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

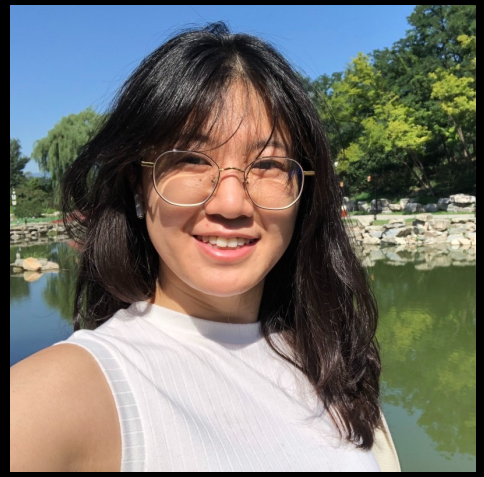
Vatsal Sharan (USC)

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

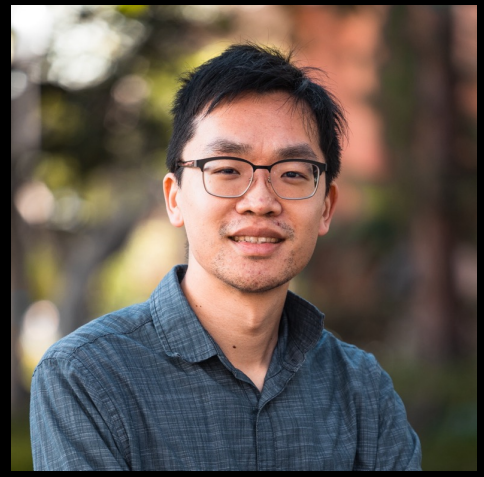
- How can we use understanding of computational and information theoretic landscape to understand Transformers?
- How can we use Transformers to understand and discover algorithms and data structures?

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

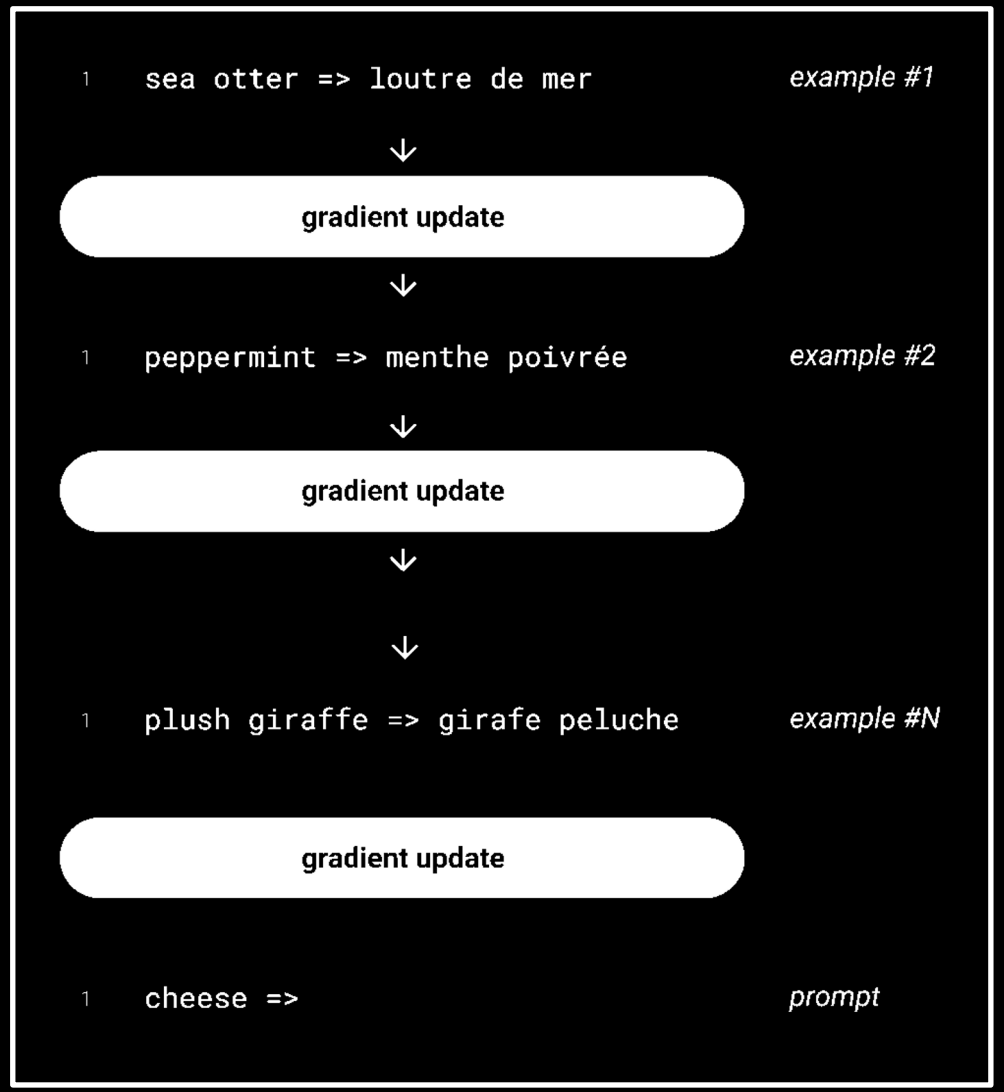
```
sea otter => loutre de mer

peppermint => menthe poivrée

plush girafe => girafe peluche

cheese =>
```

examples

prompt

In-context learning

vs.



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

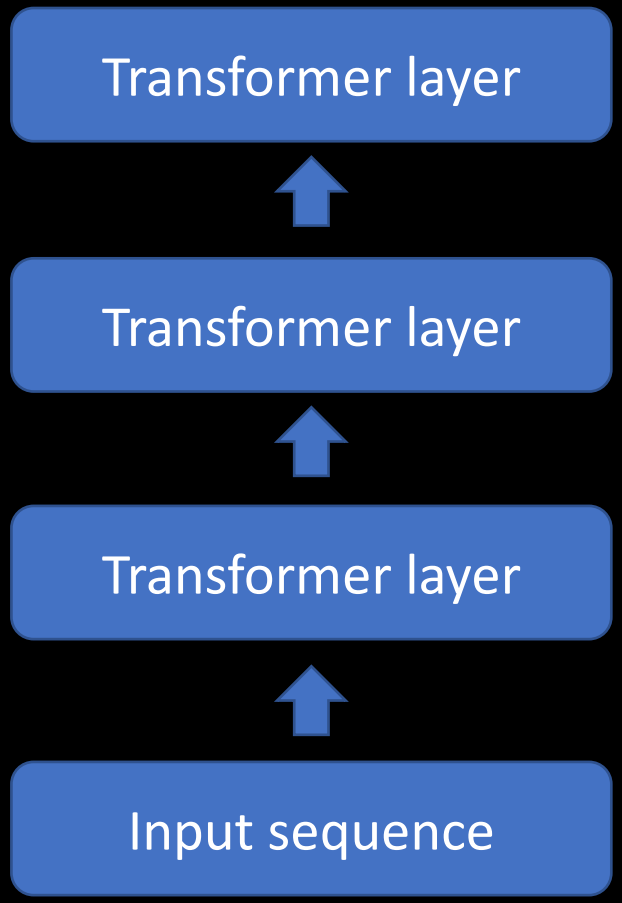
Linear models:

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

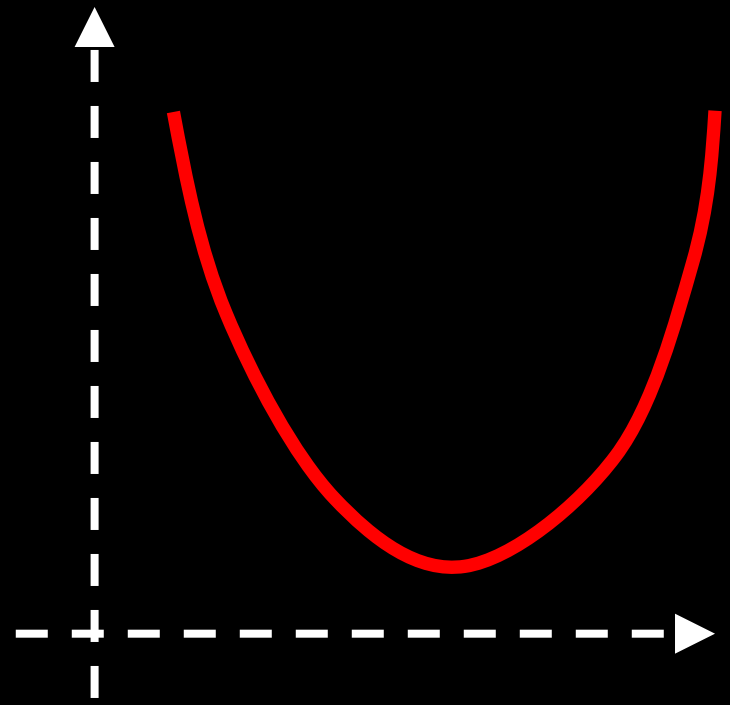
$x_2 = (-2, 2), y_2 = 8$

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

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



$\approx$



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

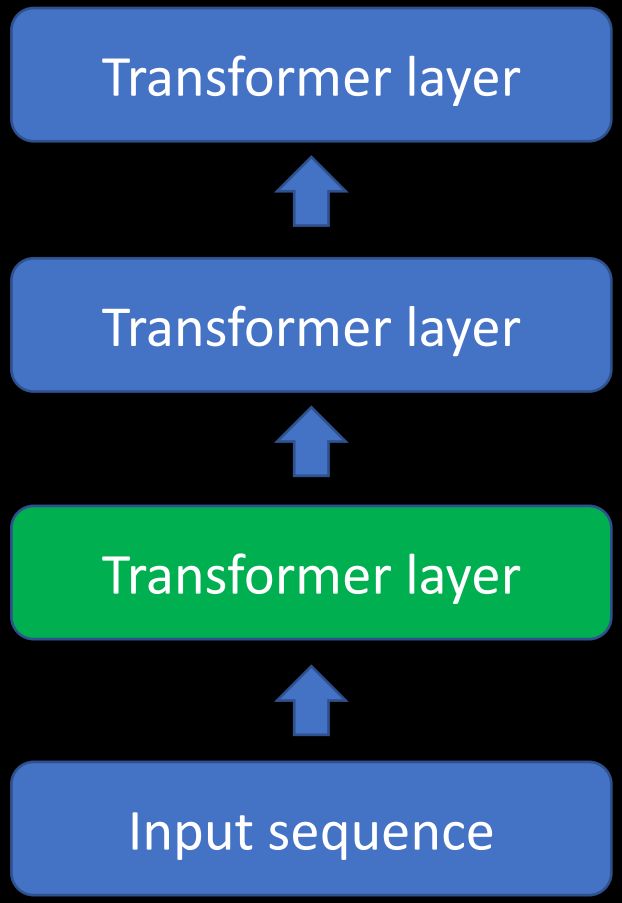
Linear models:

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

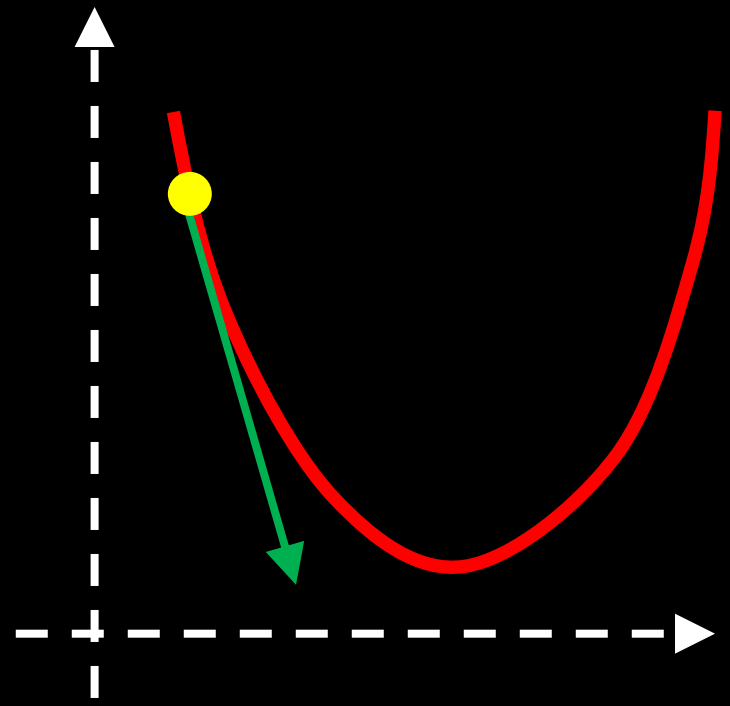
$x_2 = (-2, 2), y_2 = 8$

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

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



$\approx$



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

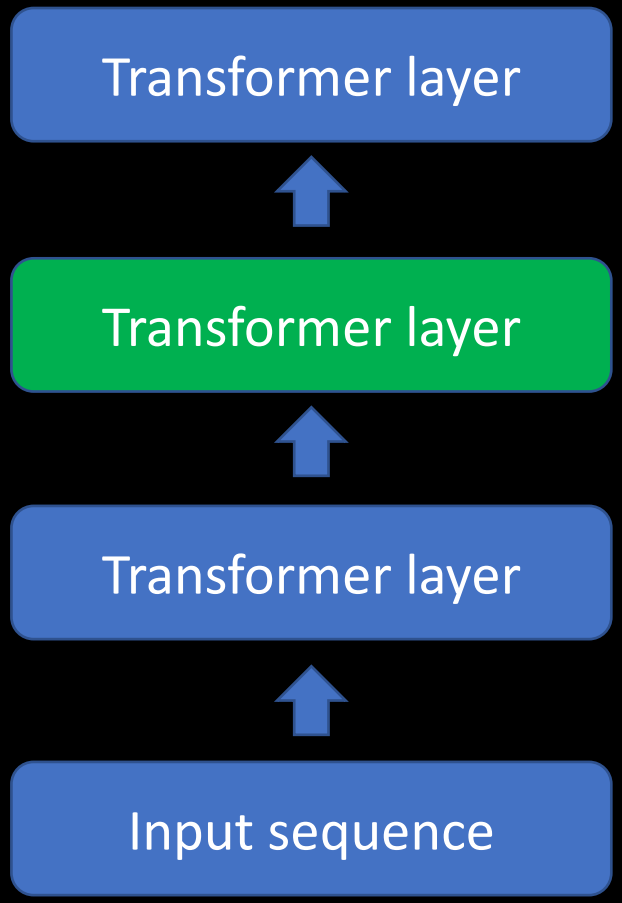
Linear models:

