

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

Vatsal Sharan (USC)

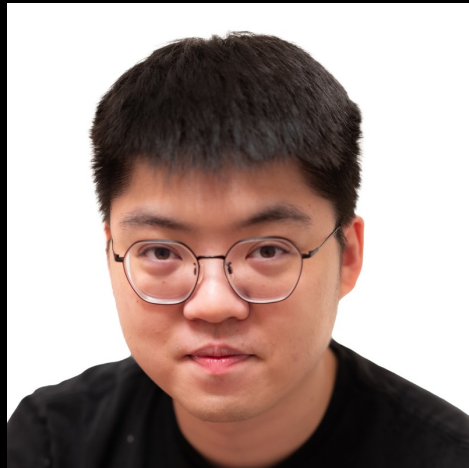




Image source: Simons program on “Computational Complexity of Statistical Inference”

- 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?

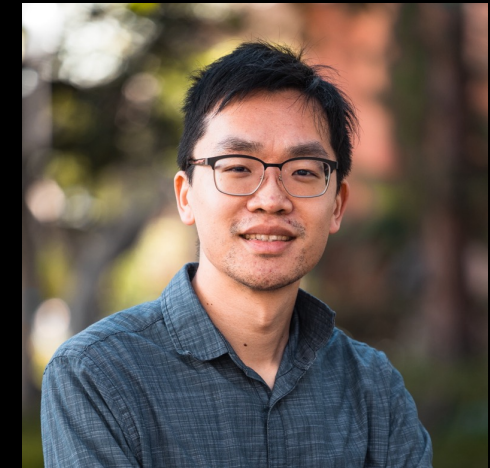
How do Transformers do linear regression?



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 =>
```

In-context learning

examples

VS.

prompt

1 sea otter => loutre de mer *example #1*



gradient update



1 peppermint => menthe poivrée *example #2*



gradient update



1 plush giraffe => girafe peluche *example #N*

gradient update

1 cheese => *prompt*

Usual fine-tuning

How do Transformers do in-context learning?

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

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

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

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

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

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

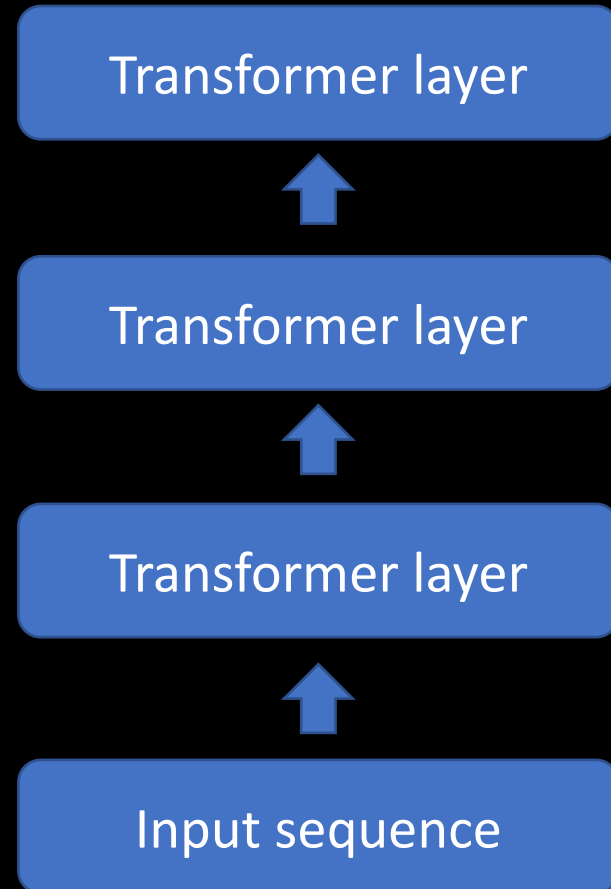
Linear models:

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

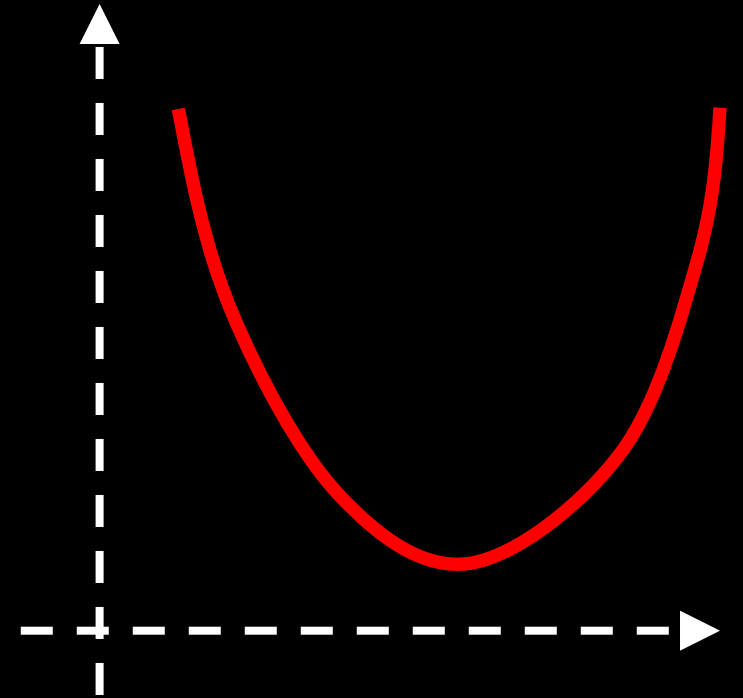
$$x_2 = (-2, 2), y_2 = 8$$

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

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



\approx



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

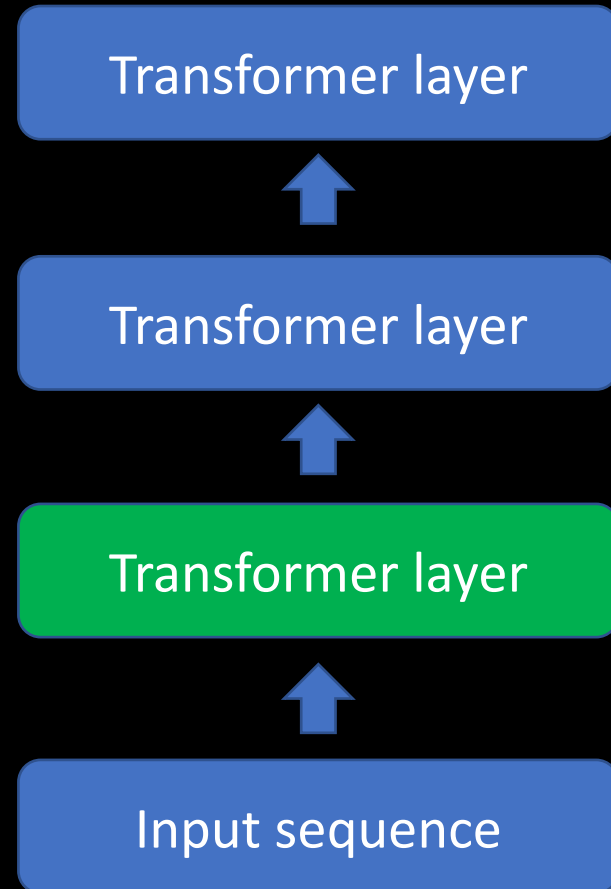
Linear models:

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

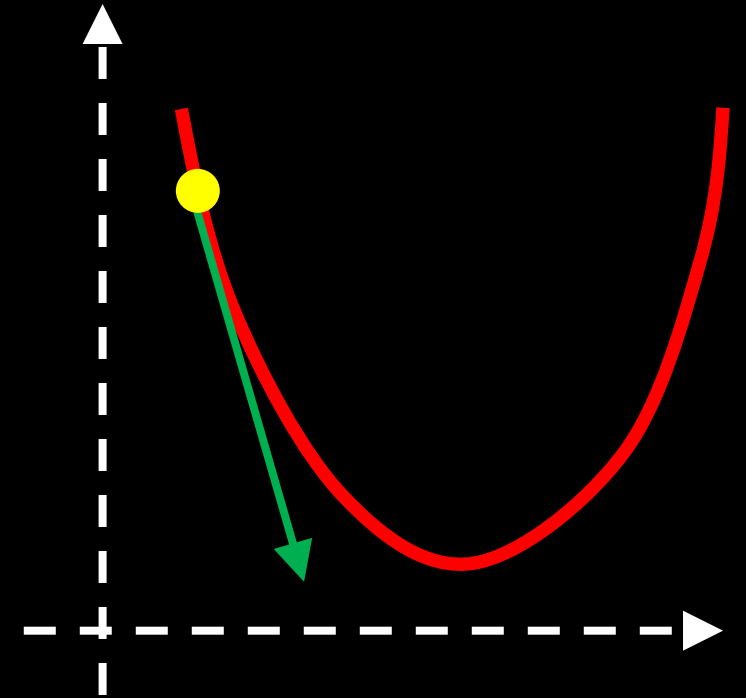
$$x_2 = (-2, 2), y_2 = 8$$

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

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



\approx



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

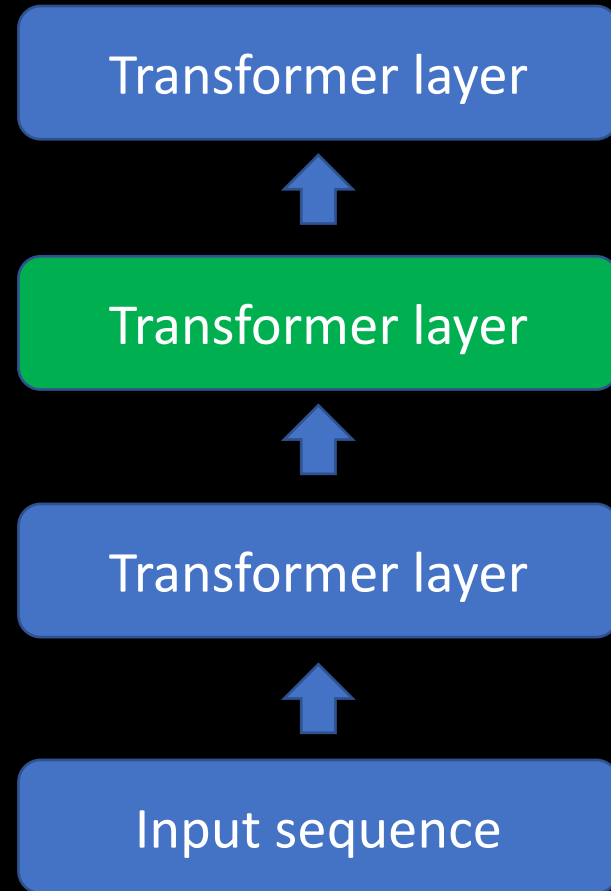
Linear models:

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

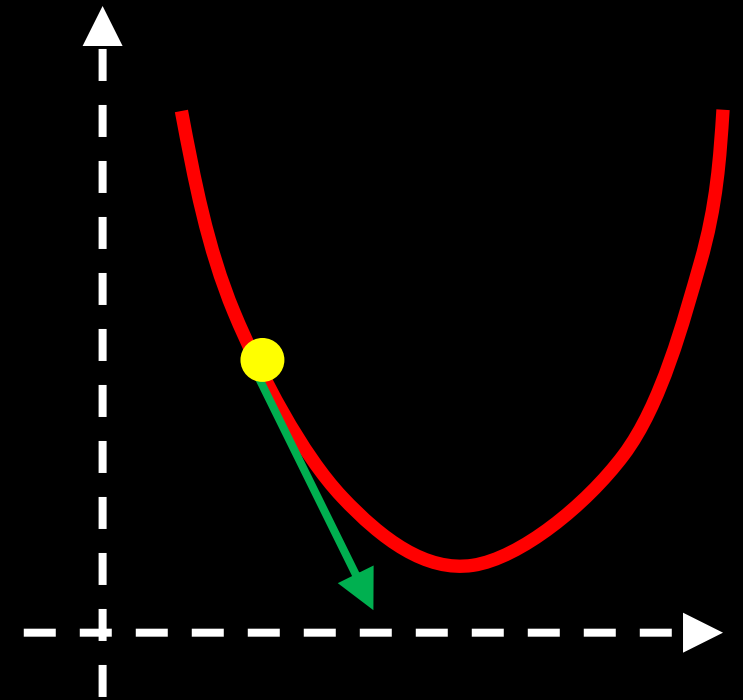
$$x_2 = (-2, 2), y_2 = 8$$

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

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



\approx



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

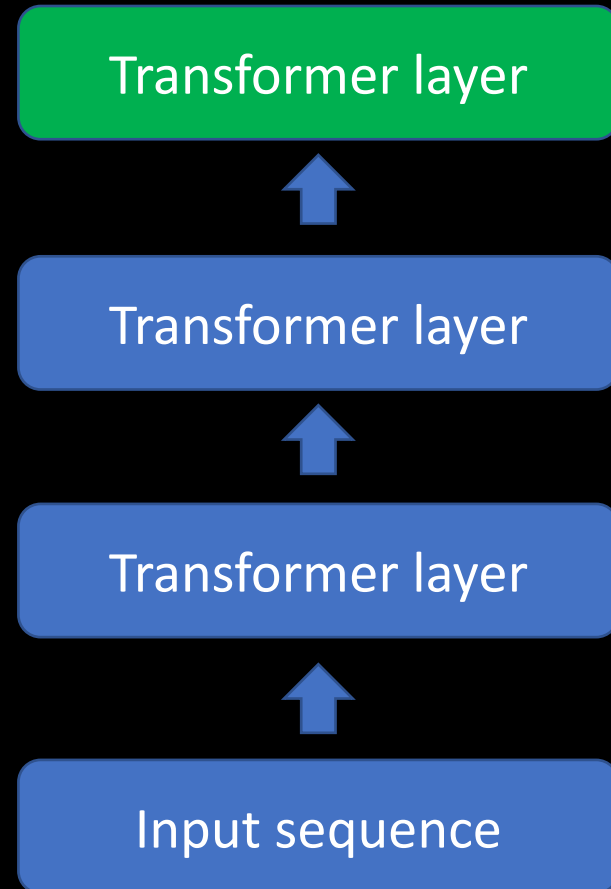
Linear models:

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

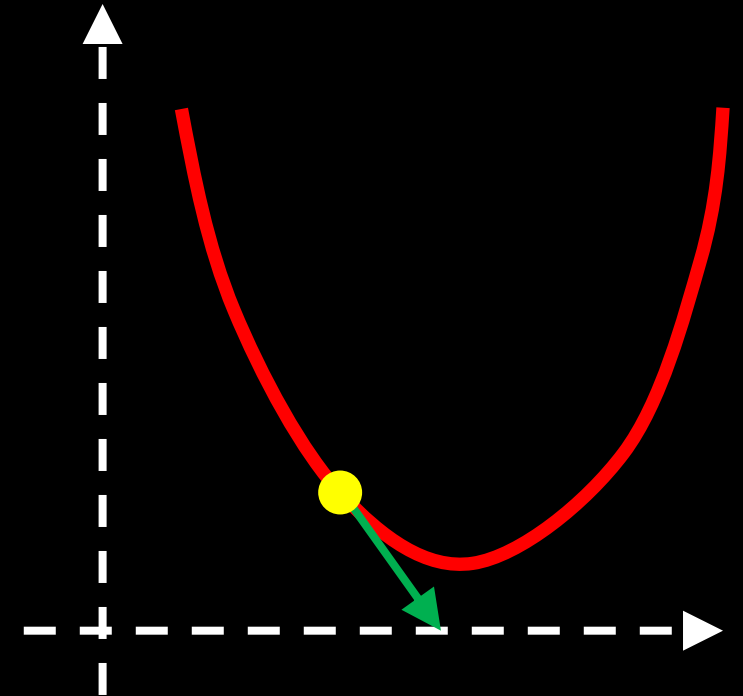
$$x_2 = (-2, 2), y_2 = 8$$

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

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



\approx

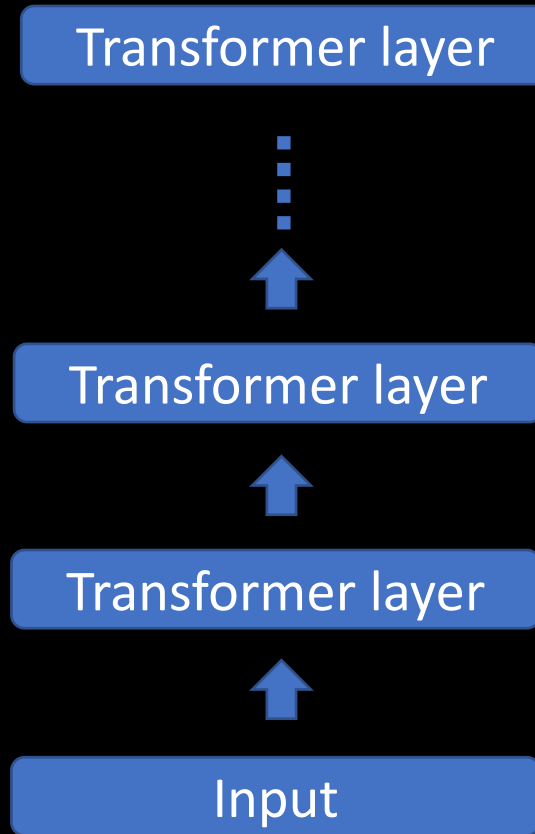




Techniques: “Applied theory”?

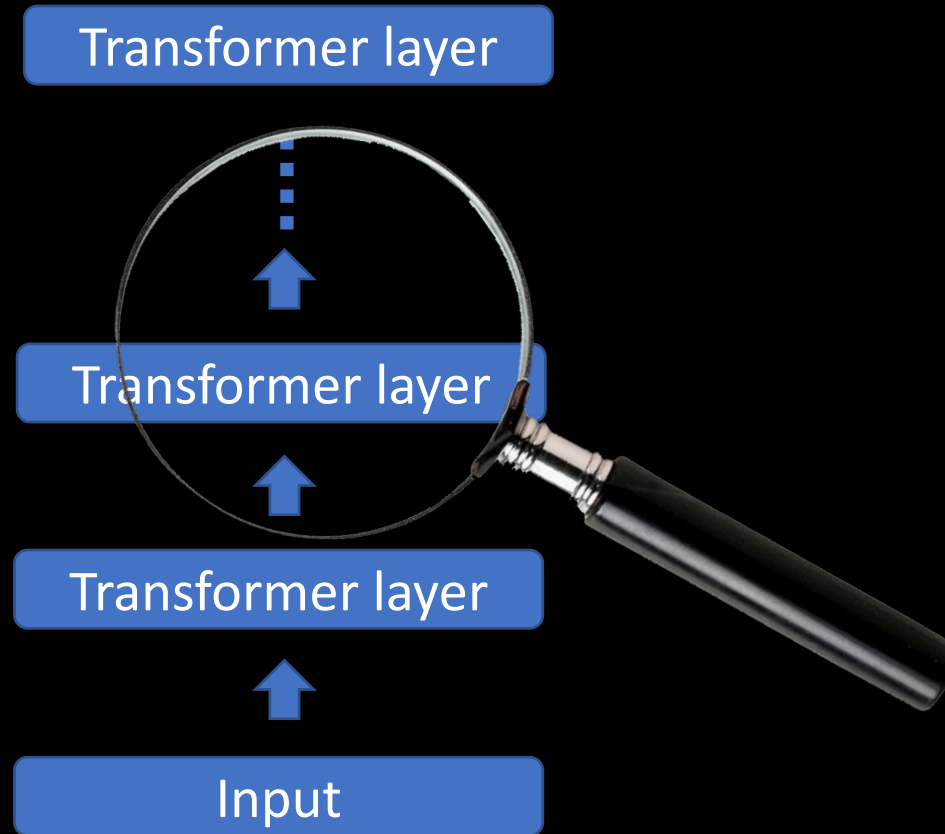
Techniques: “Applied theory”?

How should we understand how Transformers solve a problem?



Techniques: “Applied theory”?

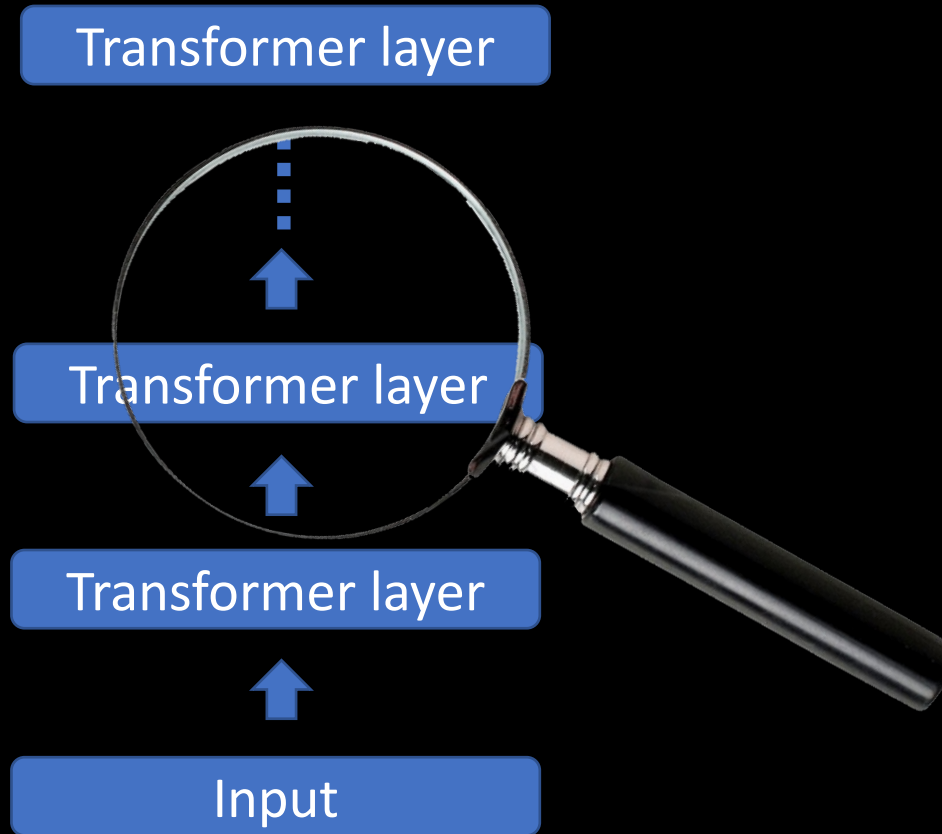
How should we understand how Transformers solve a problem?



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

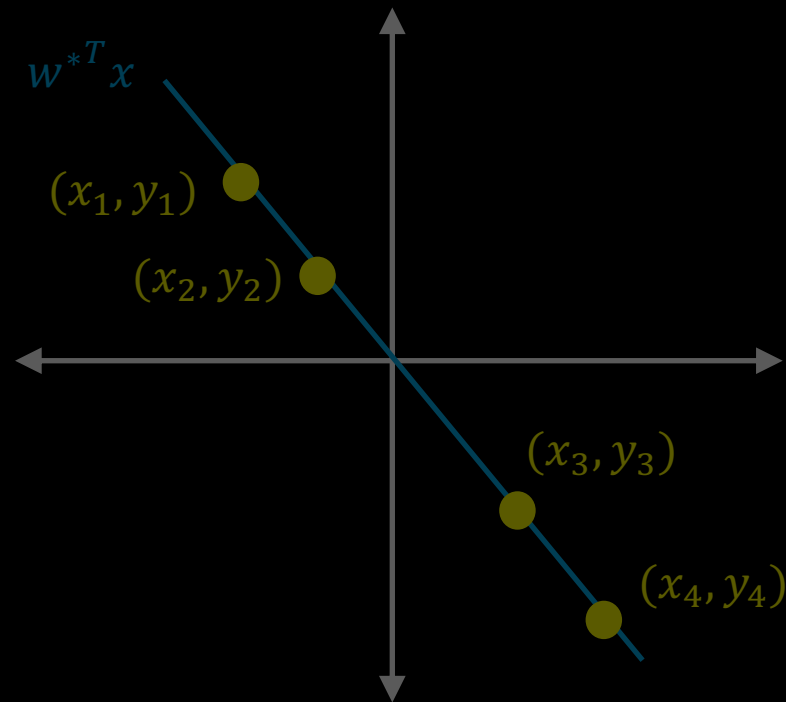
Techniques: “Applied theory”?

