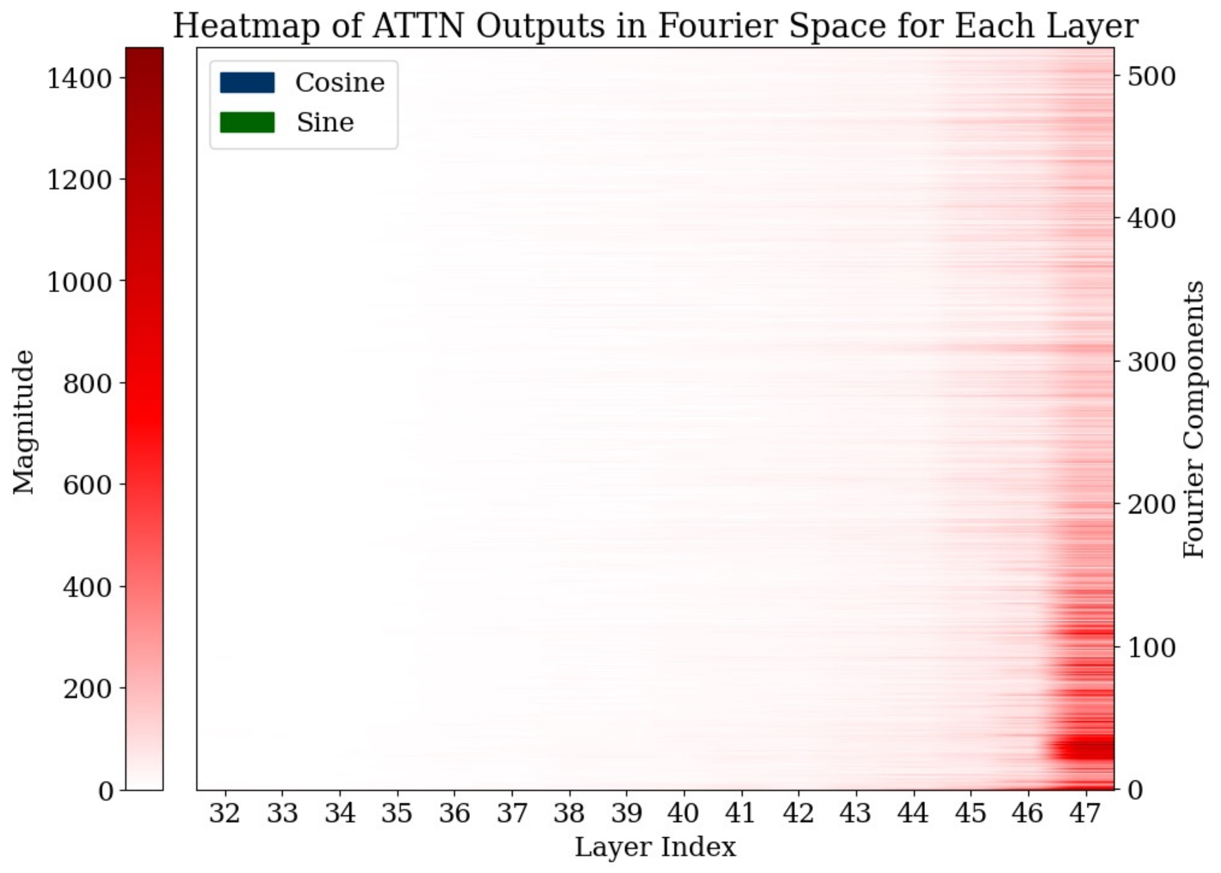
Appear to arise due to token embeddings from pre-training



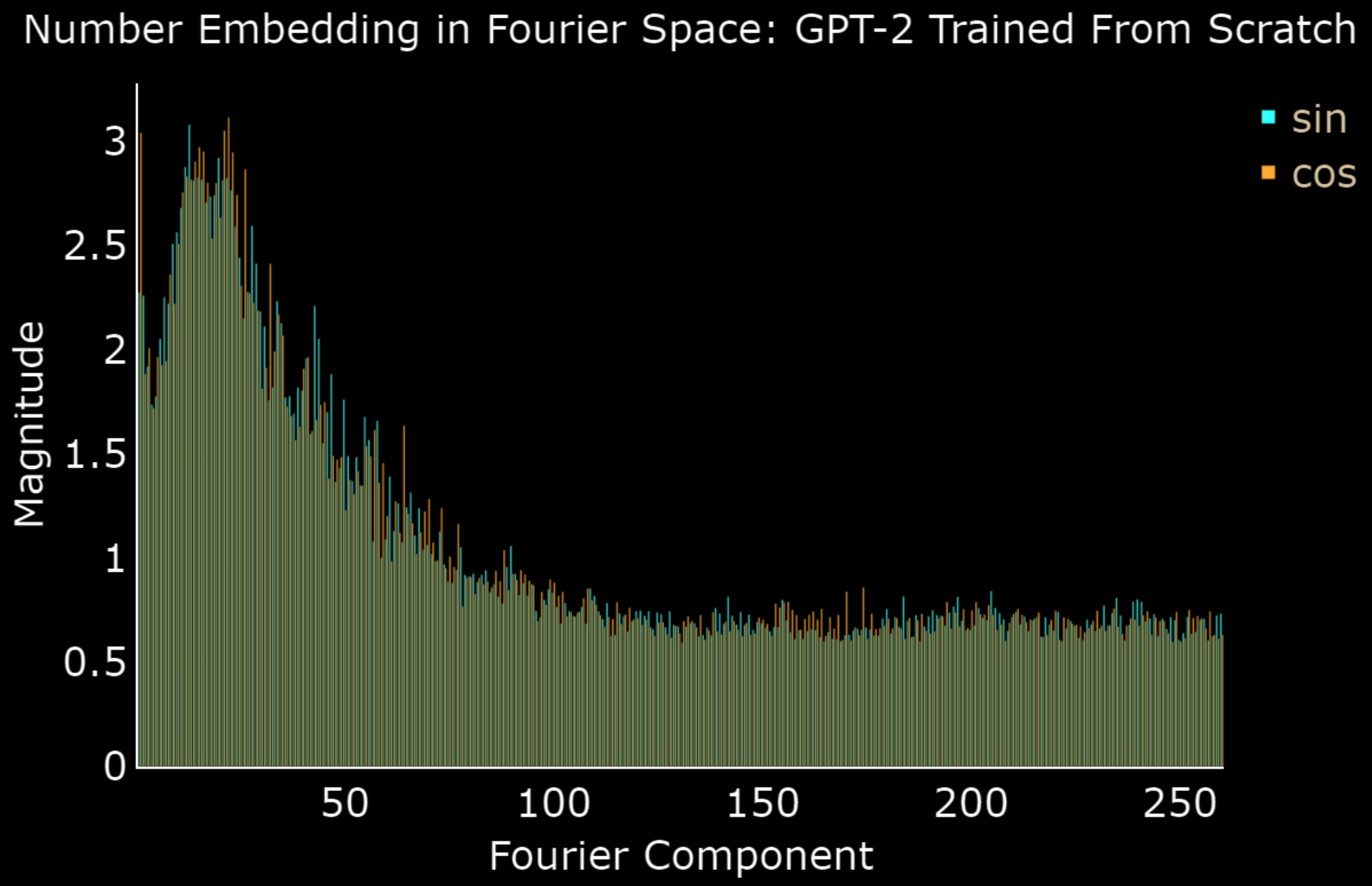
Also see similar behavior for other pre-trained models (Phi-2, RoBERTa).

Model trained from scratch does not exhibit Fourier features

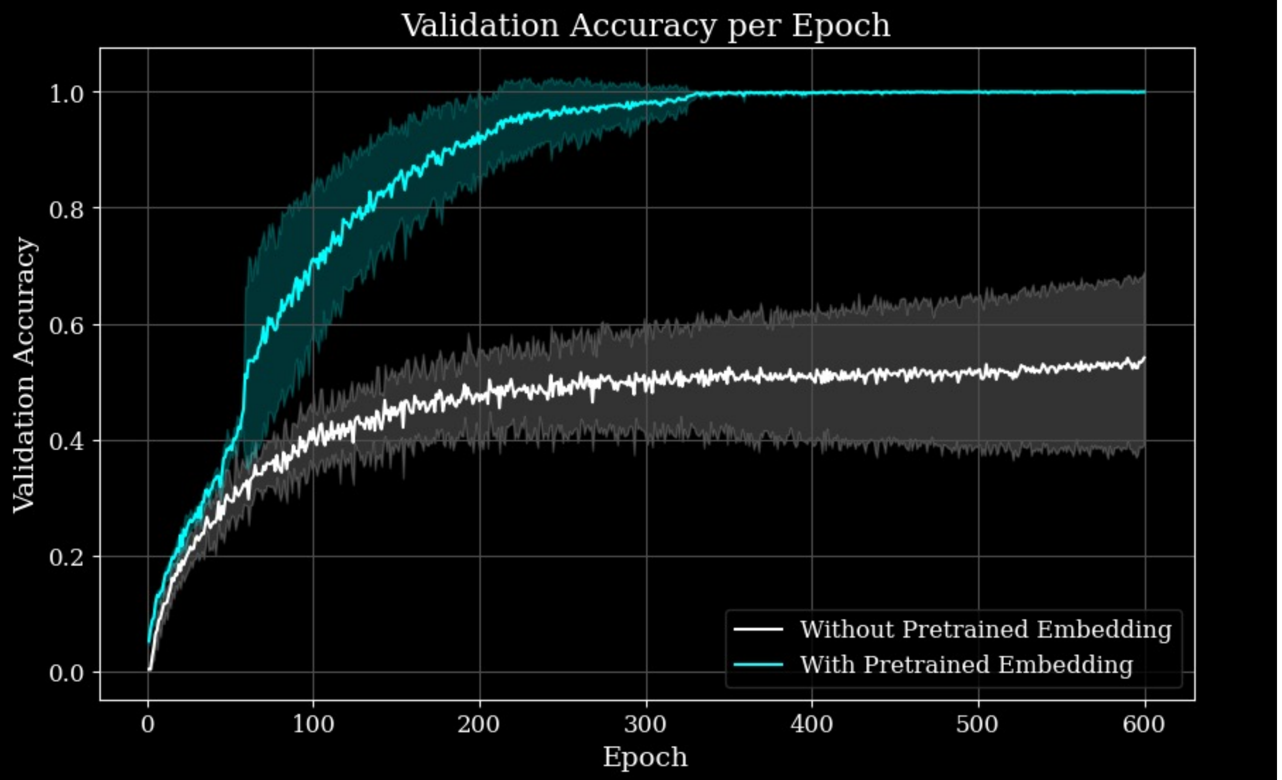


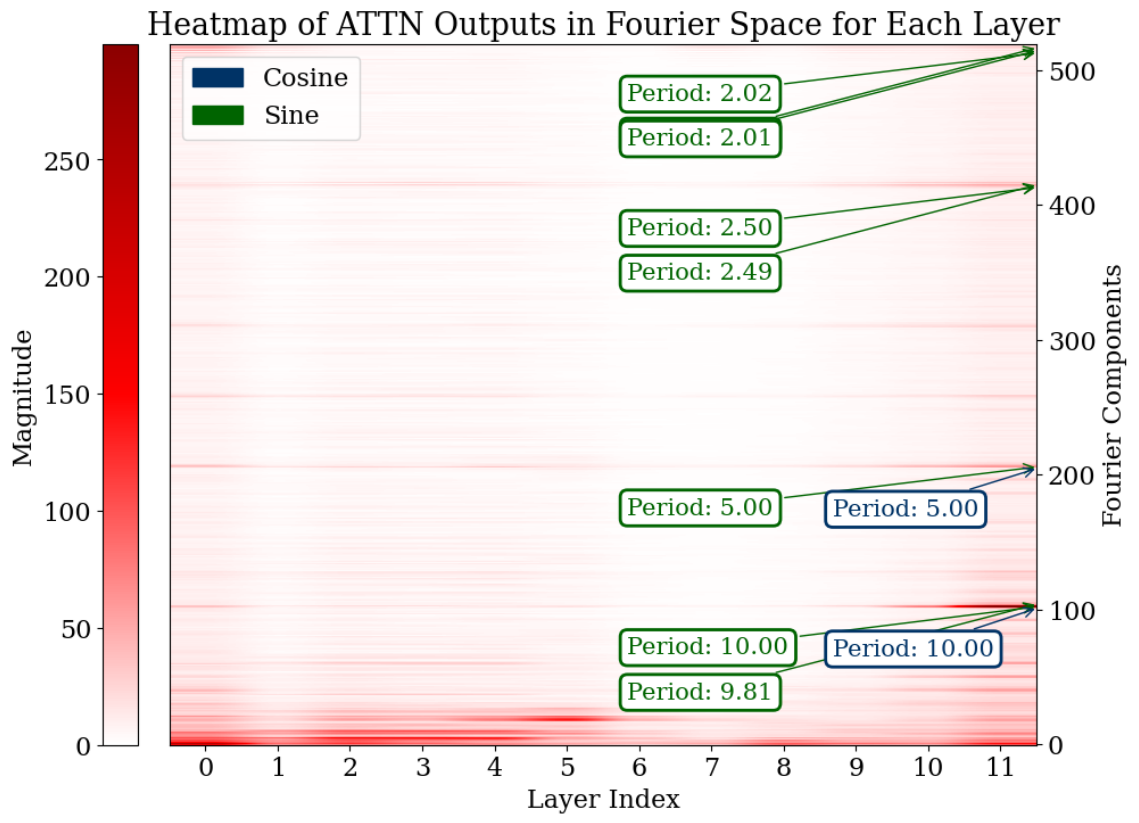


Token embeddings of model trained from scratch do not have Fourier features either