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

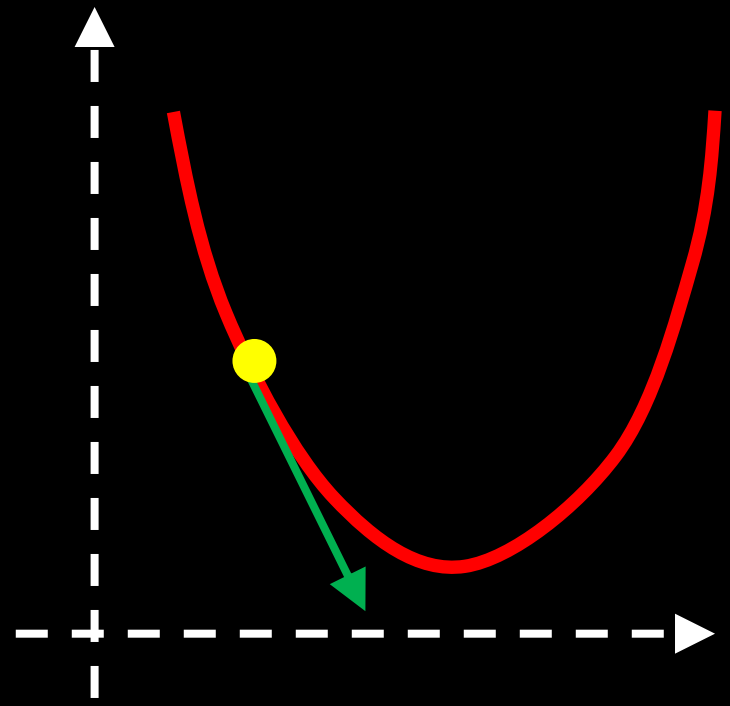
$x_2 = (-2, 2), y_2 = 8$

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

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



$\approx$



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

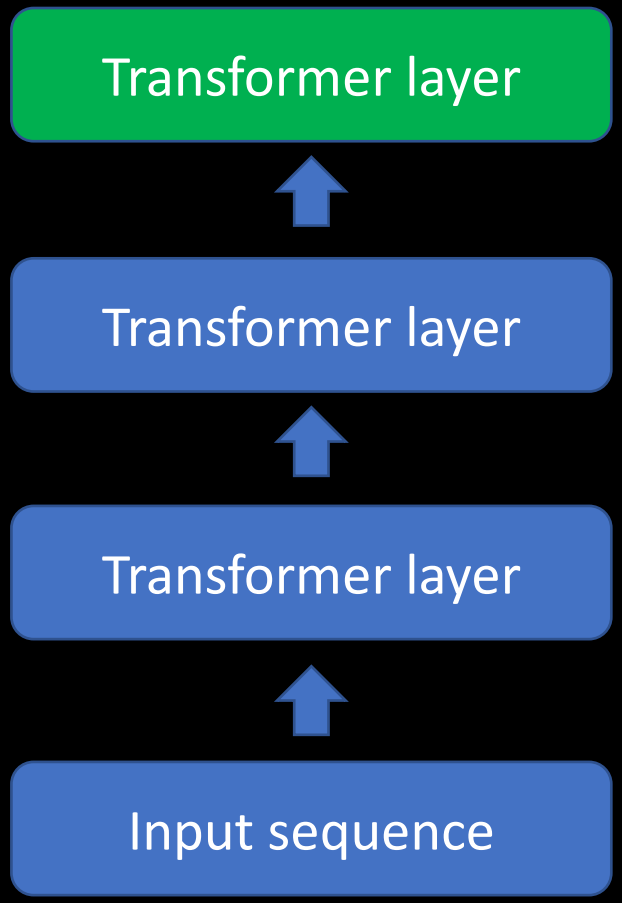
Linear models:

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

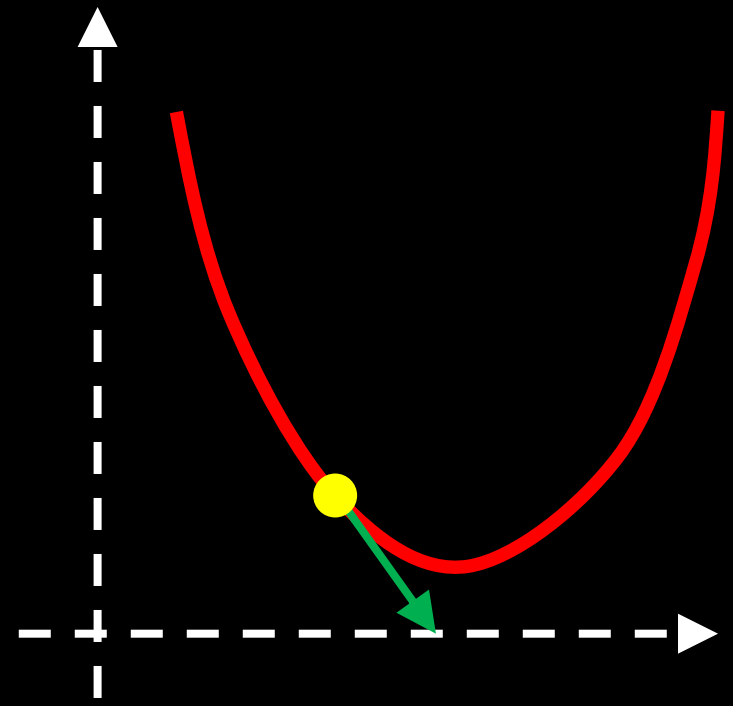
$x_2 = (-2, 2), y_2 = 8$

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

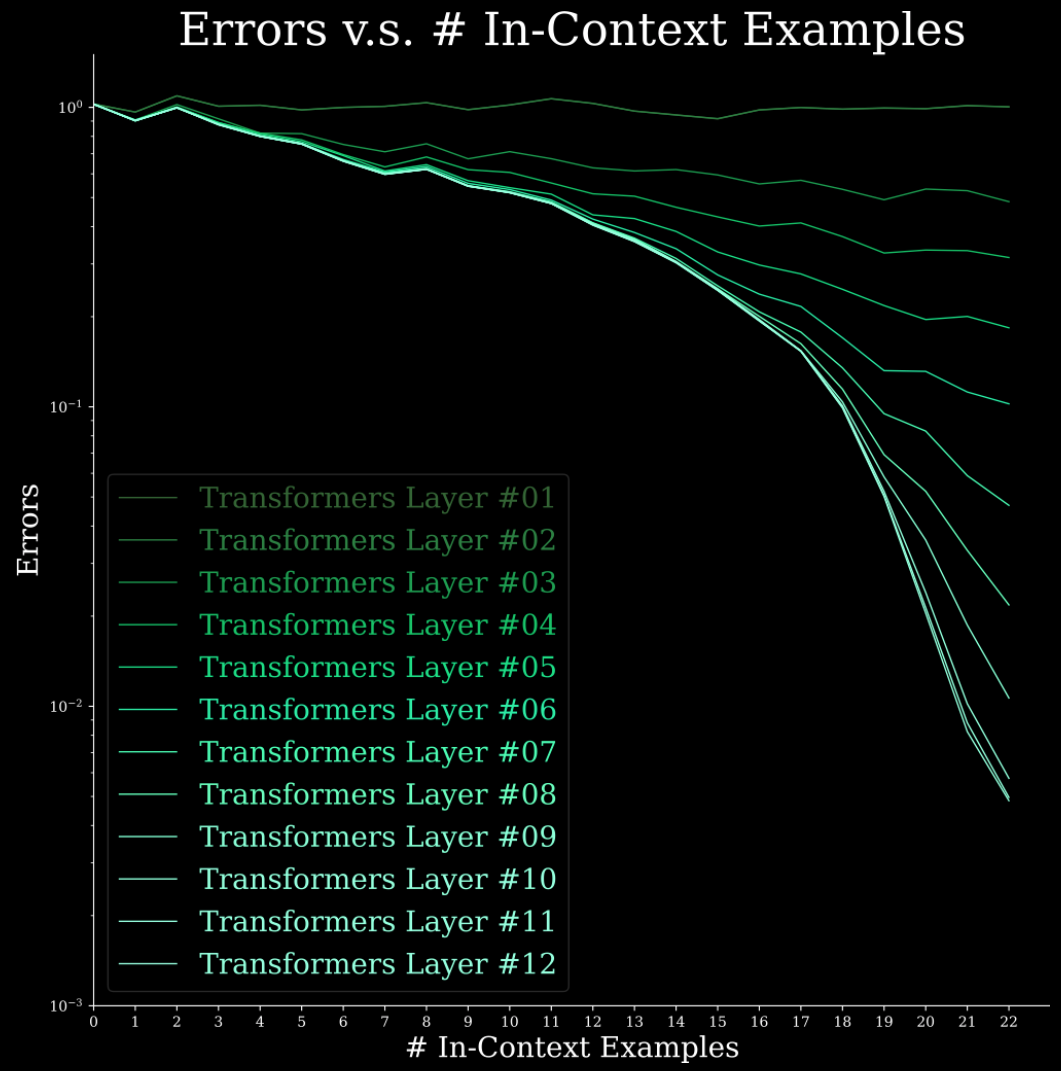
$x_4 = (4, -1), y_4 = ?$

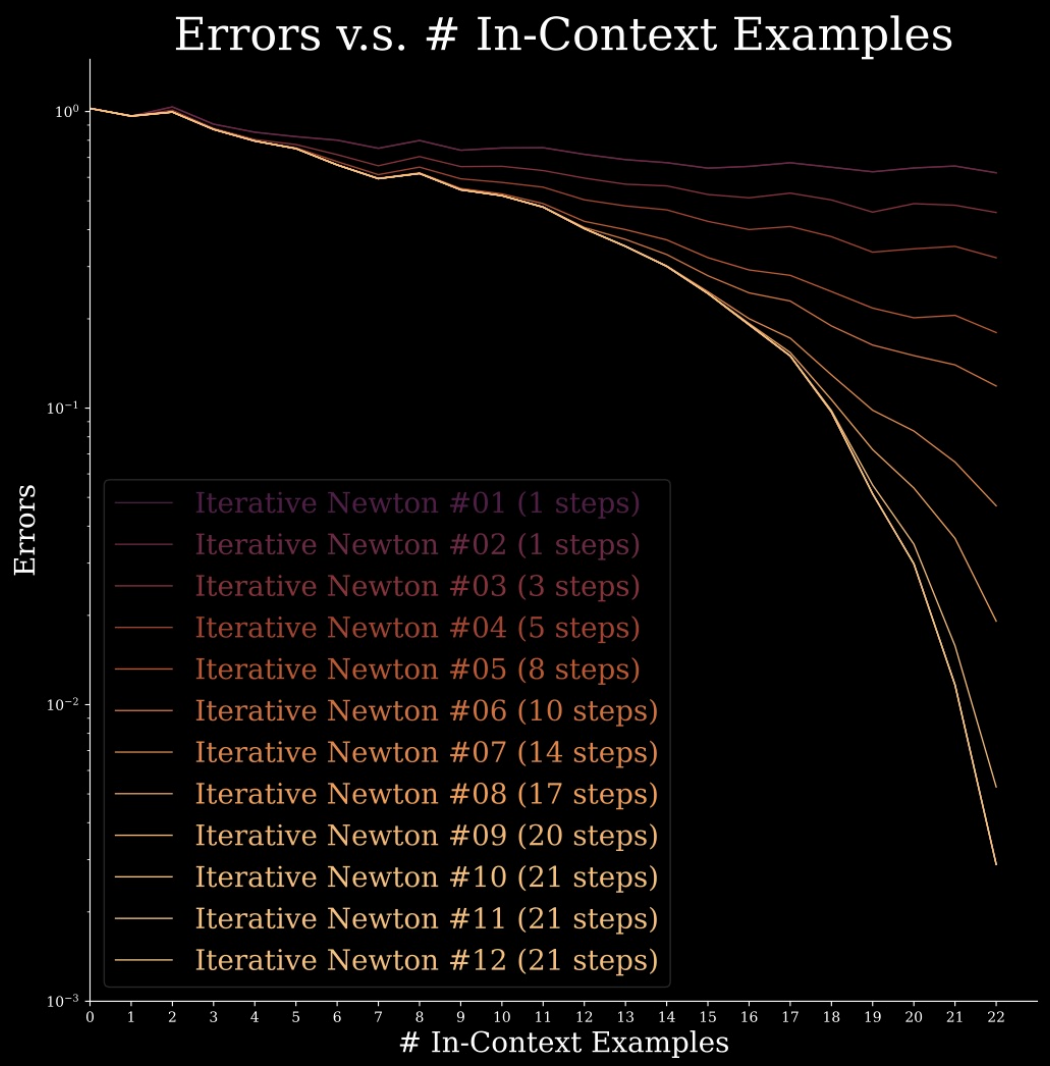


$\approx$



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

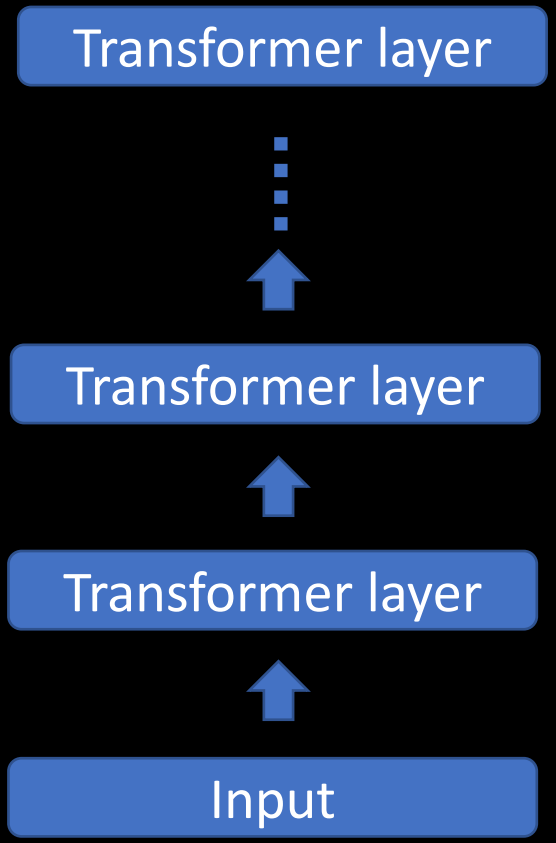





Techniques: “Applied theory”?

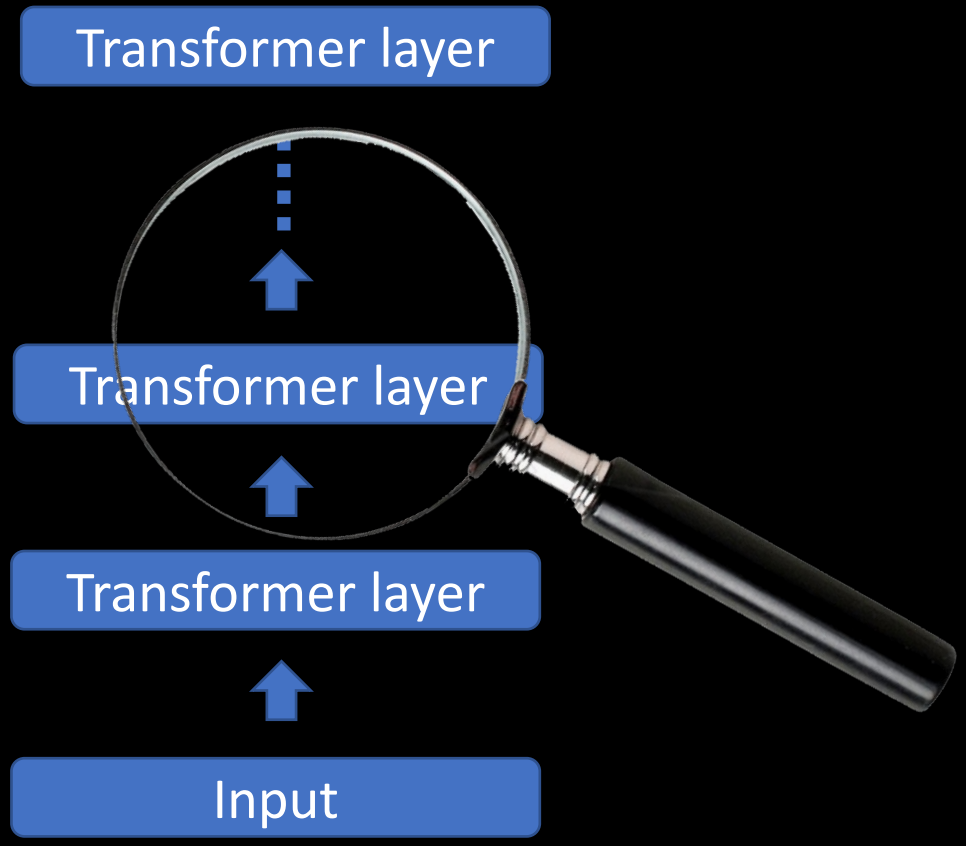
Techniques: “Applied theory”?

How should we understand how Transformers solve a problem?



Techniques: “Applied theory”?