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

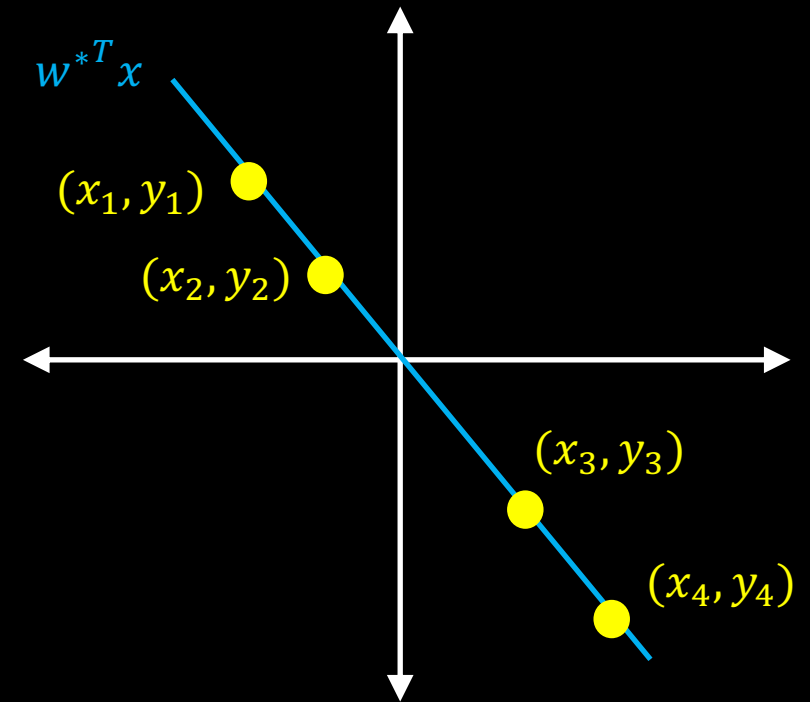
Data distribution

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)

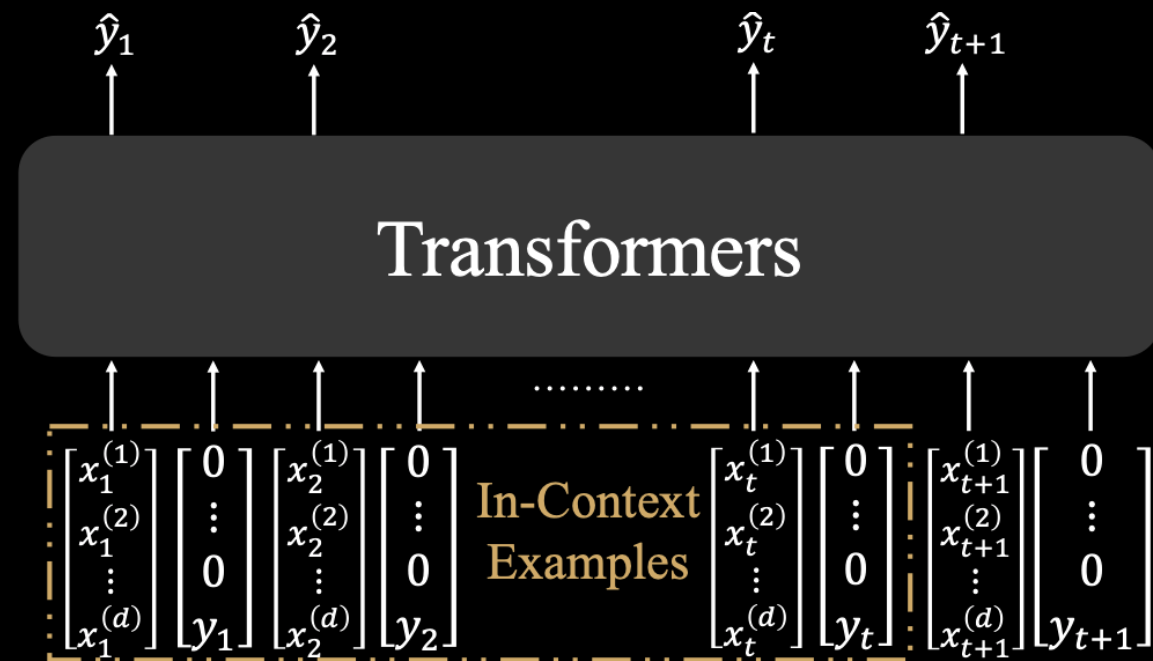
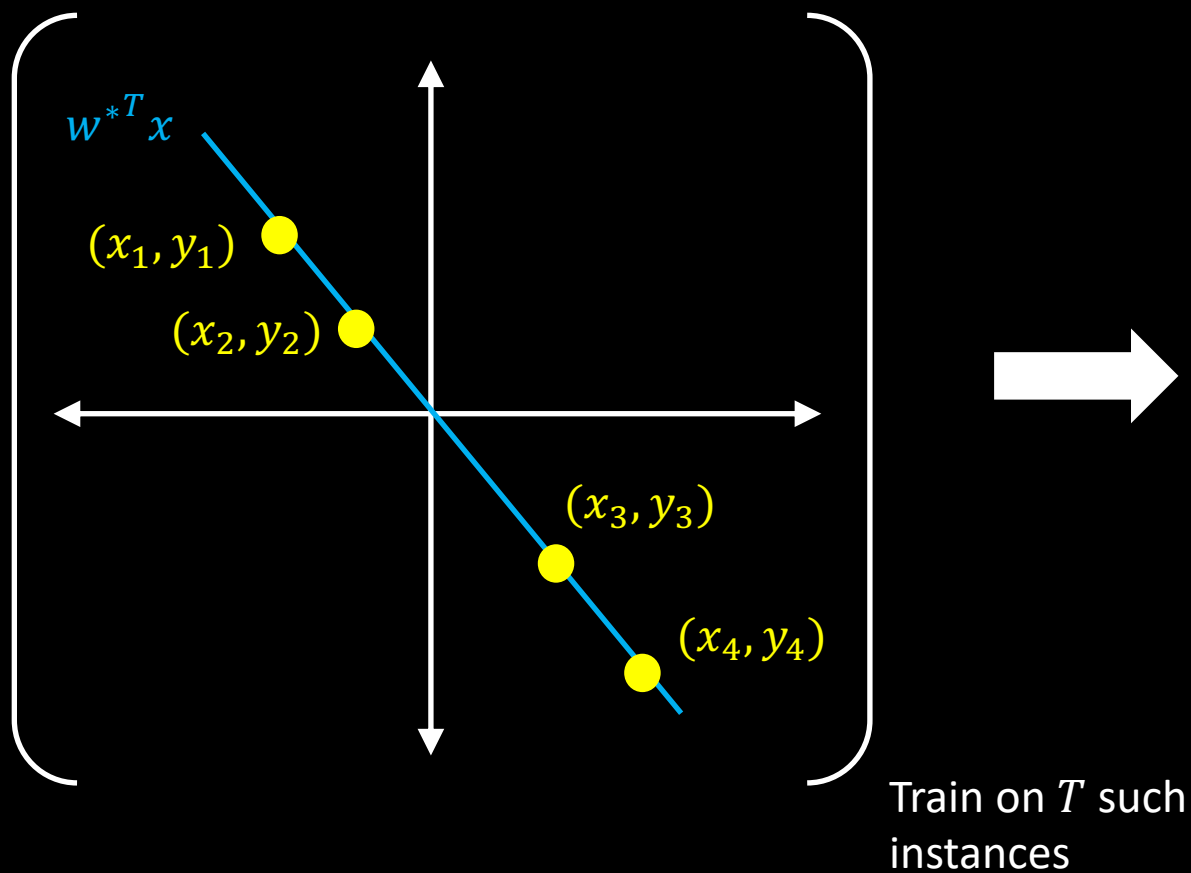
$$\text{Let } S = X^T X$$

$$M_0 = \alpha S, M_{k+1} = 2M_k - M_k S M_k$$

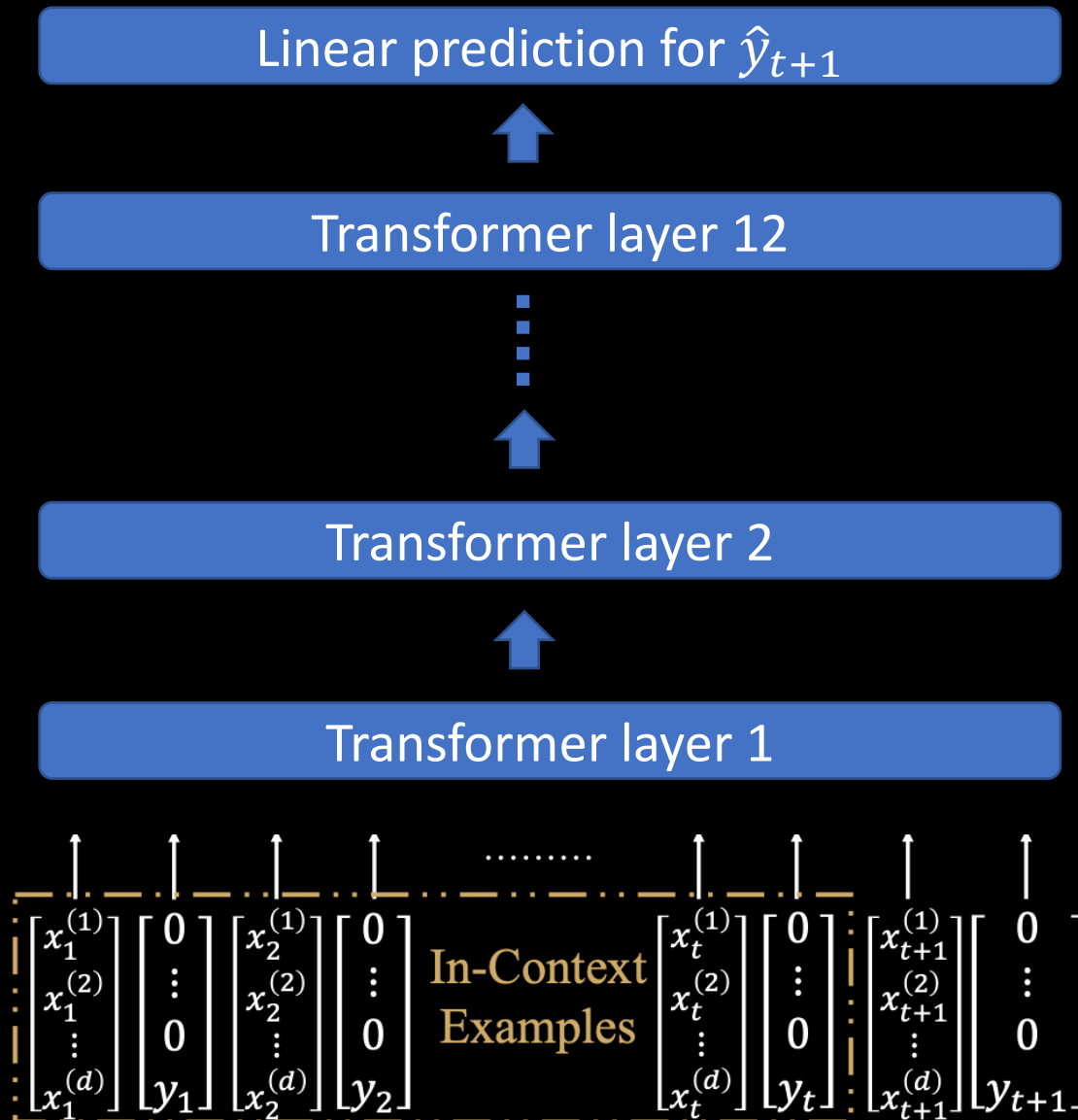
$$w_{Newton}^{(k)} = M_k X^T y$$

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

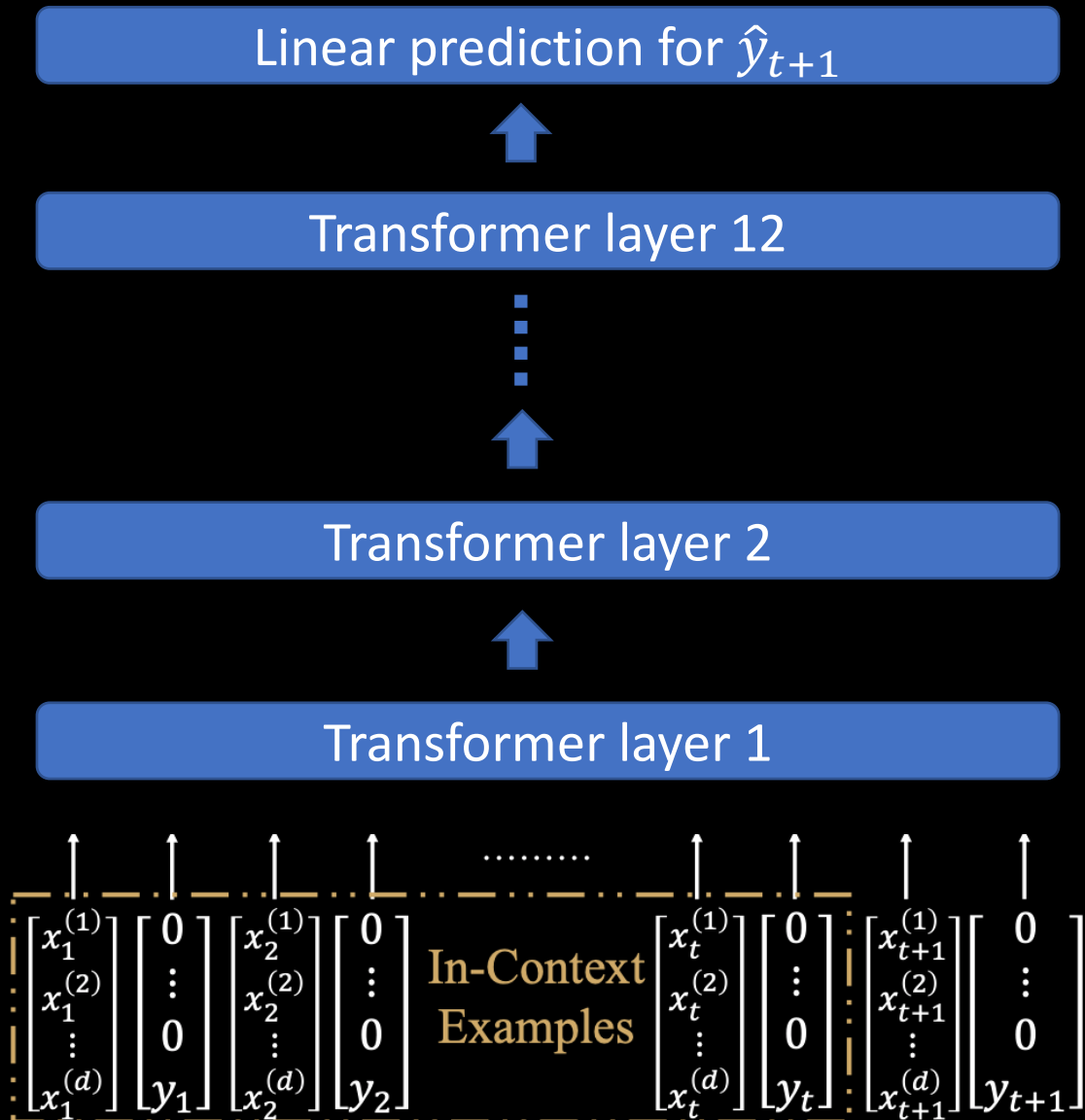
Transformers for linear regression



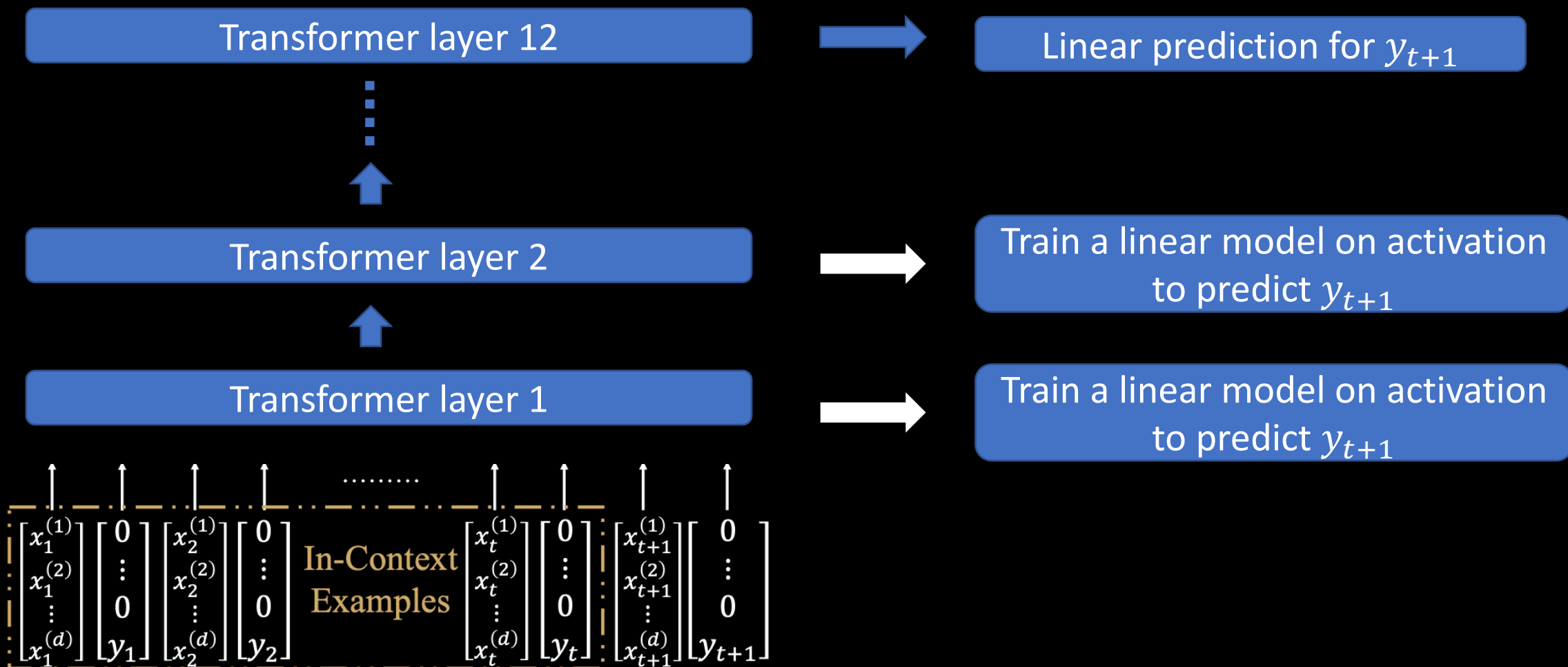
Transformers for linear regression



Transformers as an iterative algorithm: probing layers



Transformers as an iterative algorithm: probing layers



Metric: Similarity of errors

$x_1, y_1, x_2, y_2, x_3, y_3, \dots, x_n, y_n,$

Algorithm A $y_1^A, y_2^A, y_3^A, \dots, y_n^A,$

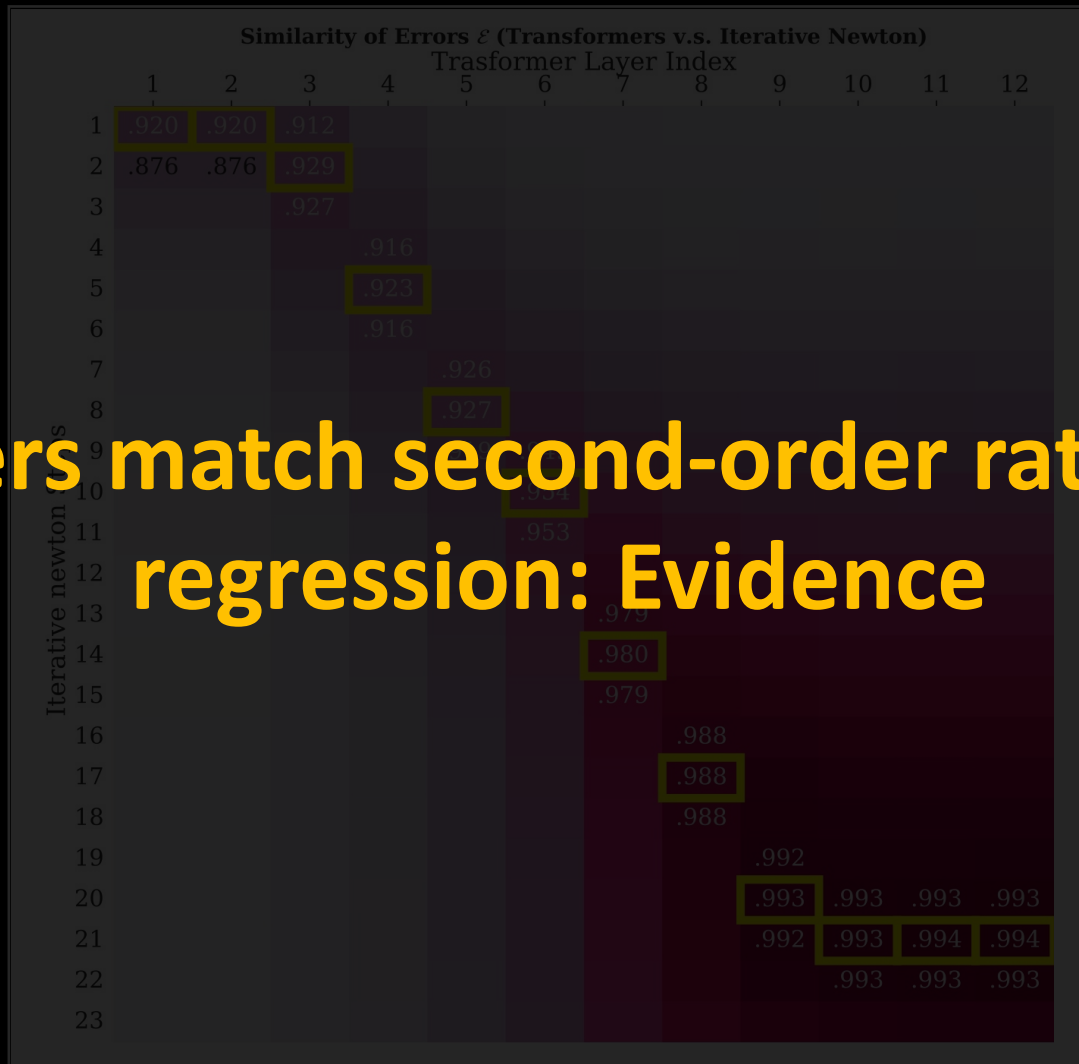
Algorithm B $y_1^B, y_2^B, y_3^B, \dots, y_n^B,$

Algorithm A residuals $(y_1 - y_1^A), (y_2 - y_2^A), (y_3 - y_3^A), \dots, (y_n - y_n^A),$

Algorithm B residuals $(y_1 - y_1^B), (y_2 - y_2^B), (y_3 - y_3^B), \dots, (y_n - y_n^B),$

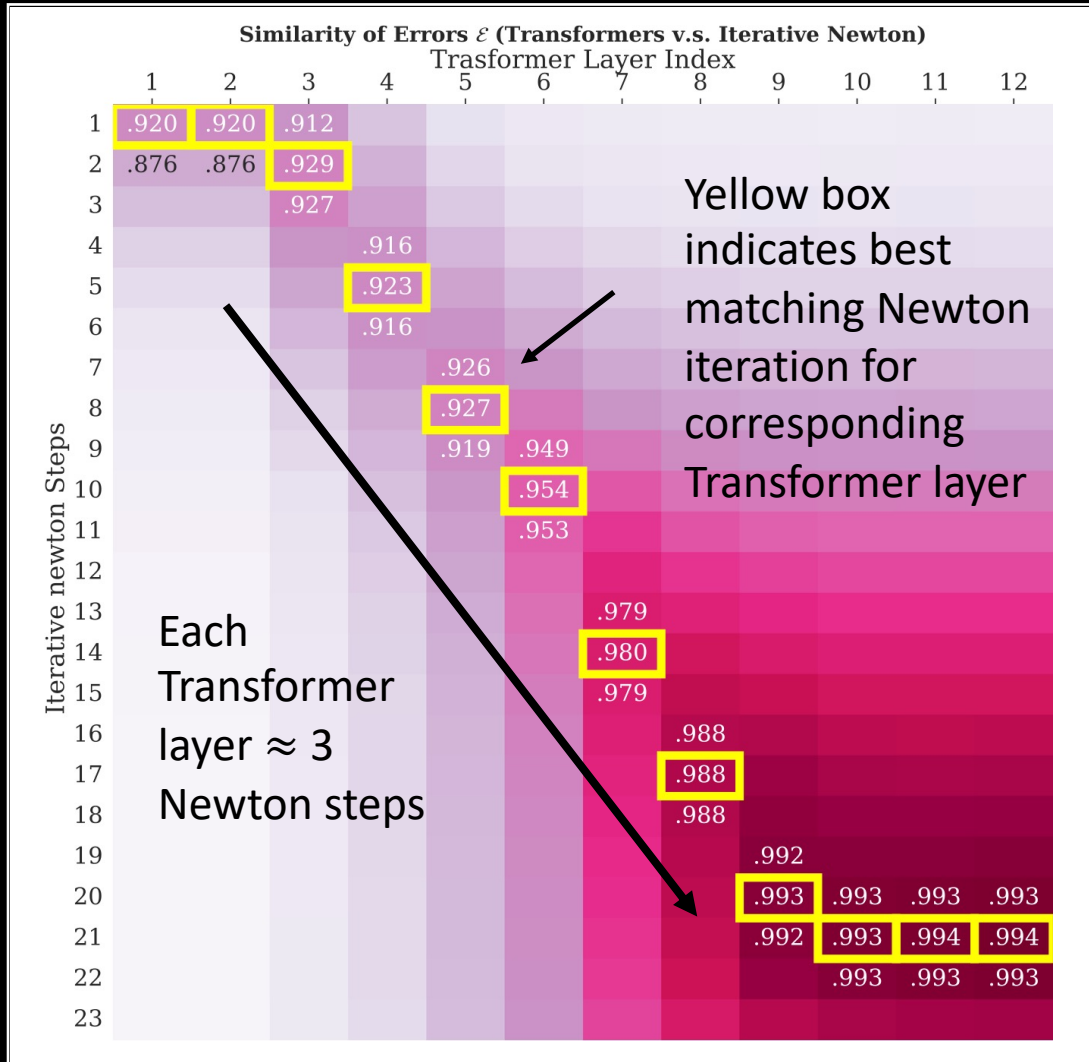