Training model from scratch but with token embeddings from pre-trained models (a) improves training (b) leads to Fourier features





Training model from scratch, but with token embeddings from pre-trained models improves training, leads to Fourier features

Related work: Fourier features in modular addition

Nanda-Chan-Lieberum-Smith-Steinhardt’2023 shows Fourier features are used in modular arithmetic

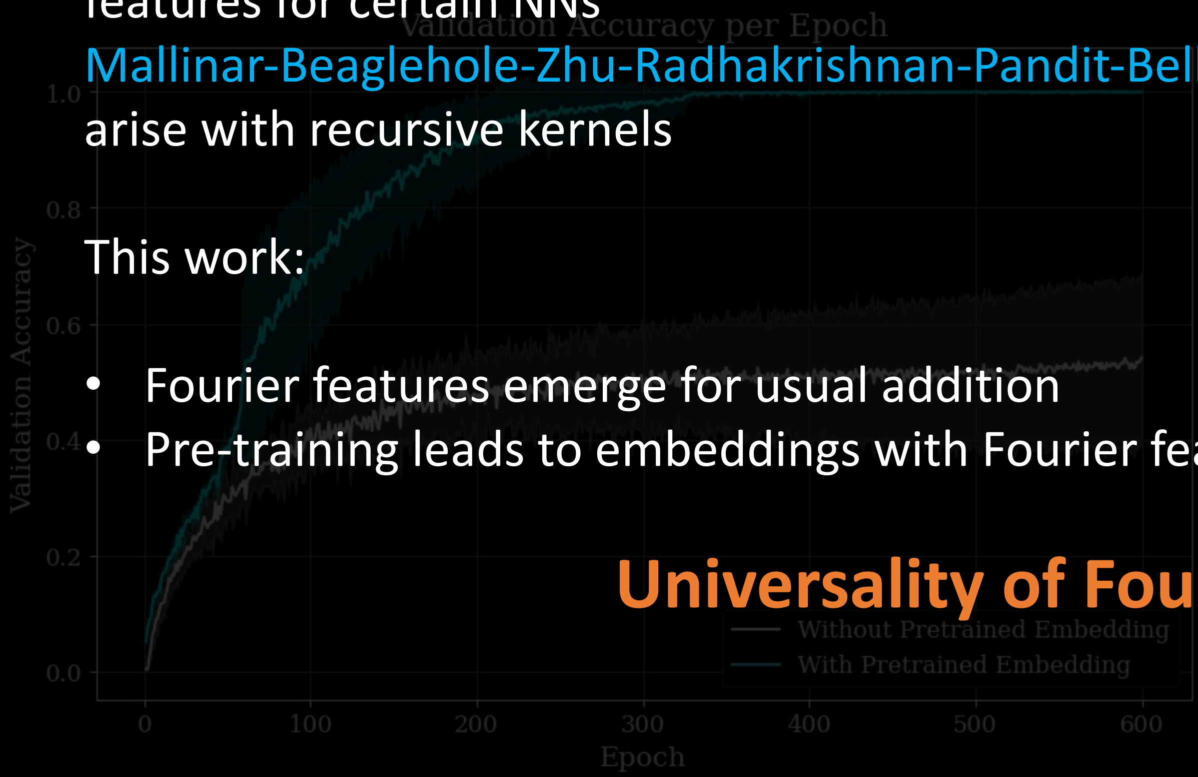
Morwani-Edelman-Oncescu-Zhao-Kakade’2023 proves margin maximization leads to Fourier features for certain NNs

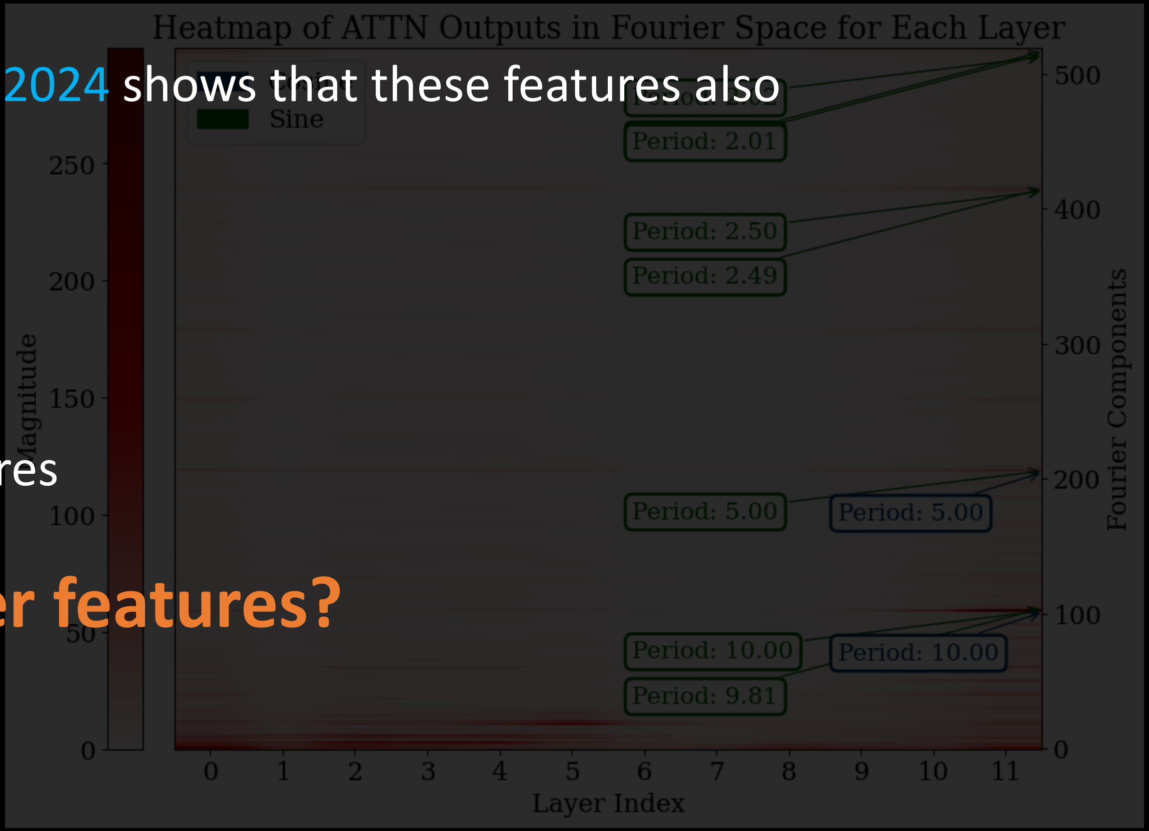
Mallinar-Beaglehole-Zhu-Radhakrishnan-Pandit-Belkin’2024 shows that these features also arise with recursive kernels

This work:

- Fourier features emerge for usual addition
- Pre-training leads to embeddings with Fourier features

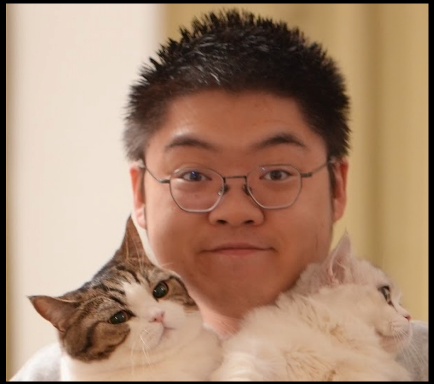
Universality of Fourier features?





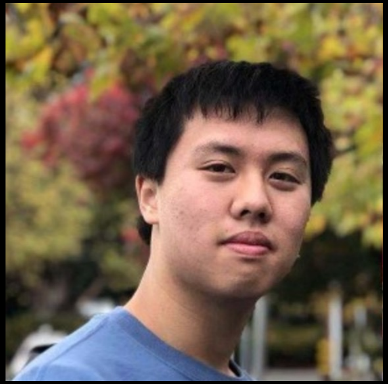
What classes of functions do Transformers prefer to learn?

Bhavya Vasudeva (USC)

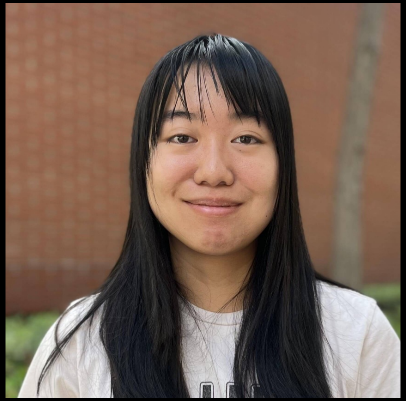


Deqing Fu (USC)

Tianyi Zhou (USC)



Elliot Kau (USC)



You-Qi Huang (USC)

Simplicity Bias of Transformers to Learn Low Sensitivity Functions, arXiv, 2024

Sensitivity from Boolean function analysis

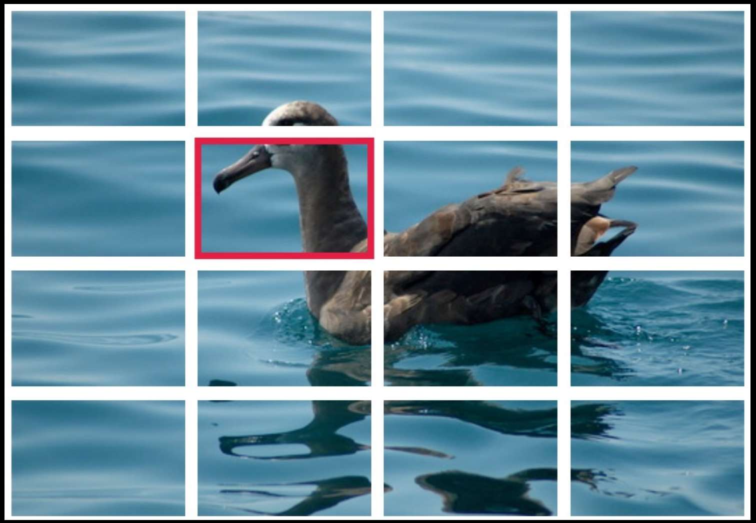
Consider some function f defined on the Boolean hypercube H_d

$$Sensitivity(f) = \mathbb{E}_{x\sim H_d}\left[\frac{1}{d}\sum_{i=1}^{d} \underbrace{\mathbf{1}(f(x) \neq f(x^{\oplus i}))}_{\text{Does flipping the i-th coordinate change the function?}}\right]$$

Related to measures such as degree, noise stability etc.

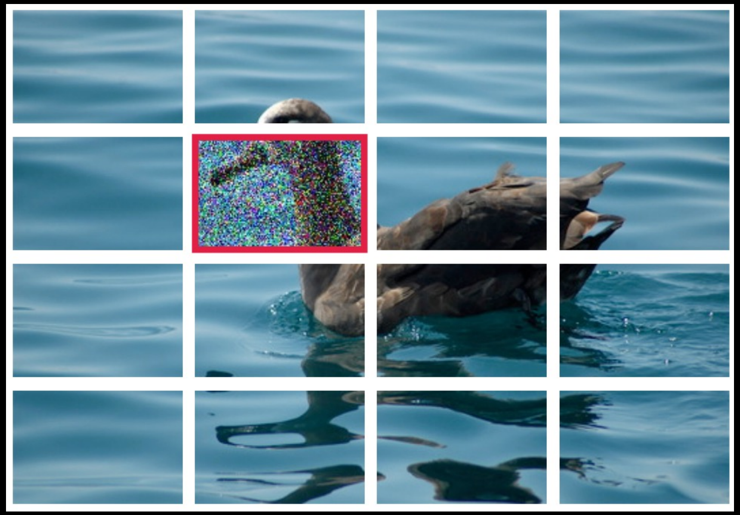
Bhattimishra-Patel-Kanade-Blunsom’23 shows that
Transformers prefer to learn low-sensitivity Boolean functions

Sensitivity beyond Boolean data



Evaluate model on original input

If model's predictions change, model is **sensitive** to that token



Evaluate model on perturbation to random token

Observations: Transformers learn lower sensitivity functions

- **Image** (Fashion MNIST, CIFAR-10, SVHN, ImageNet-1k)
  - For same accuracy, Transformers learn solutions with lower sensitivity than MLPs, CNN, and also other patch-based architectures such as ConvMixer

- **Language** (Paraphrasing tasks: MRPC, QQP)
  - For same accuracy, Transformers learn solutions with lower sensitivity than LSTMs
  - LSTMs are more sensitive to recent tokens, Transformers have more uniform sensitivity across context

- **Advantages of low sensitivity**
  - Adding sensitivity as a regularizer improves robustness
  - Adding sensitivity as a regularizer also leads to flatter minima

Sensitivity as a measure to understand inductive bias?

Can we use Transformers to discover data structures from scratch?