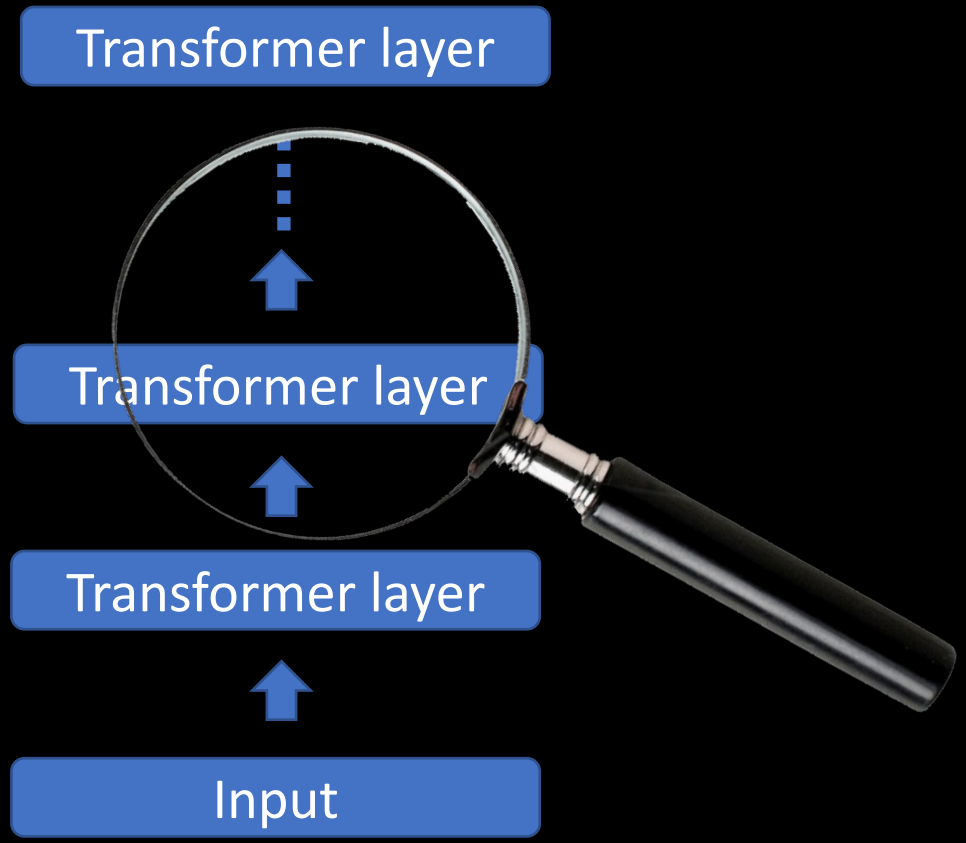
How should we understand how Transformers solve a problem?



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

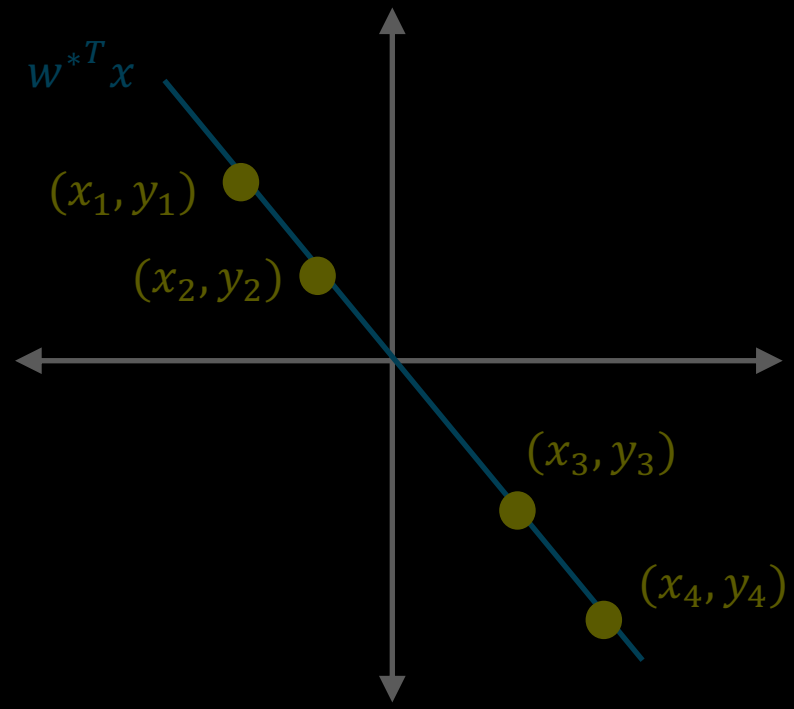
Techniques: “Applied theory”?

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

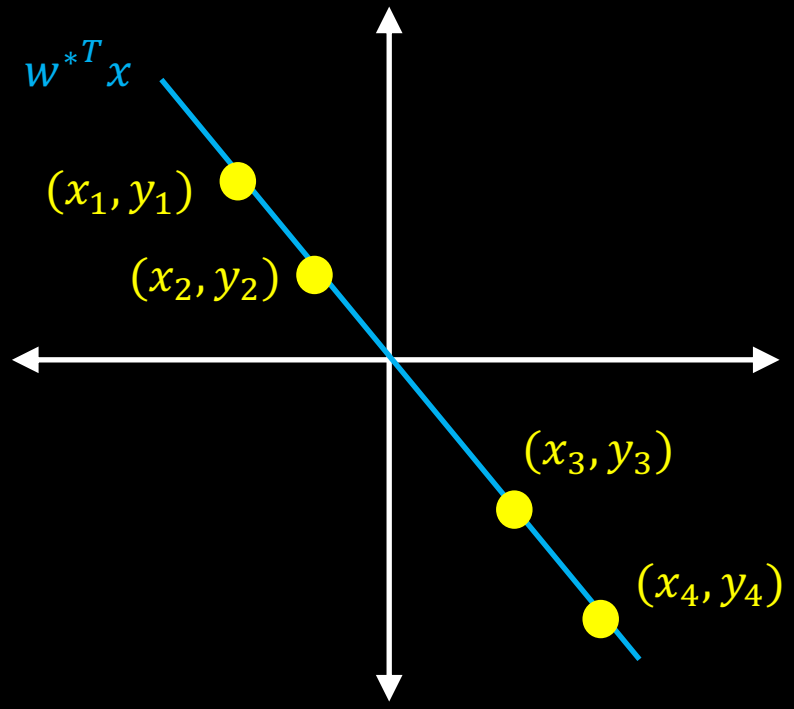


The Setup

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

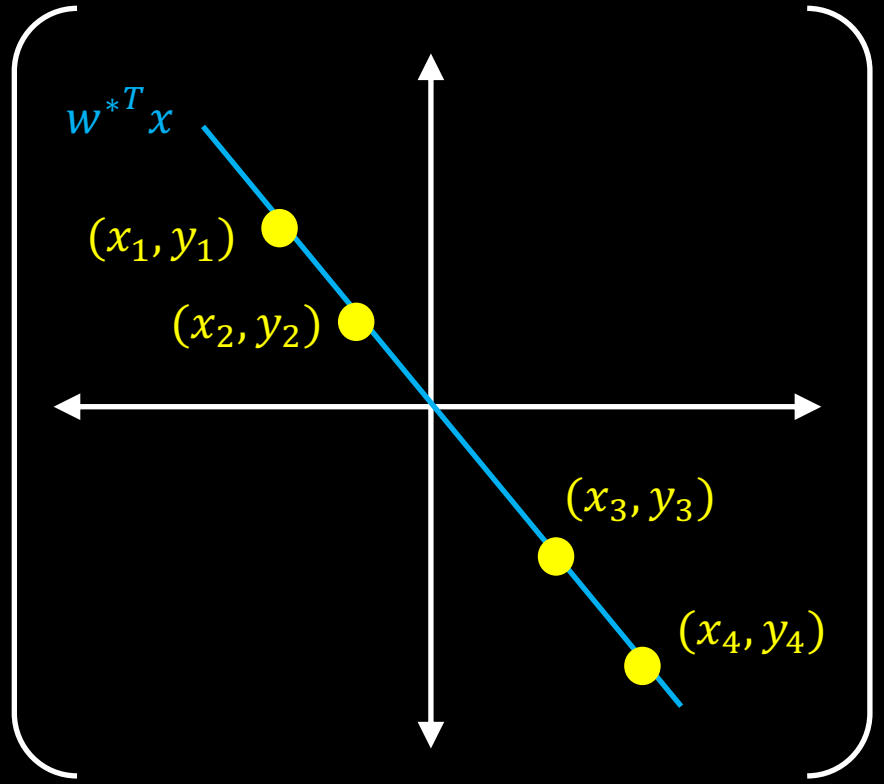
Iterative Newton's: Iterative 2nd order method to find inverse (≈ matrix Taylor series)

Let $S = X^T X$

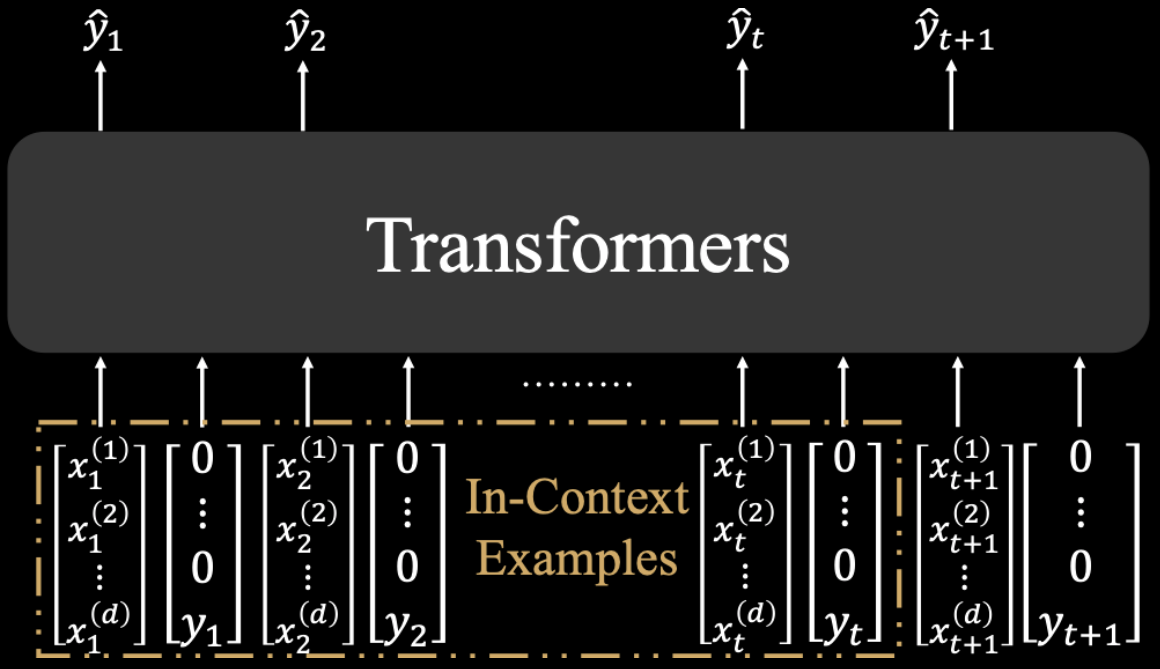
$$M_0 = \alpha S, M_{k+1} = 2M_k - M_k S M_k$$
$$w_{Newton}^{(k)} = M_k X^T y$$