Similarity of errors on $\{x_i, y_i\}_{i=1}^n$ between Algorithm A, Algorithm B
= Cosine similarity between residuals of A, B

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

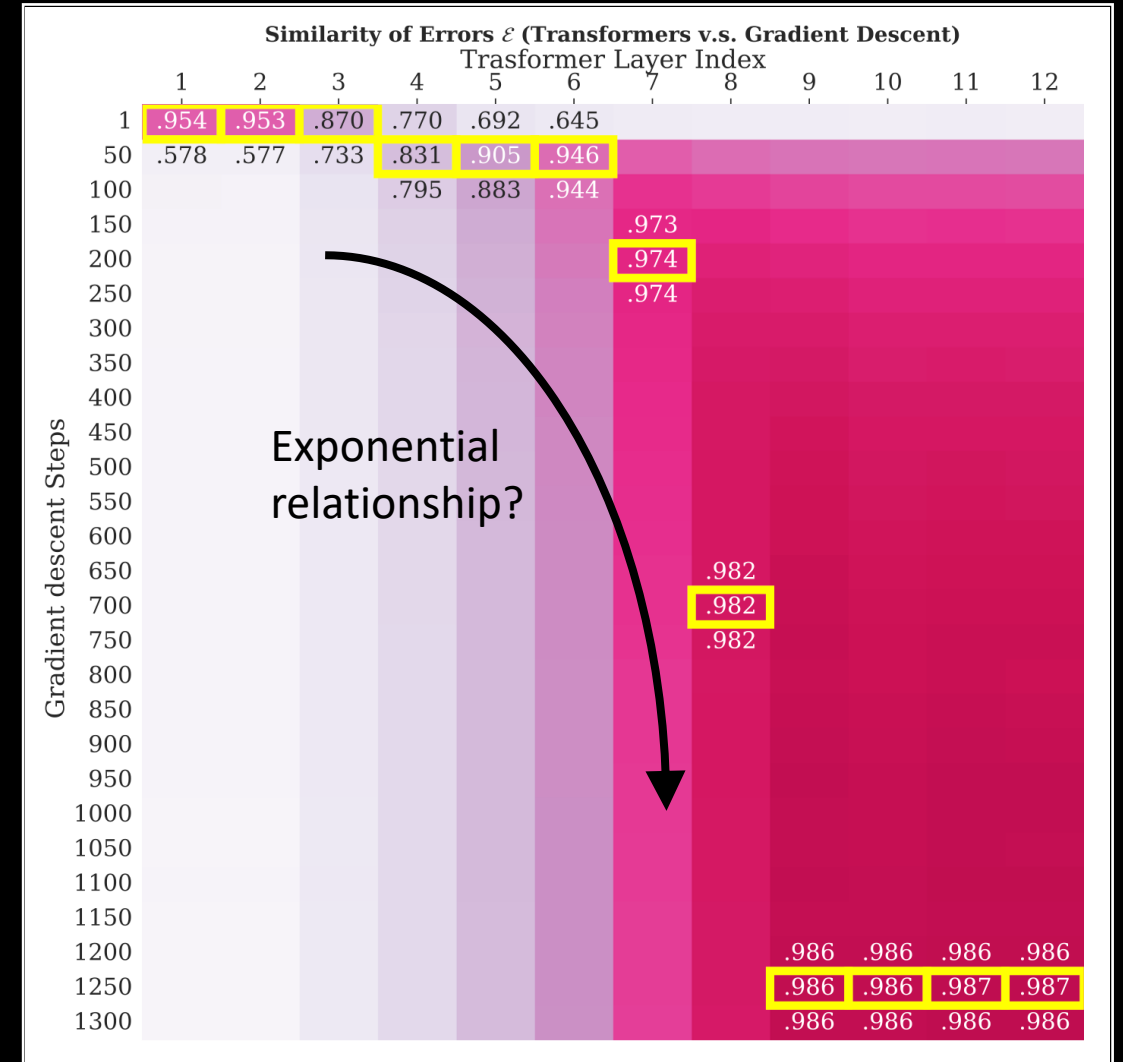


Transformers match second-order rates for linear regression: Evidence

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

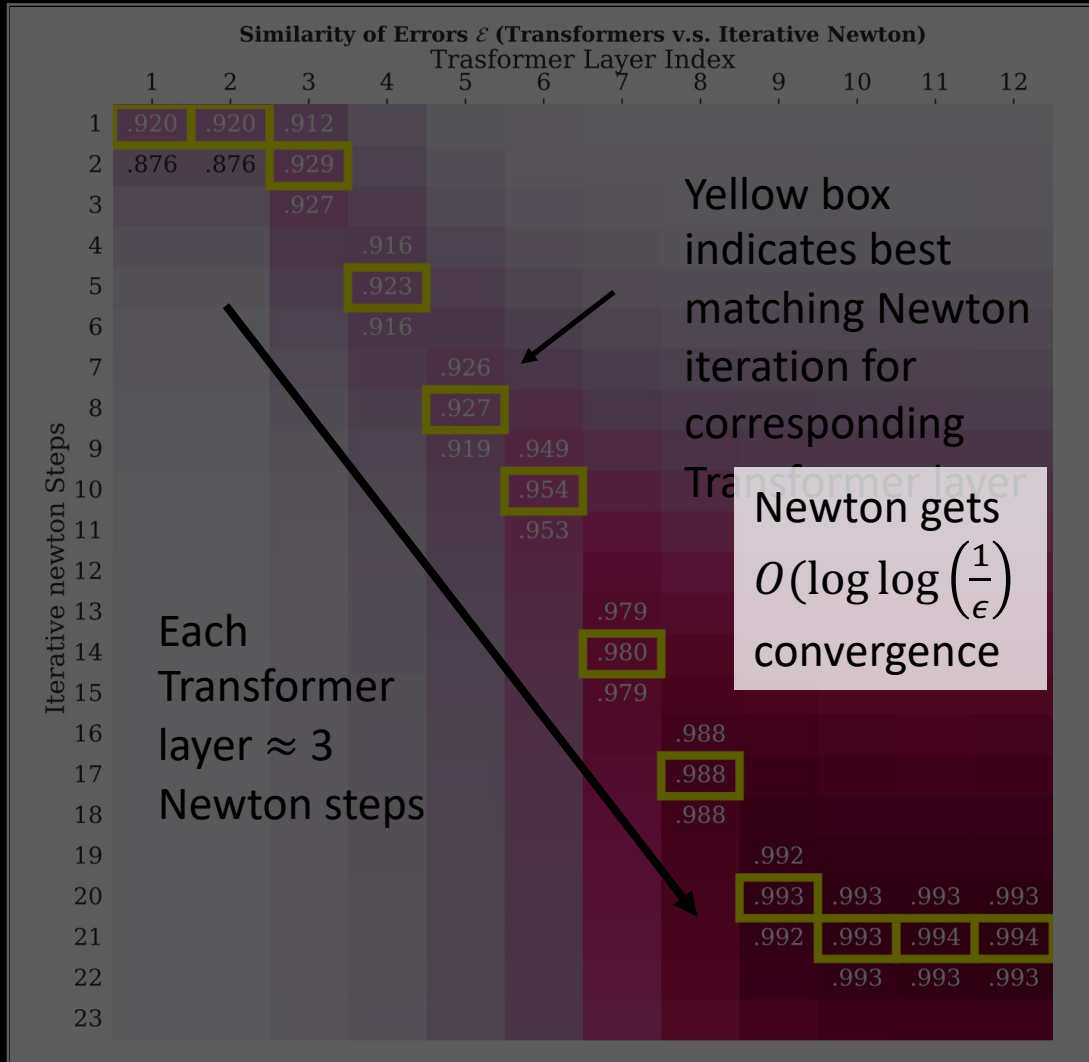


Transformers vs Newton

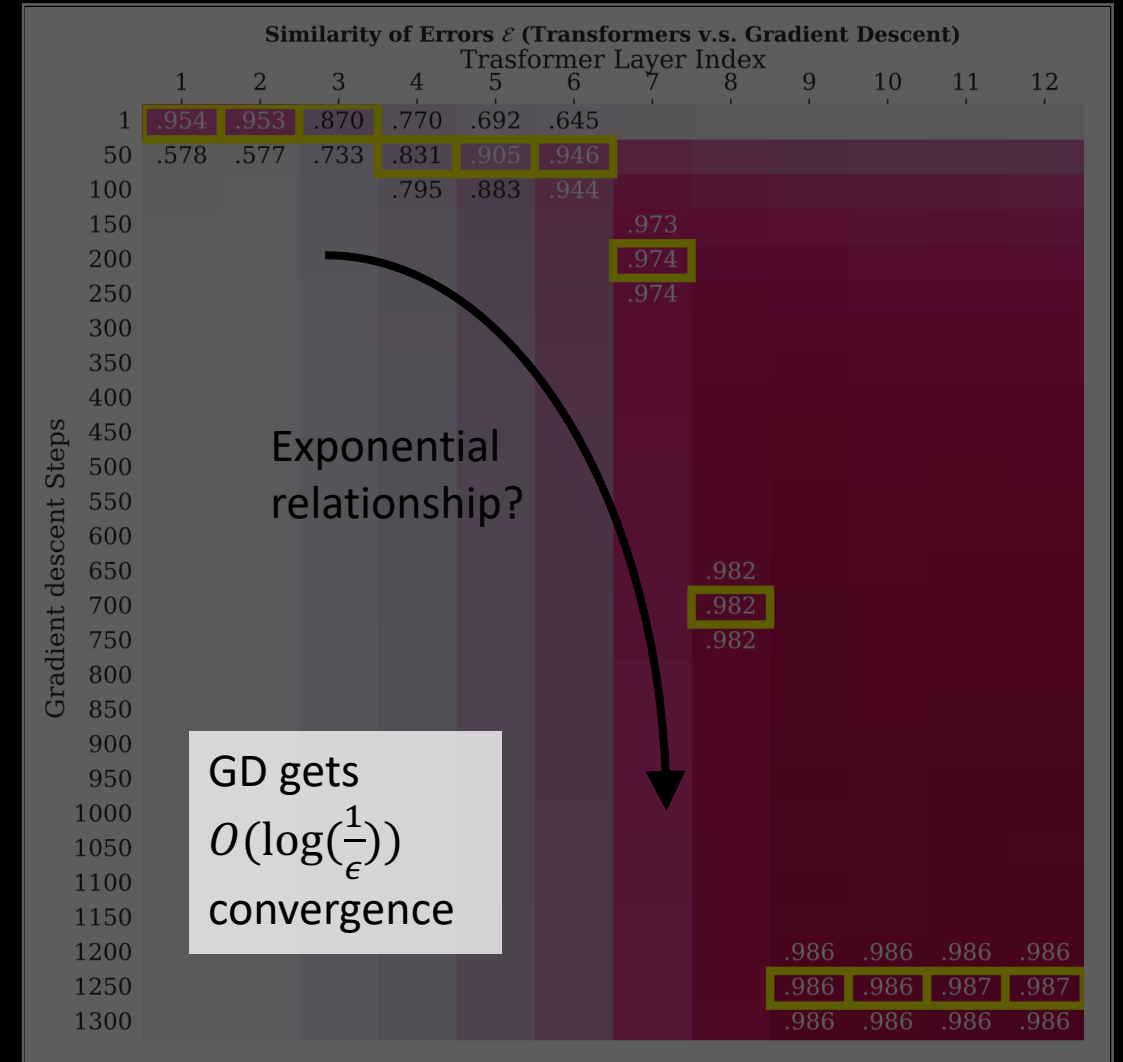


Transformers vs Gradient Descent

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

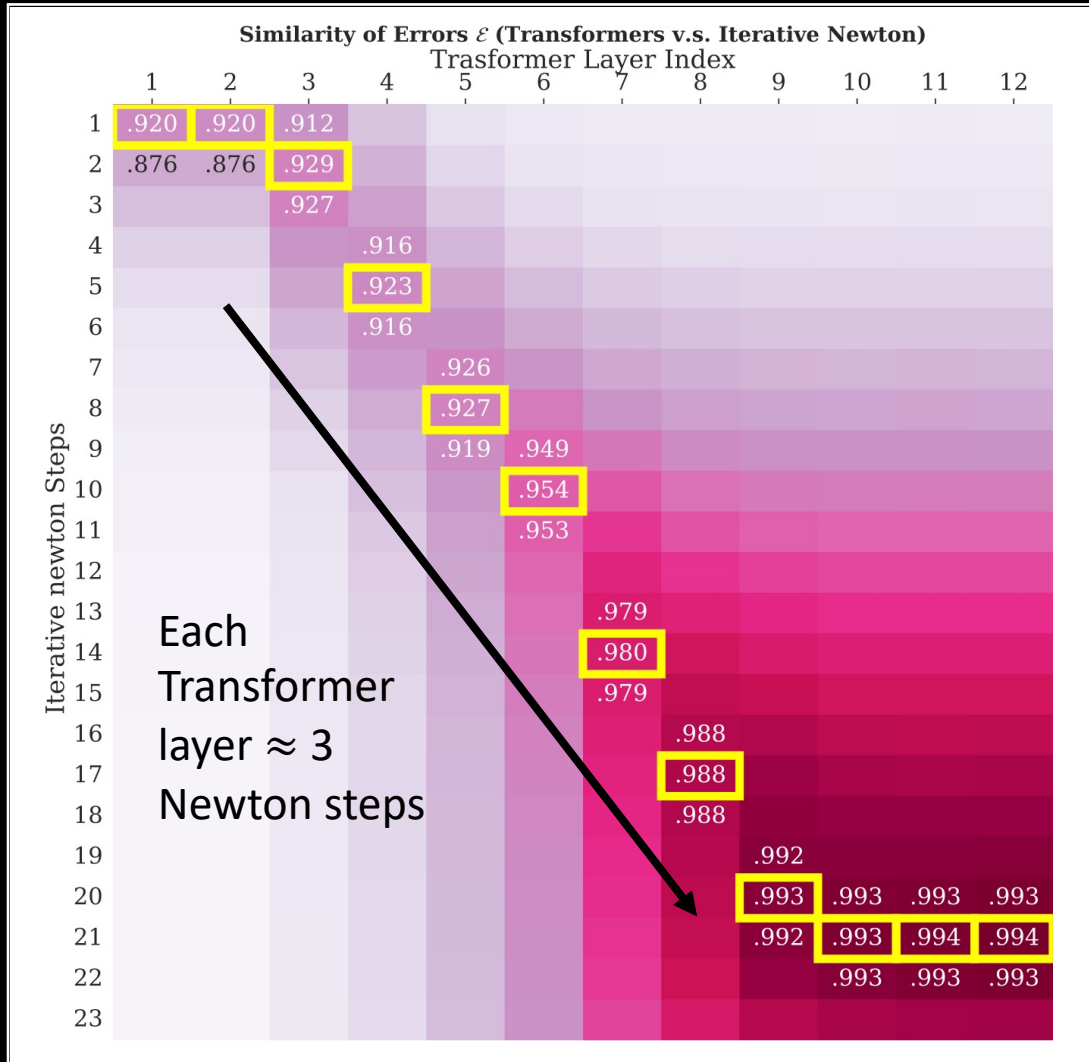


Transformers vs Newton

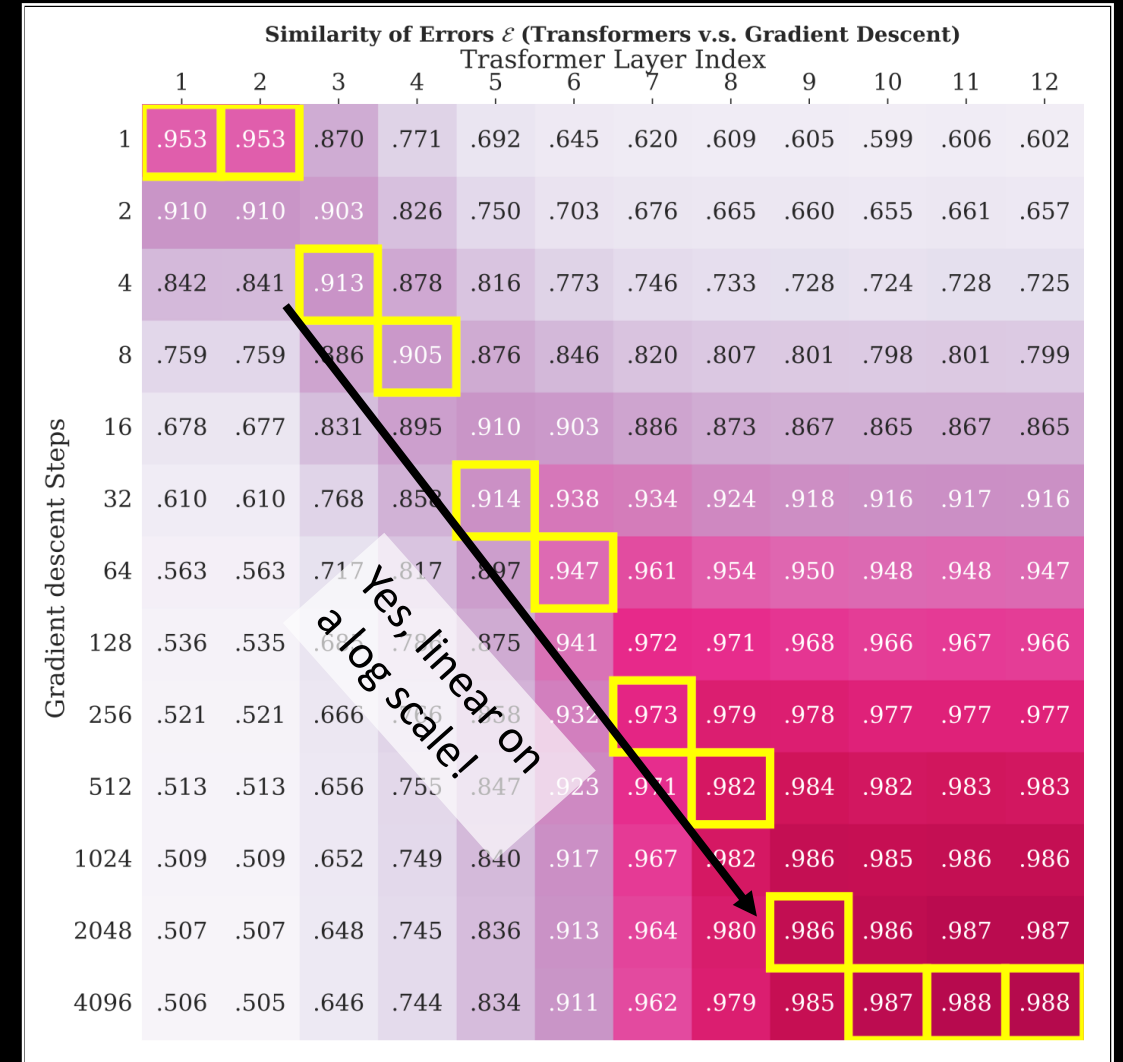


Transformers vs Gradient Descent

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



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?

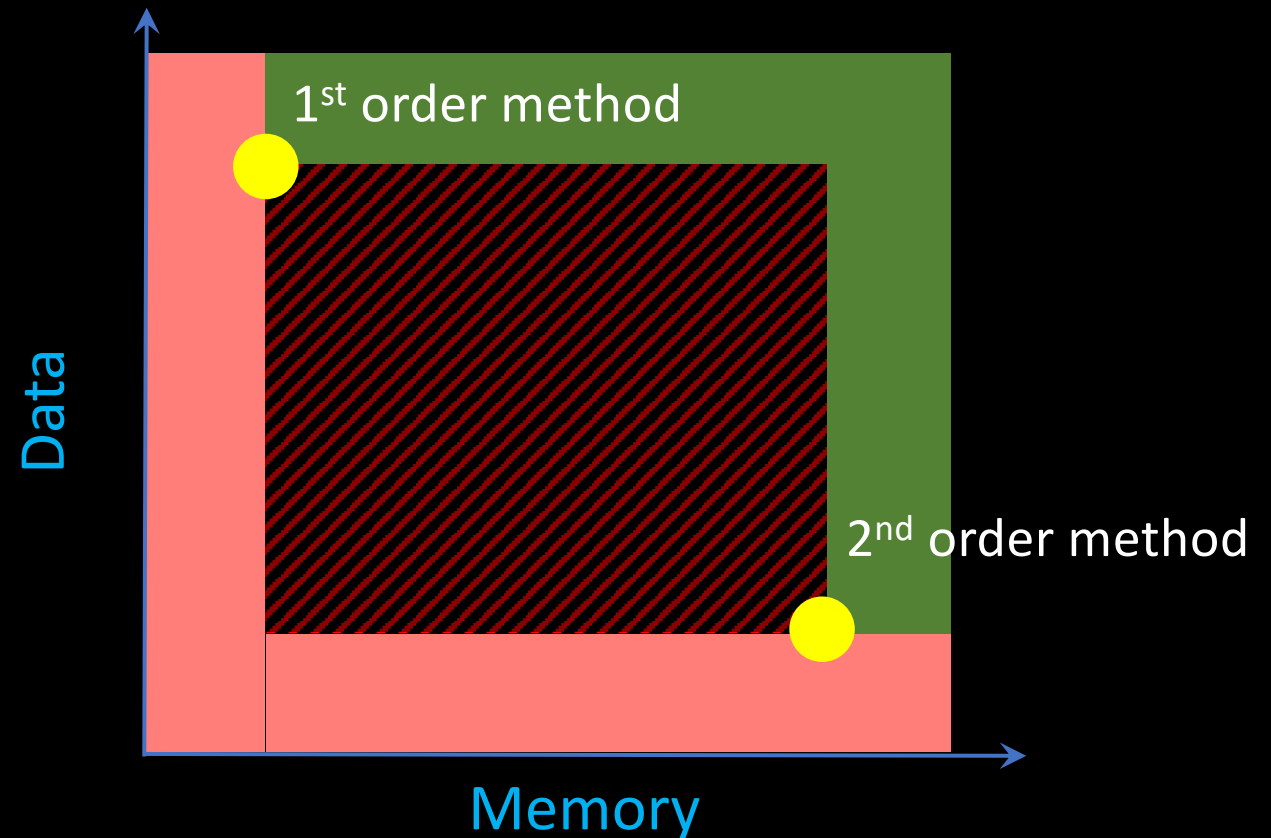
On **ill-conditioned instances**, gradient descent (or its variants) get *poly*(κ) dependence on the condition number of the linear system κ , 2nd order methods get *polylog*(κ) dependence.

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 $\text{poly}(\kappa)$ dependence on the condition number of the linear system κ , 2nd order methods get $\text{polylog}(\kappa)$ dependence.