Omar Salemohamed

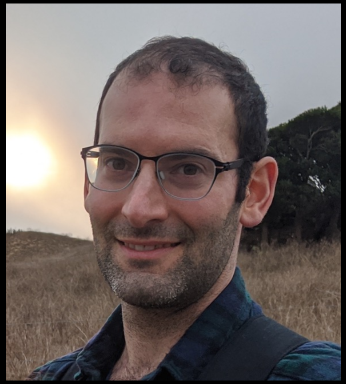
(Universite de Montreal/MILA)

Laurent Charlin

(HEC Montreal/MILA)

Shivam Garg

(MSR NYC)

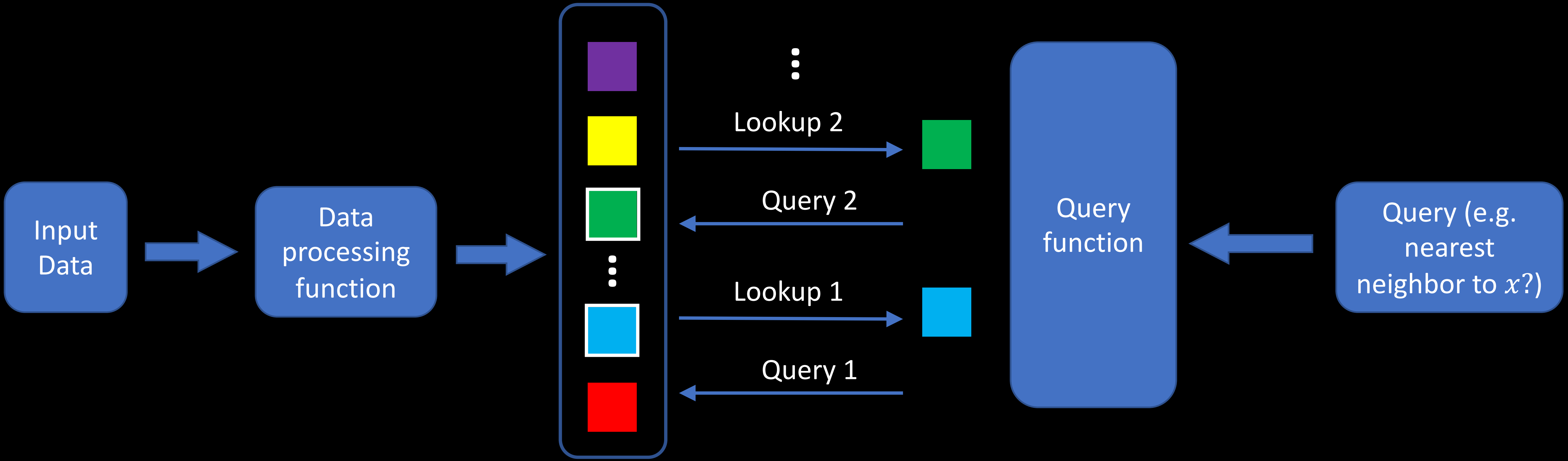


Greg Valiant

(Stanford)

Discovering Data Structures: Nearest Neighbor Search and Beyond, ongoing

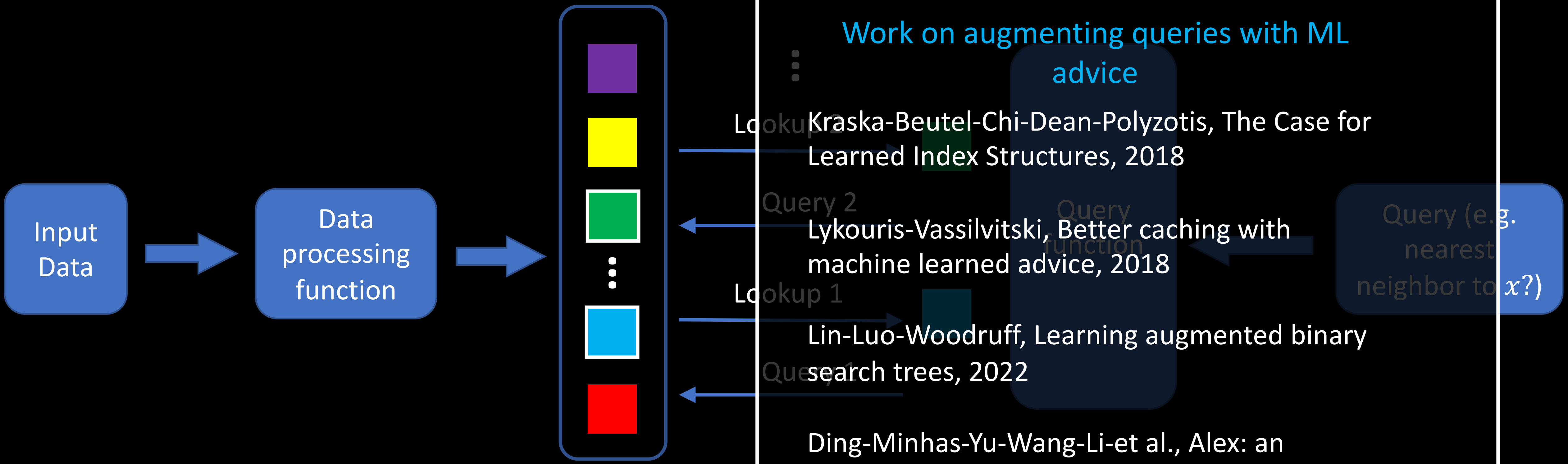
Data structures (think nearest neighbor lookup in 1D)



We will focus on:

- No. of lookups
- Space usage

Recent work has tried to augment data structures with ML



We will focus on:

- No. of lookups
- Space usage

Work on augmenting queries with ML advice

Kraska-Beutel-Chi-Dean-Polyzotis, The Case for Learned Index Structures, 2018

Lykouris-Vassilvitski, Better caching with machine learned advice, 2018

Lin-Luo-Woodruff, Learning augmented binary search trees, 2022

Ding-Minhas-Yu-Wang-Li-et al., Alex: an updatable adaptive learned index., 2020

…

Recent work has tried to augment data structures with ML

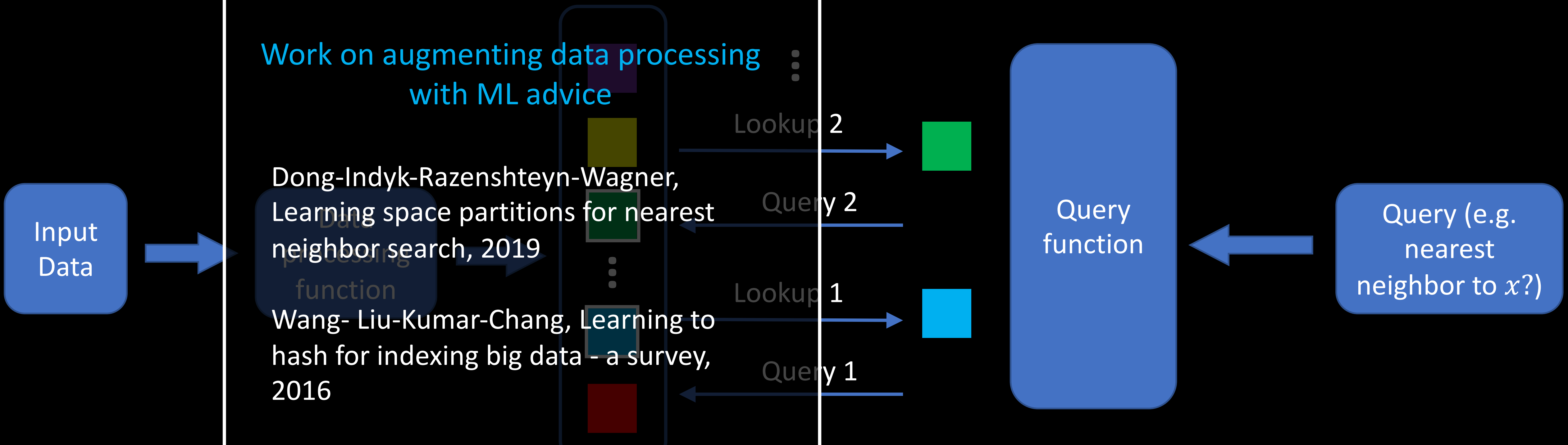
Work on augmenting data processing with ML advice

Dong-Indyk-Razenshteyn-Wagner, Learning space partitions for nearest neighbor search, 2019

Wang- Liu-Kumar-Chang, Learning to hash for indexing big data - a survey, 2016

Sabek-Vaidya-Horn-Kipf-Kraska, When Are Learned Models Better Than Hash Functions?, 2021

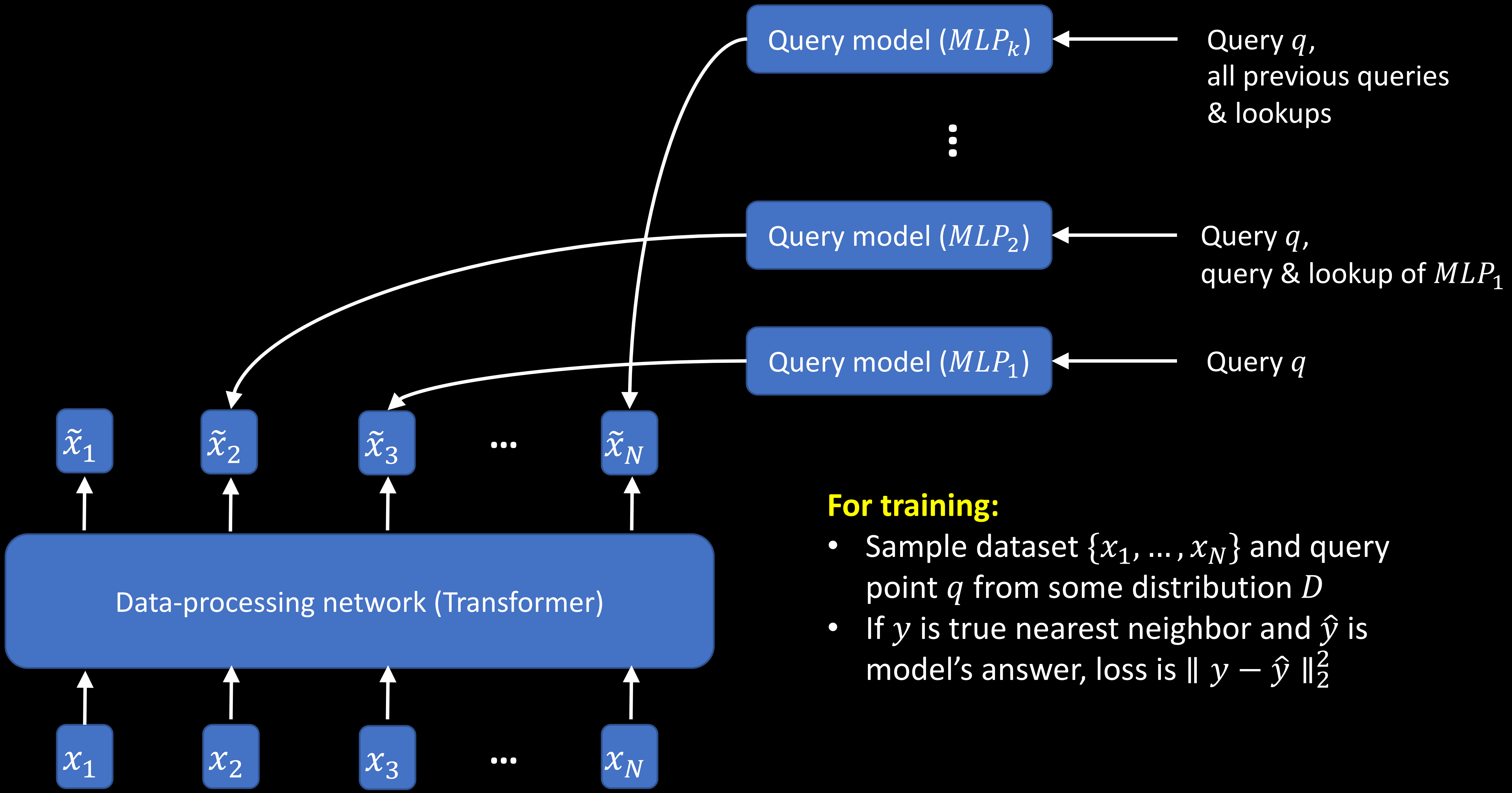
…



We will focus on:

- No. of lookups
- Space usage

What if we learn everything end to end with ML, with no algorithmic priors?



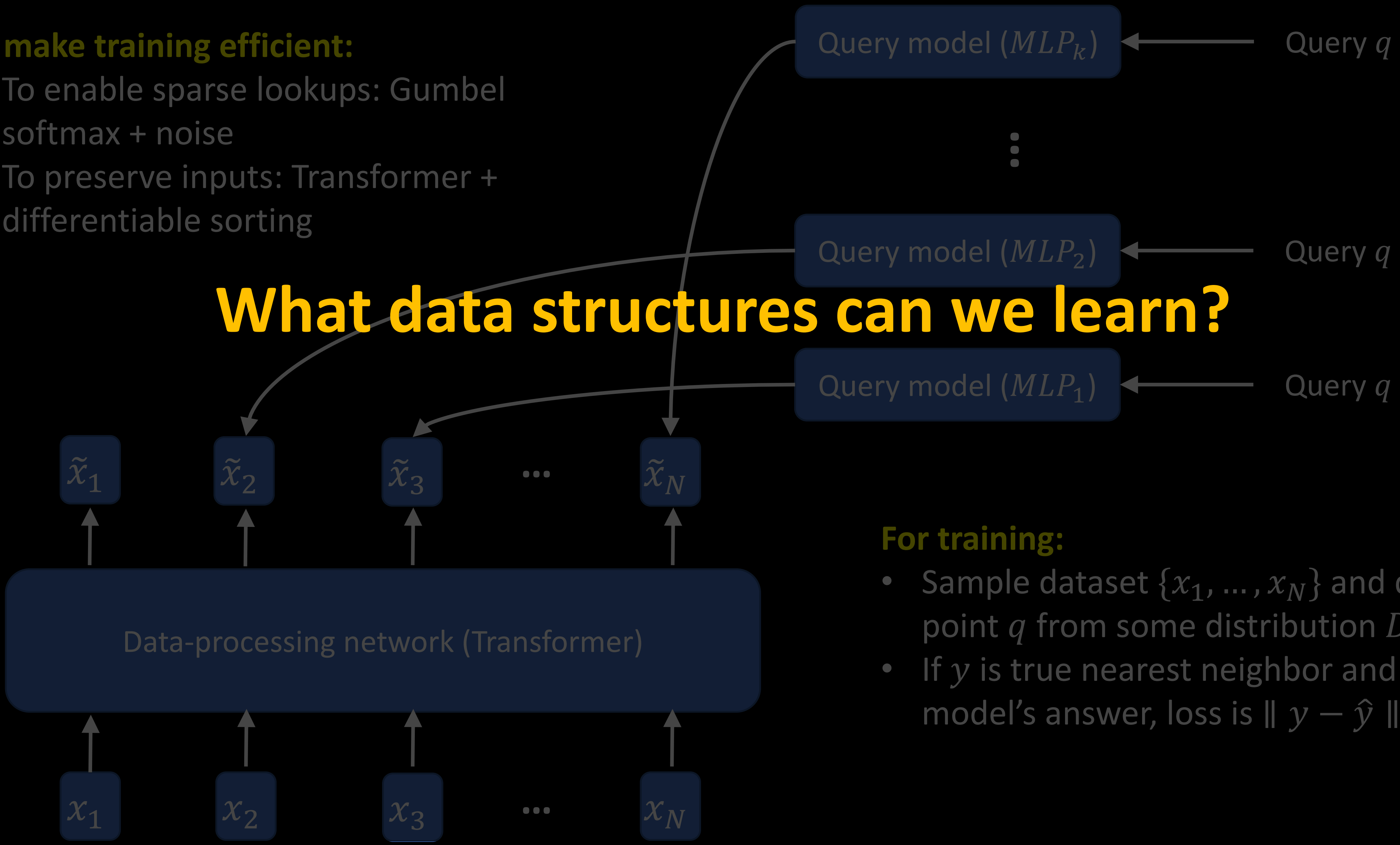
For training:

- Sample dataset $\{x_1, \ldots, x_N\}$ and query point q from some distribution D
- If y is true nearest neighbor and $\hat{y}$ is model's answer, loss is $\| y - \hat{y} \|_2^2$

What if we learn everything end to end with ML, with no algorithmic priors?