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

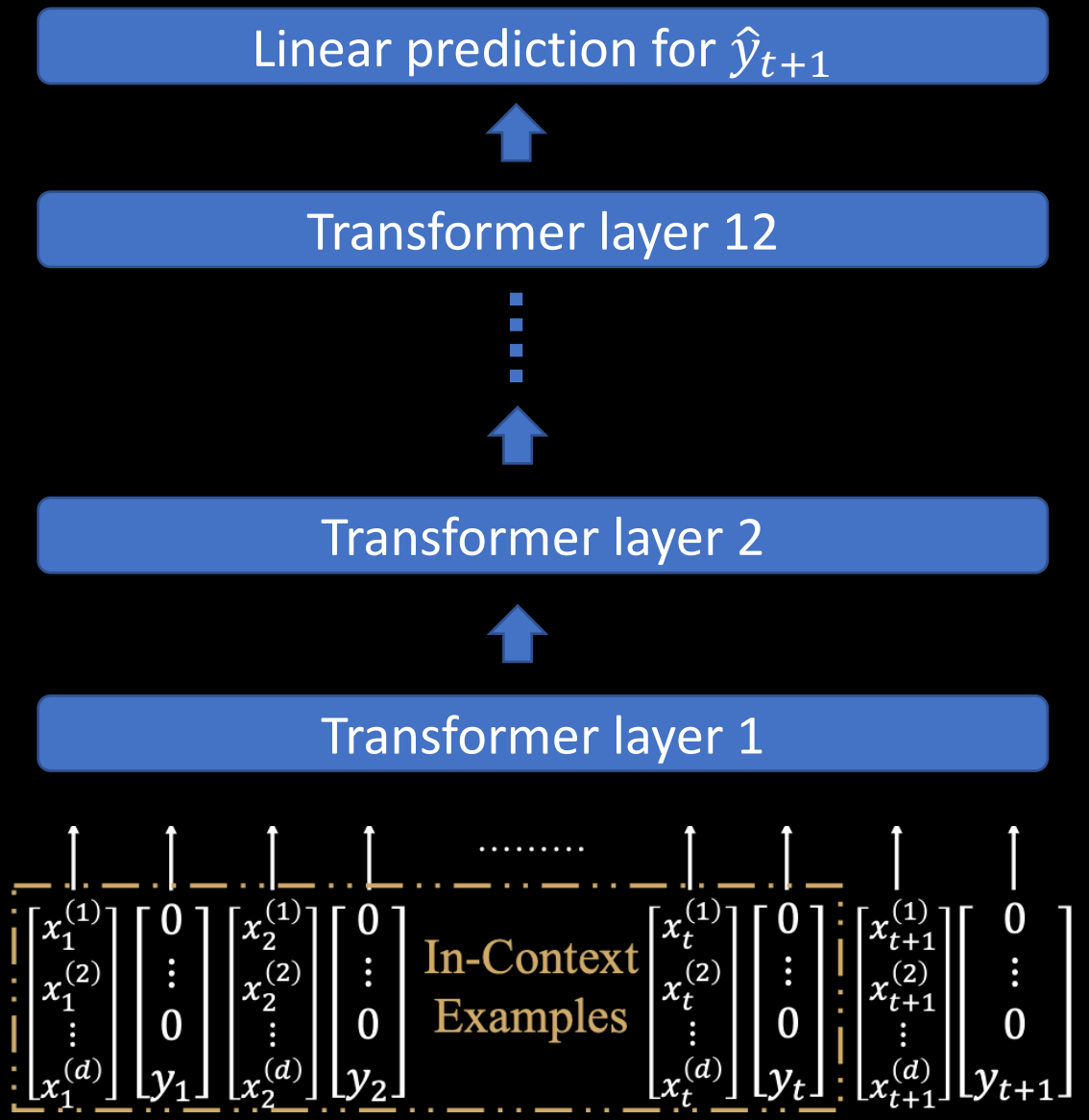
Transformers for linear regression



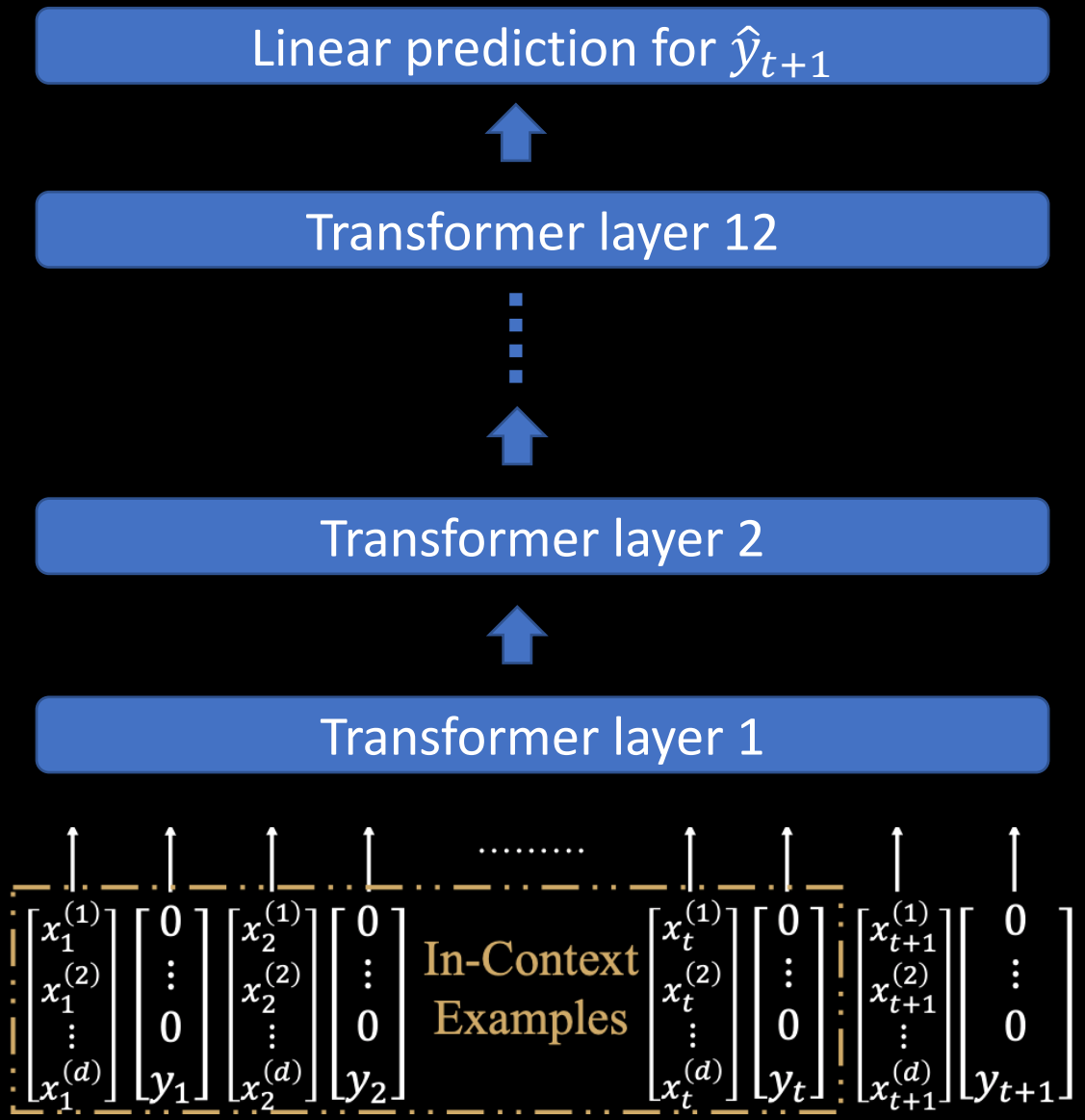
Train on T such instances



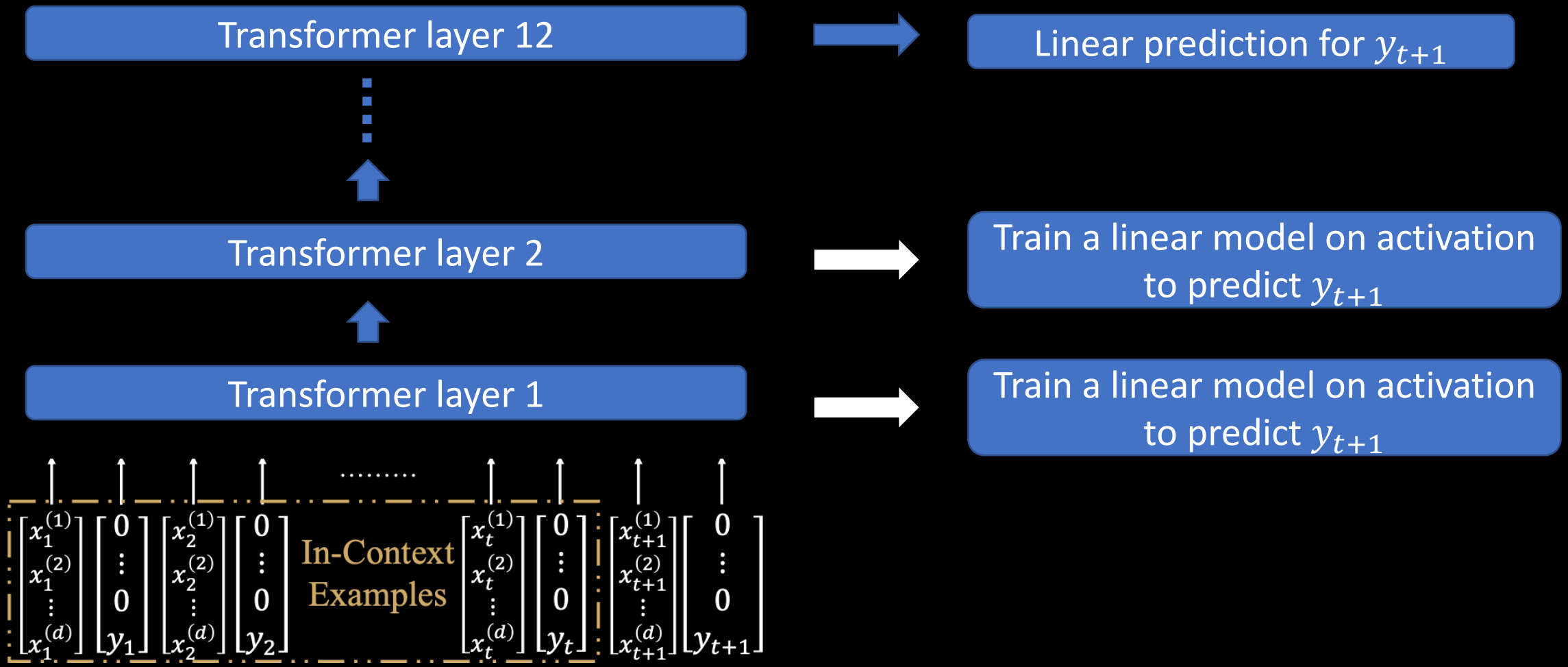
Transformers for linear regression



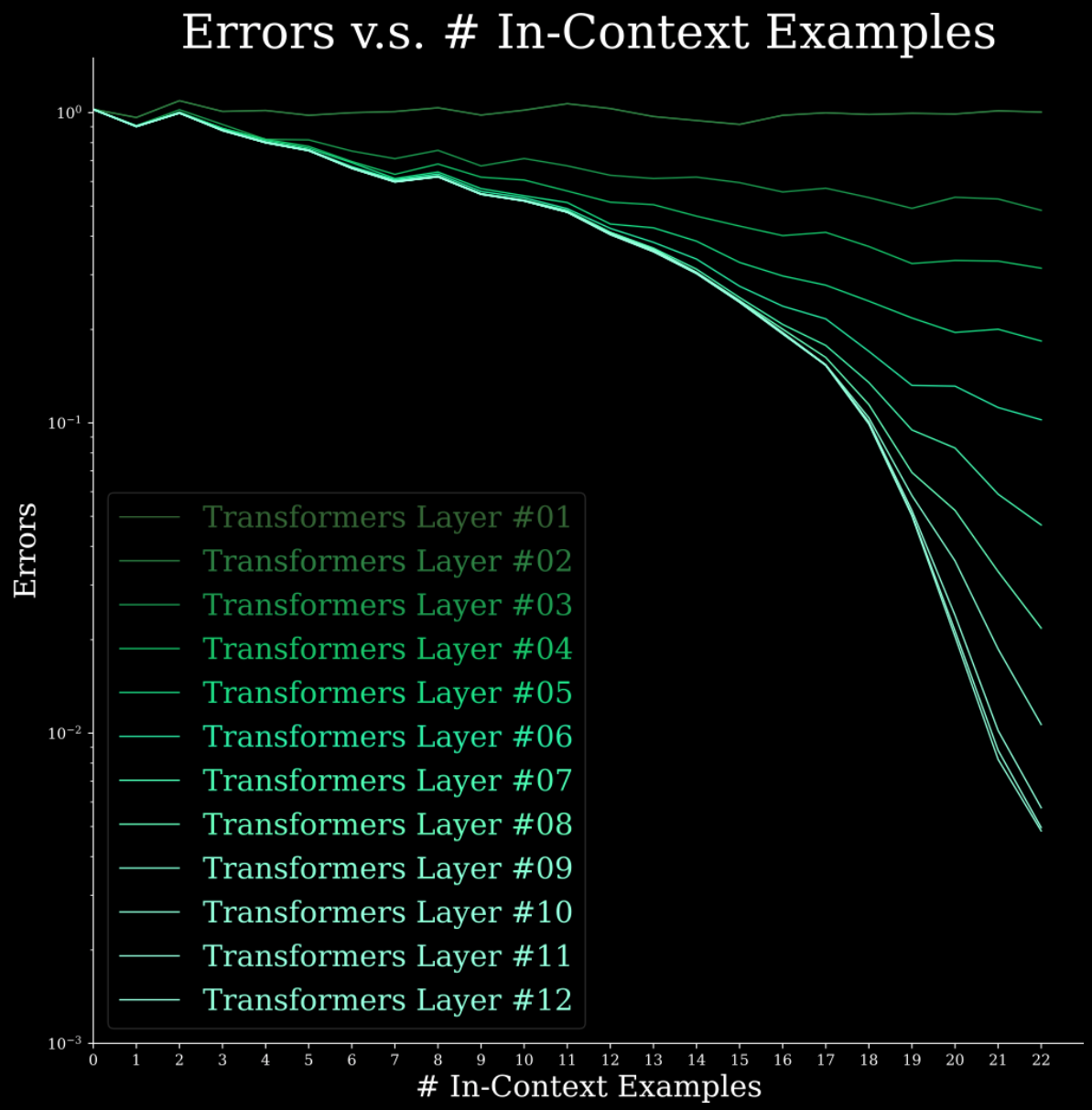
Transformers as an iterative algorithm: probing layers



Transformers as an iterative algorithm: probing layers



Transformers as an iterative algorithm: probing layers



Metric: Similarity of errors

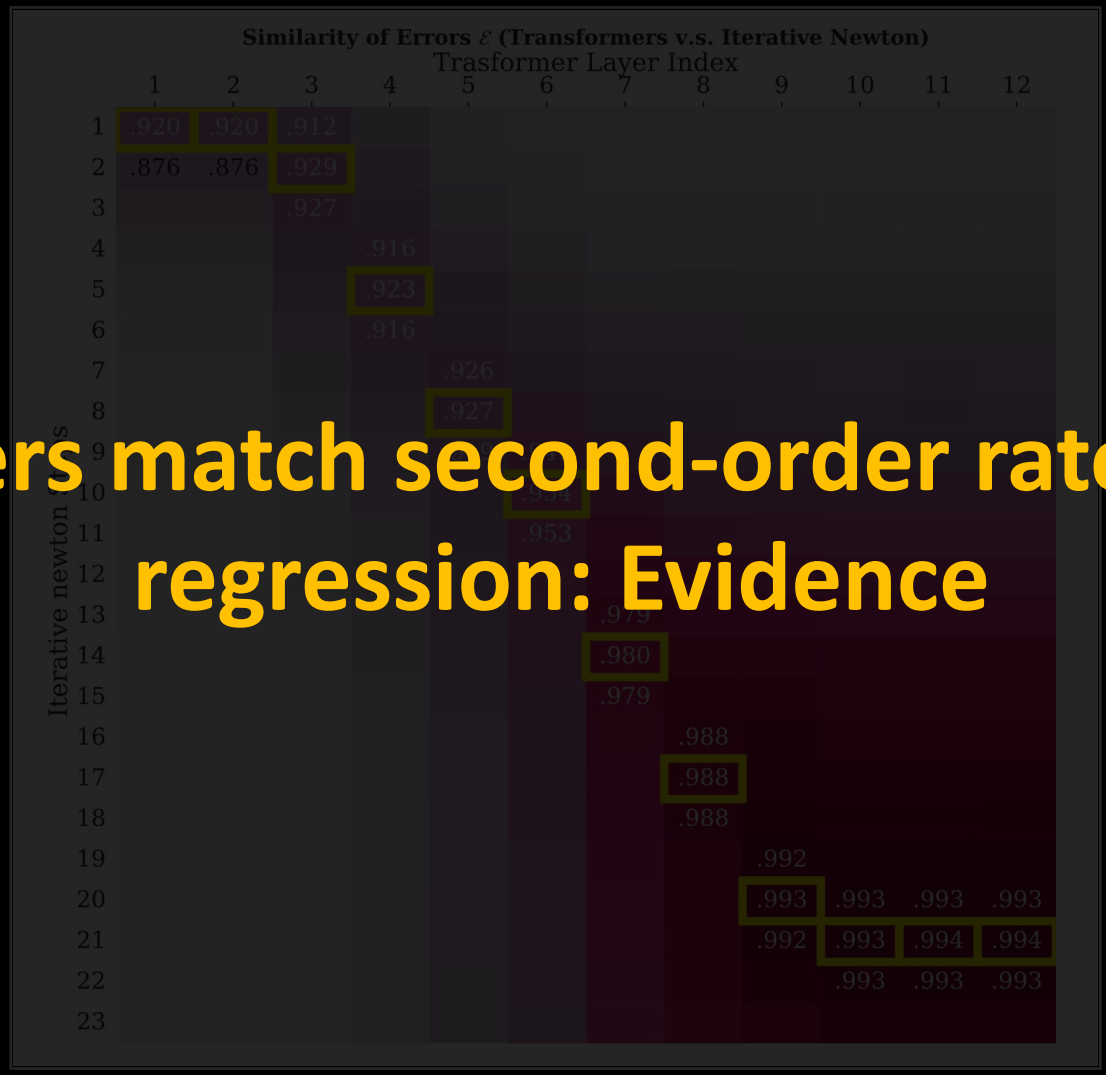
$x_1,\ y_1,\ x_2, y_2,\ x_3, y_3,\quad \ldots, x_n,\ y_n,$

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	$(y_1-y_1^B),\ (y_2-y_2^B,\ \ (y_3-y_3^B),\ \ldots,\ (y_n-y_n^B),$