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



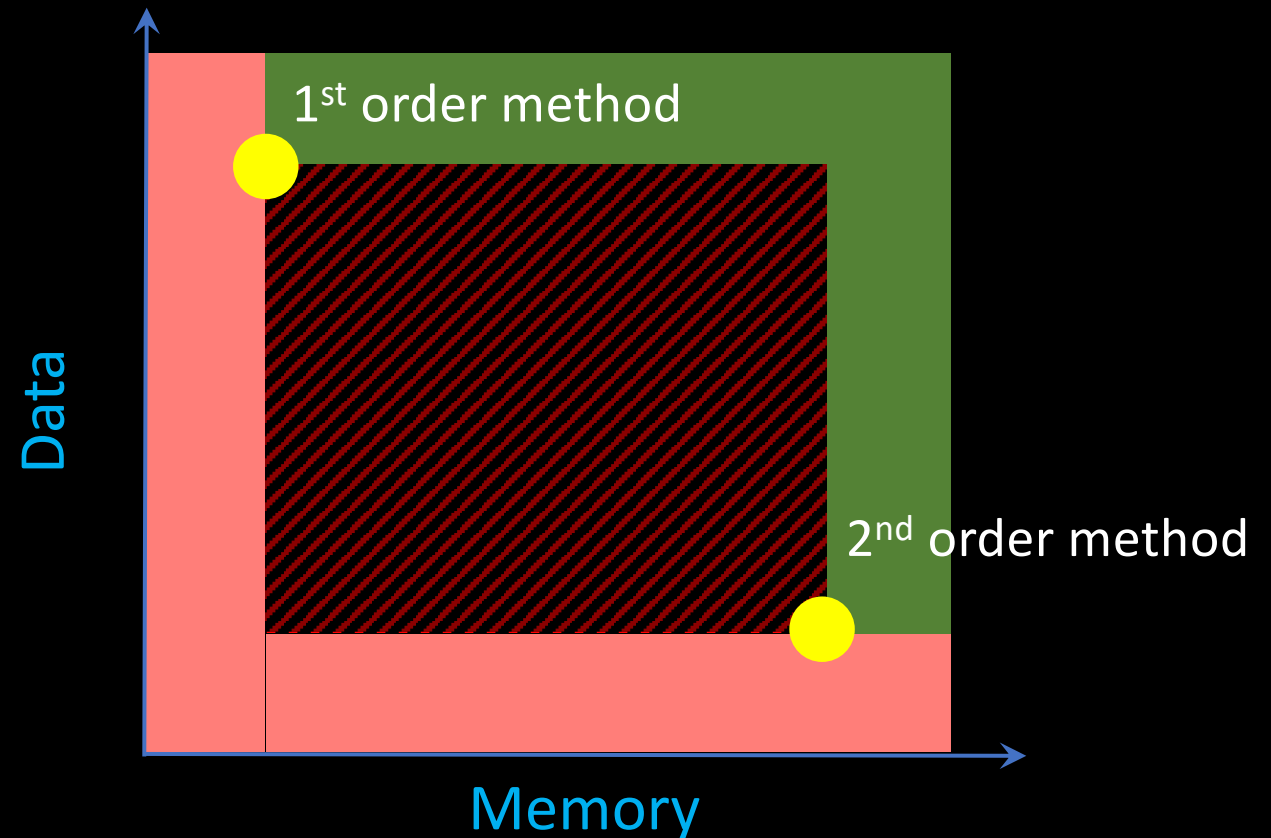
Claim 3: Transformers are still able to match Newton on harder distributions

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

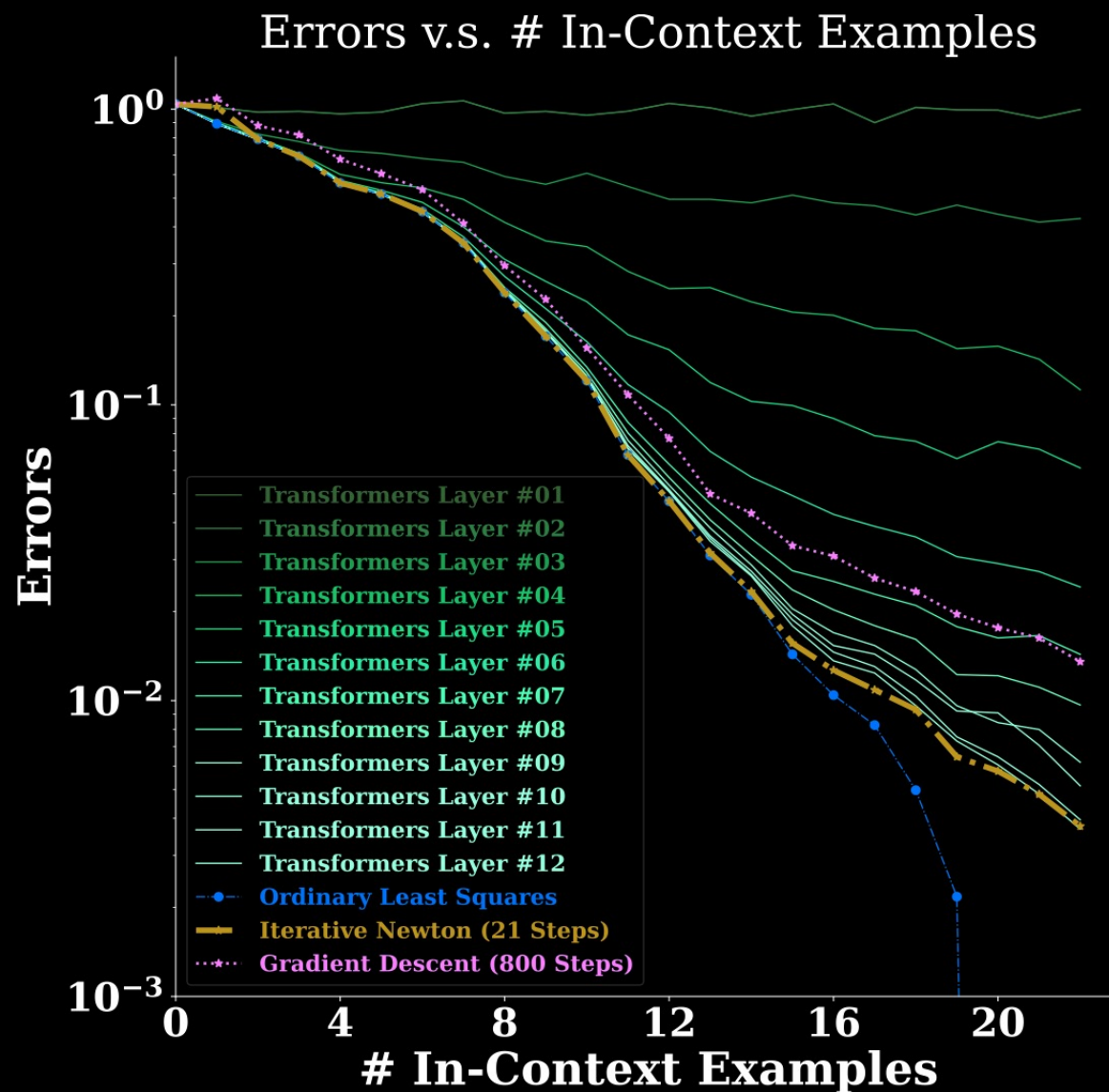
On **ill-conditioned instances**, gradient descent (or its variants) get $\text{poly}(\kappa)$ dependence on the condition number of the linear system κ , 2nd order methods get $\text{polylog}(\kappa)$ dependence.

Conjecture (S.-Sidford-Valiant'19): No first-order (linear memory method) can avoid a $\text{poly}(\kappa)$ 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 ε (Transformers v.s. Iterative Newton)

		Transformer Layer Index											
		1	2	3	4	5	6	7	8	9	10	11	12
Iterative newton Steps	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	.677	.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	.800	.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	.917	.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	.958	.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 ε (Transformers v.s. Gradient Descent)

		Transformer Layer Index											
		1	2	3	4	5	6	7	8	9	10	11	12
Gradient descent Steps	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	.422	.423	.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	.971	.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](#)

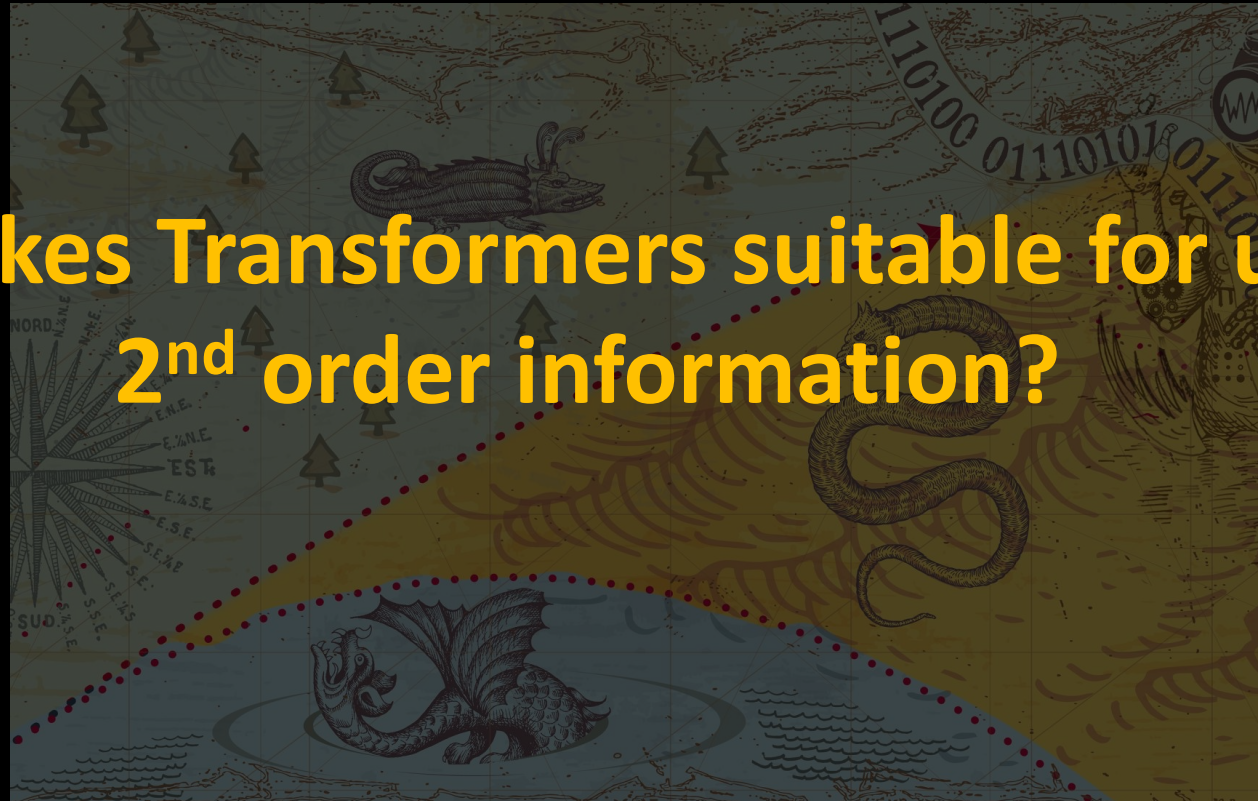
Some more related work

[Ahn-Cheng-Daneshmand-Sra'2023](#), [Zhang-Frei-Bartlett'2023](#) & [Mahankali-Hashimoto-Ma'2024](#) analyze dynamics of trained one-layer Transformers

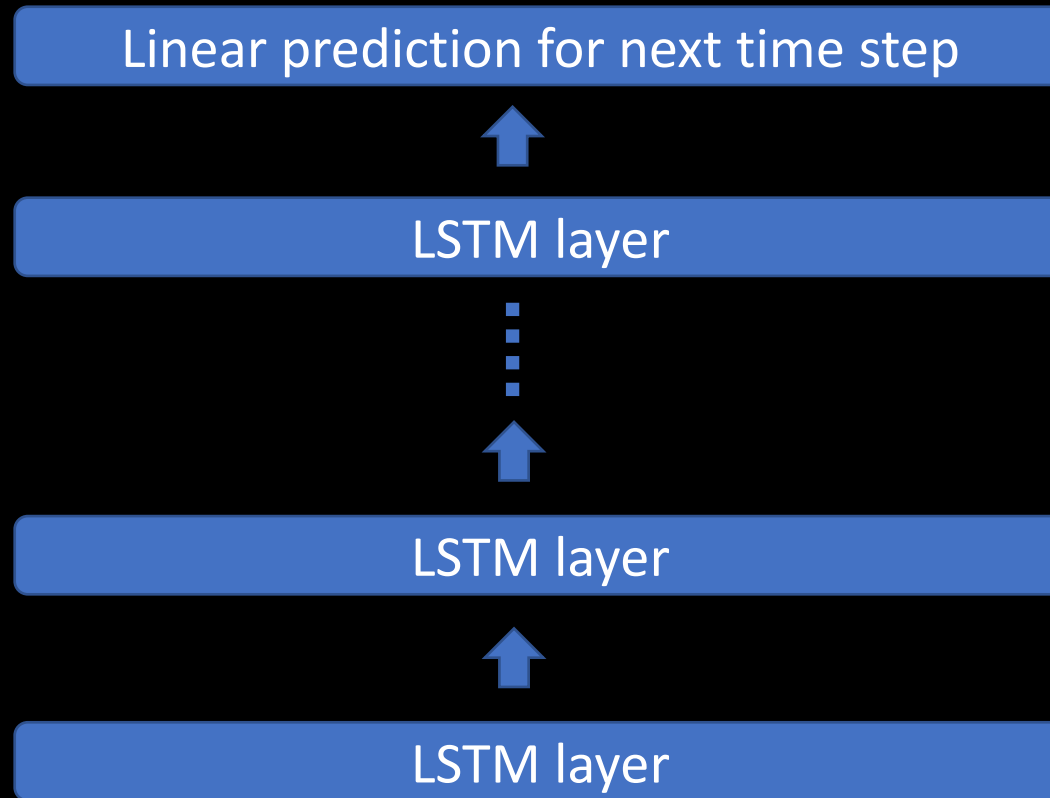
[Vladymyrov-von Oswald-Sandler-Ge'2024](#) show that a second-order variant of GD can mimic Iterative Newton by implicitly approximating inverse

[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?**

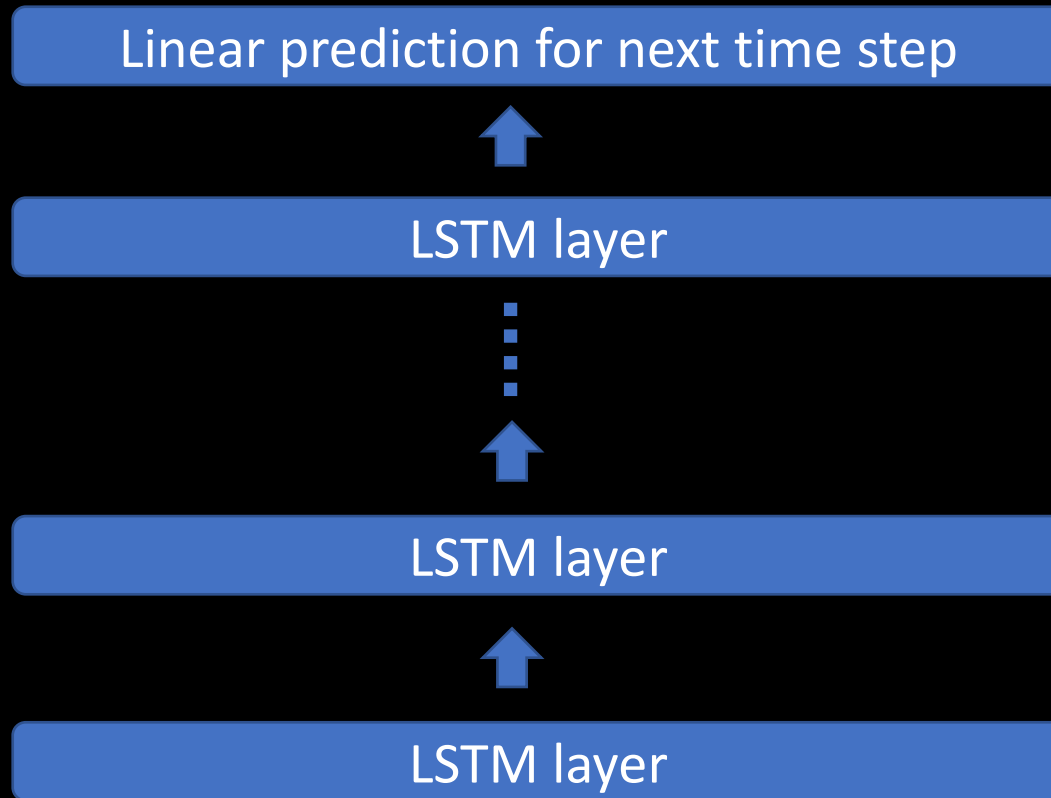


LSTMs for linear regression



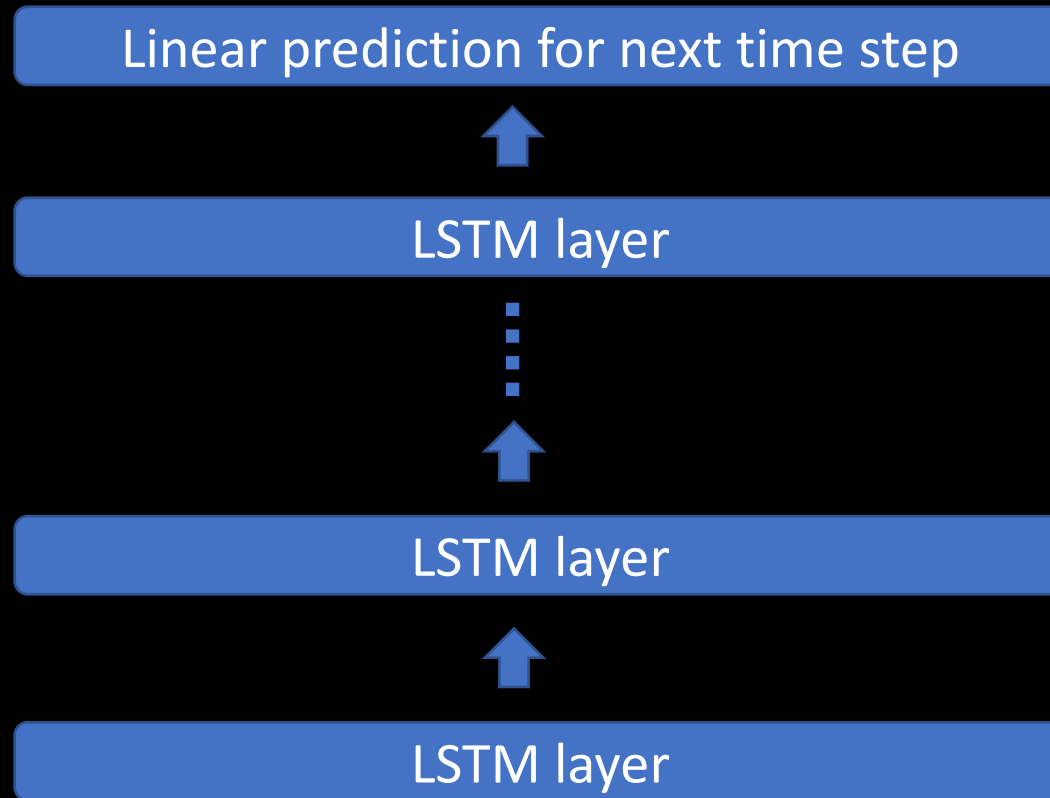
$$\begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(d)} \end{bmatrix}$$

LSTMs for linear regression



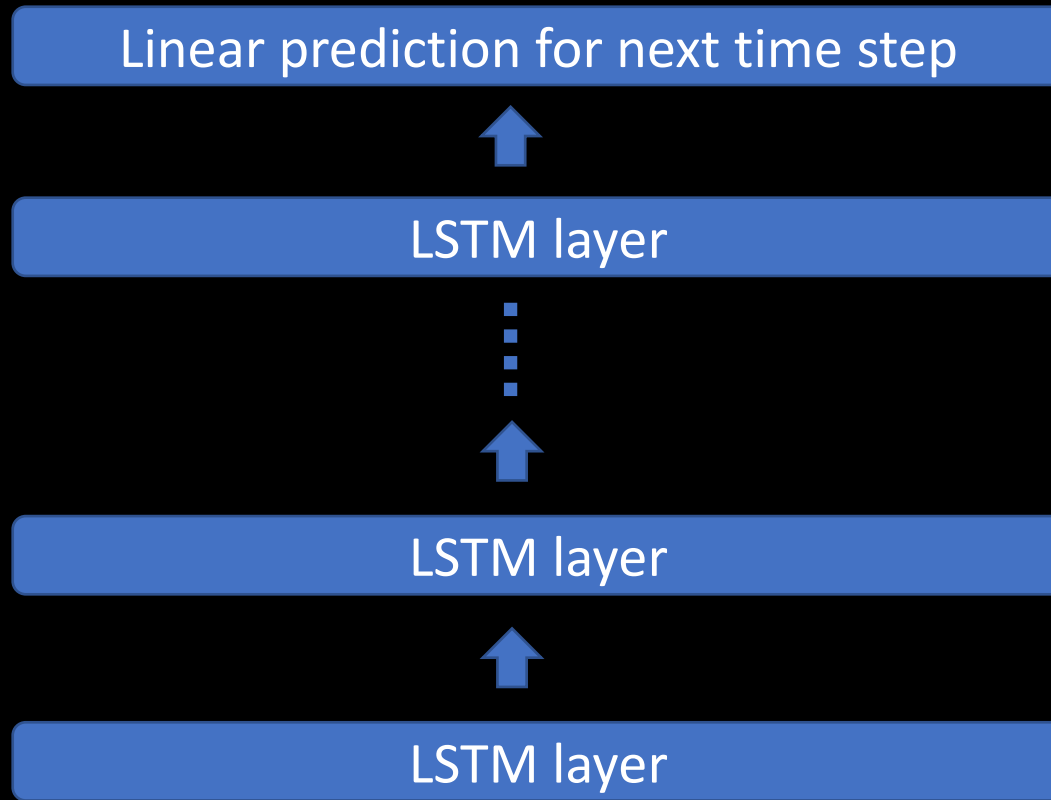
$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_1 \end{bmatrix}$$

LSTMs for linear regression



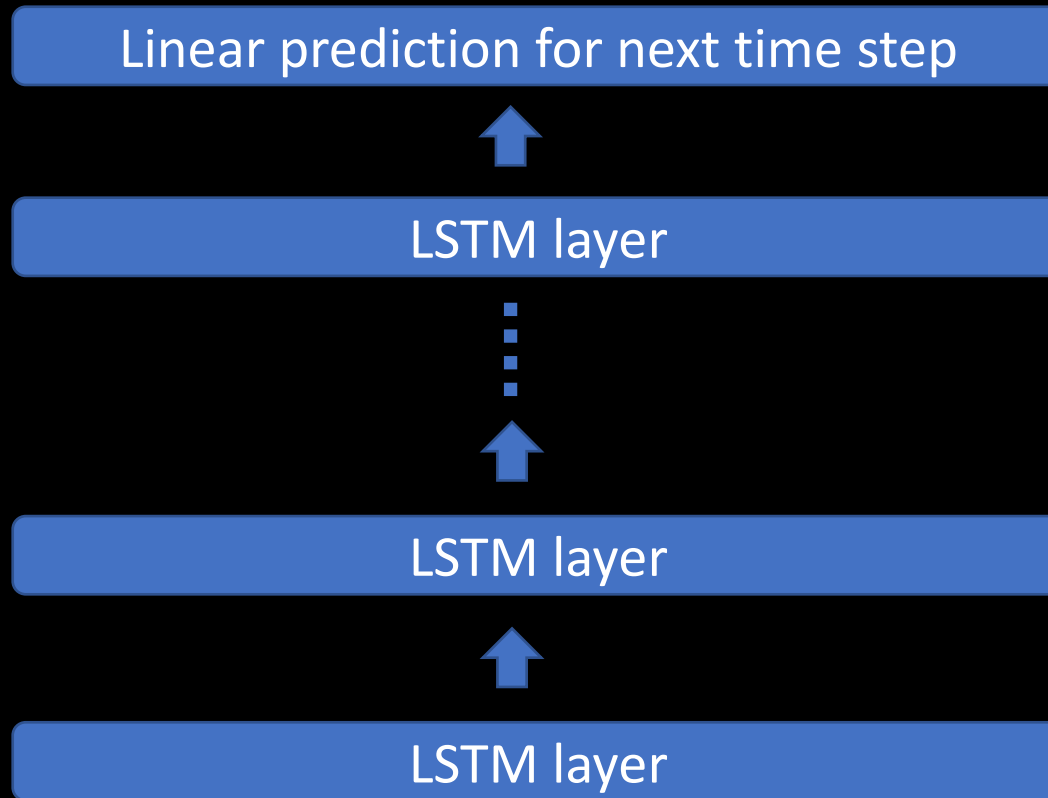
$$\begin{bmatrix} x_2^{(1)} \\ x_2^{(2)} \\ \vdots \\ x_2^{(d)} \end{bmatrix}$$

LSTMs for linear regression



$$\begin{matrix} \uparrow \\ \dots \\ \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \underline{y_2} \end{array} \right] \end{matrix}$$

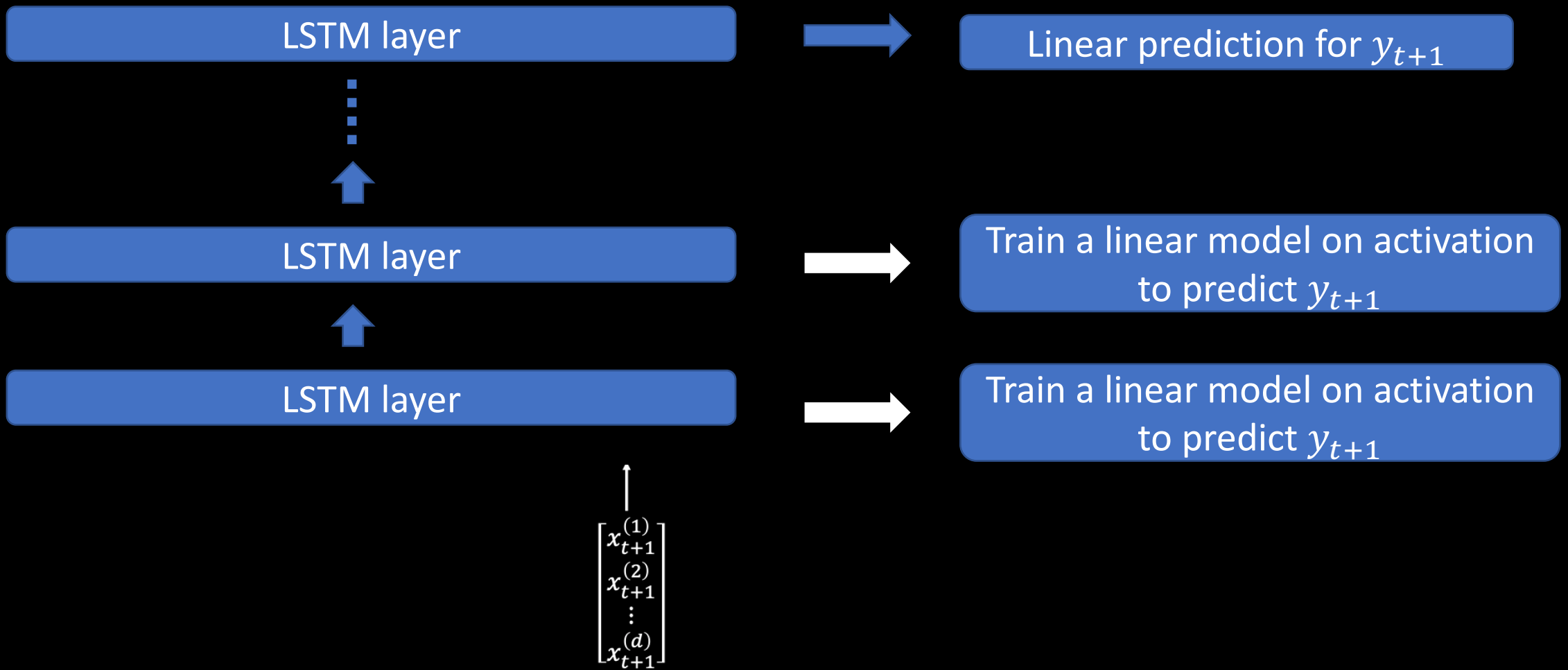
LSTMs for linear regression



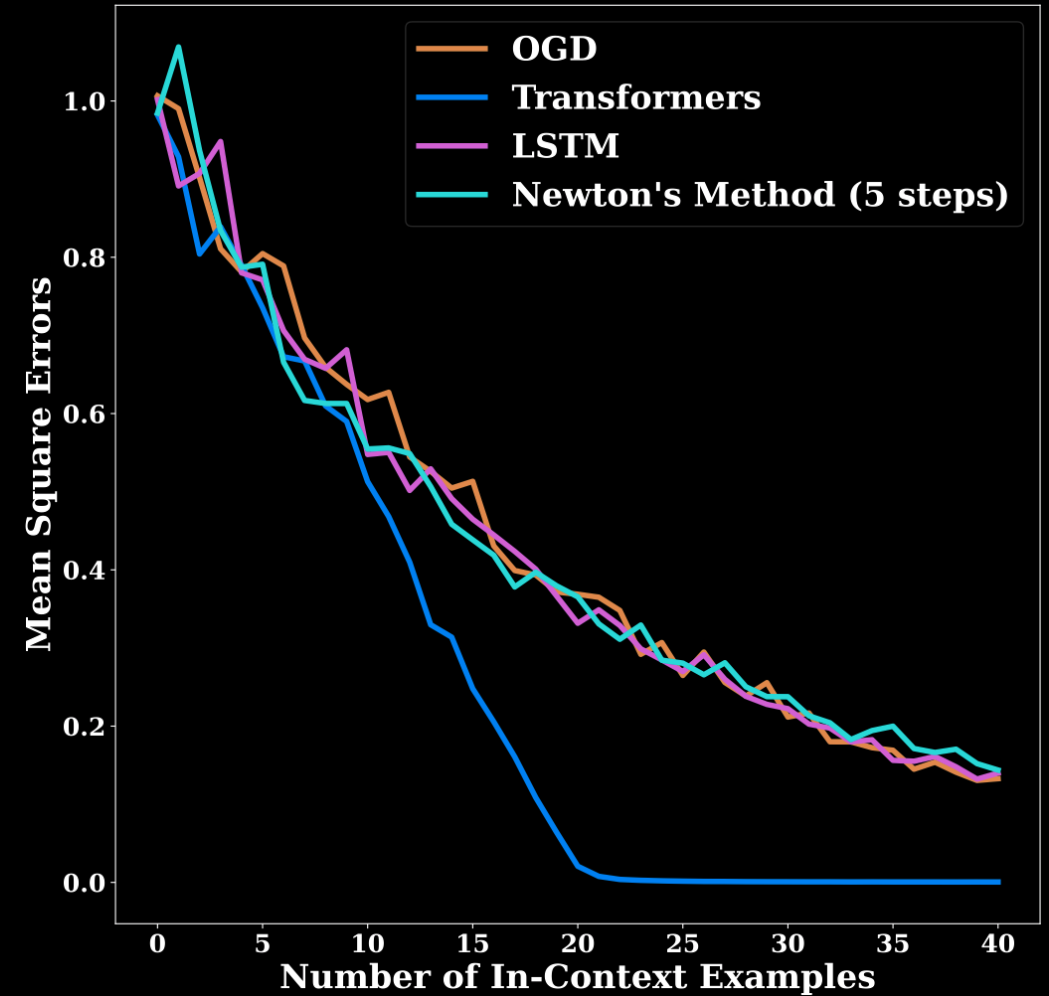
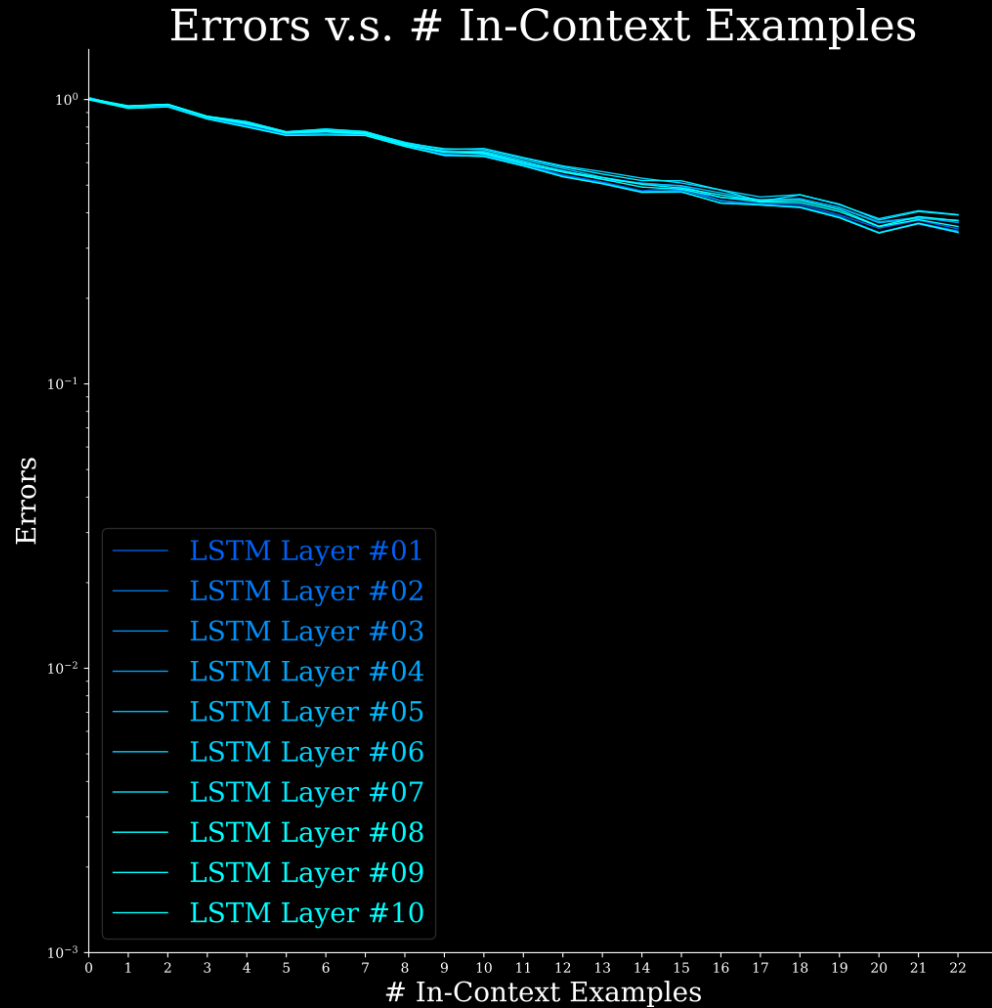
...