To make training efficient:

- To enable sparse lookups: Gumbel softmax + noise
- To preserve inputs: Transformer + differentiable sorting

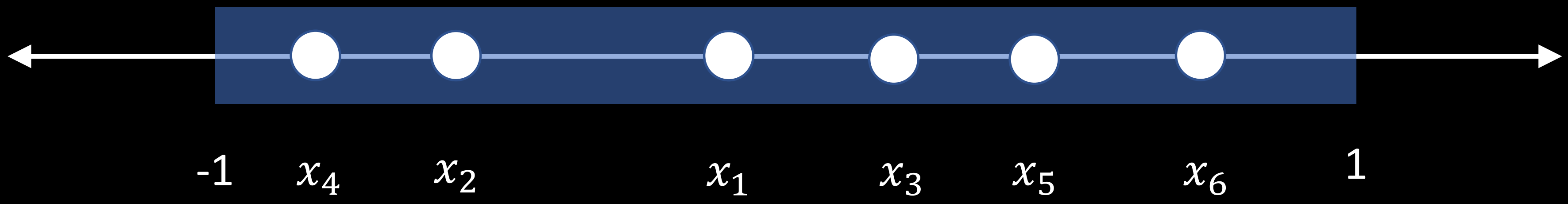


What data structures can we learn?

For training:

- Sample dataset $\{x_1, \dots, x_N\}$ and query point q from some distribution D
- If y is true nearest neighbor and $\hat{y}$ is model's answer, loss is $\| y - \hat{y} \|_2^2$

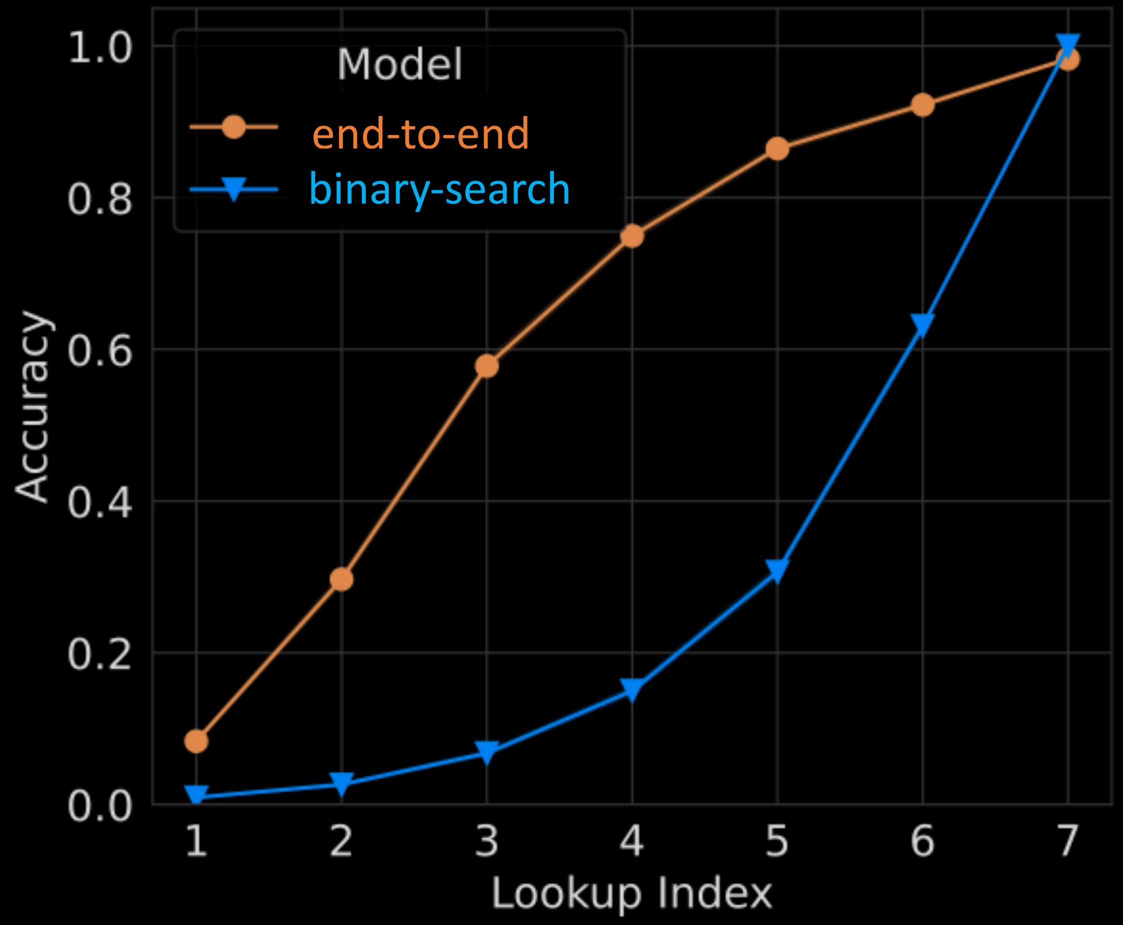
Uniform distribution in 1D

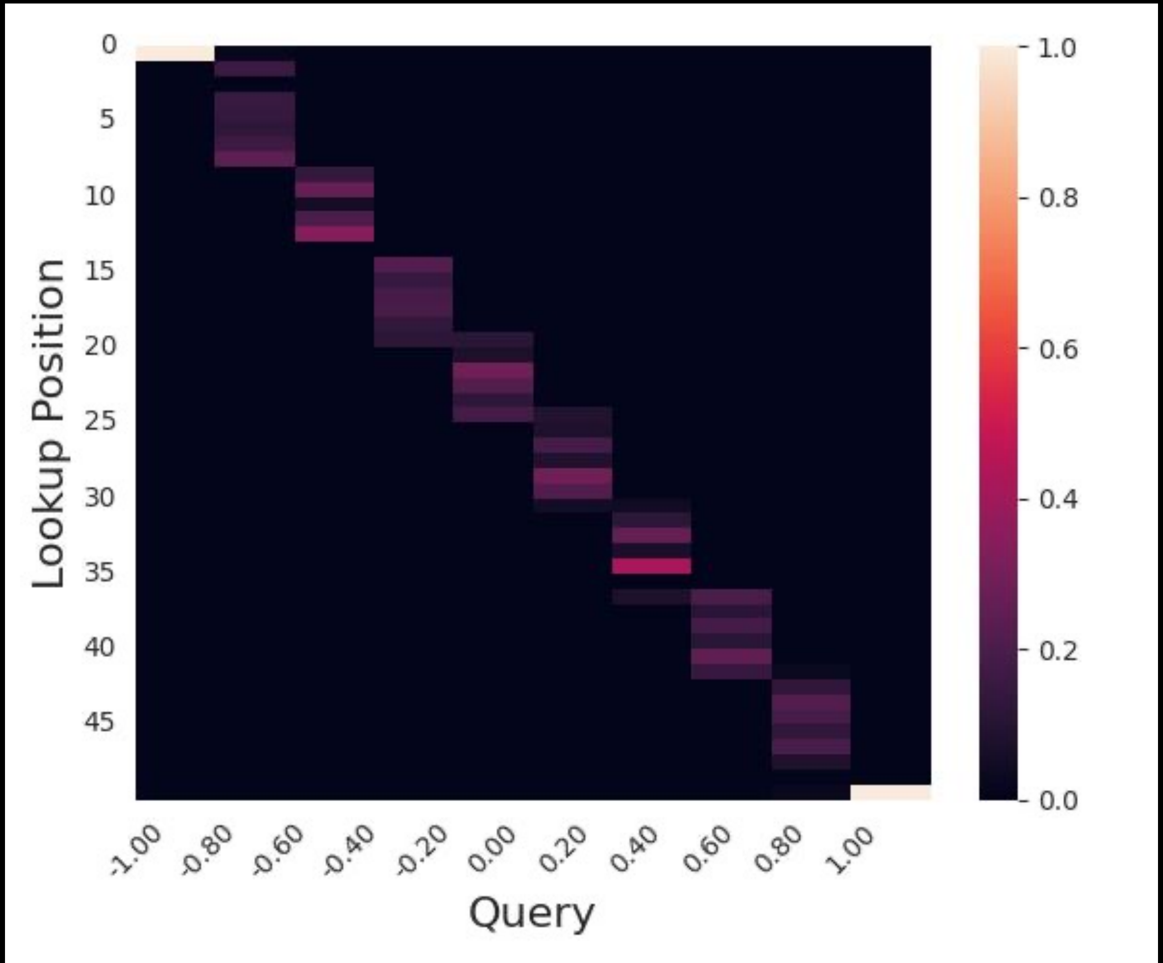


Model trained on this distribution:

- **Learns to sort, with small error**
- **Does better than binary search**

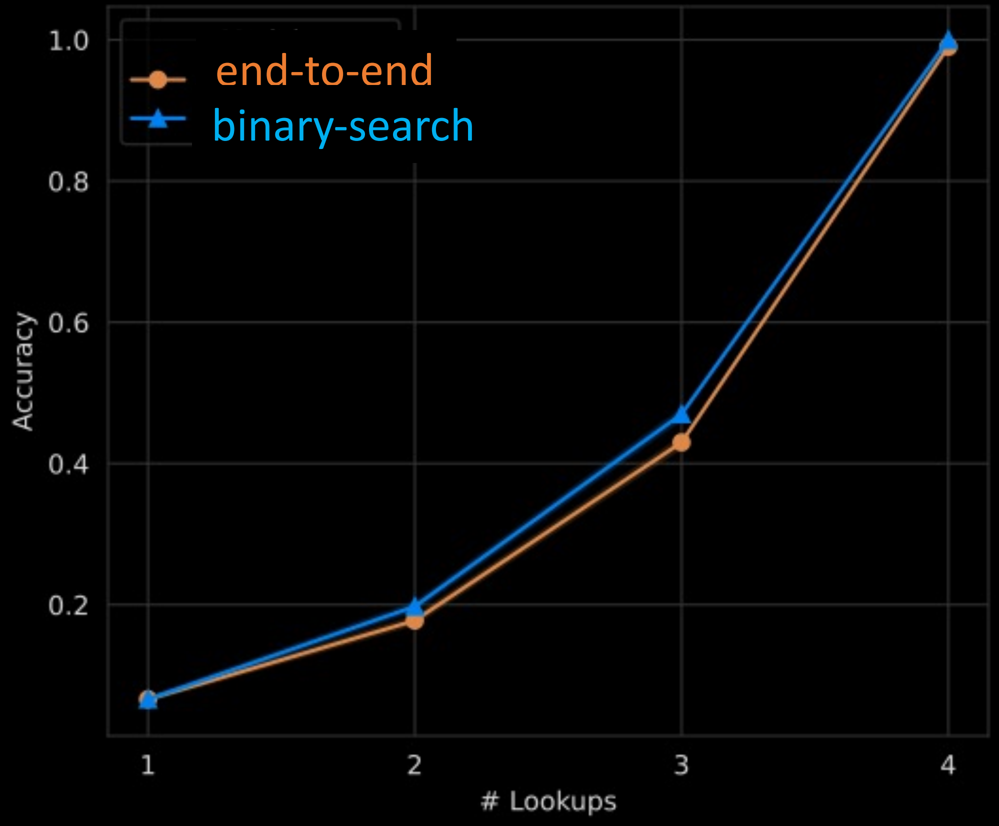
Model outperforms binary search

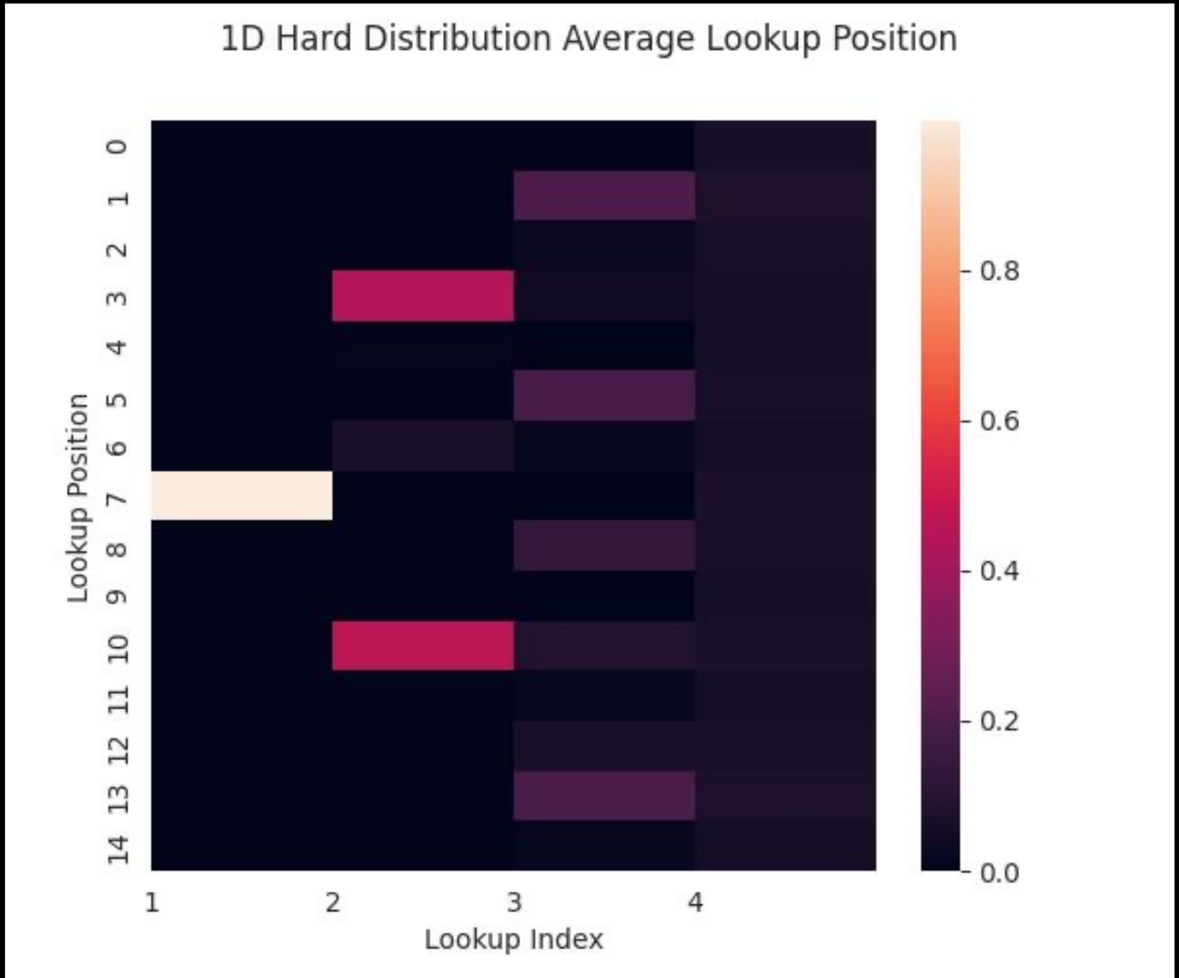




Query model begins search not far from nearest neighbor

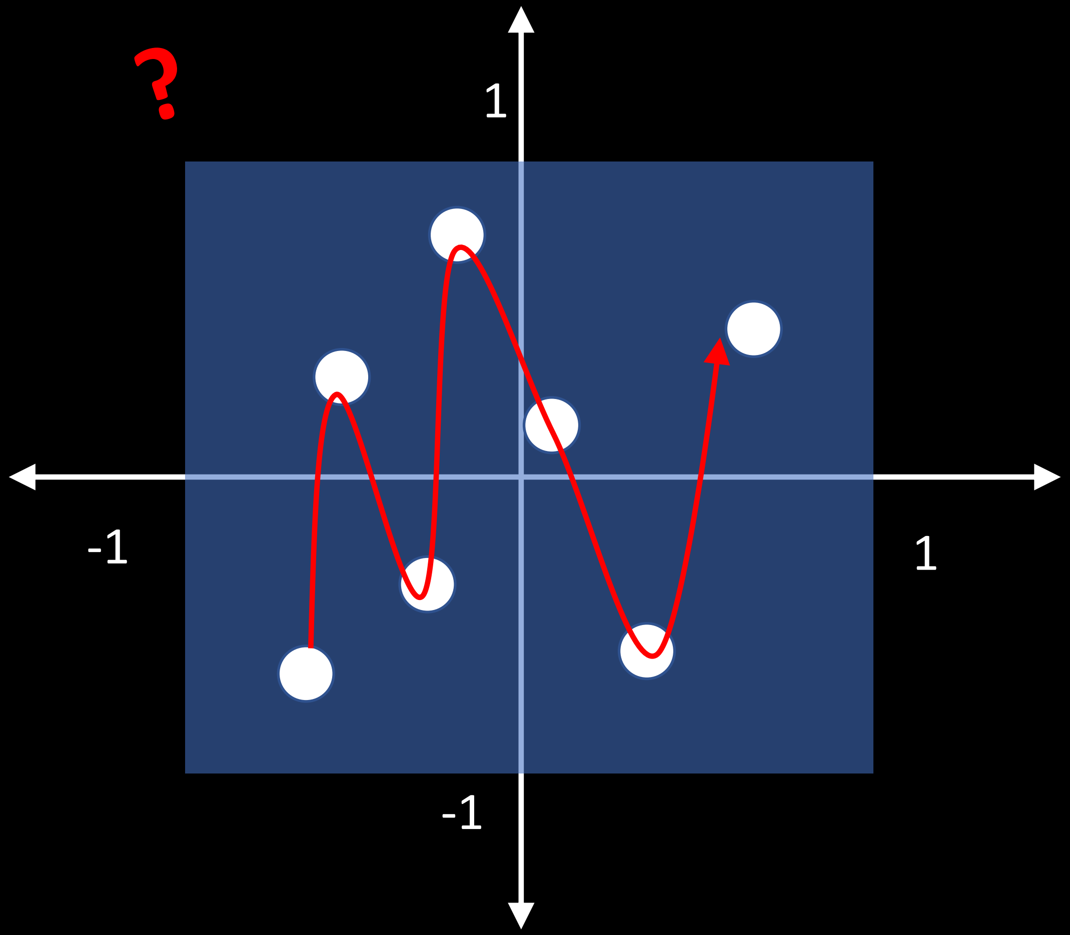
Harder 1D distribution where quantiles don’t concentrate



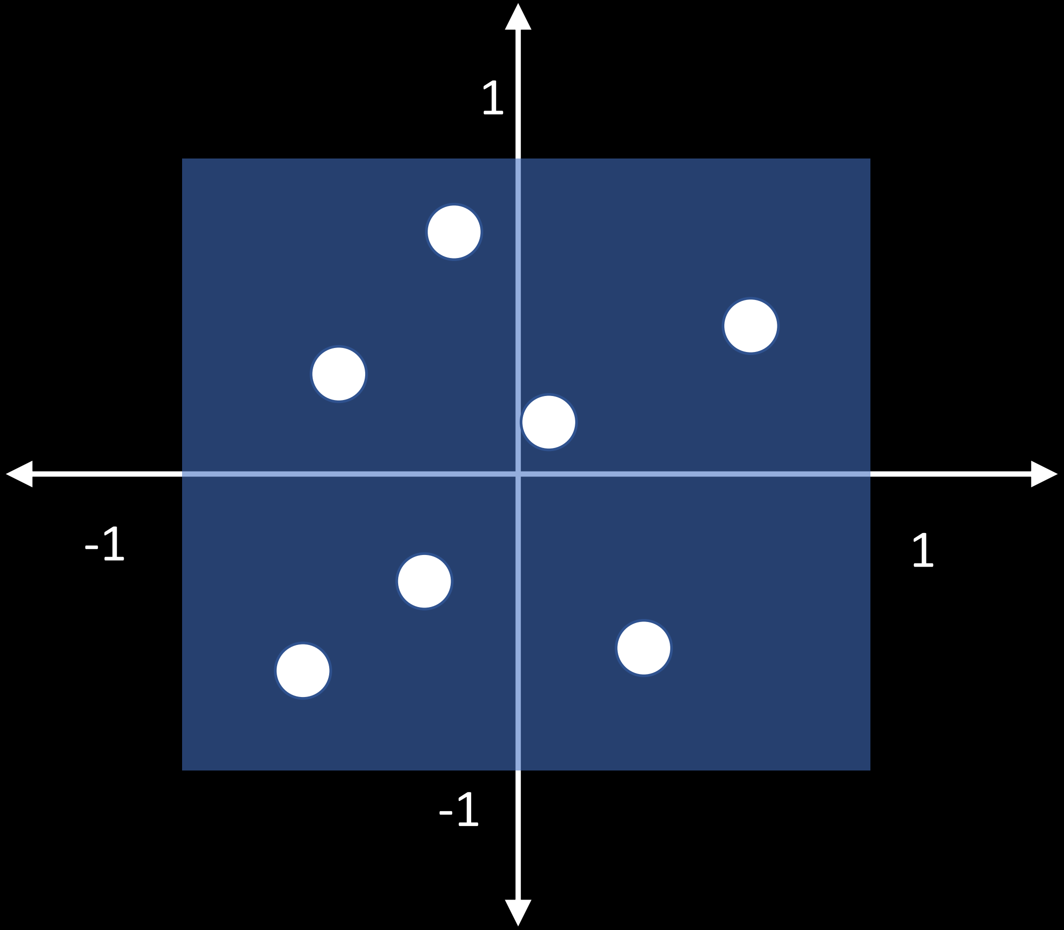


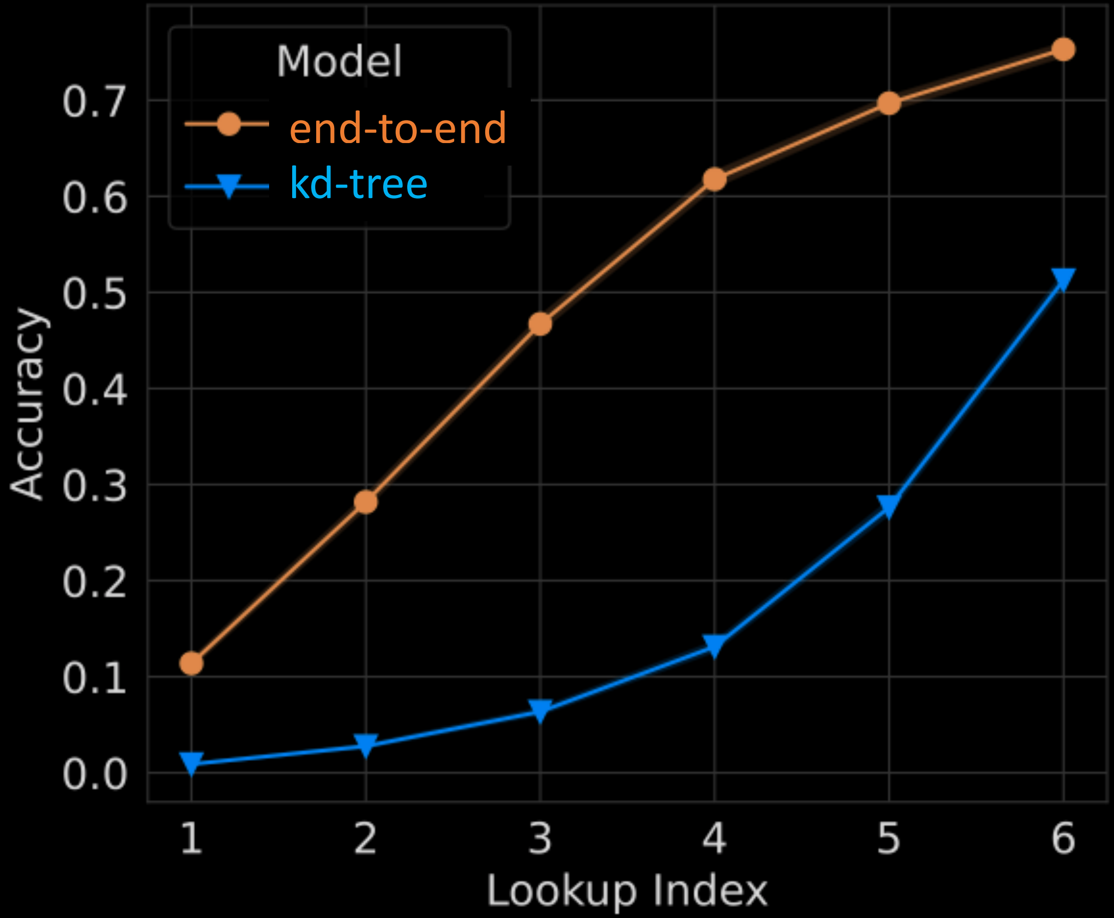
Model learns binary search!

Uniform distribution in 2D: What is the right permutation?