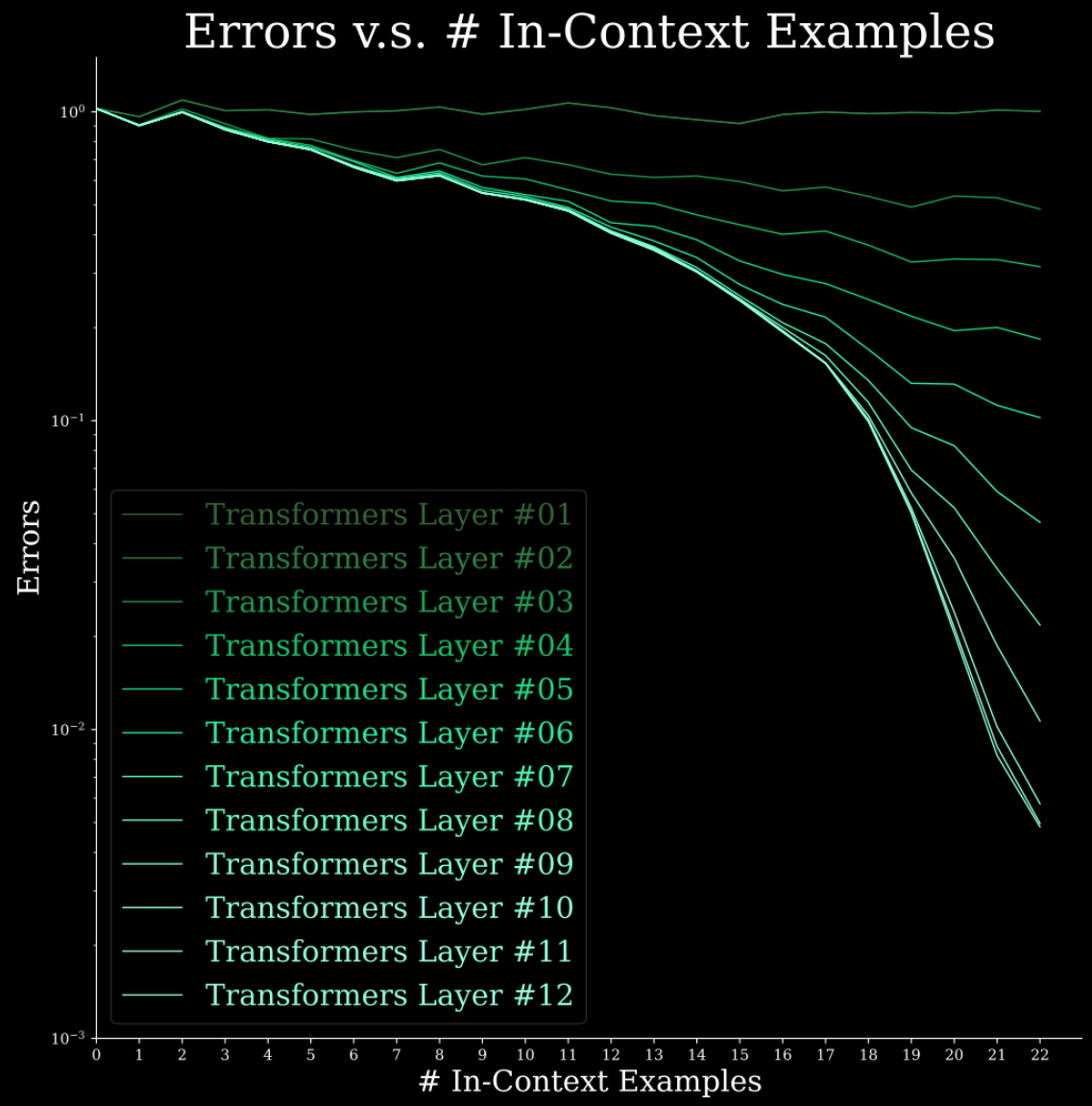
Similarity of errors on $\{x_i, y_i\}_{i=1}^n$ between Algorithm A, Algorithm B
= Cosine similarity between residuals of A , B

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

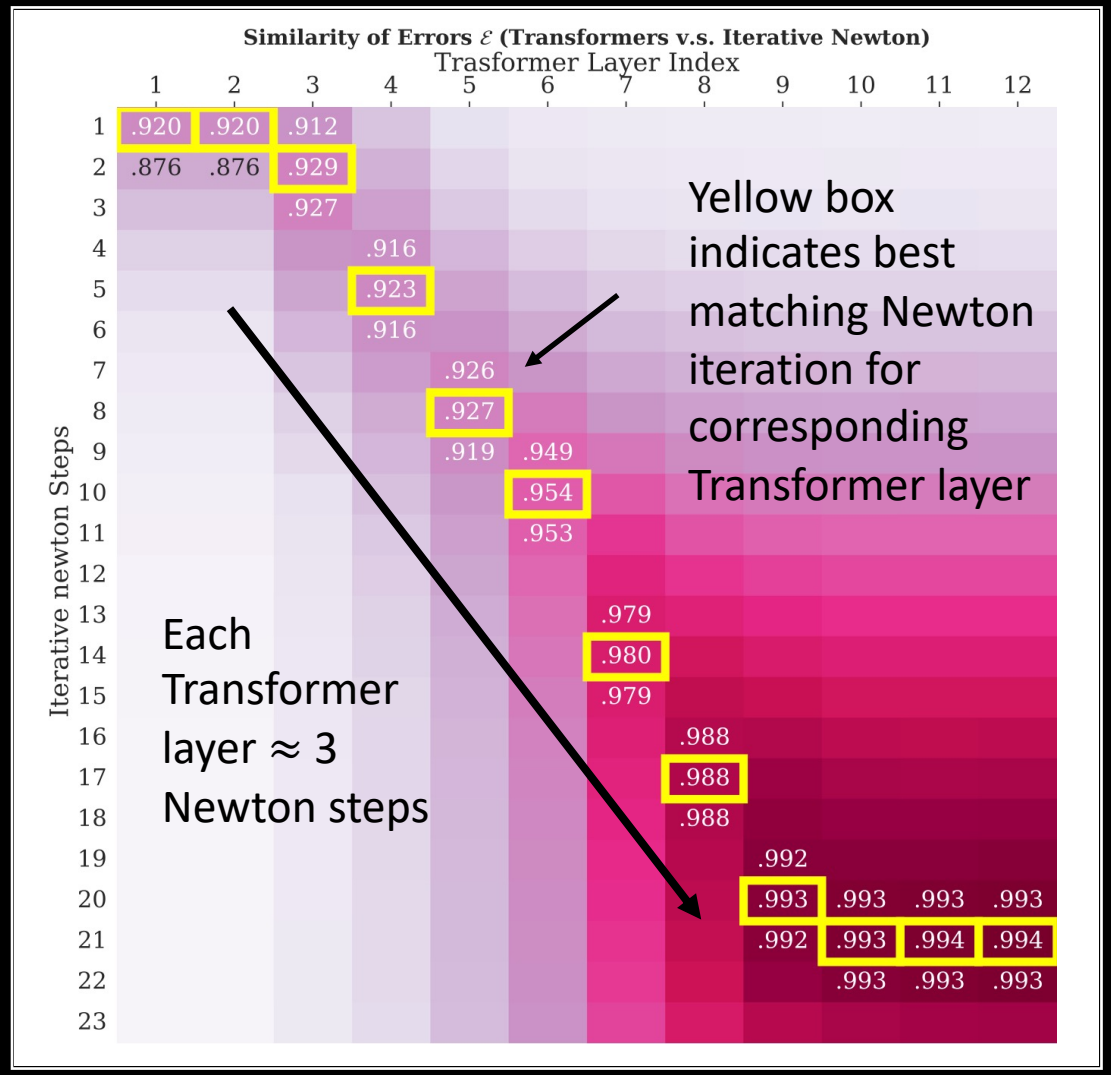
Transformers match second-order rates for linear regression: Evidence



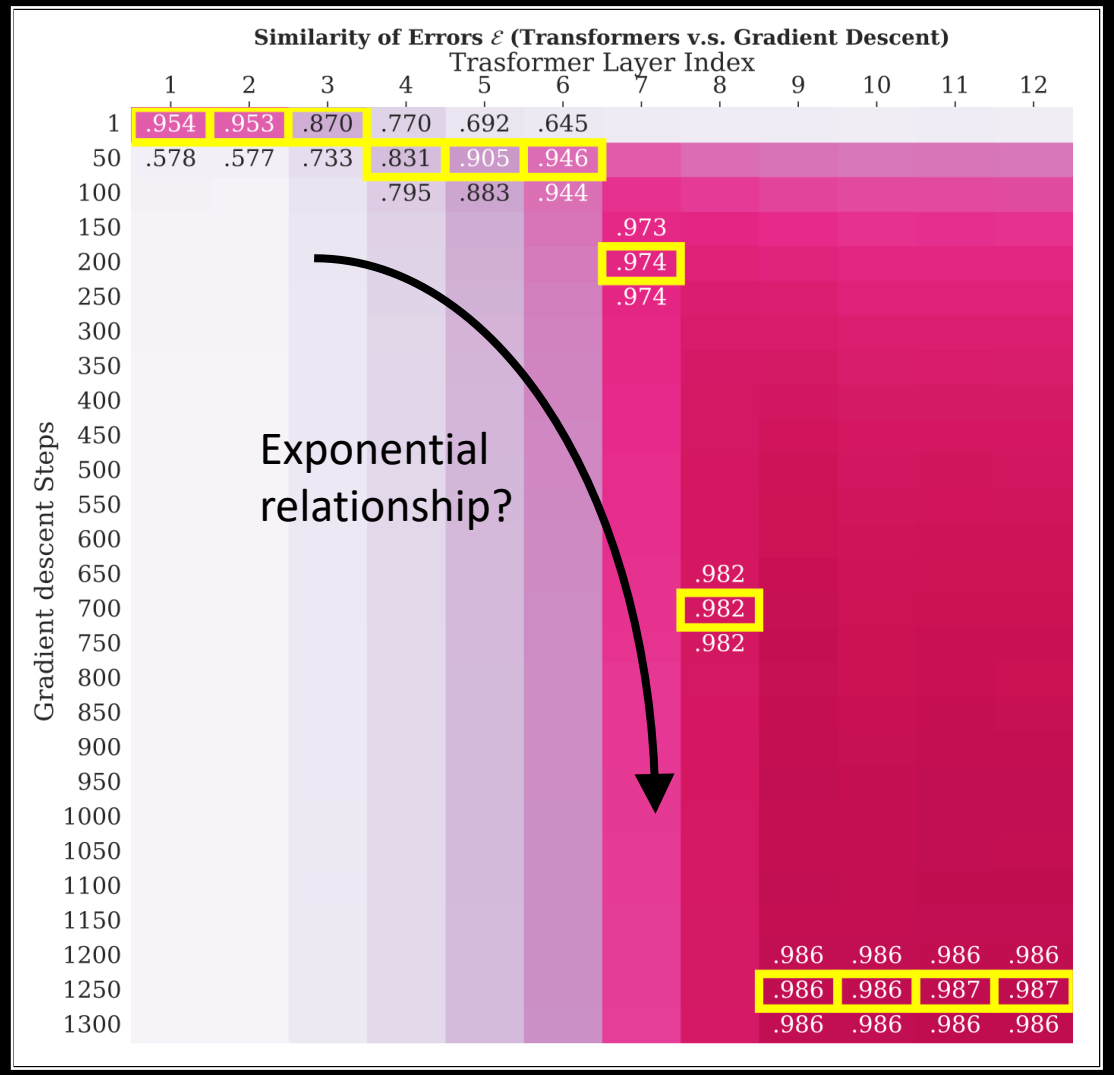
Claim 1: Transformers improve across layers