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

LSTMs as an iterative algorithm: probing layers

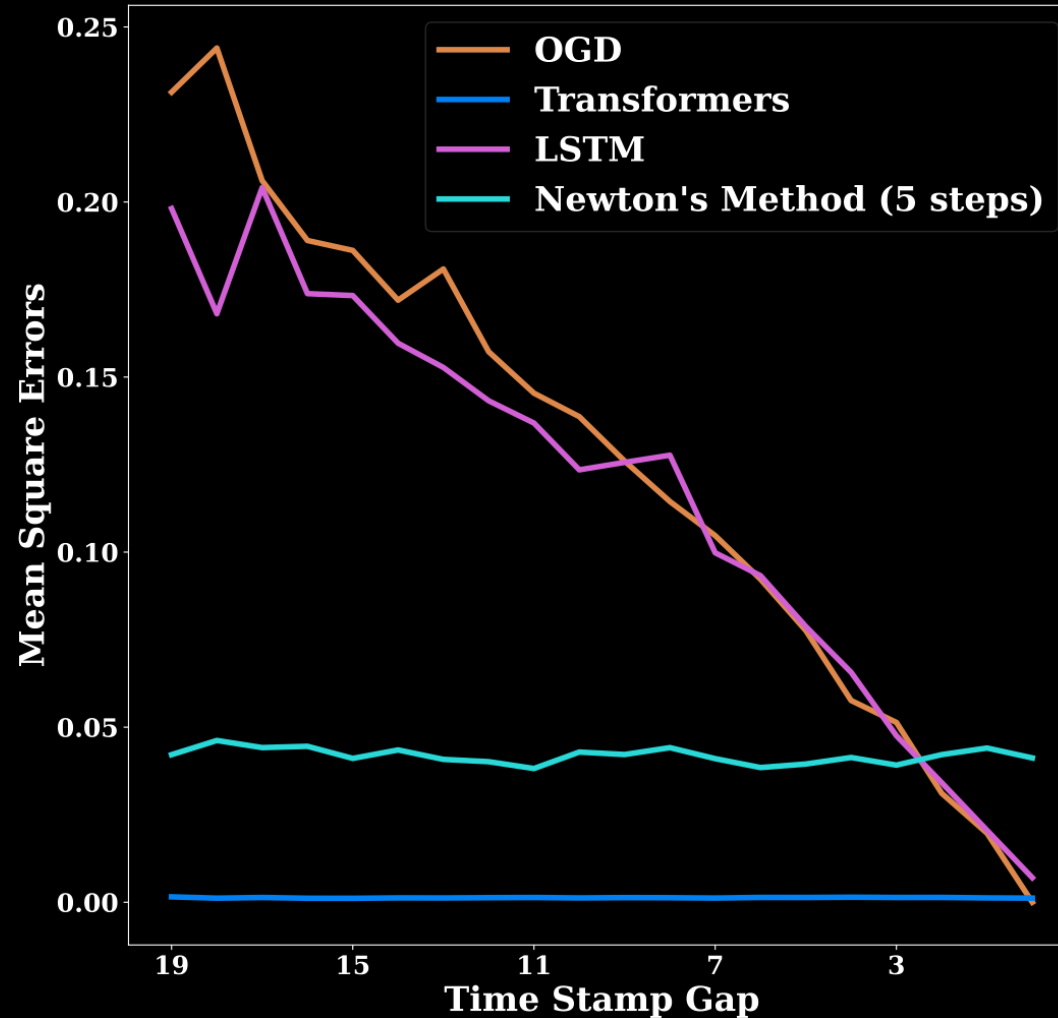


What do LSTMs implement?



LSTMs seem similar to online gradient descent

Like OGD, LSTMs 'forget' previous examples

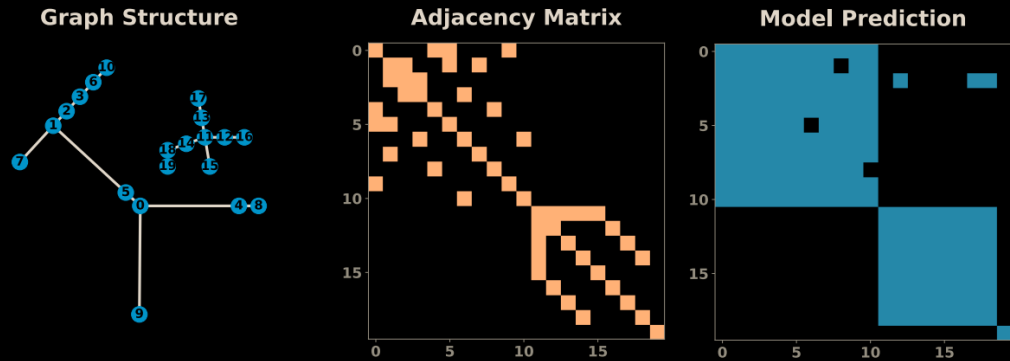


Error when input from t time steps ago is given as query point

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

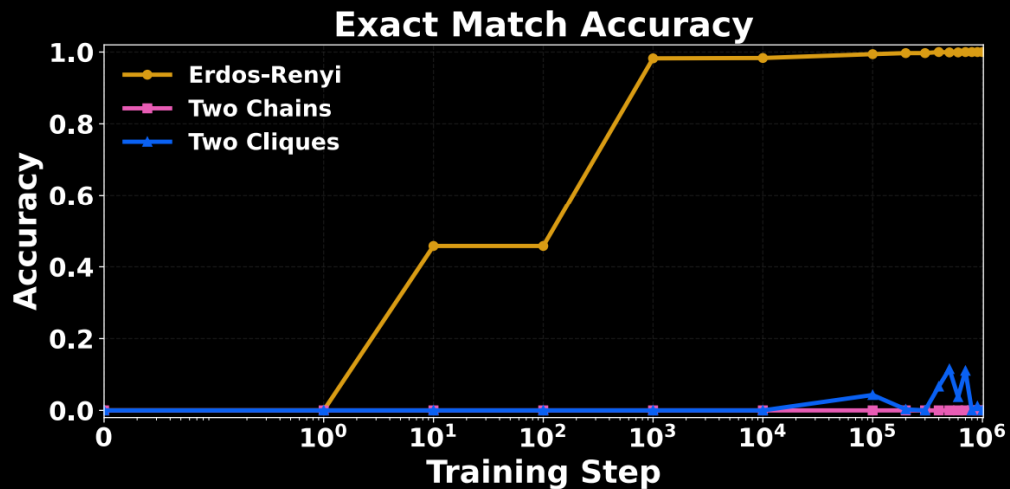
Recent line of theoretical work suggests that the available memory determines the best possible convergence rate, is gap between architectures an instantiation of this?

Recent work: Understanding graph algorithms



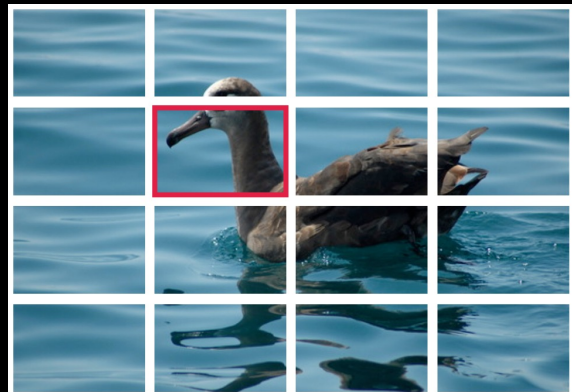
Algorithms vs. Data-driven Heuristics

- Investigate when transformers learn robust algorithmic solutions for graph connectivity, as opposed to learning distribution specific-heuristics
- Empirically and theoretically show how it depends on the data distribution



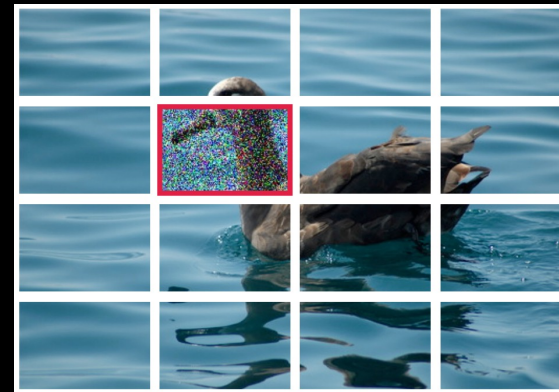
Transformers Provably Learn Algorithmic Solutions for Graph Connectivity, But Only with the Right Data, Qilin Ye, Deqing Fu, Robin Jia, Vatsal Sharan, ICML, 2026

Beyond Specific Tasks: What classes of functions do Transformers prefer to learn?



Evaluate model on original input

If model's predictions change, model is **sensitive** (as in Boolean function analysis) to that token



Evaluate model on perturbation to random token

Across data modalities, transformers learn lower sensitivity functions

High Level Message

- Tools from theoretical computer science can say interesting things about LLMs
- Often without very complicated proofs!
- Hard part: figuring out the right questions to ask