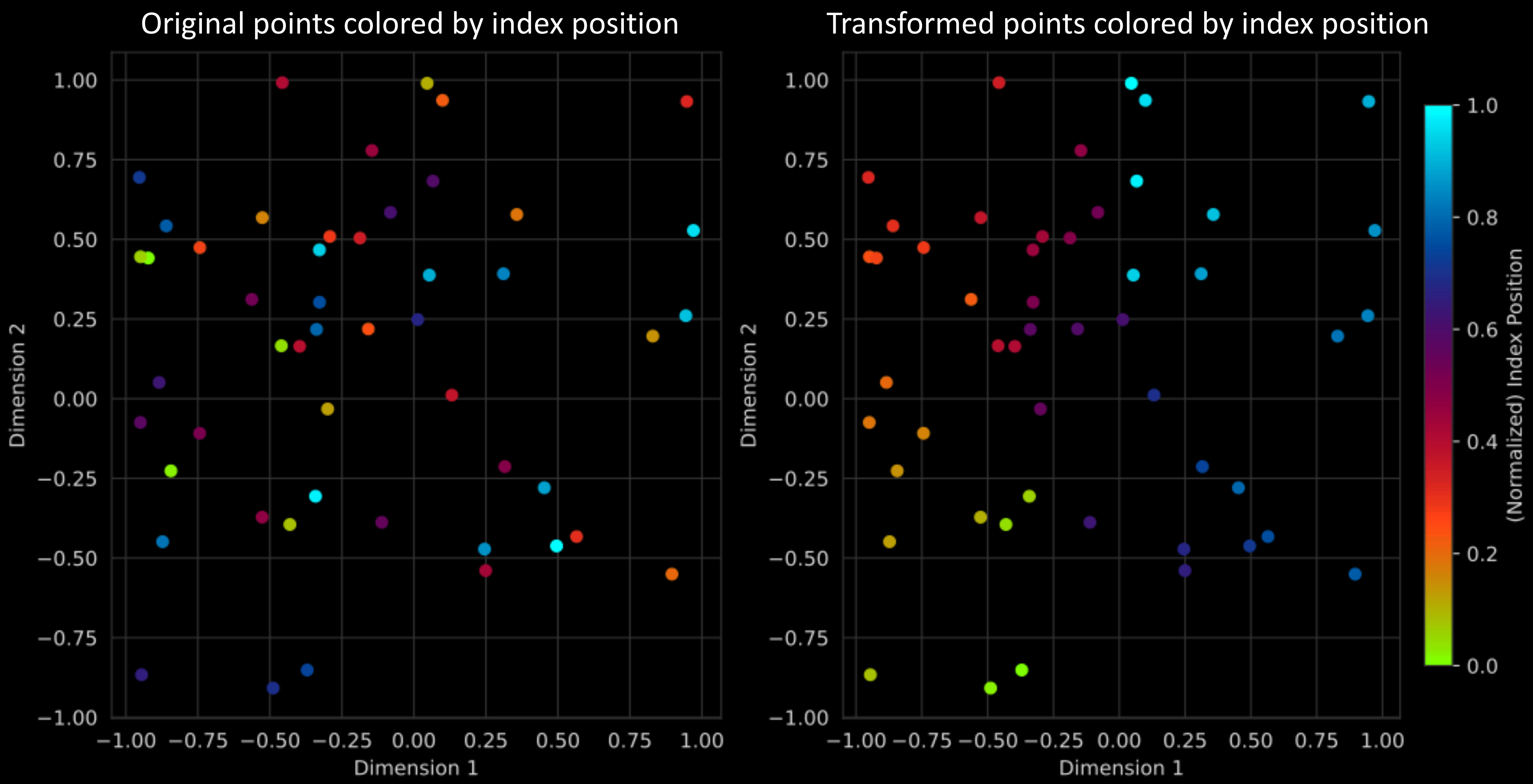


Uniform distribution in 2D: Outperforms kd-trees

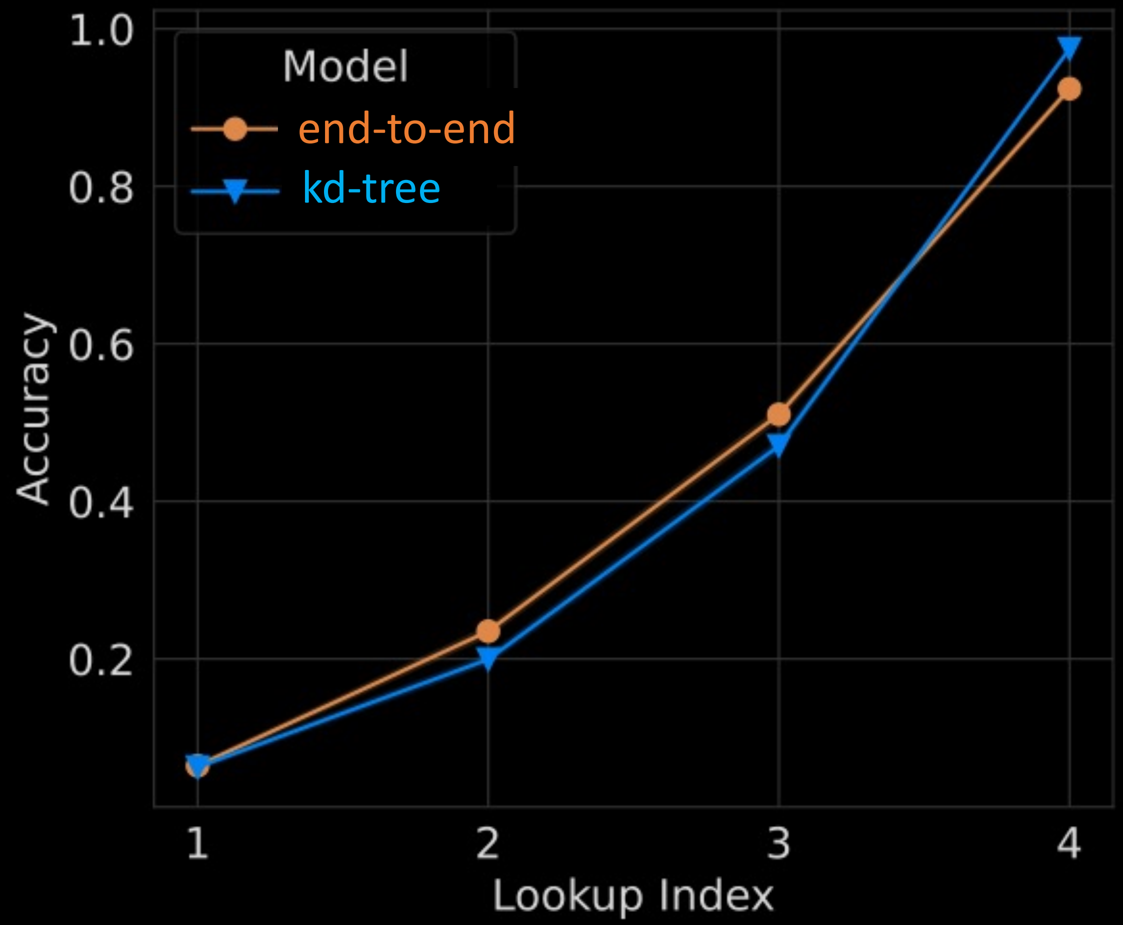




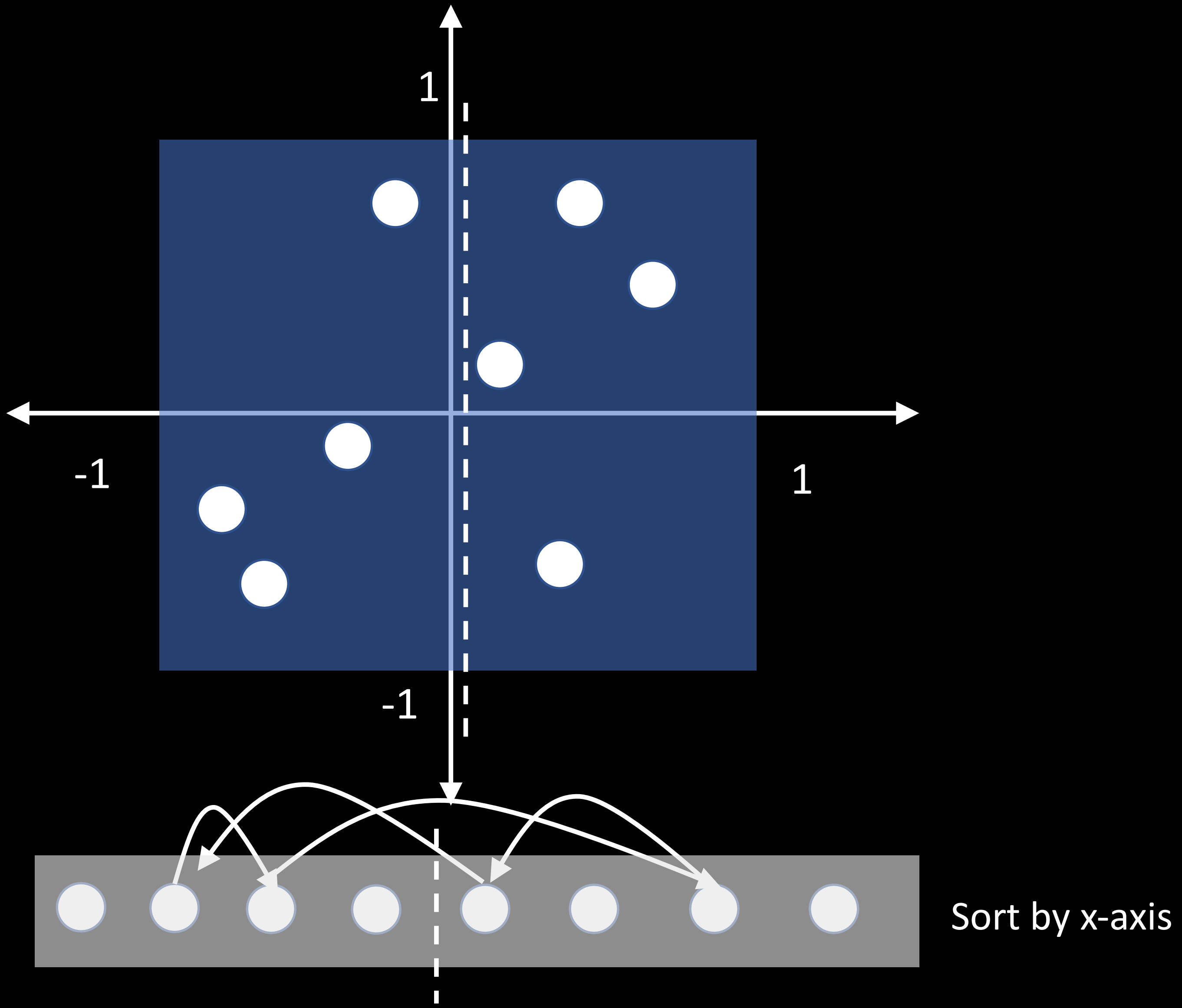
Model learns to index nearby points together



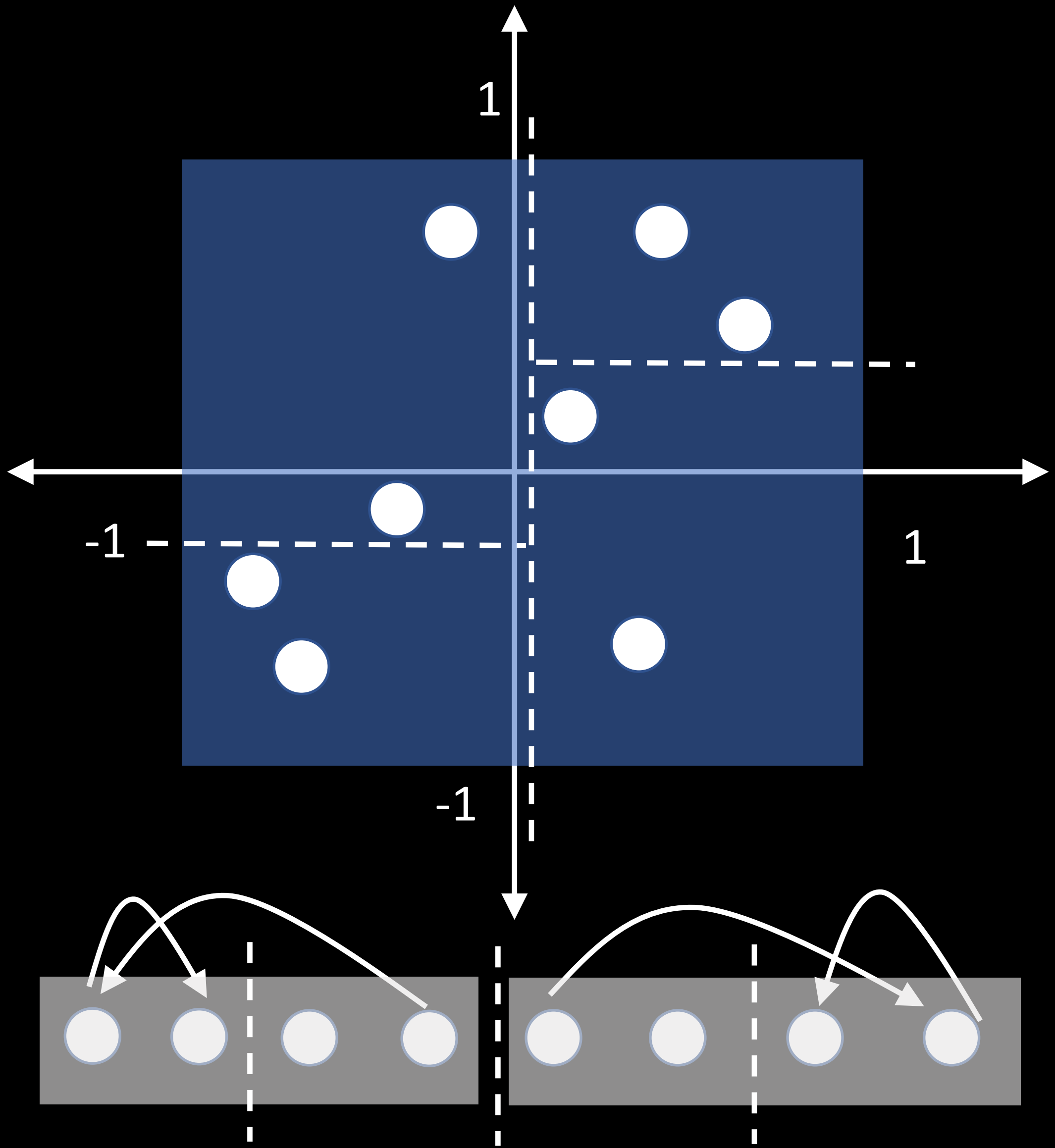
Hard distribution in 2D: Matches kd-trees



Can see that the model is essentially recovering a kd-tree!