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

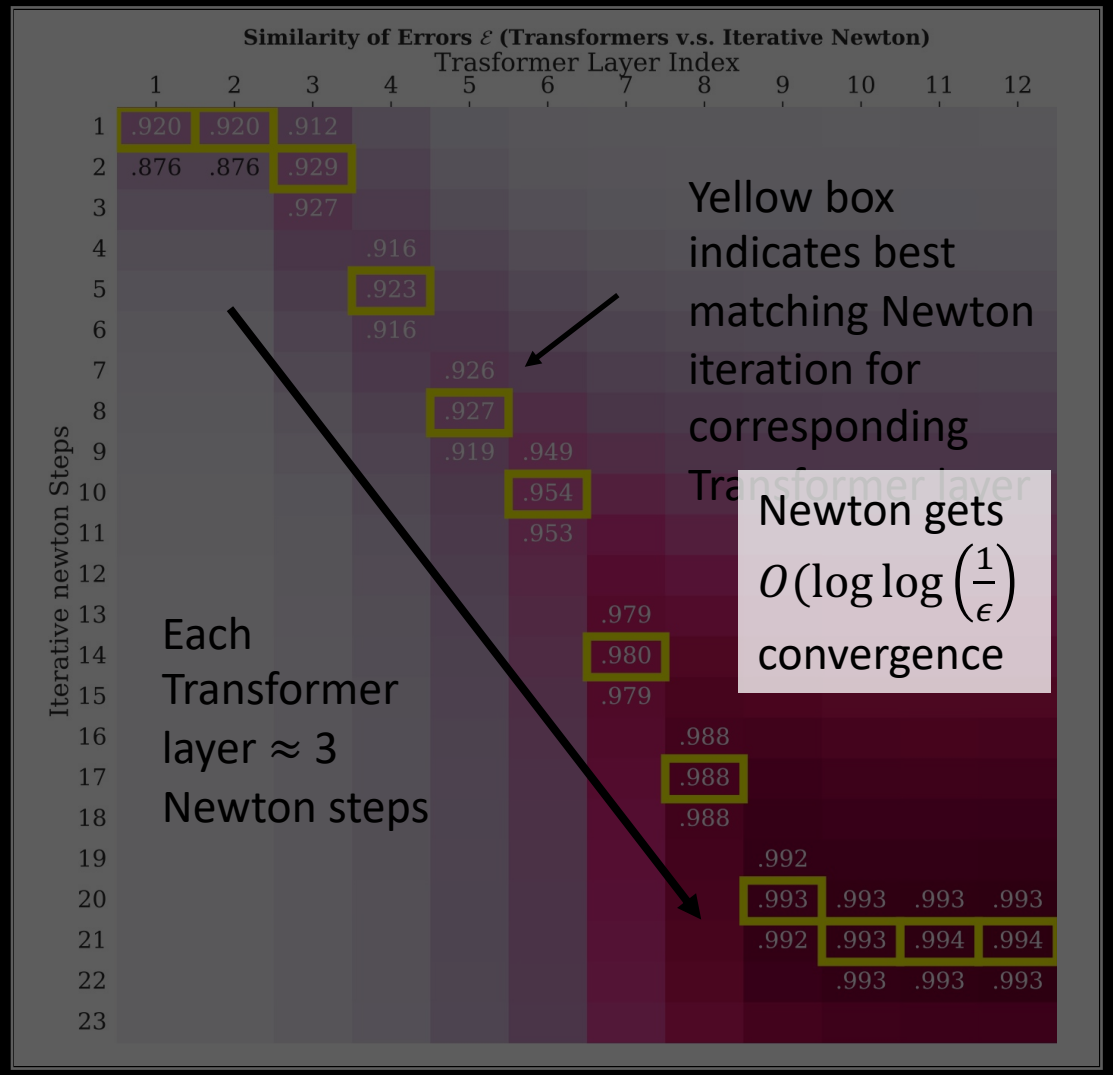


Transformers vs Newton

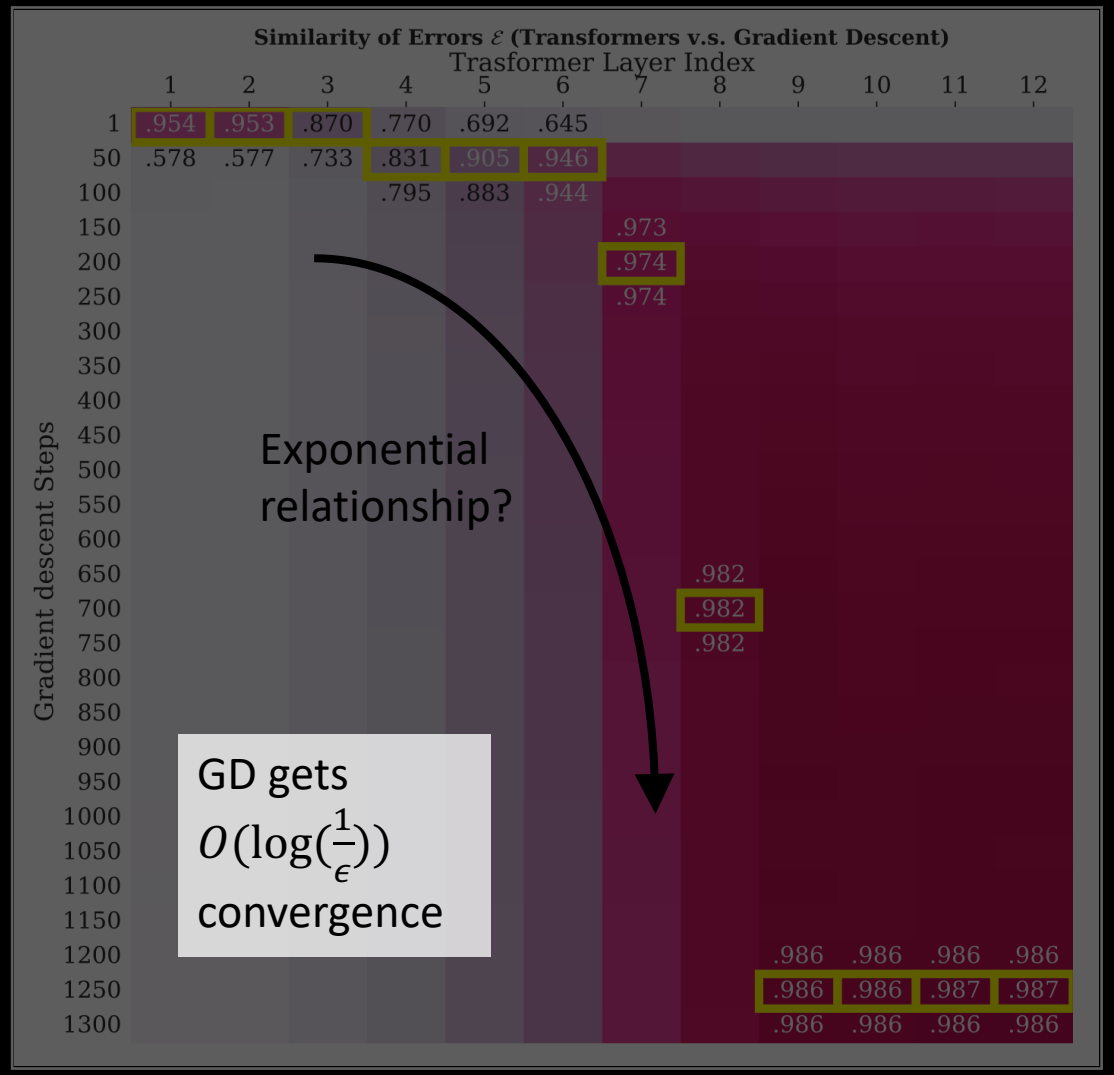


Transformers vs Gradient Descent

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

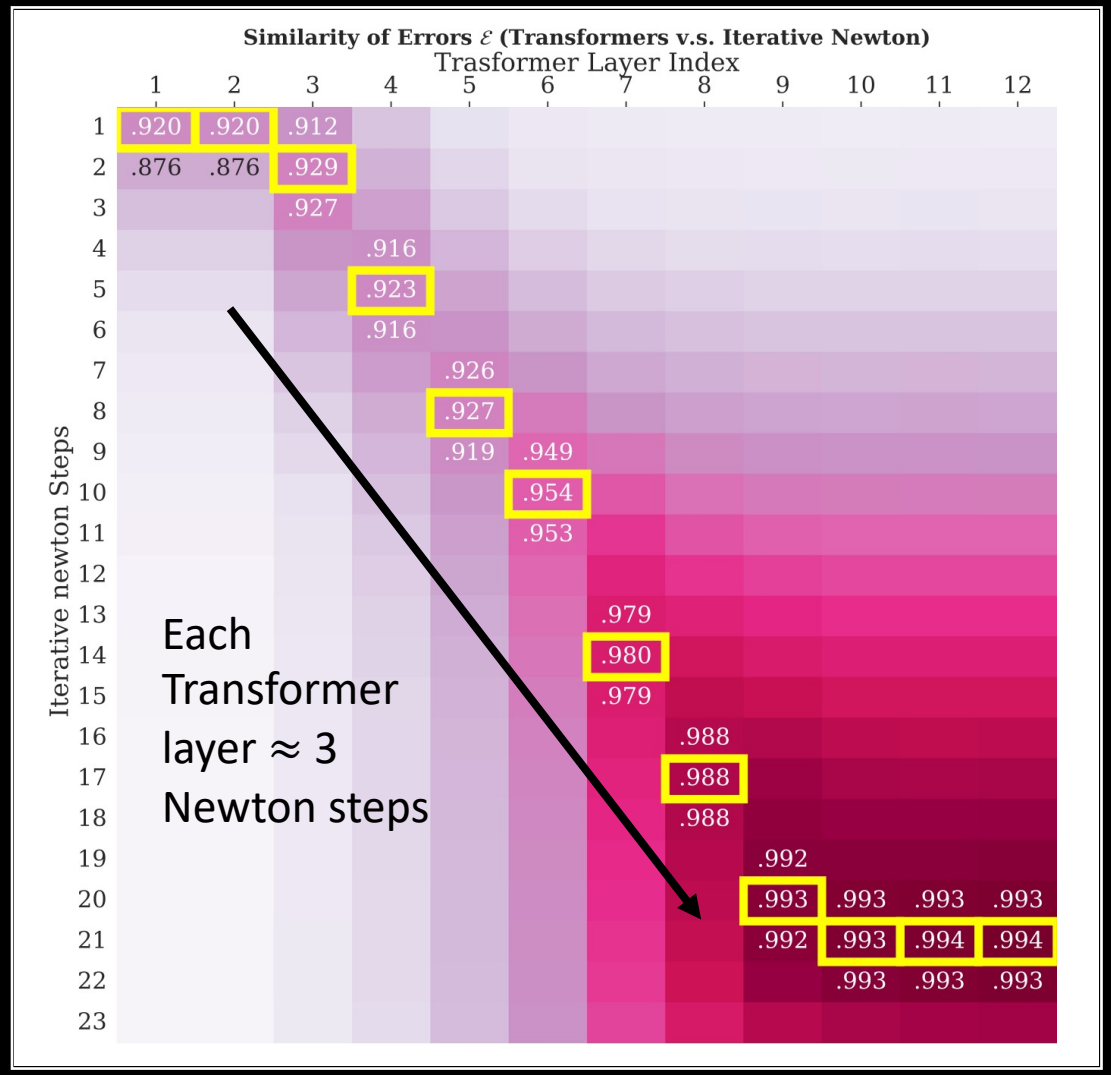


Transformers vs Newton

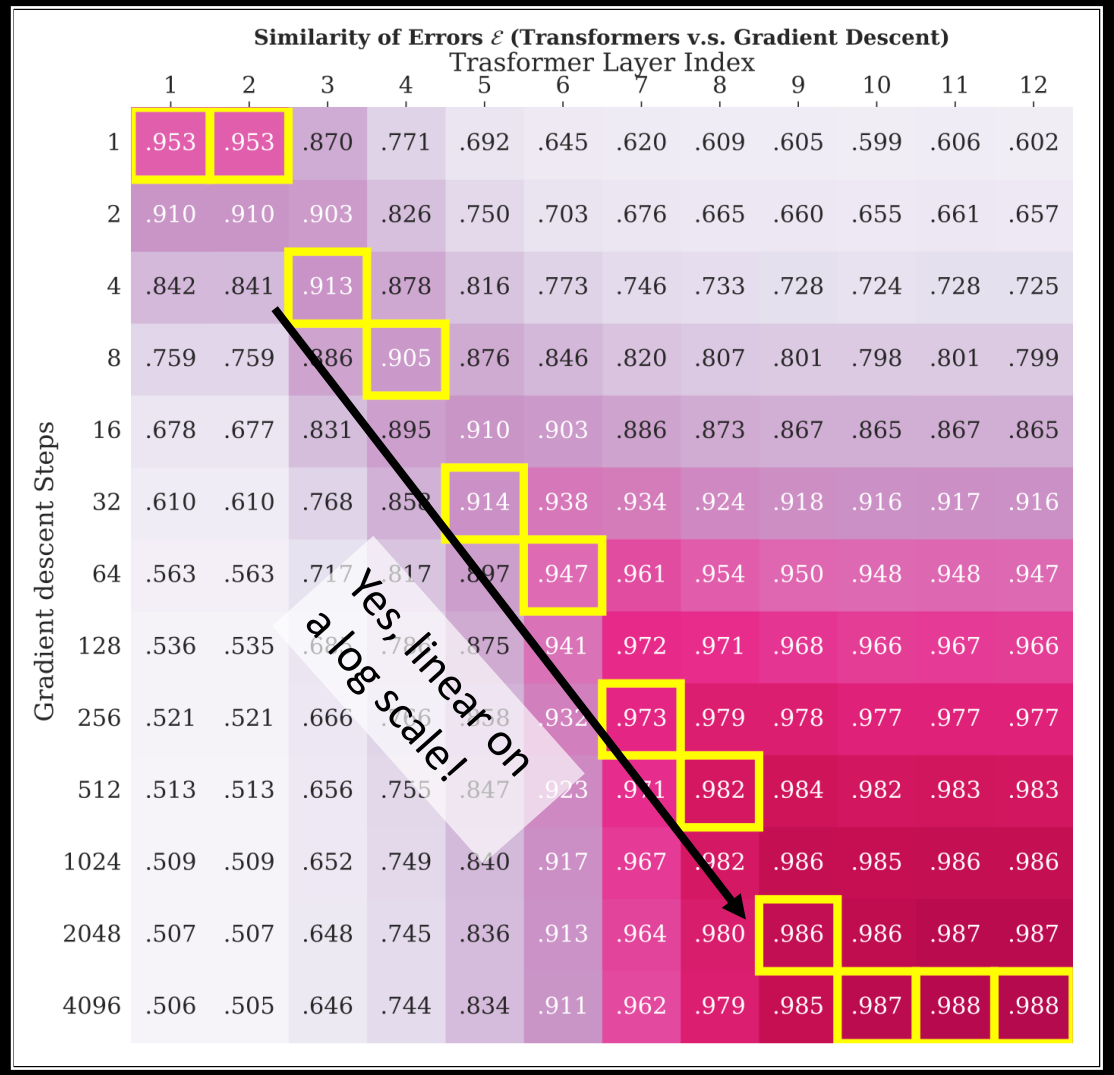


Transformers vs Gradient Descent

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?

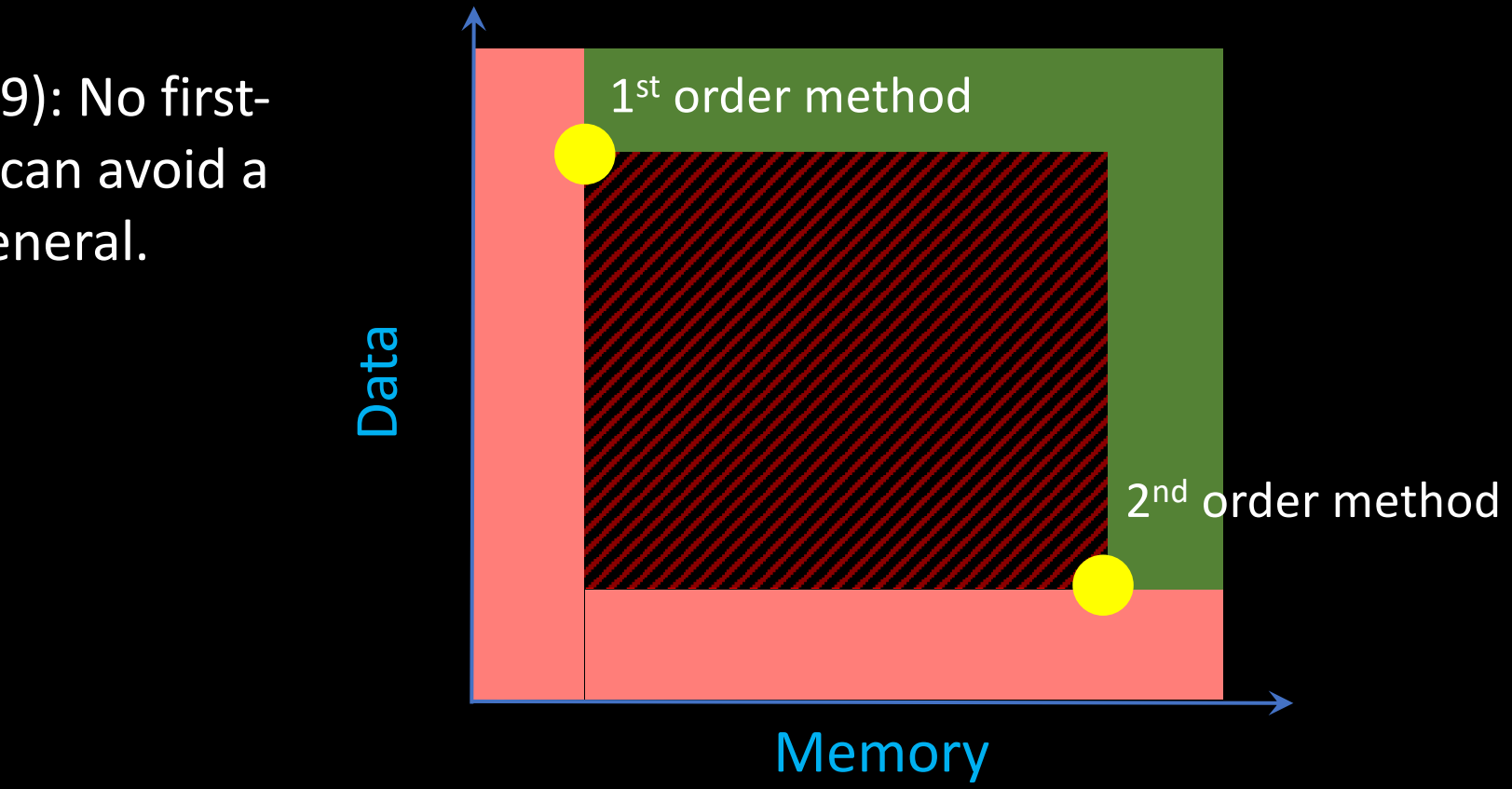
On **ill-conditioned instances**, gradient descent (or its variants) get $poly(\kappa)$ dependence on the condition number of the linear system κ, 2nd order methods get $polylog(\kappa)$ dependence.