 - One of the best approaches: conversations and collaboration!



Can we use Transformers to discover data structures from scratch?



Omar Salemhamed
(Universite de Montreal/MILA)



Laurent Charlin
(HEC Montreal/MILA)

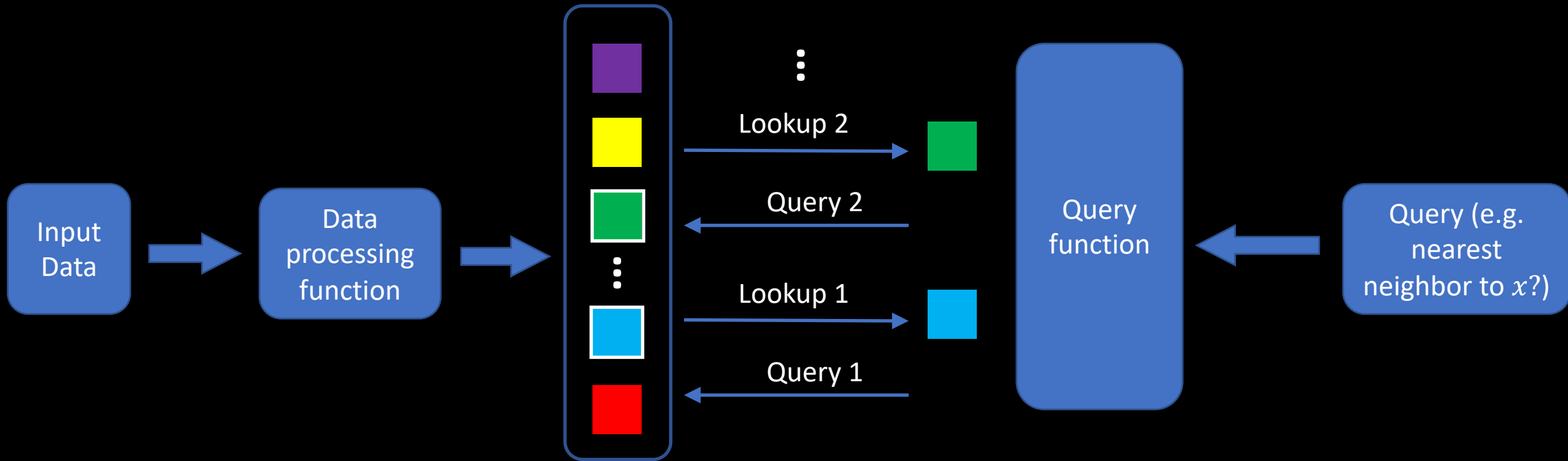


Shivam Garg
(MSR NYC)



Greg Valiant
(Stanford)

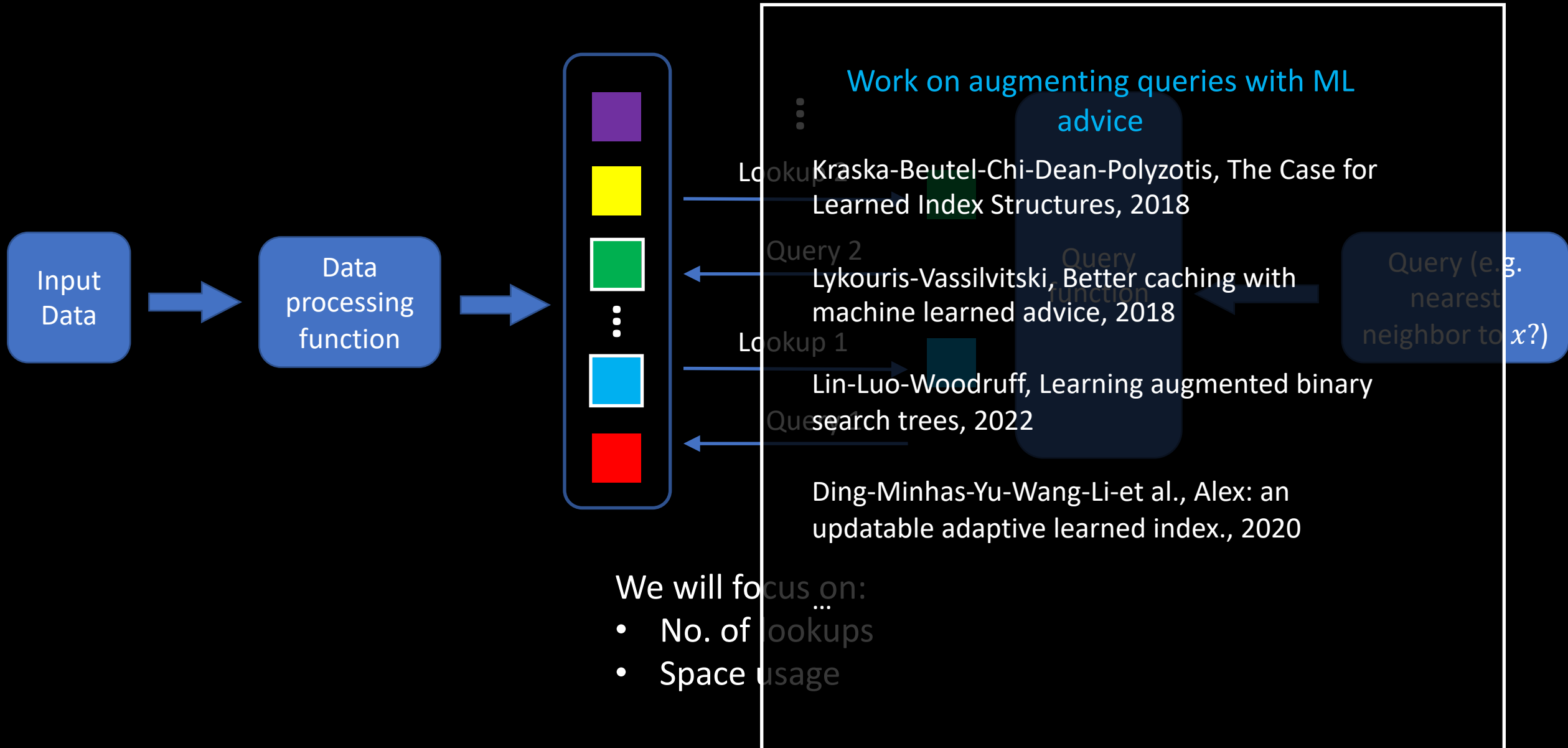
Data structures (think nearest neighbor lookup in 1D)



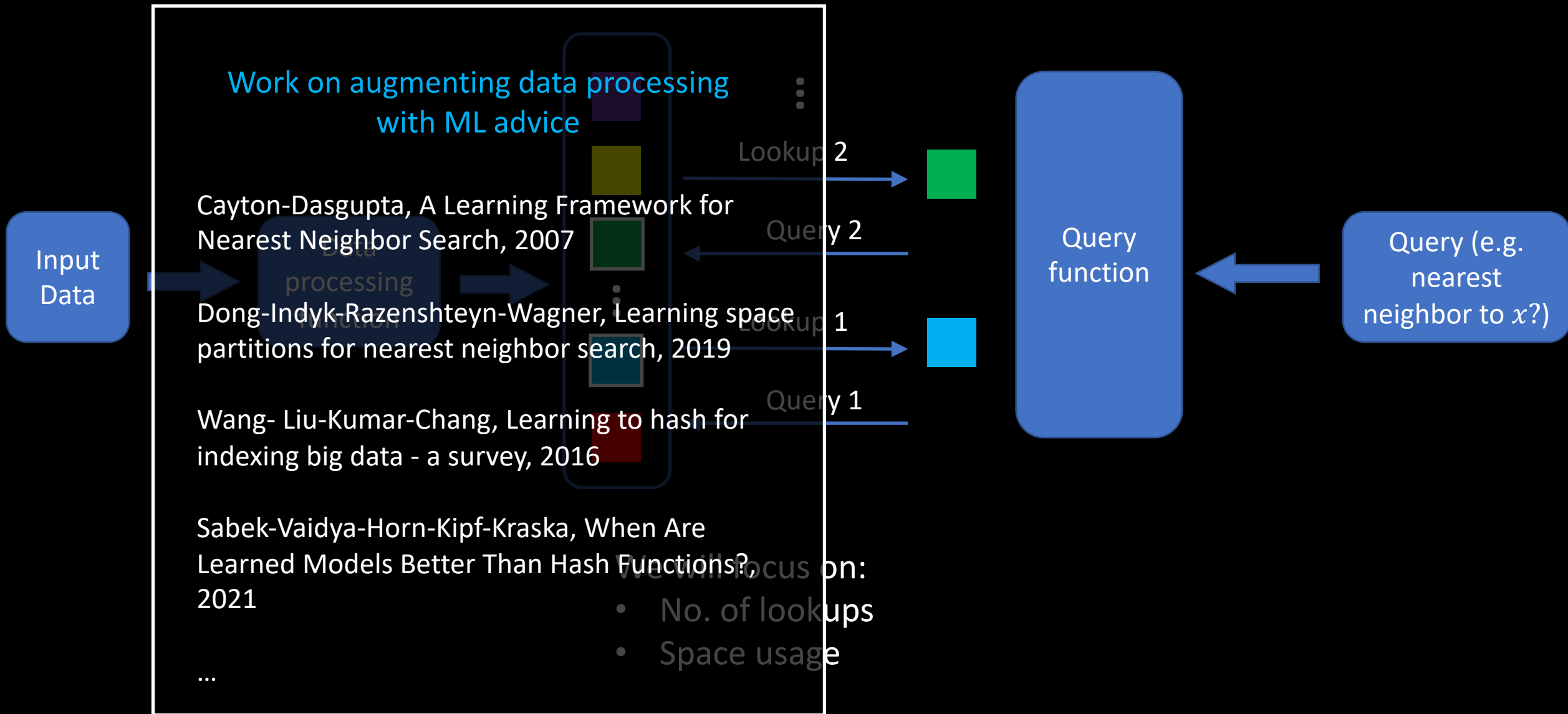
We will focus on:

- No. of lookups
- Space usage

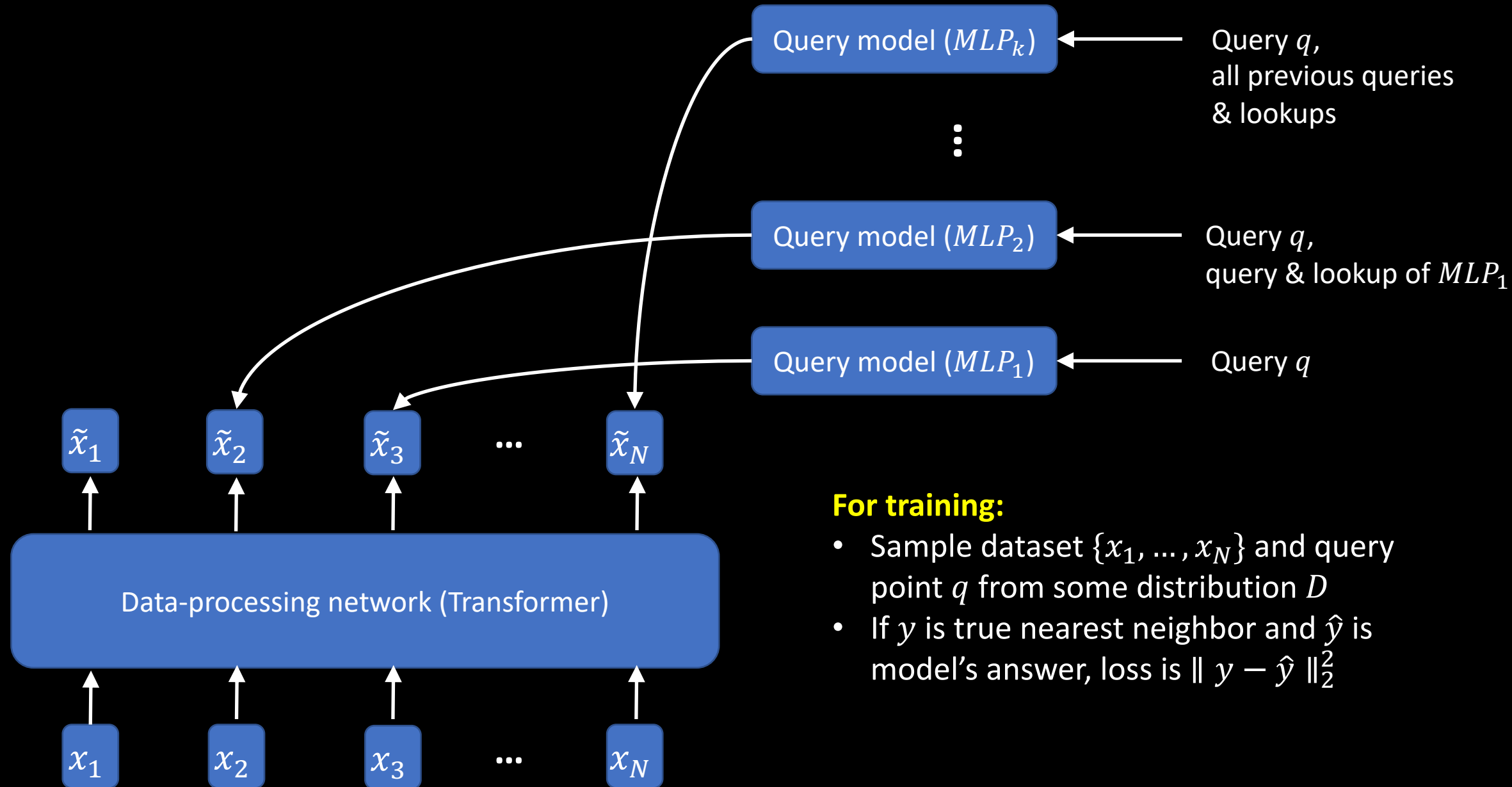
Recent work has tried to augment data structures with ML



Recent work has tried to augment data structures with ML



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

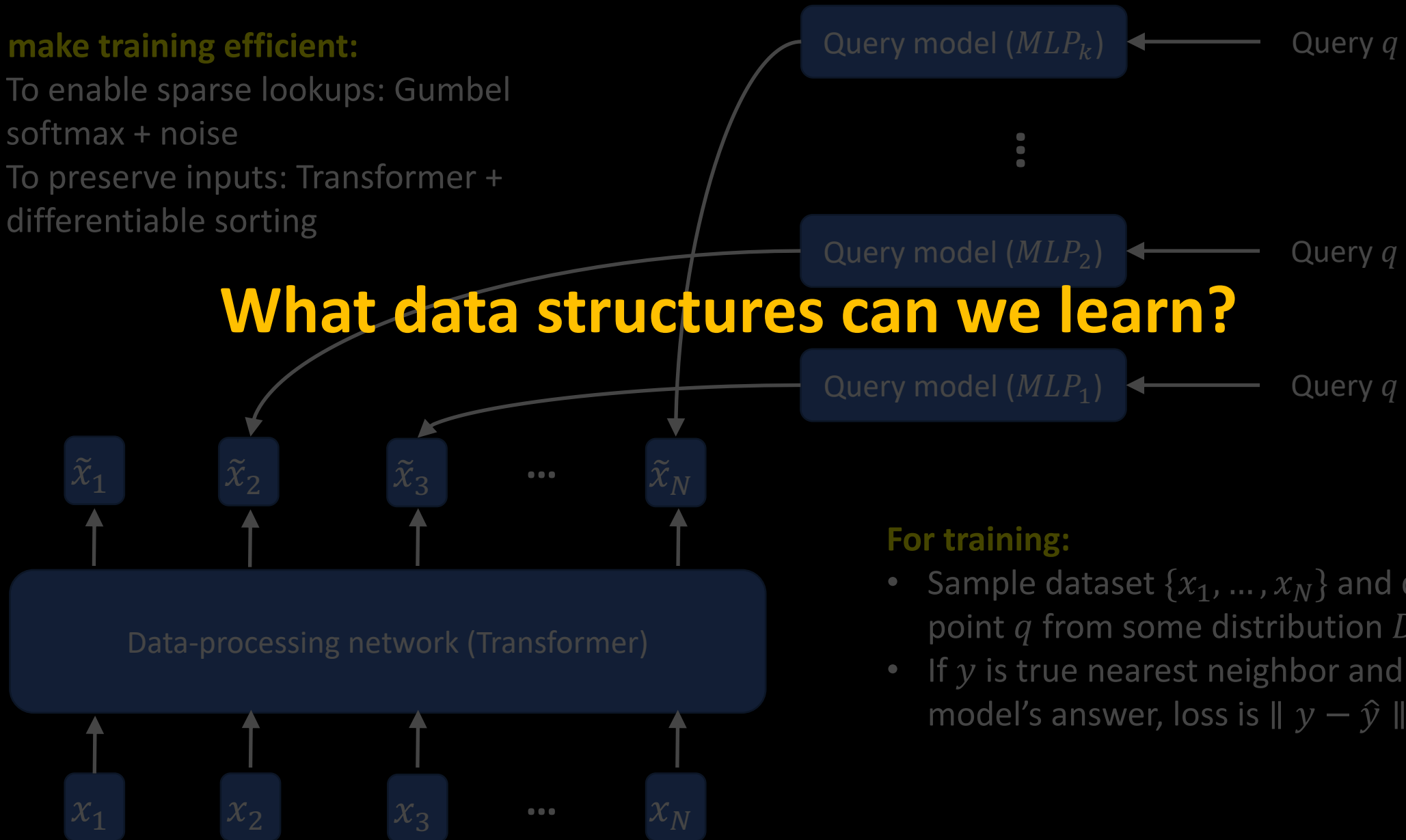


What if we learn everything end to end with ML, with no algorithmic priors?

To make training efficient:

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

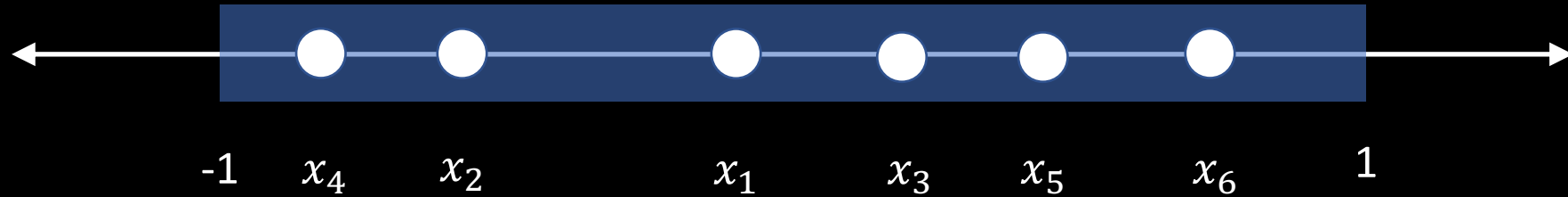
What data structures can we learn?



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 $\|y - \hat{y}\|_2^2$

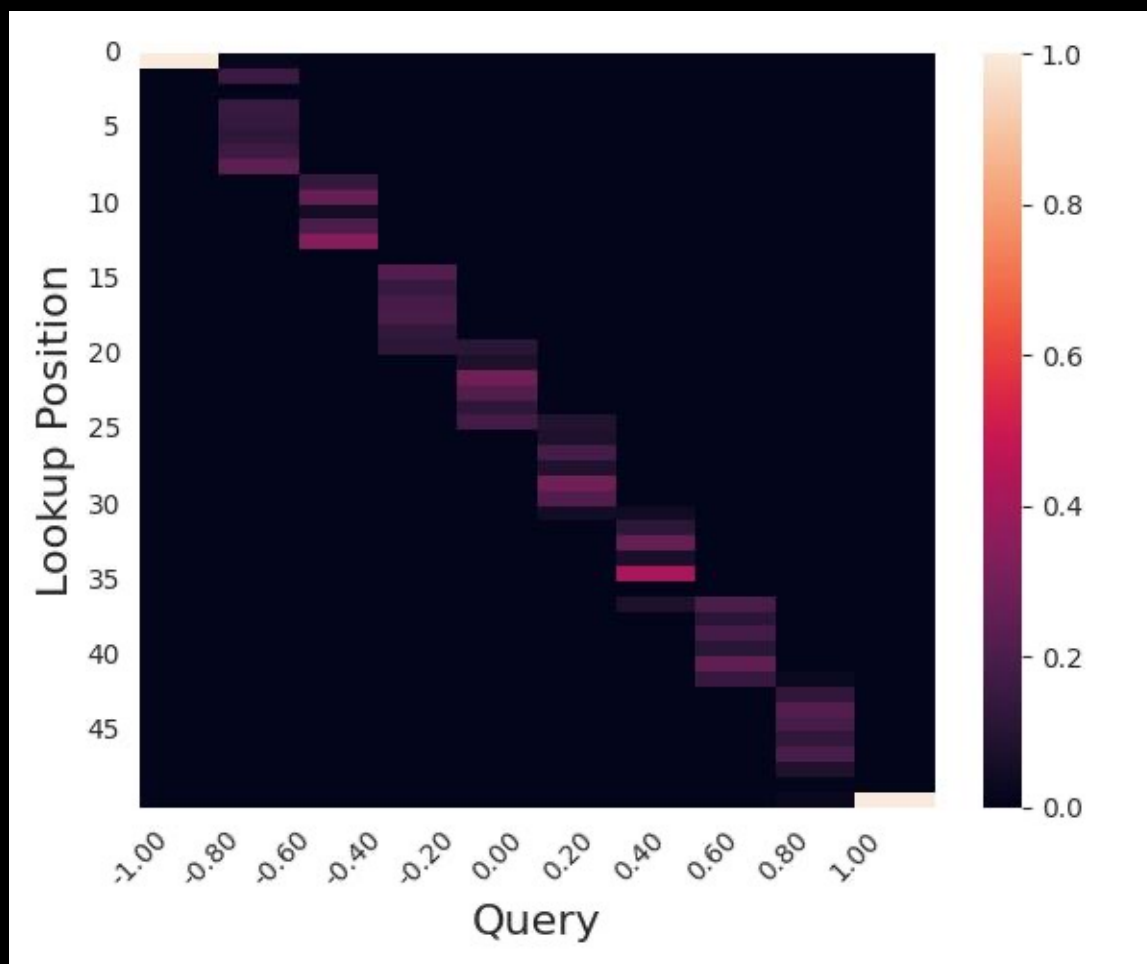
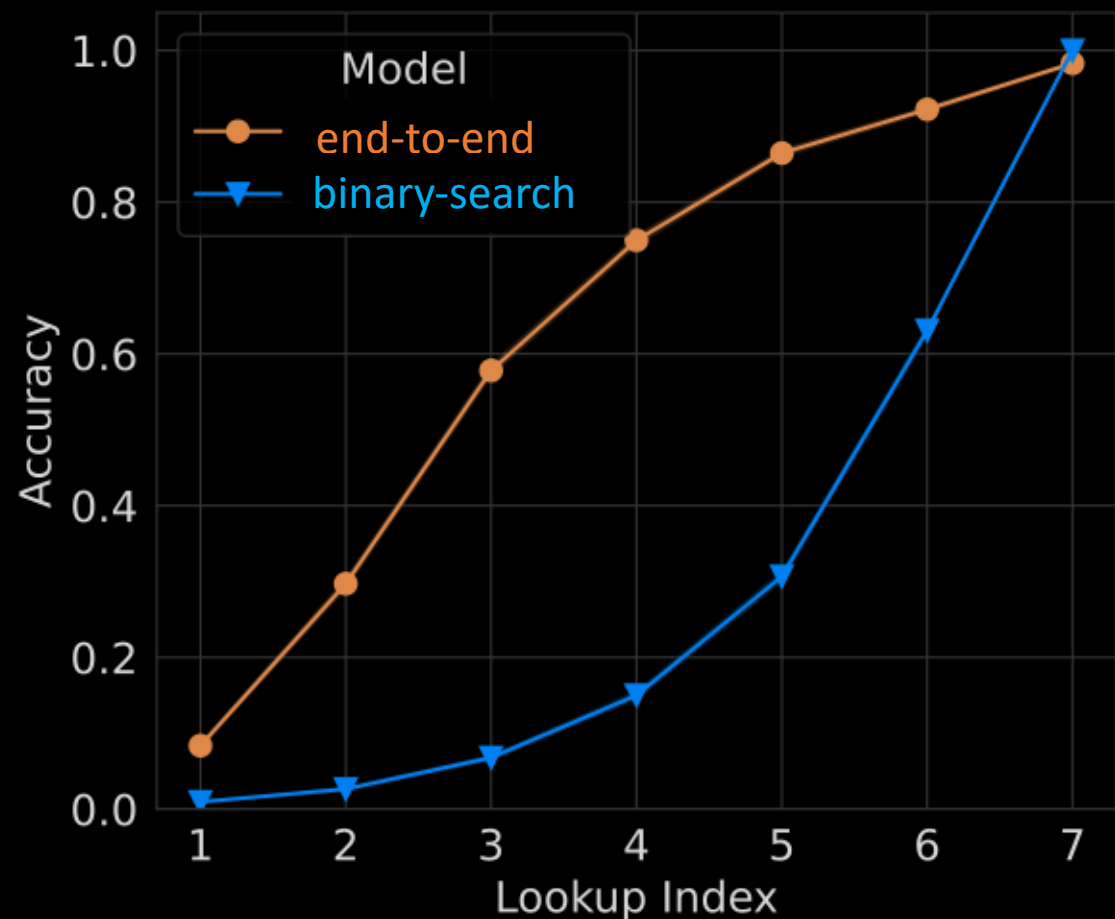
Uniform distribution in 1D



Model trained on this distribution:

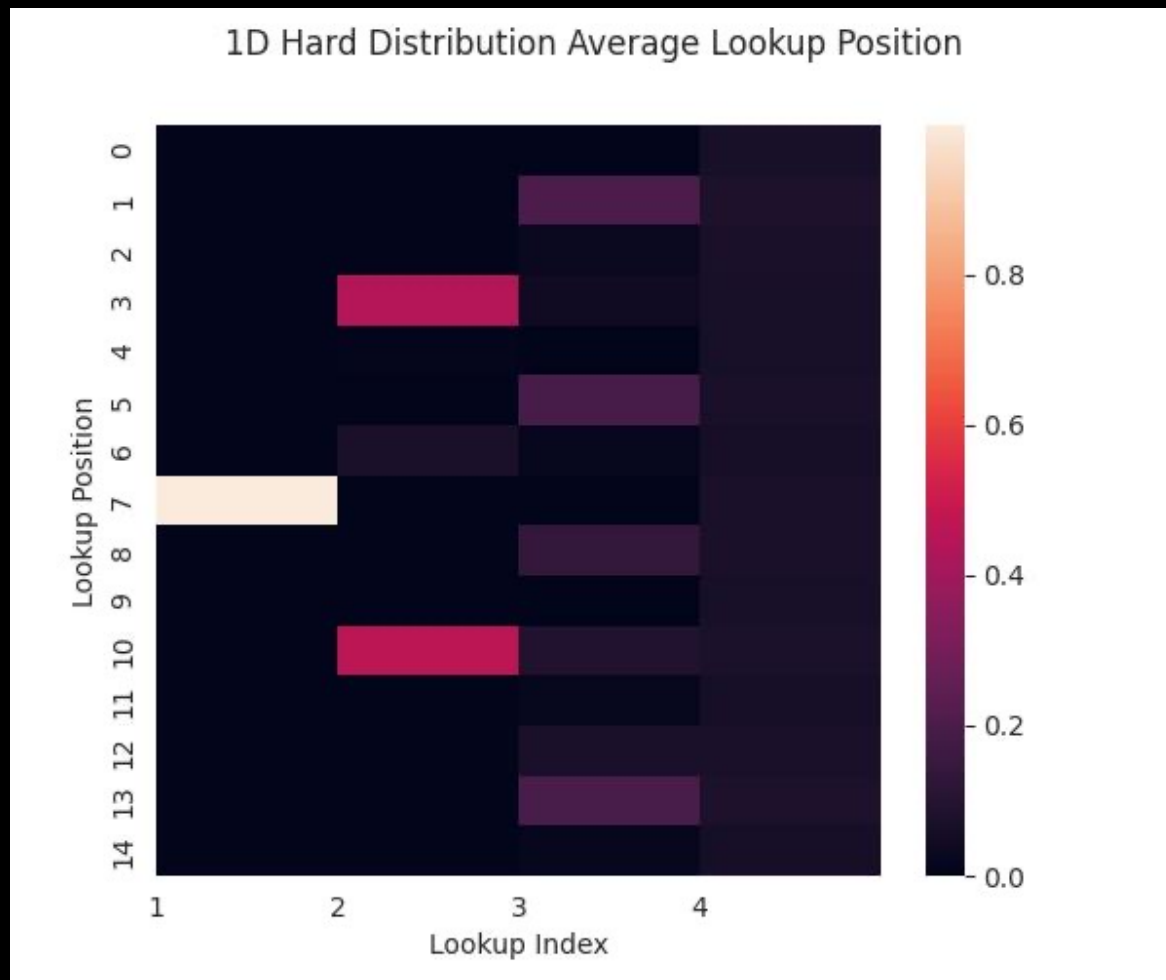
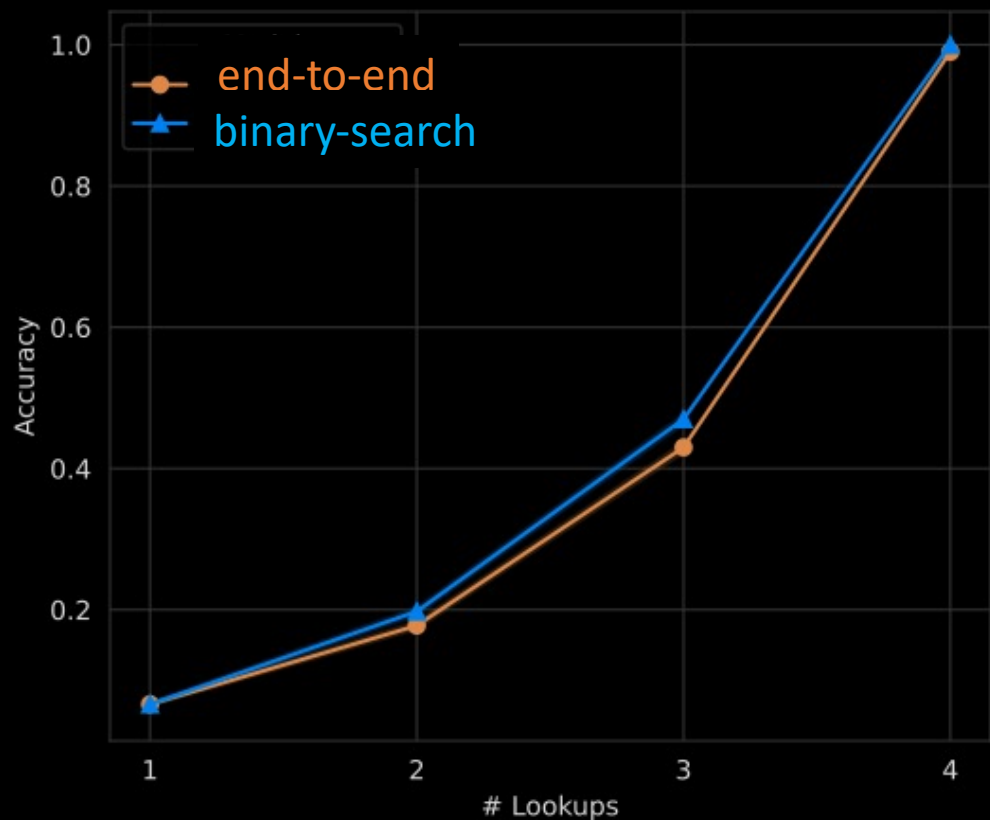
- **Learns to sort, with small error**
- **Does better than binary search**

Model outperforms binary search



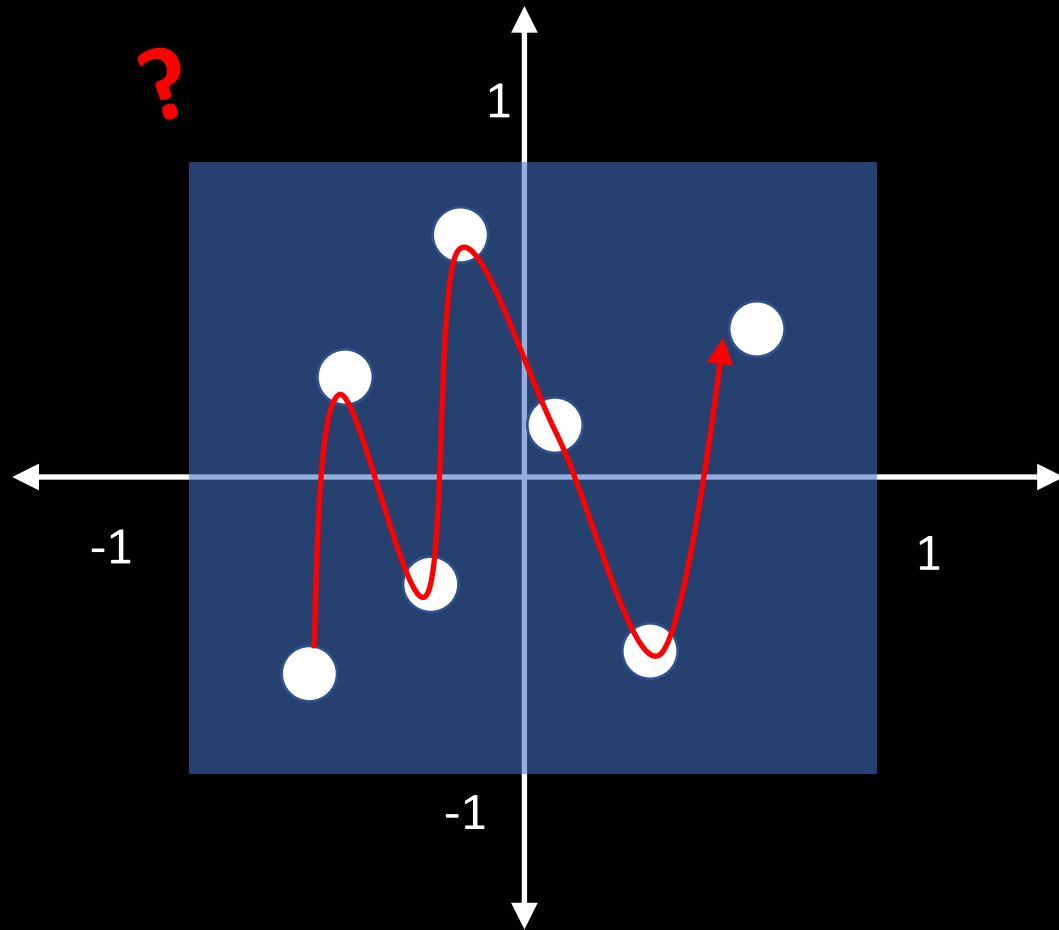
Query model begins search not far from nearest neighbor

Harder 1D distribution where quantiles don't concentrate

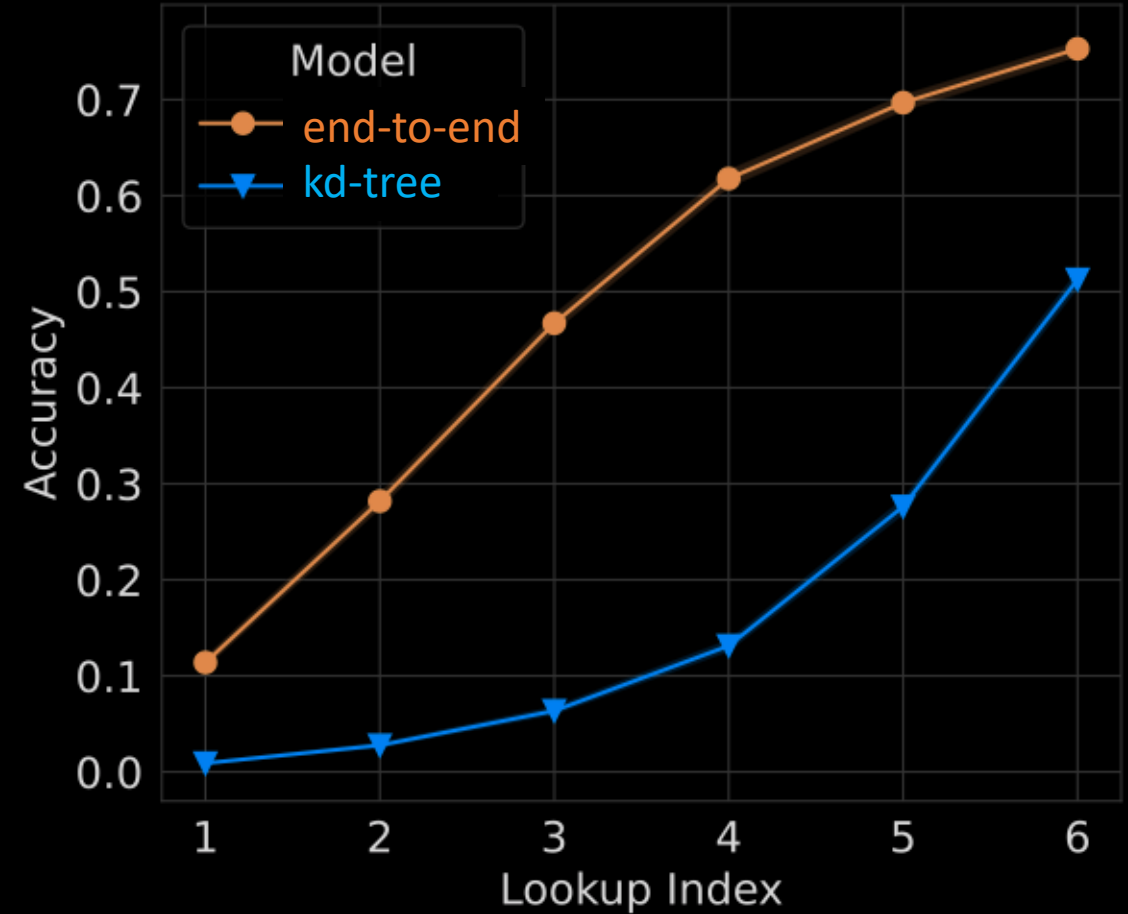
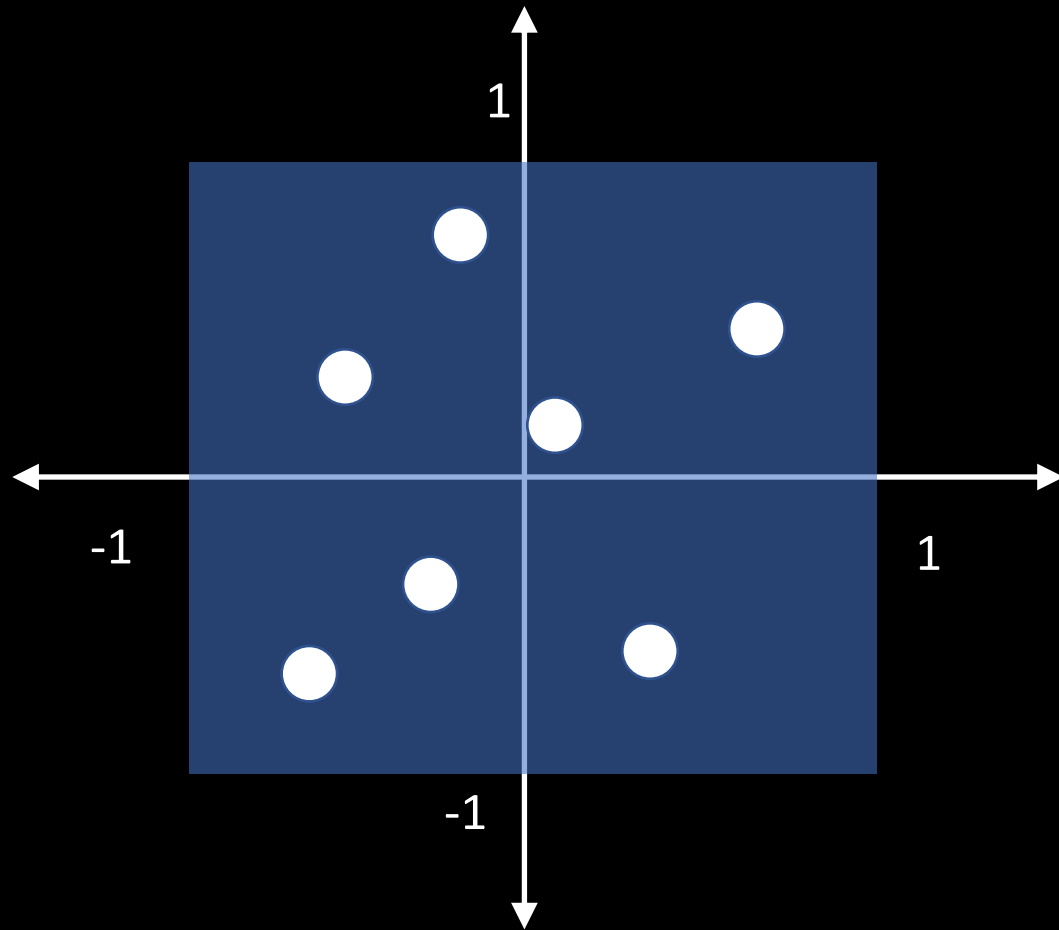


Model learns binary search!