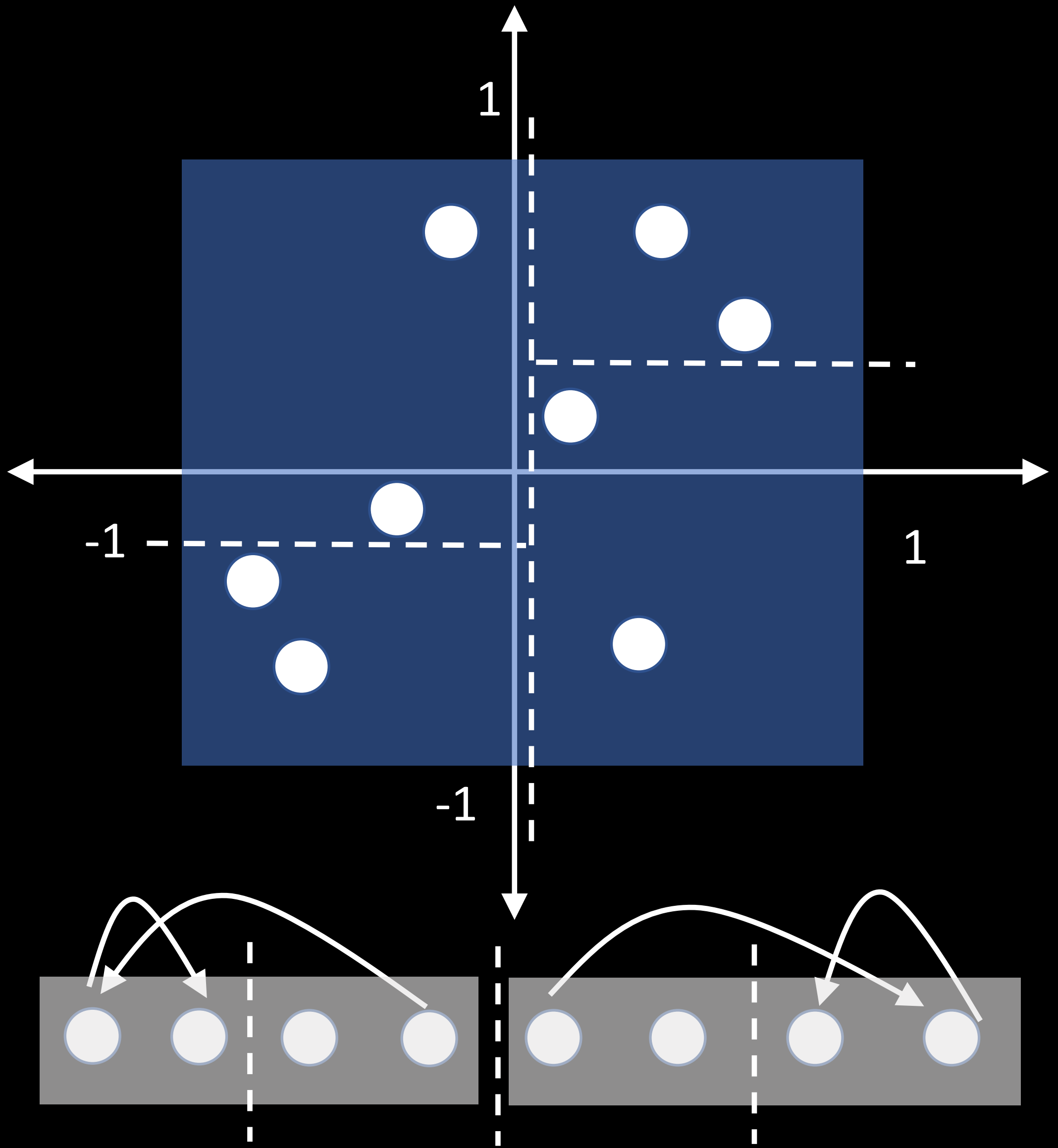


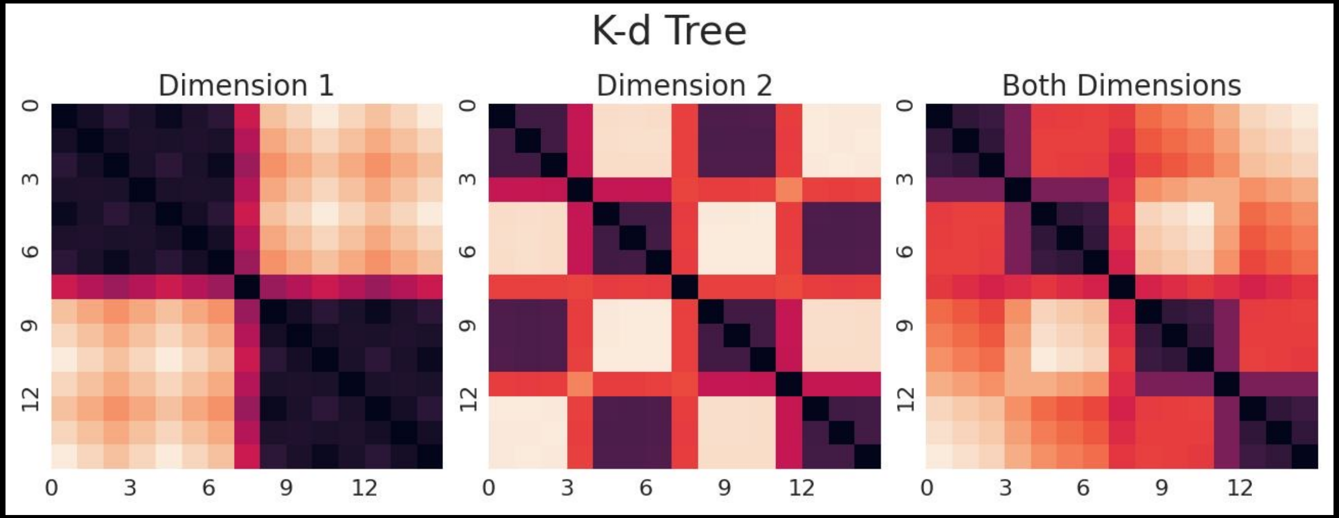
Can see that the model is essentially recovering a kd-tree!

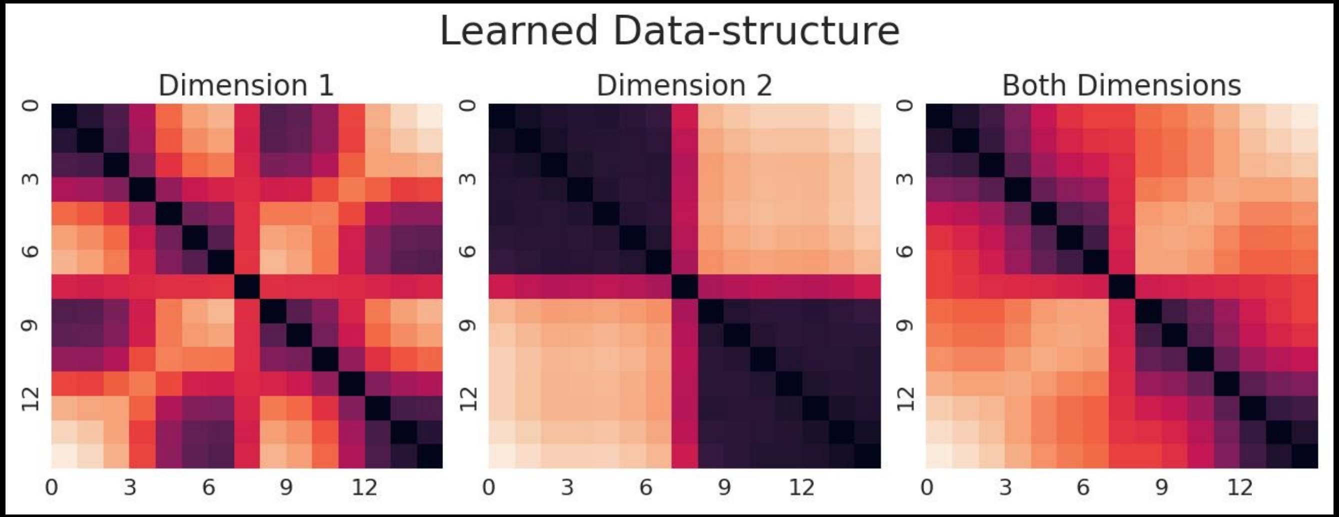


Sort each half
by y-axis

Can see that the model is essentially recovering a kd-tree!