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



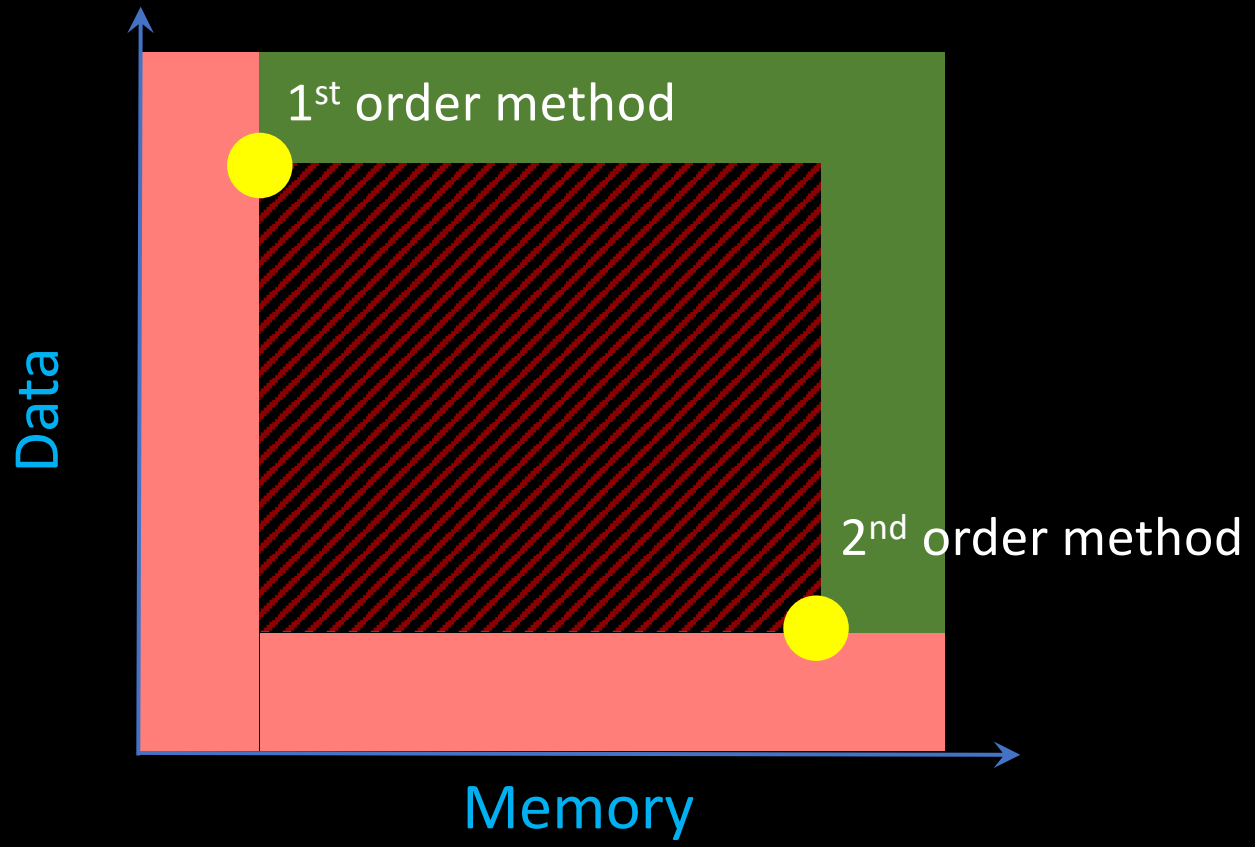
Claim 3: Transformers are still able to match Newton on harder distributions

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

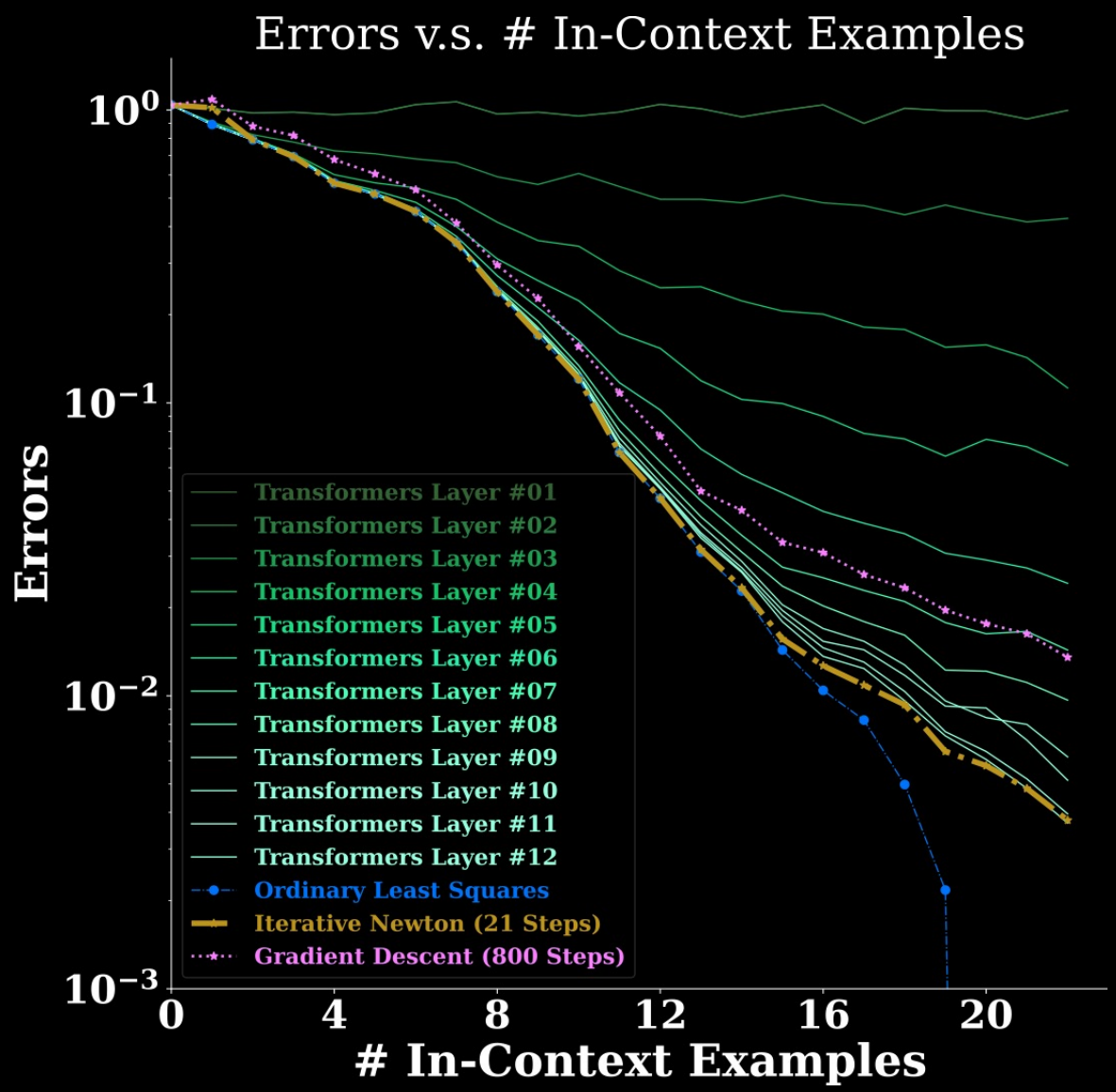
On **ill-conditioned instances**, gradient descent (or its variants) get $poly(\kappa)$ dependence on the condition number of the linear system κ, 2nd order methods get $polylog(\kappa)$ dependence.

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

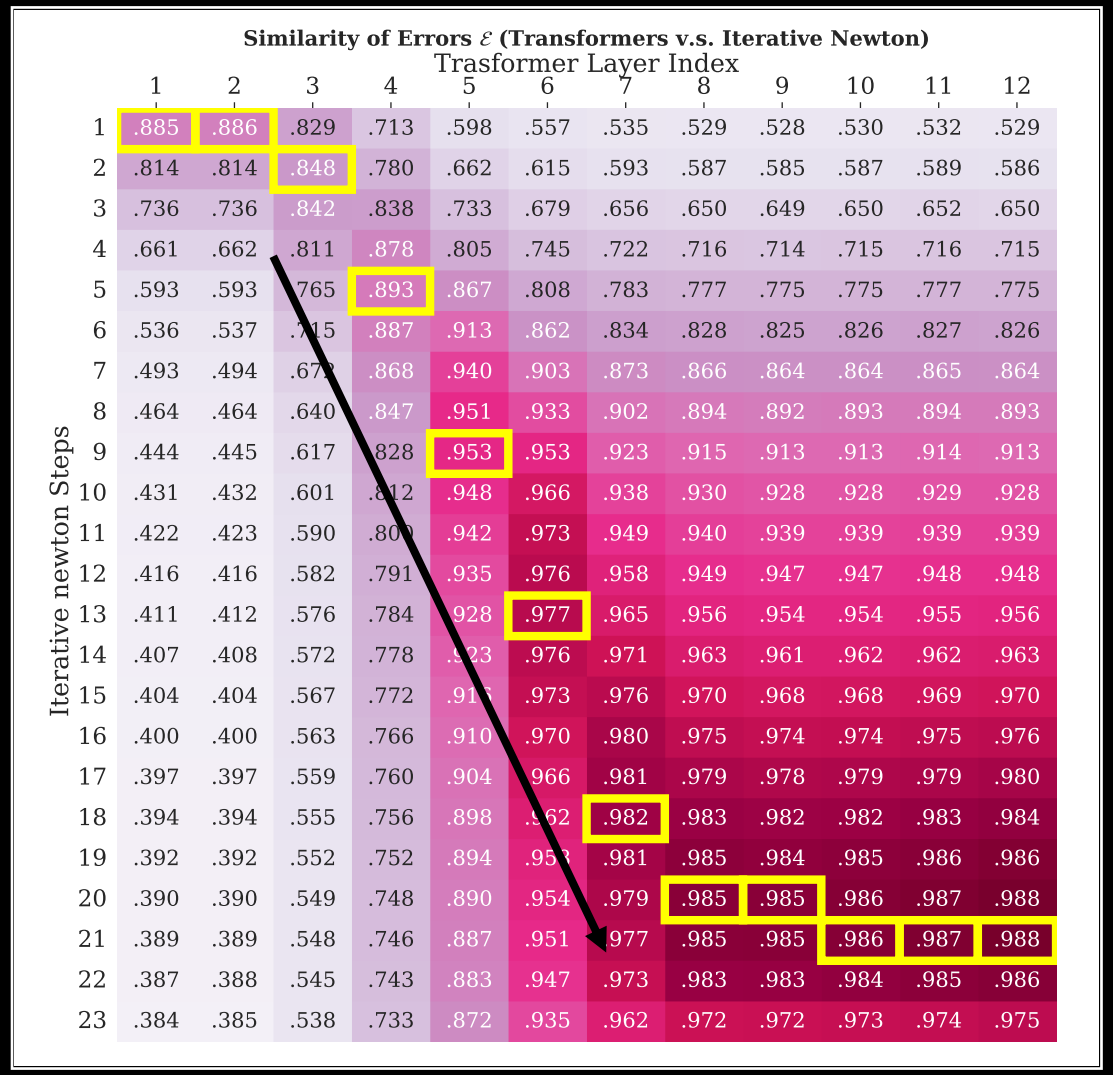
Hard distribution: Sample Σ with $d/2$ eigenvalues at 100, $d/2$ eigenvalues at 1, uniformly random eigenbasis.



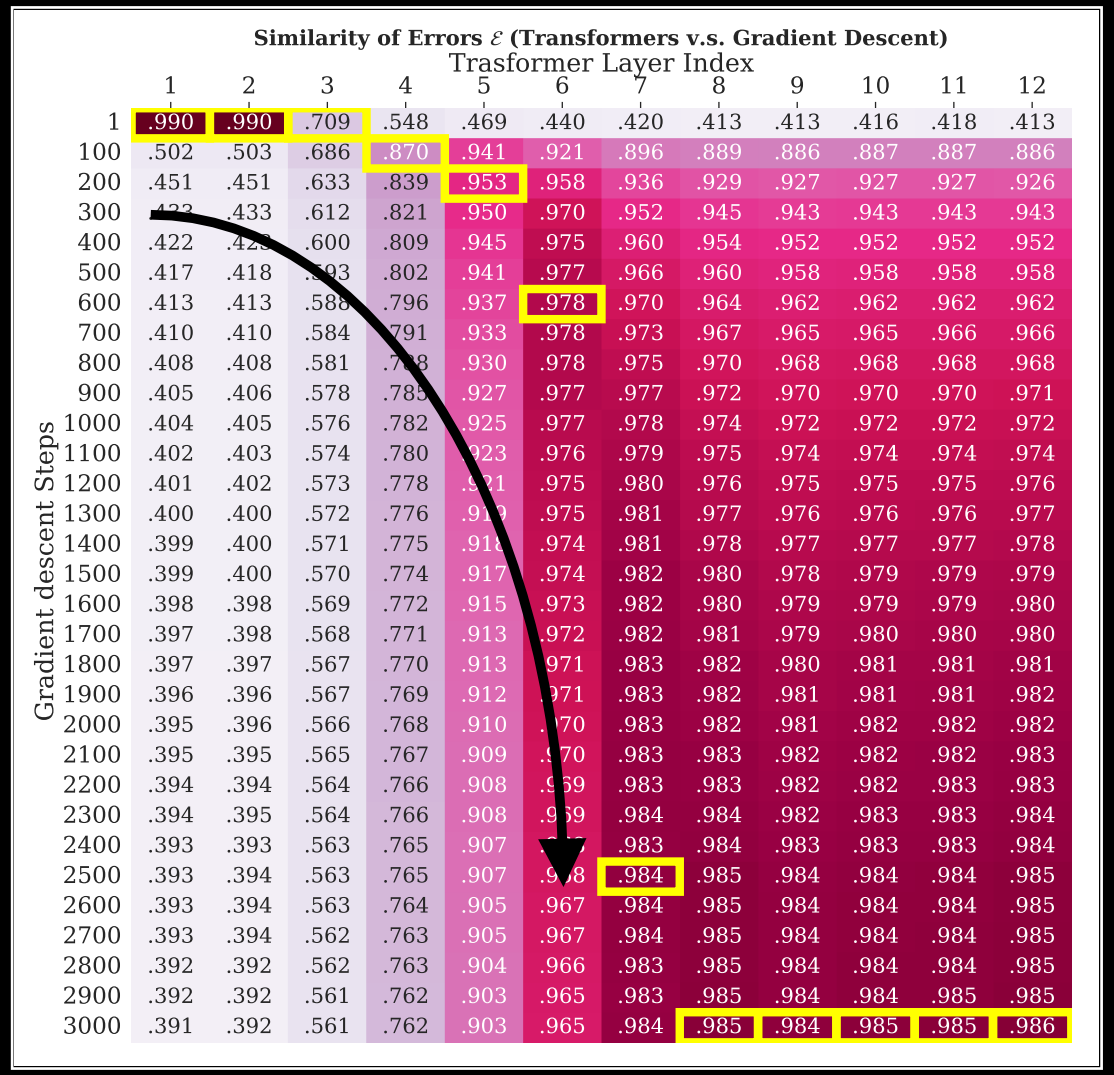
Claim 3: Transformers are still able to match Newton on ill-conditioned data