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

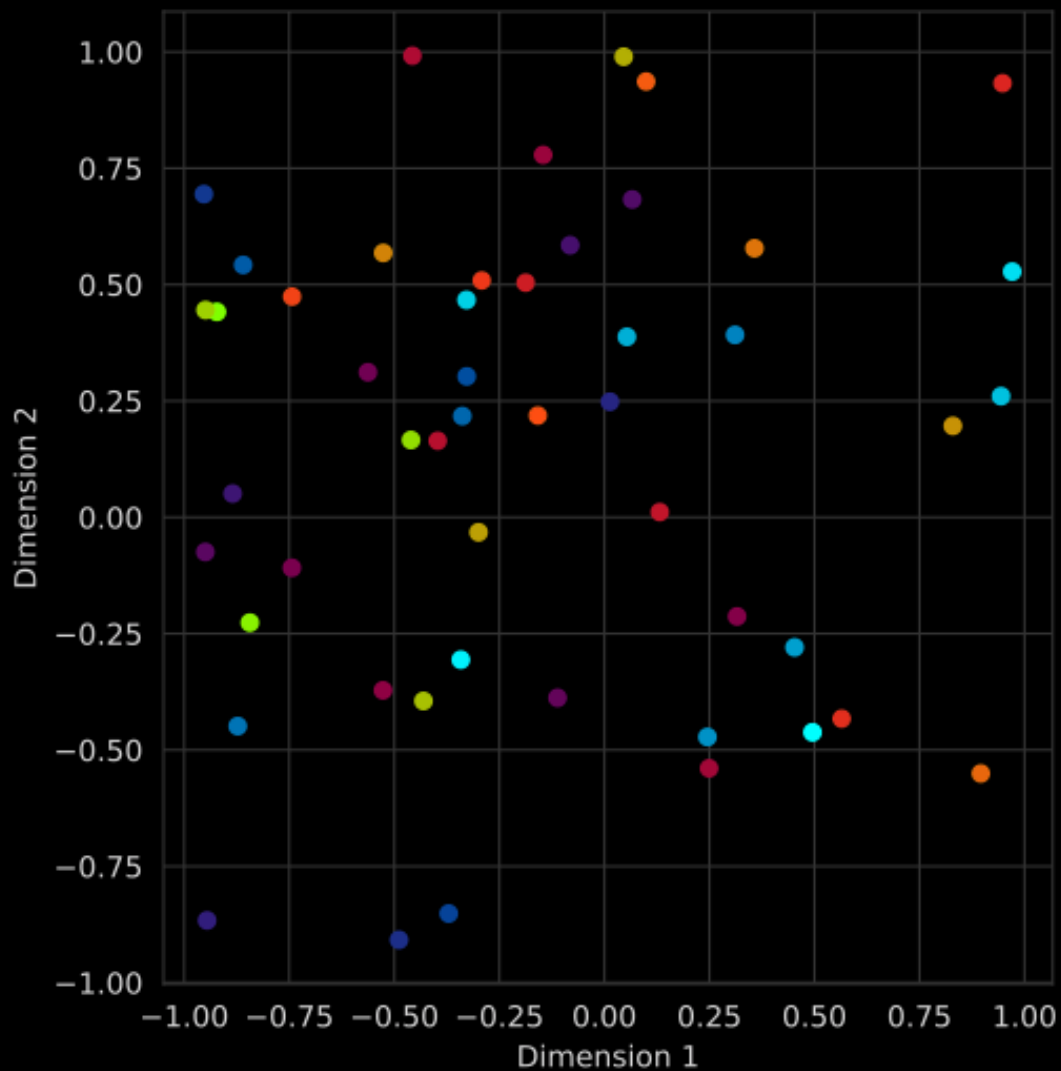


Uniform distribution in 2D: Outperforms kd-trees

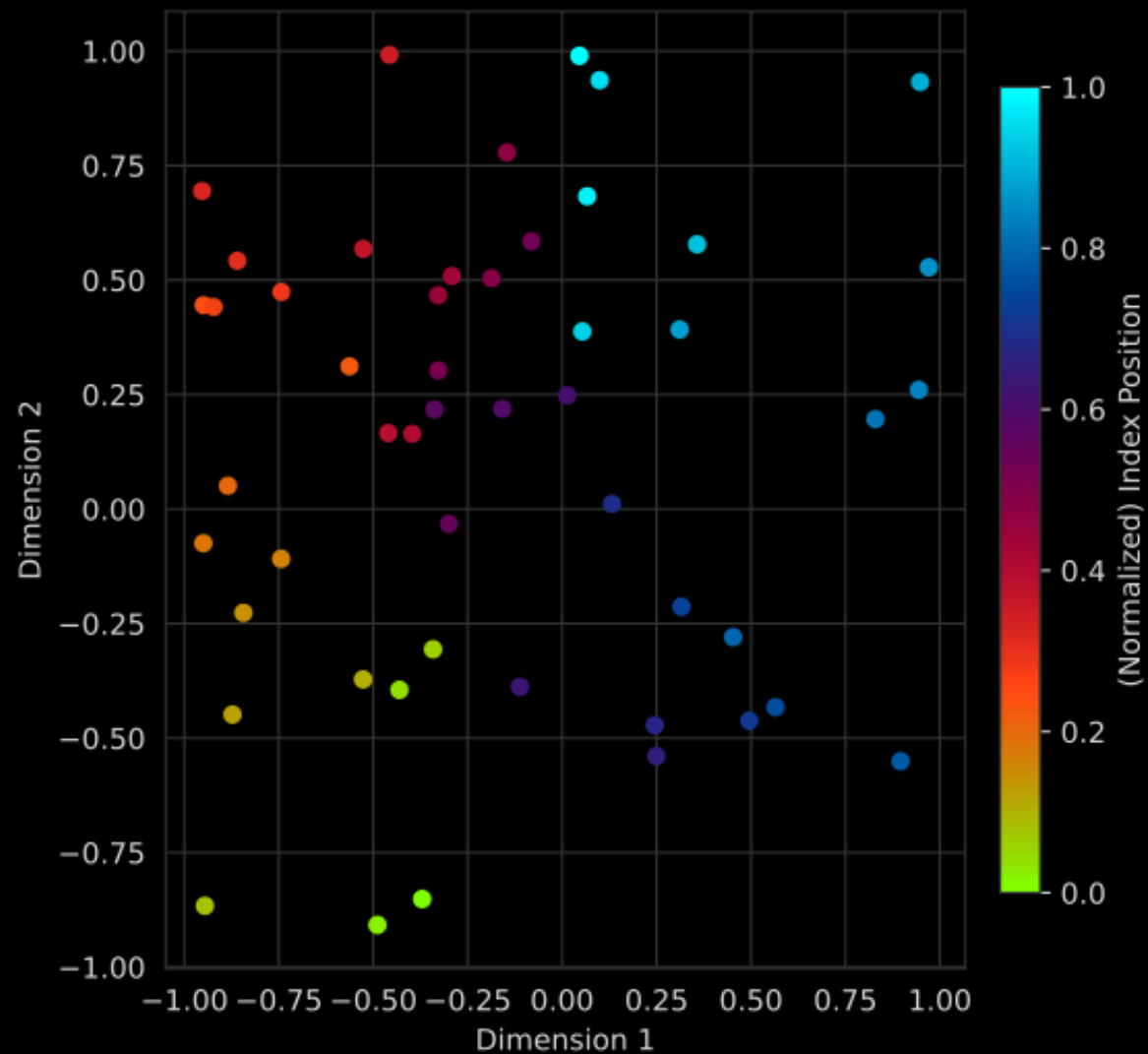


Model learns to index nearby points together

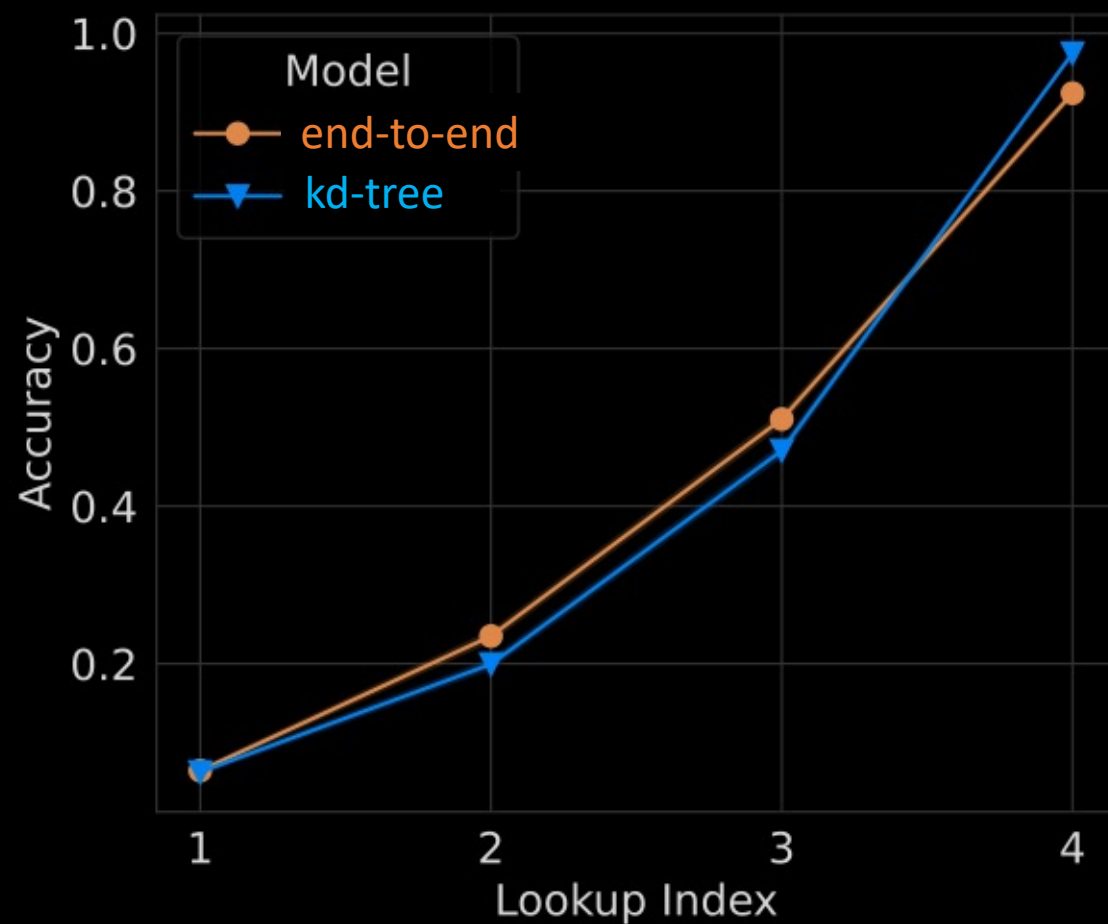
Original points colored by index position



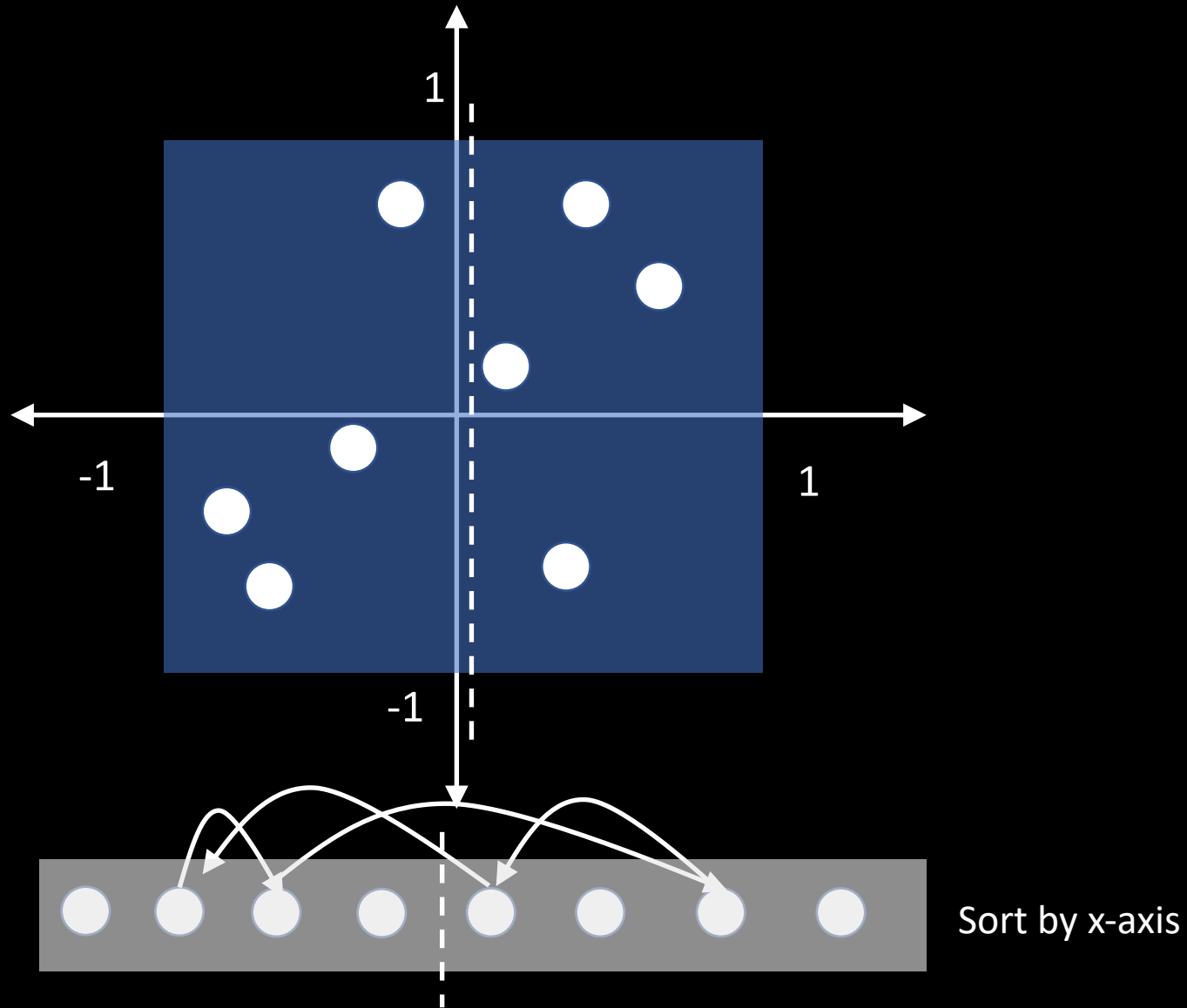
Transformed points colored by index position



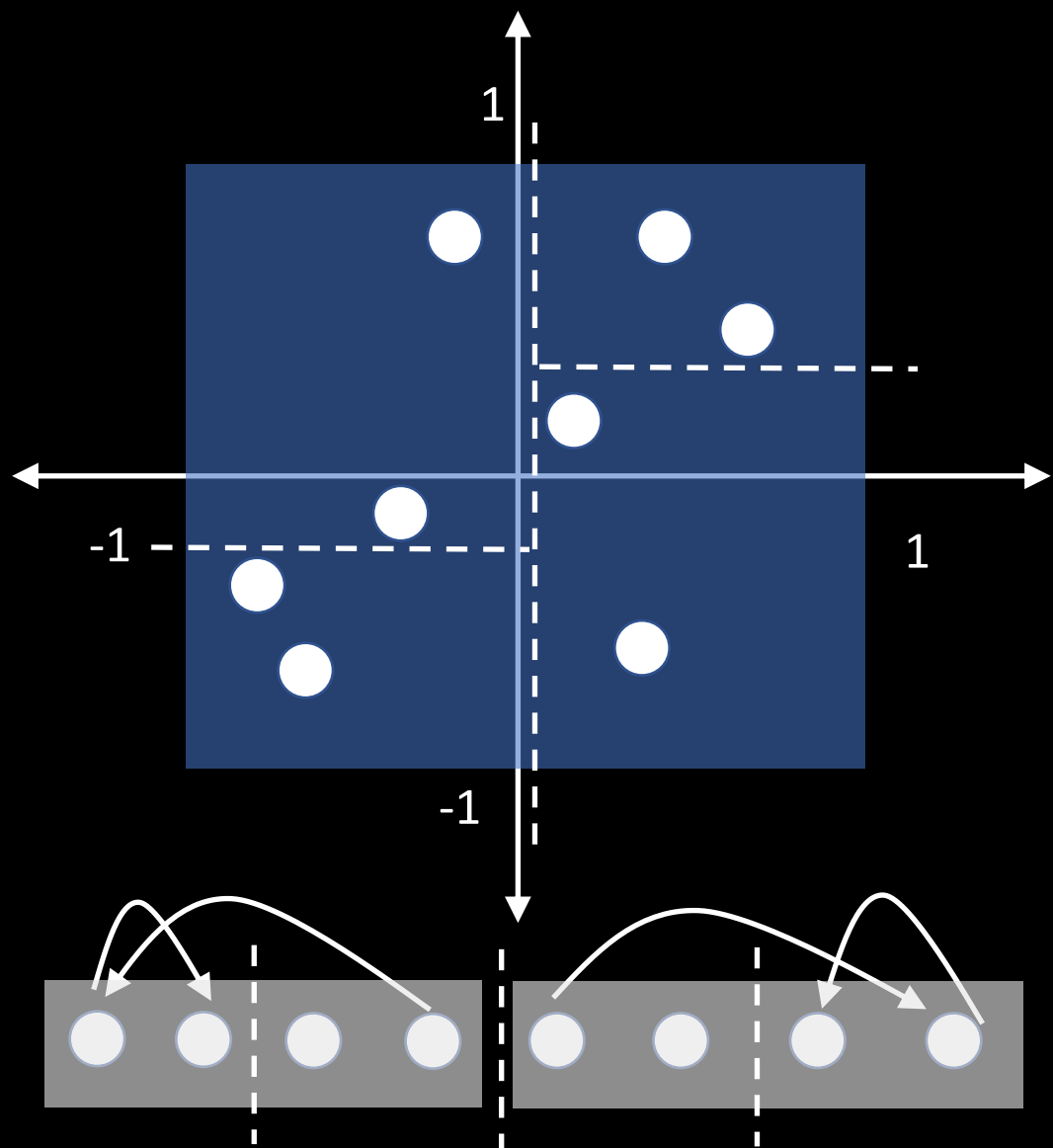
Hard distribution in 2D: Matches kd-trees



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

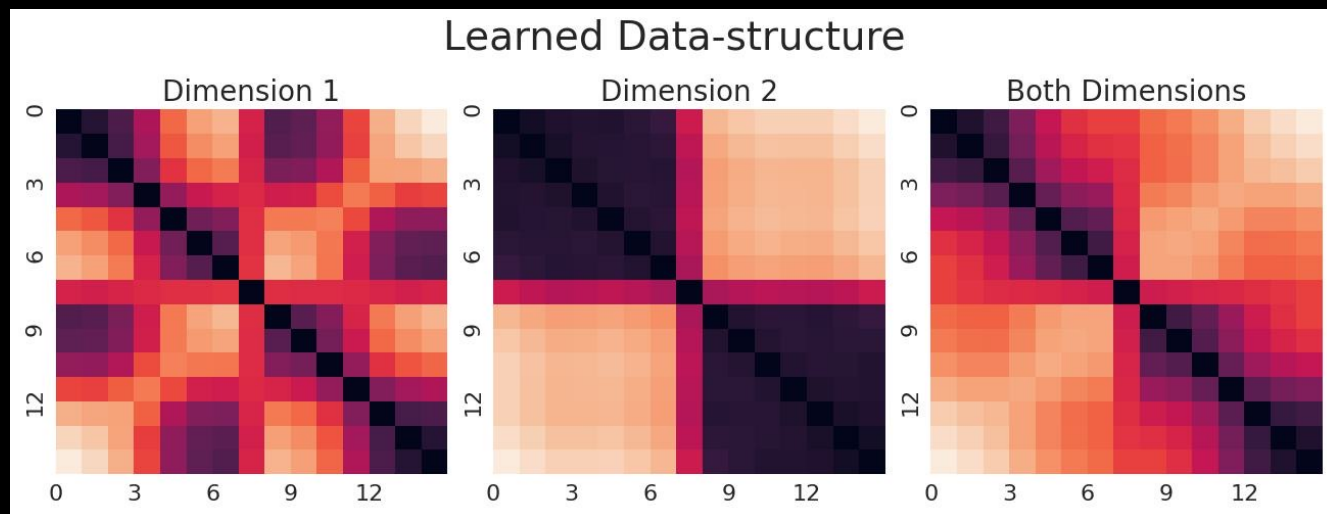
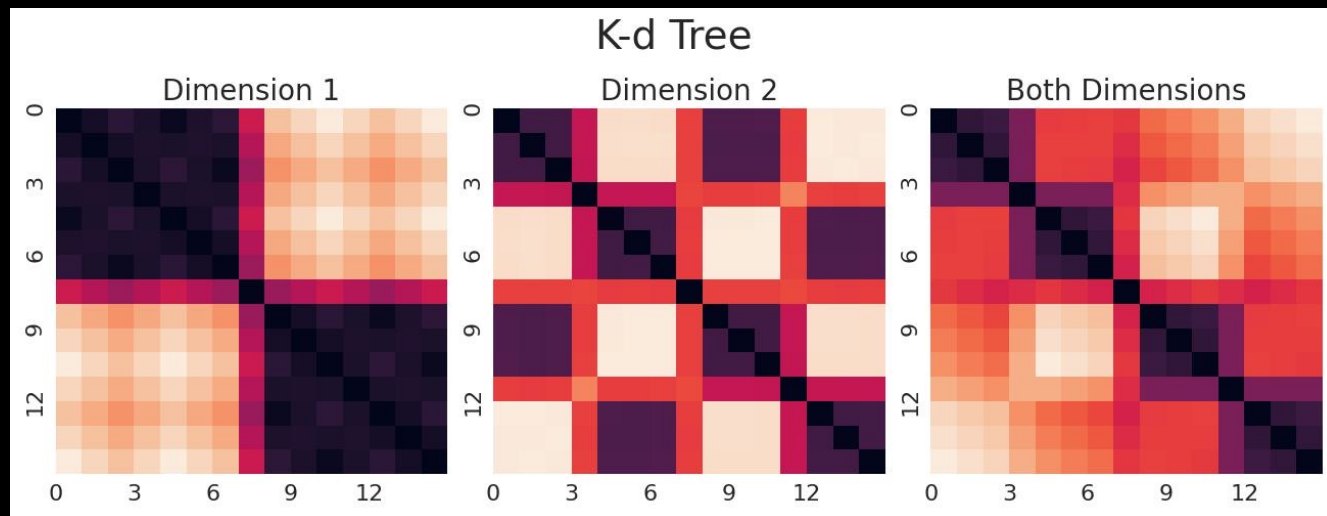
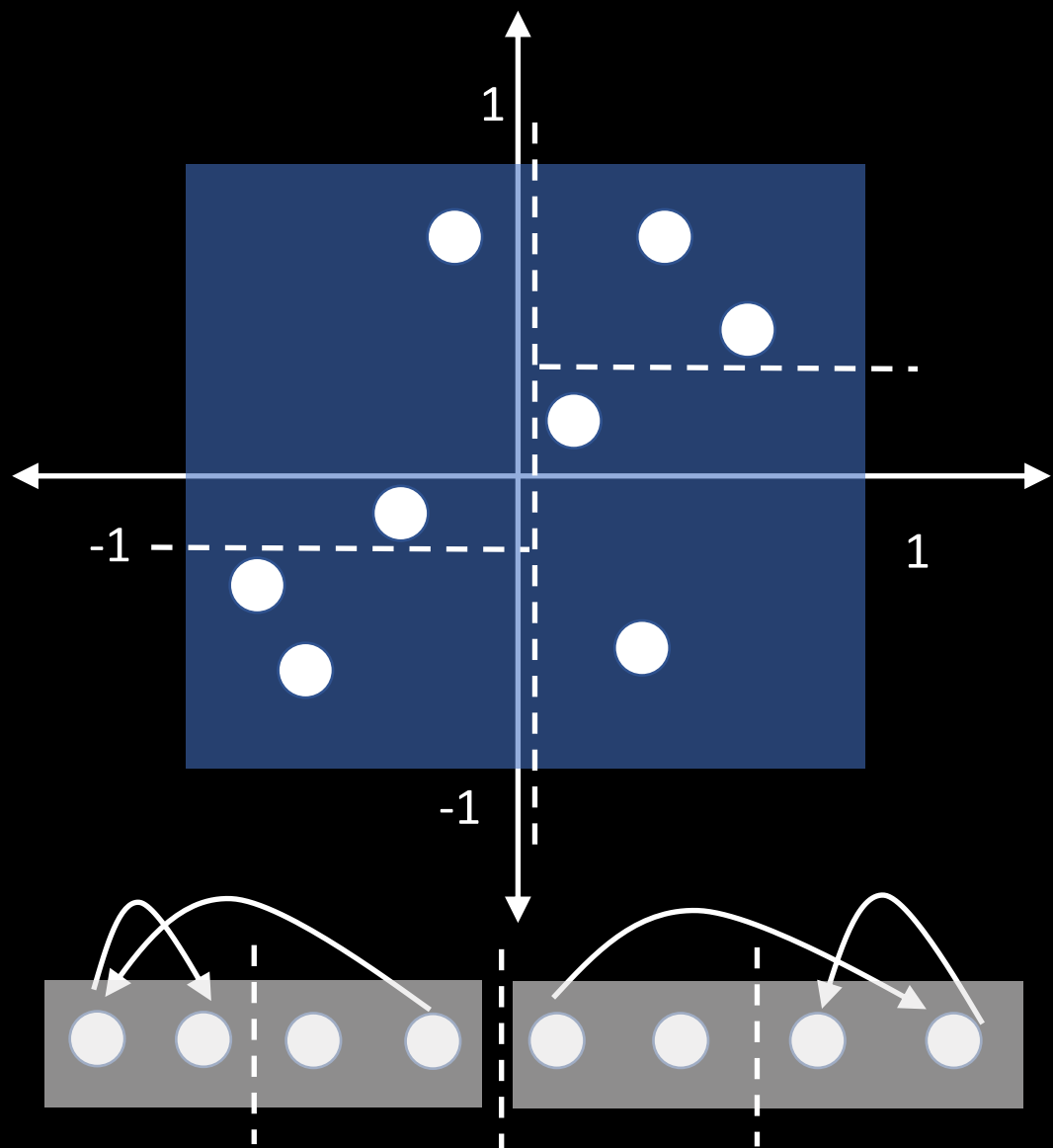


Can see that the model is essentially recovering a kd-tree!



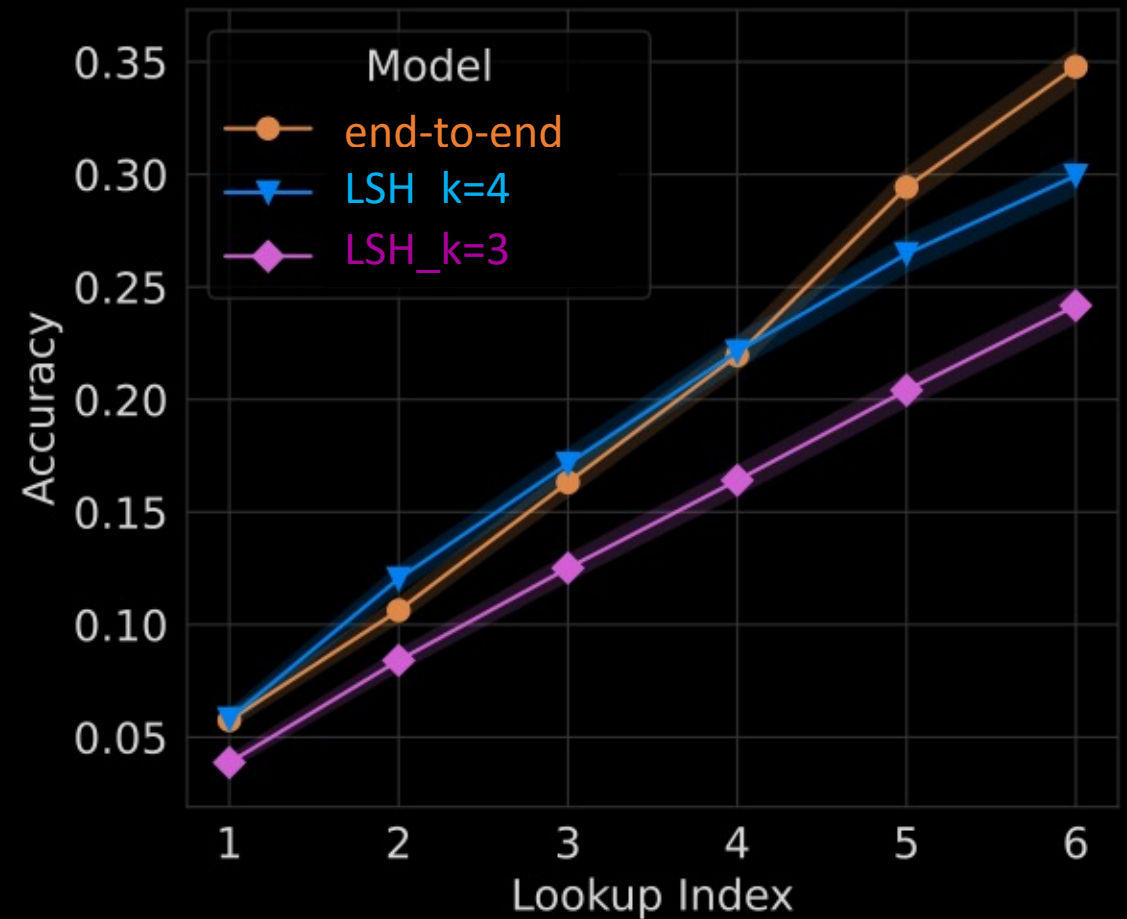
Sort each half
by y-axis

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

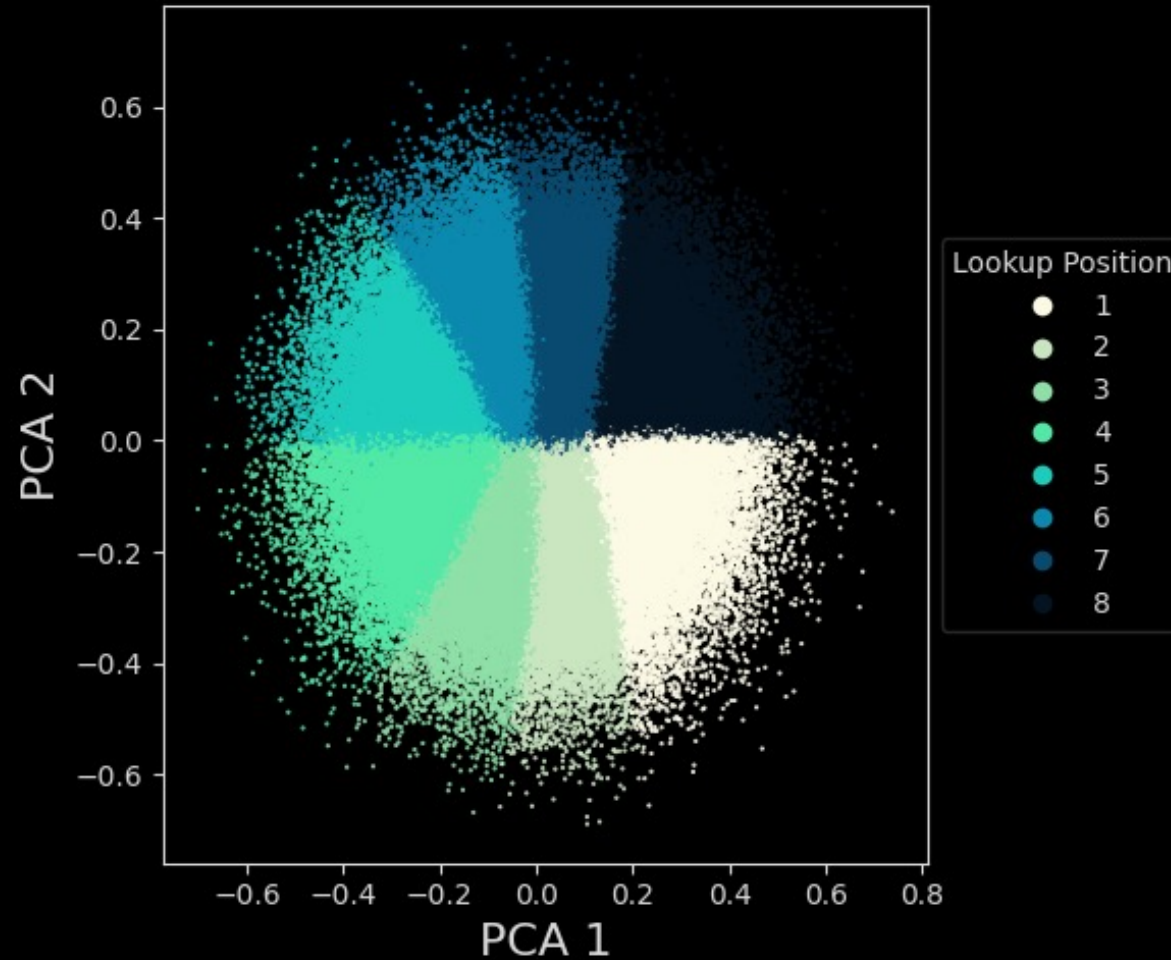


Uniform distribution in 30D: Matches LSH

- 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

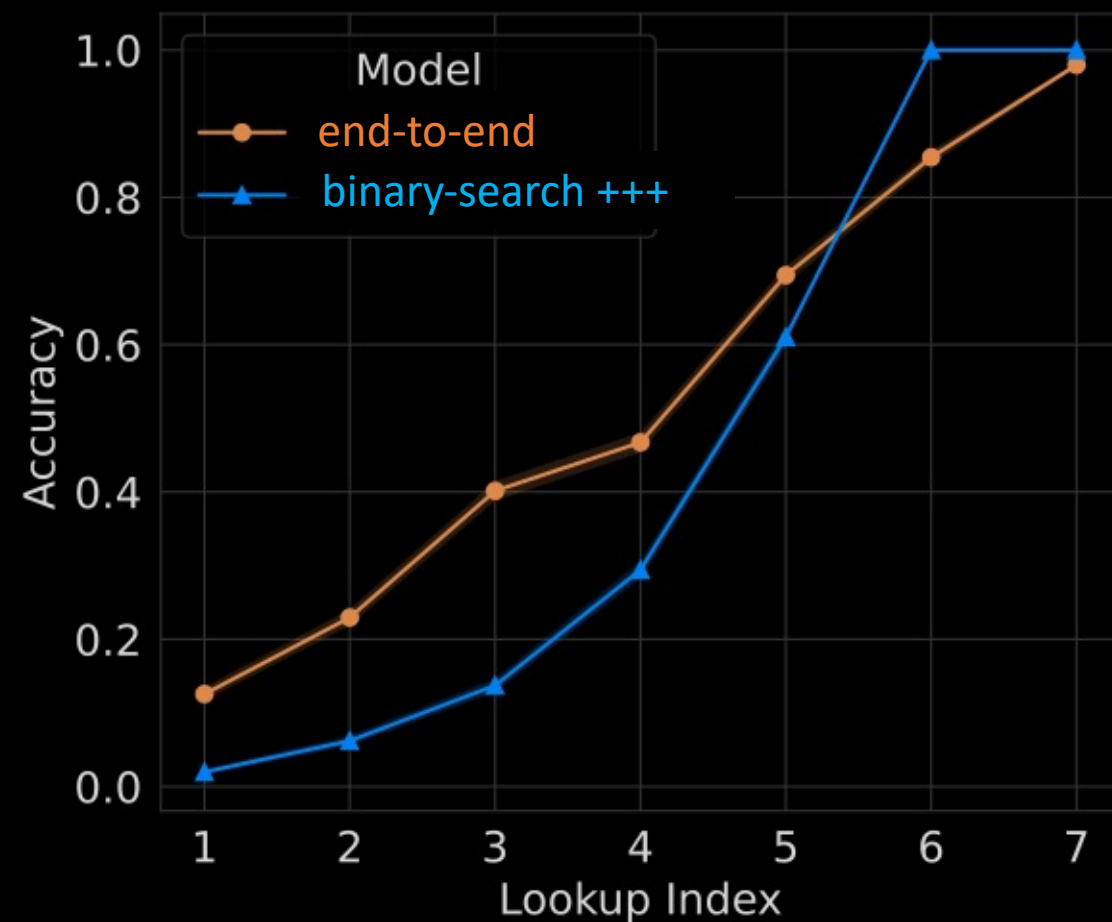
x_N

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

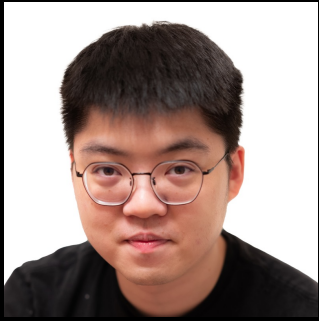
- Model also learns to use extra space