K-d Tree

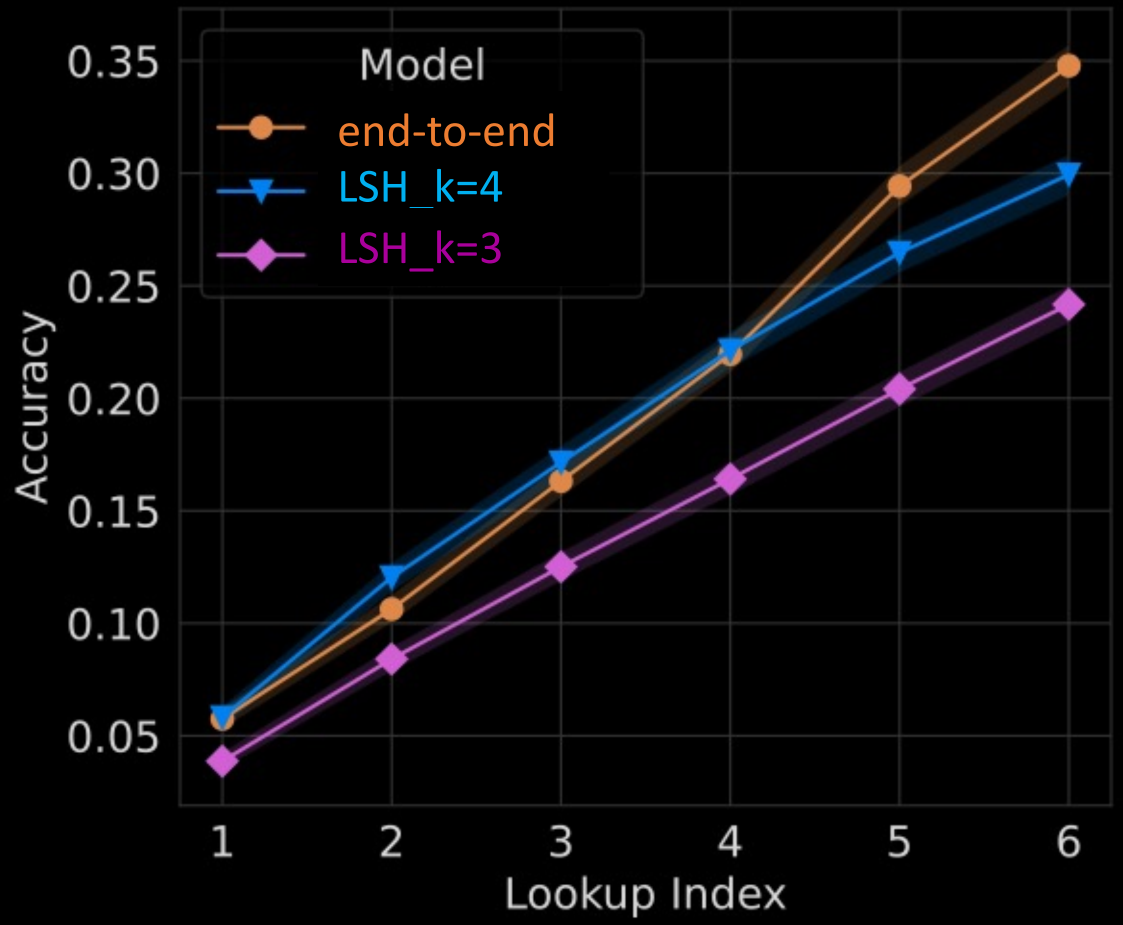

Learned Data-structure

Uniform distribution in 30D: Matches LSH

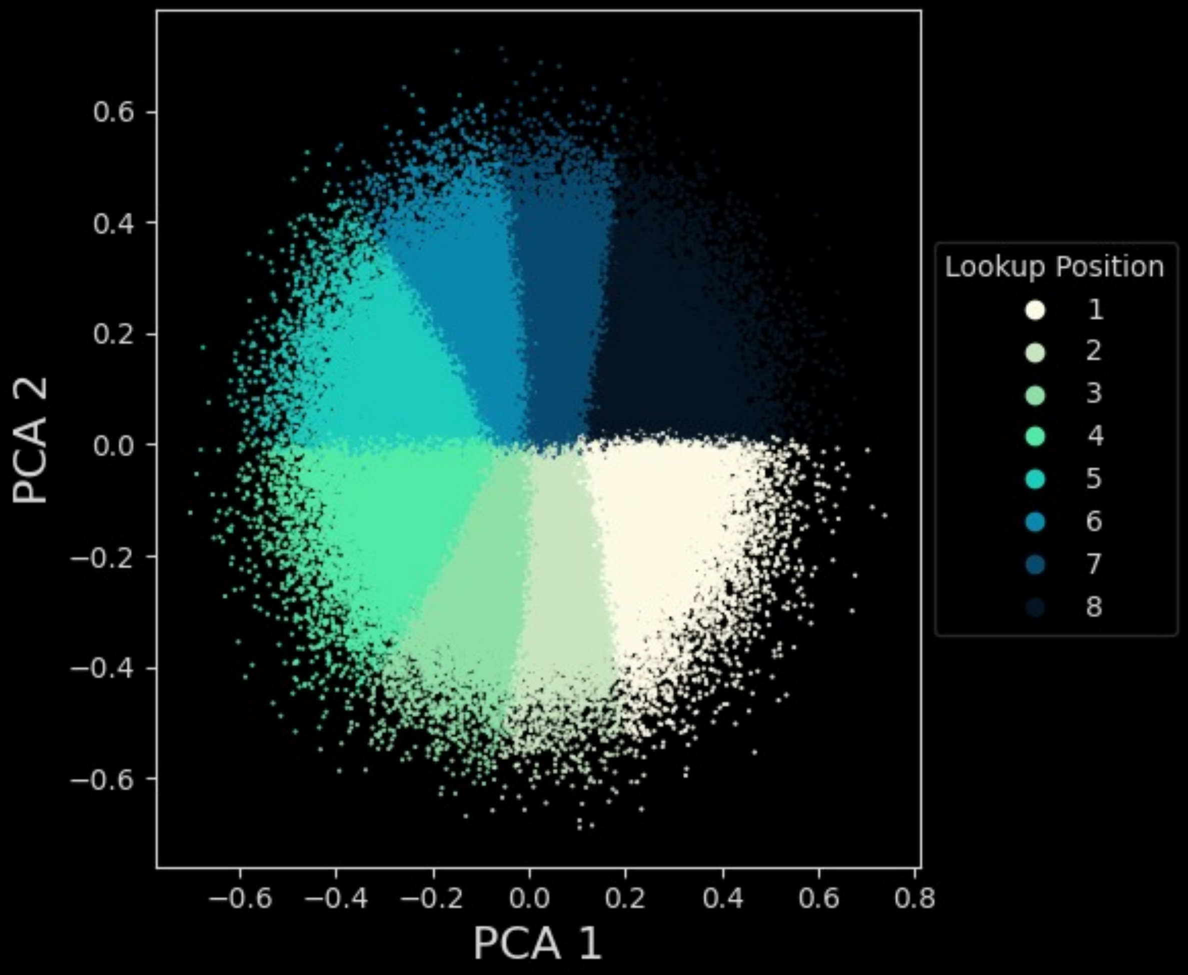
- In high dimensions (even 30), we don’t understand optimal data structures, even for the uniform distribution!

- Kd-trees suffer from curse of dimensionality

- LSH is a popular alternative



Model learns to do a projection, like LSH



Query model mainly considers projection of query onto this 2-dimensional subspace to decide where to look

Model can learn underlying metric space

Input: 50 images of numbers uniformly drawn from [0,200]

x_1

x_2

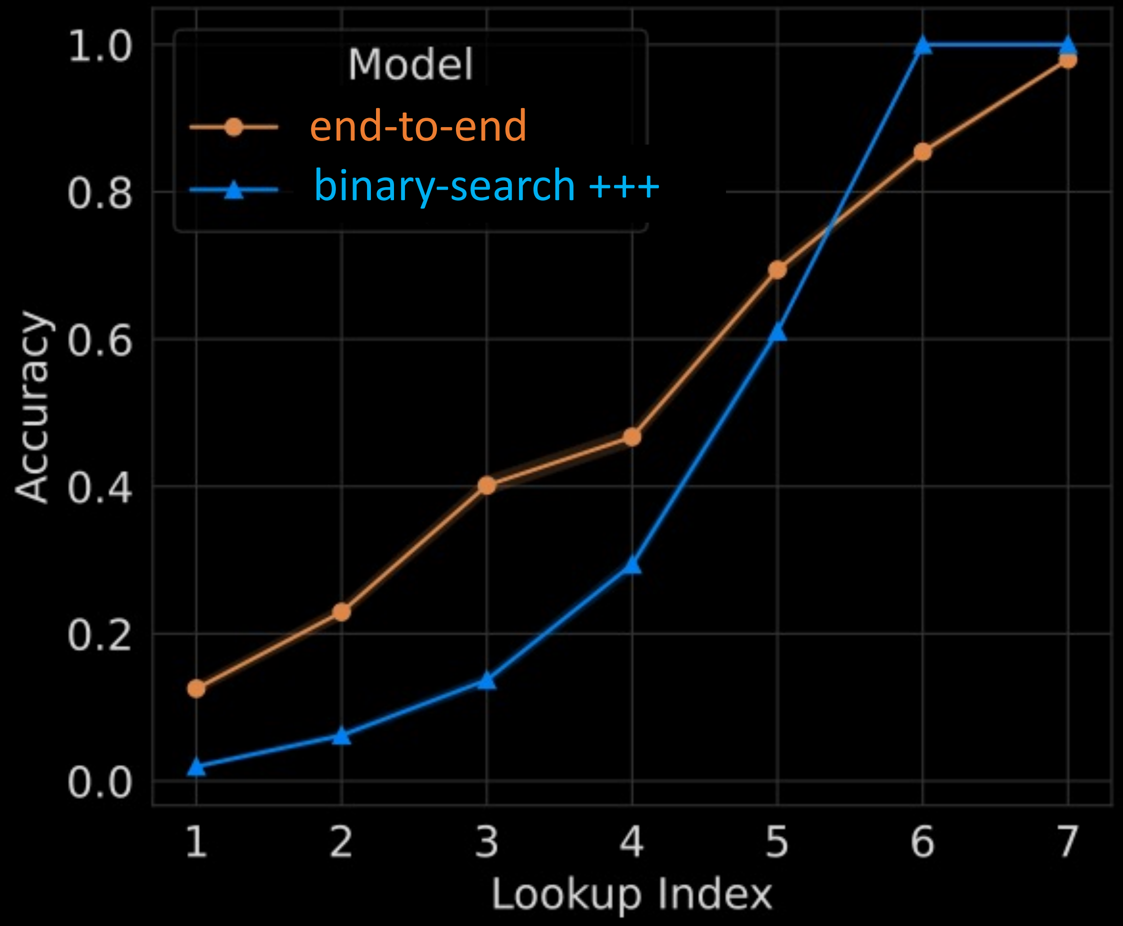
…

x_N

Query: Images of numbers uniformly drawn from [0,200]

x_q

- Train on cross-entropy loss of prediction
- Model gets no access to the labelling of the image as a number



Summary: Claims & Thoughts

We can train models end to end to learn data structures

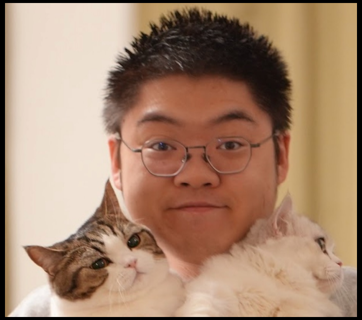
- Model also learns to use extra space
- We also show we can learn data structures for frequency estimation in a data stream, recovering/outperforming count-sketch

Models outperform data-independent baselines

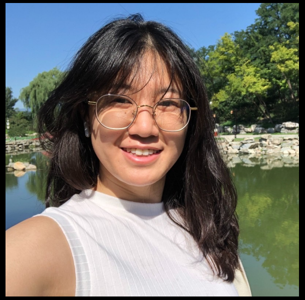
- Also consider settings with power-law distributions etc.

Learned models can be interpreted and understood, providing insights for data-structure design

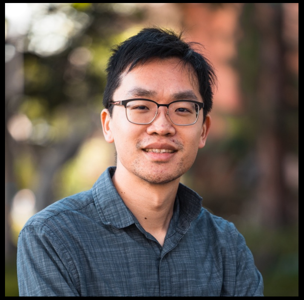
- ***Can we use these to understand tradeoffs in theory, build better strategies for high-dimensional NN search and other data structure problems?***



Deqing Fu



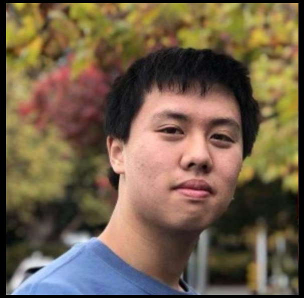
Tianqi Chen



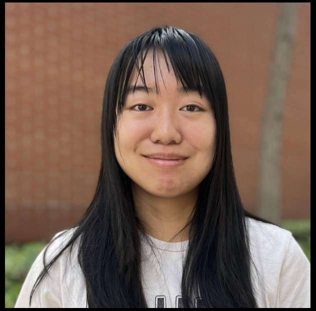
Robin Jia

Tianyi Zhou

Bhavya Vasudeva



Elliot Kau

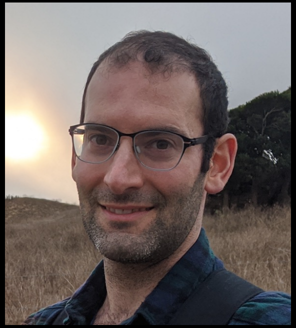


You-Qi Huang

Omar Salemohamed

Laurent Charlin

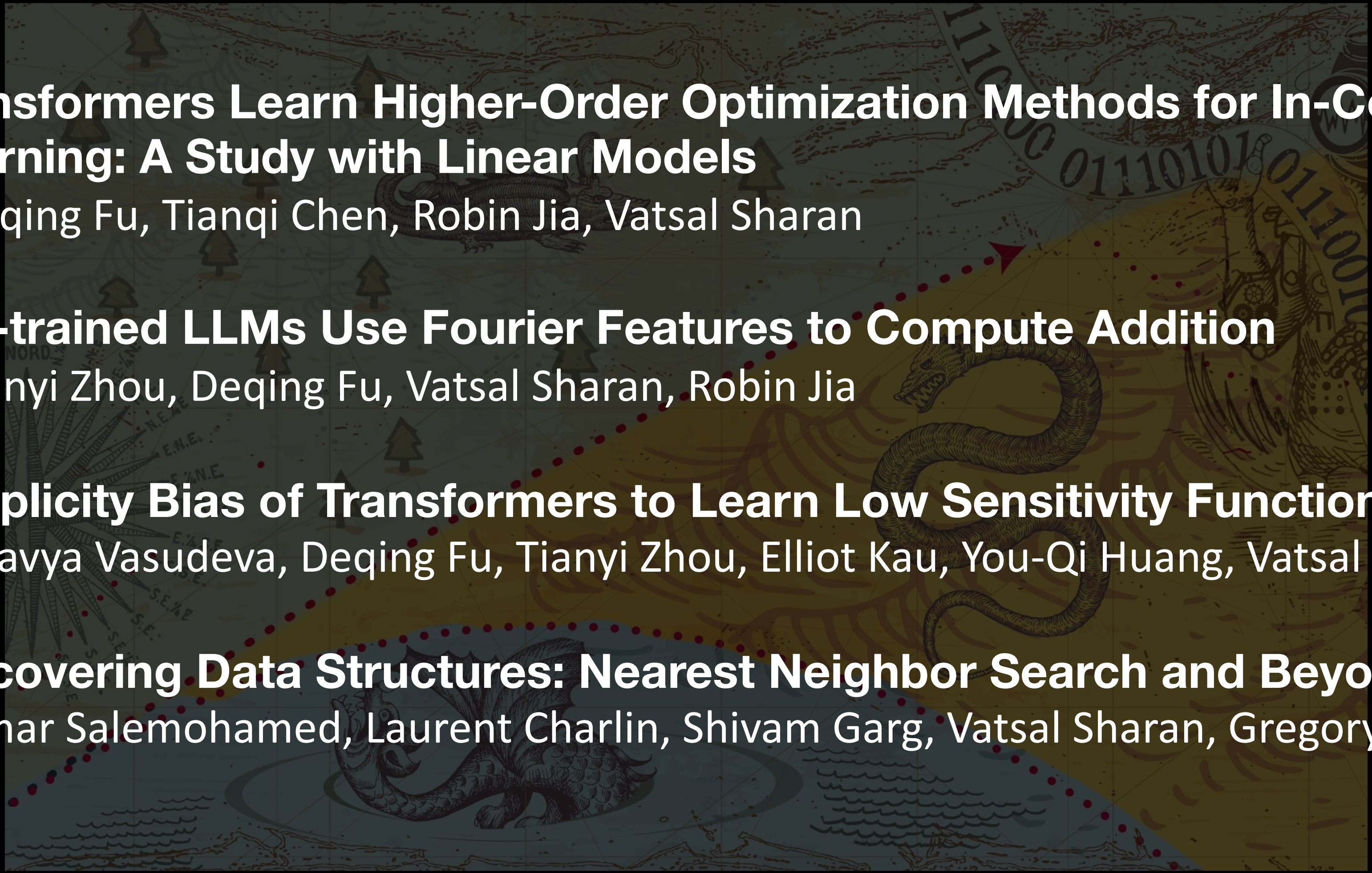
Shivam Garg



Greg Valiant

Thanks!

- How can we use understanding of computational and information theoretic landscape to understand Transformers?
- How can we use Transformers to understand and discover algorithms and data structures?



- **Transformers Learn Higher-Order Optimization Methods for In-Context Learning: A Study with Linear Models**
  Deqing Fu, Tianqi Chen, Robin Jia, Vatsal Sharan

- **Pre-trained LLMs Use Fourier Features to Compute Addition**
  Tianyi Zhou, Deqing Fu, Vatsal Sharan, Robin Jia

- **Simplicity Bias of Transformers to Learn Low Sensitivity Functions**
  Bhavya Vasudeva, Deqing Fu, Tianyi Zhou, Elliot Kau, You-Qi Huang, Vatsal Sharan

- **Discovering Data Structures: Nearest Neighbor Search and Beyond**
  Omar Salemohamed, Laurent Charlin, Shivam Garg, Vatsal Sharan, Gregory Valiant