- We also show we can learn data structures for frequency estimation in a data stream, recovering/outperforming count-sketch

Models outperform data-independent baselines

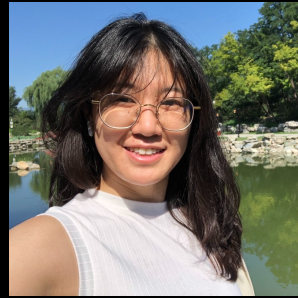
- Also consider settings with power-law distributions etc.

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

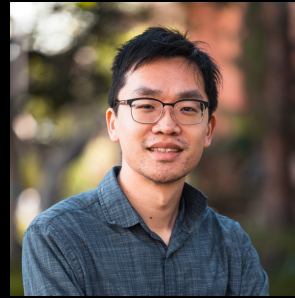
- *Can we use these to understand tradeoffs in theory, build better strategies for high-dimensional NN search and other data structure problems?*
- *Can we use a similar methodology for algorithm discovery under resource constraints more generally, for example to discover new optimization techniques in an end-to-end fashion?*



Deqing Fu



Tianqi Chen



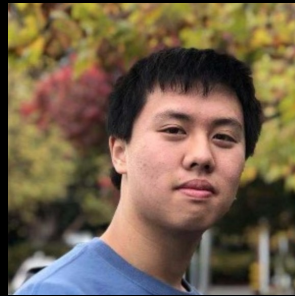
Robin Jia



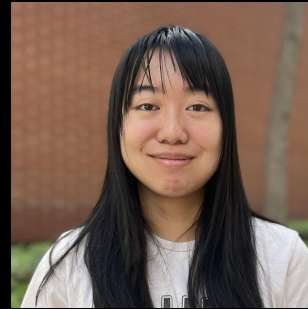
Tianyi Zhou



Bhavya Vasudeva



Elliot Kau



You-Qi Huang



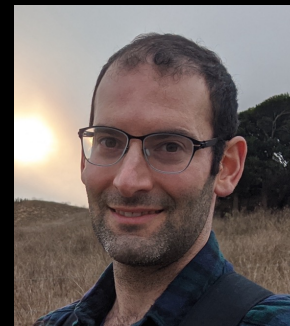
Omar Salemhamed



Laurent Charlin



Shivam Garg

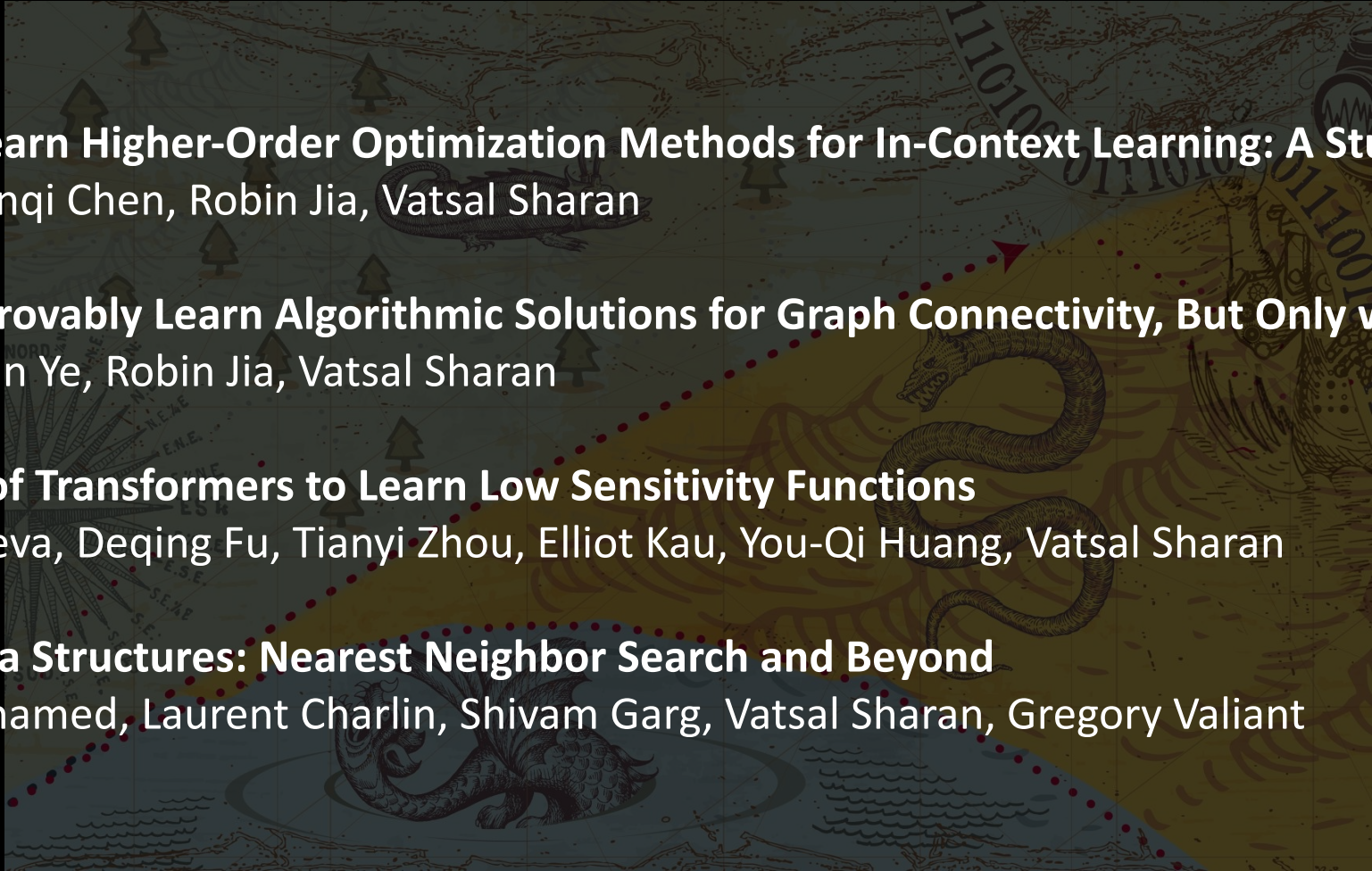


Greg Valiant

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
 - **Transformers Provably Learn Algorithmic Solutions for Graph Connectivity, But Only with the Right Data**
Deqing Fu, Qilin Ye, Robin Jia, Vatsal Sharan
 - **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