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



Transformers vs Newton



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

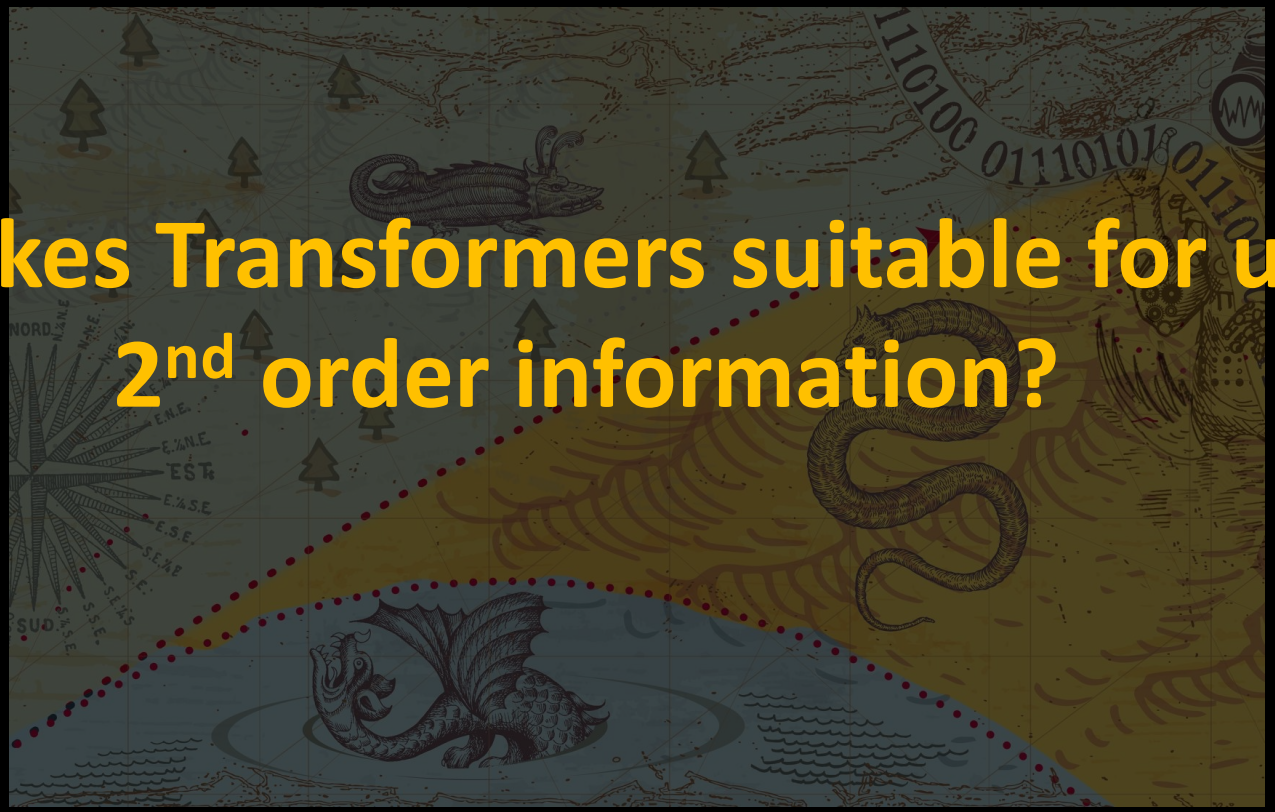
Some more related work

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

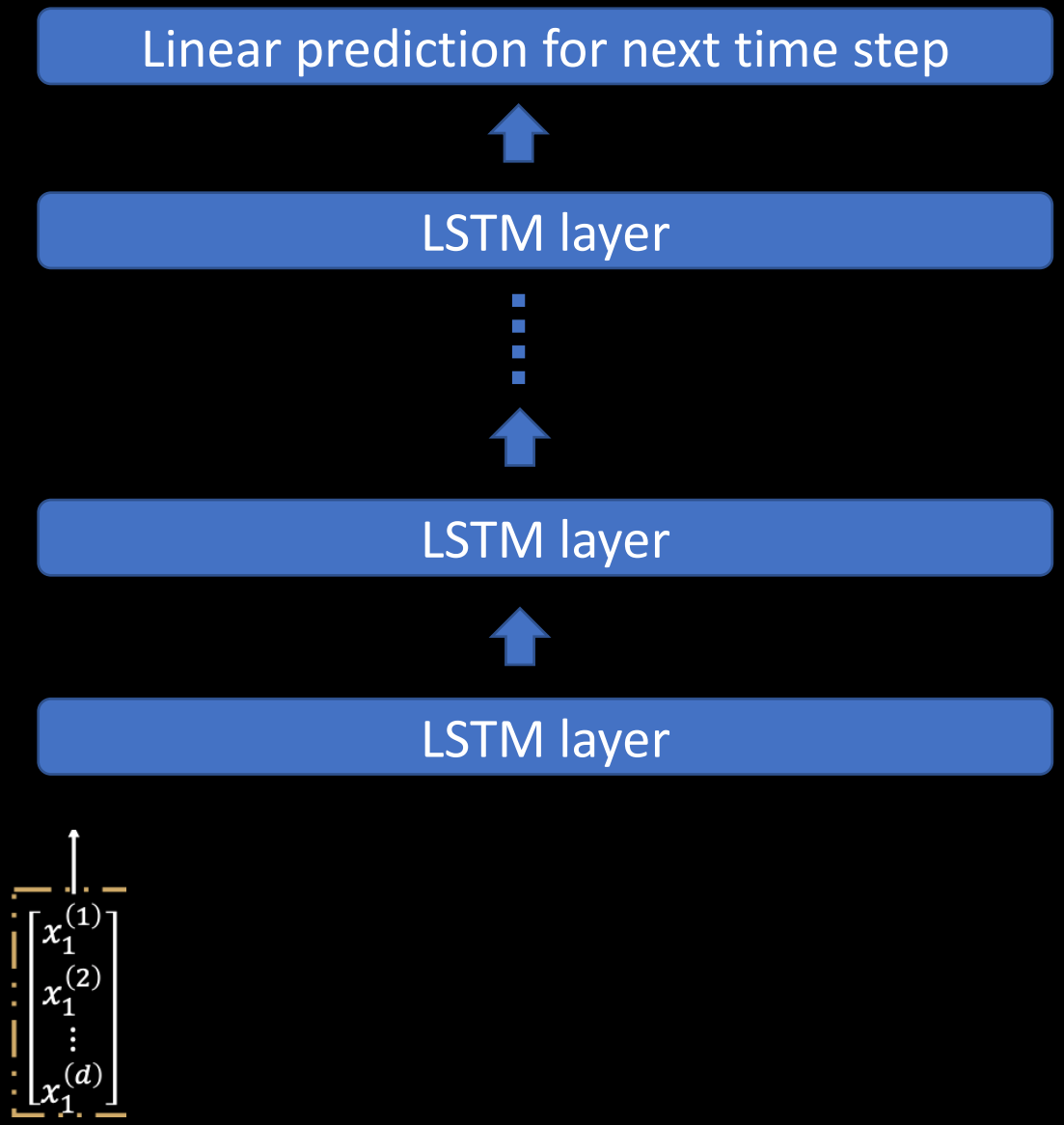
Vladymyrov-von Oswald-Sandler-Ge'2024 show that a second-order variant of GD can mimic Iterative Newton by implicitly approximating inverse

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

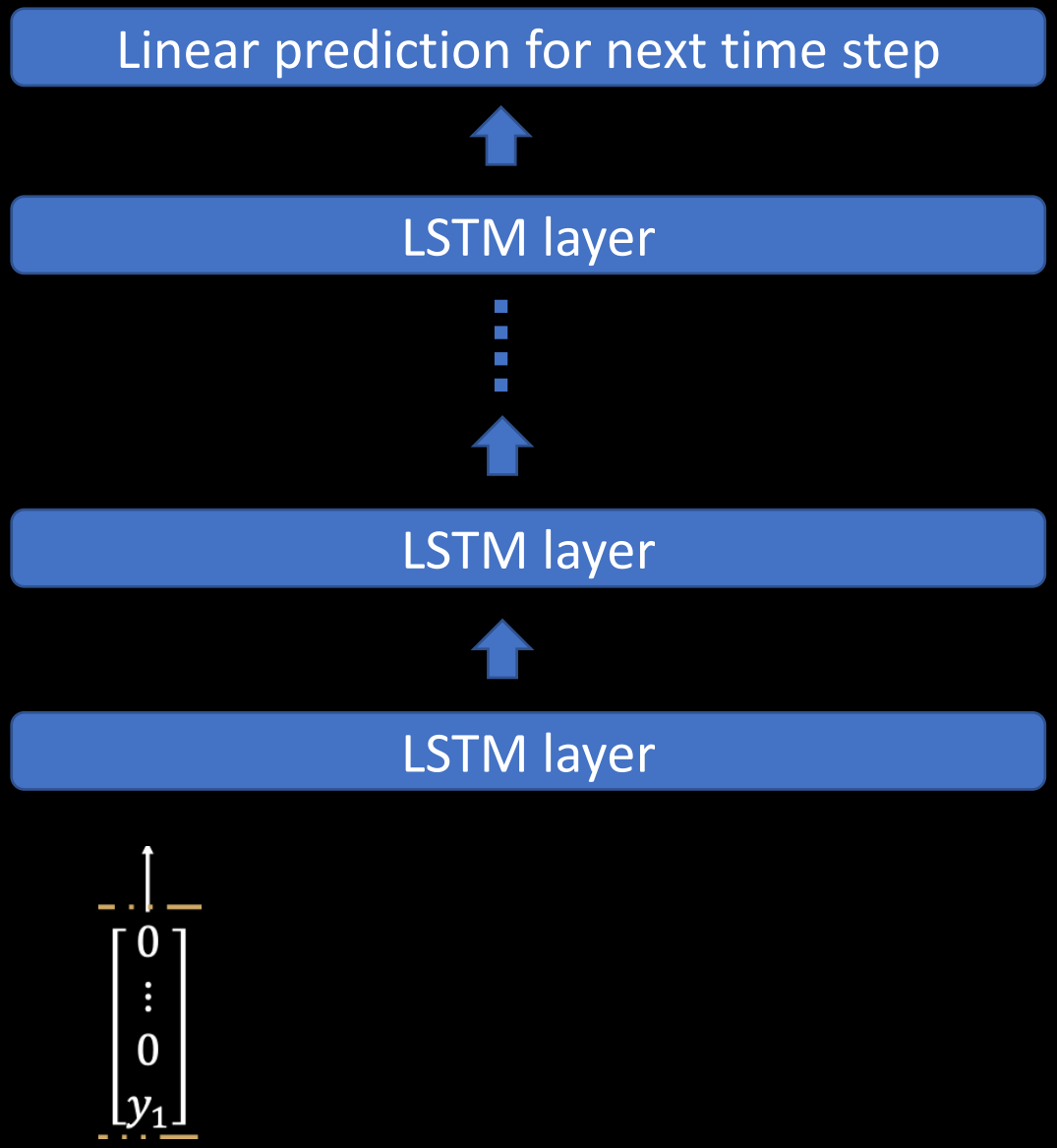
What makes Transformers suitable for utilizing 2nd order information?



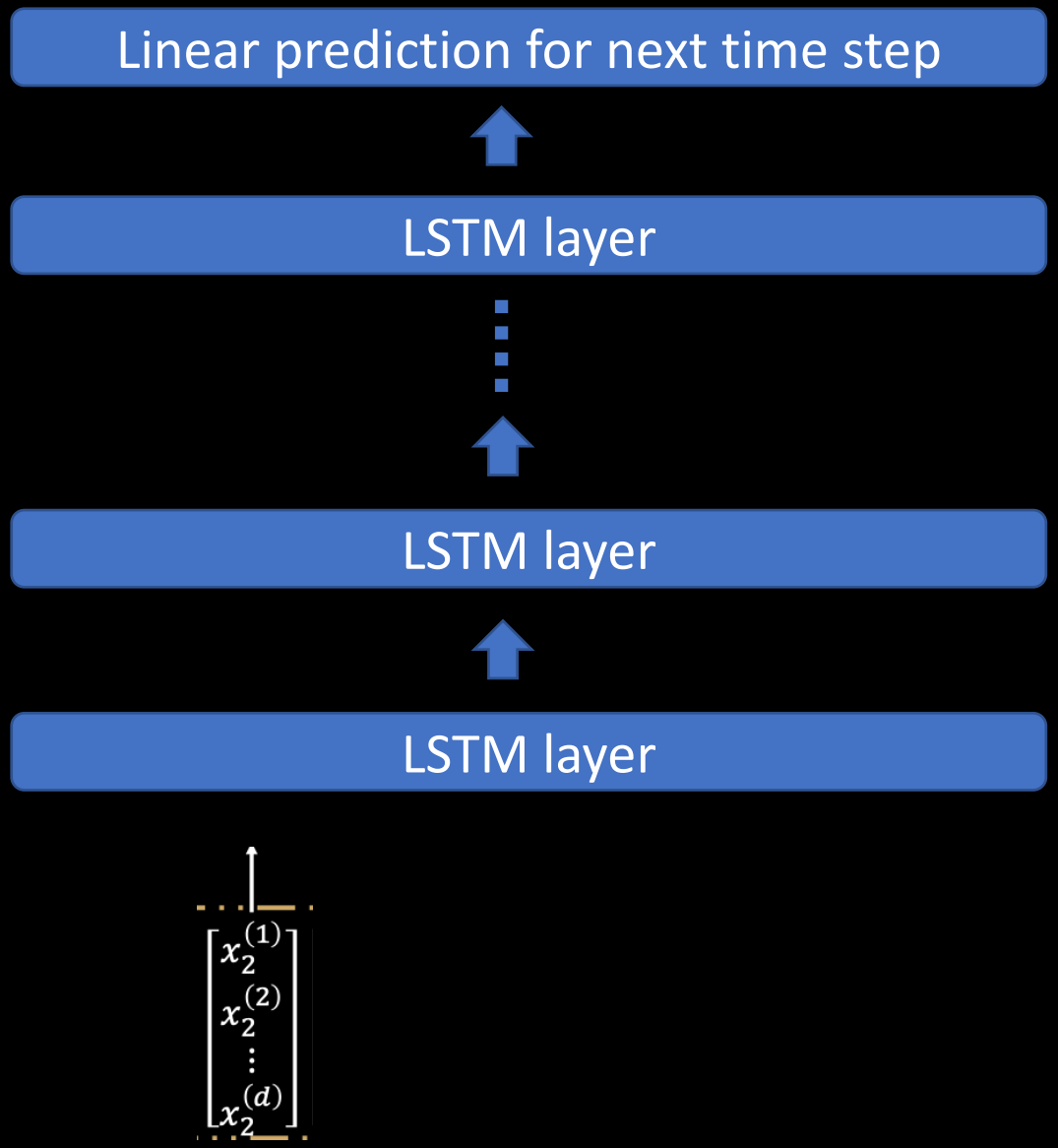
LSTMs for linear regression



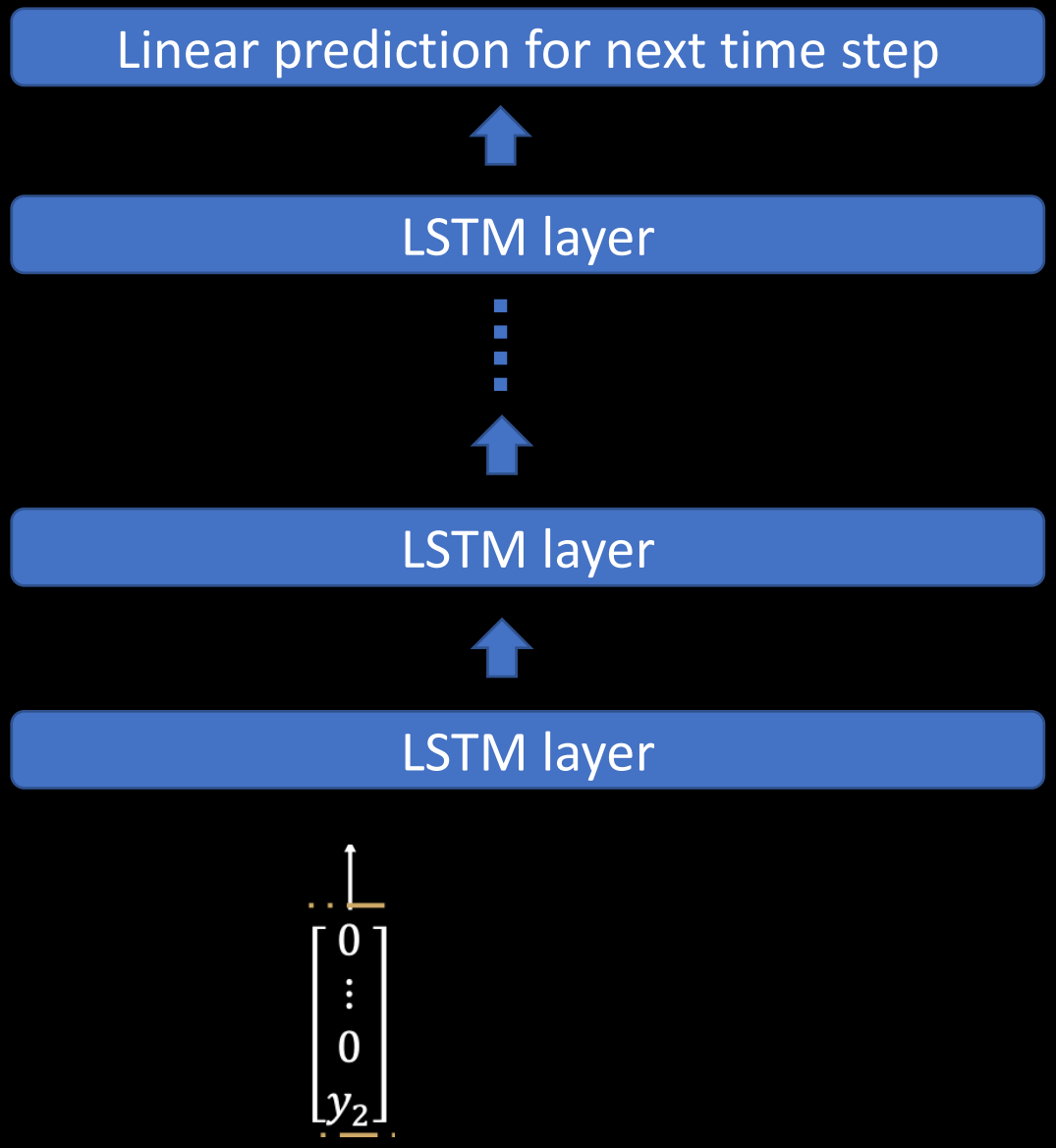
LSTMs for linear regression



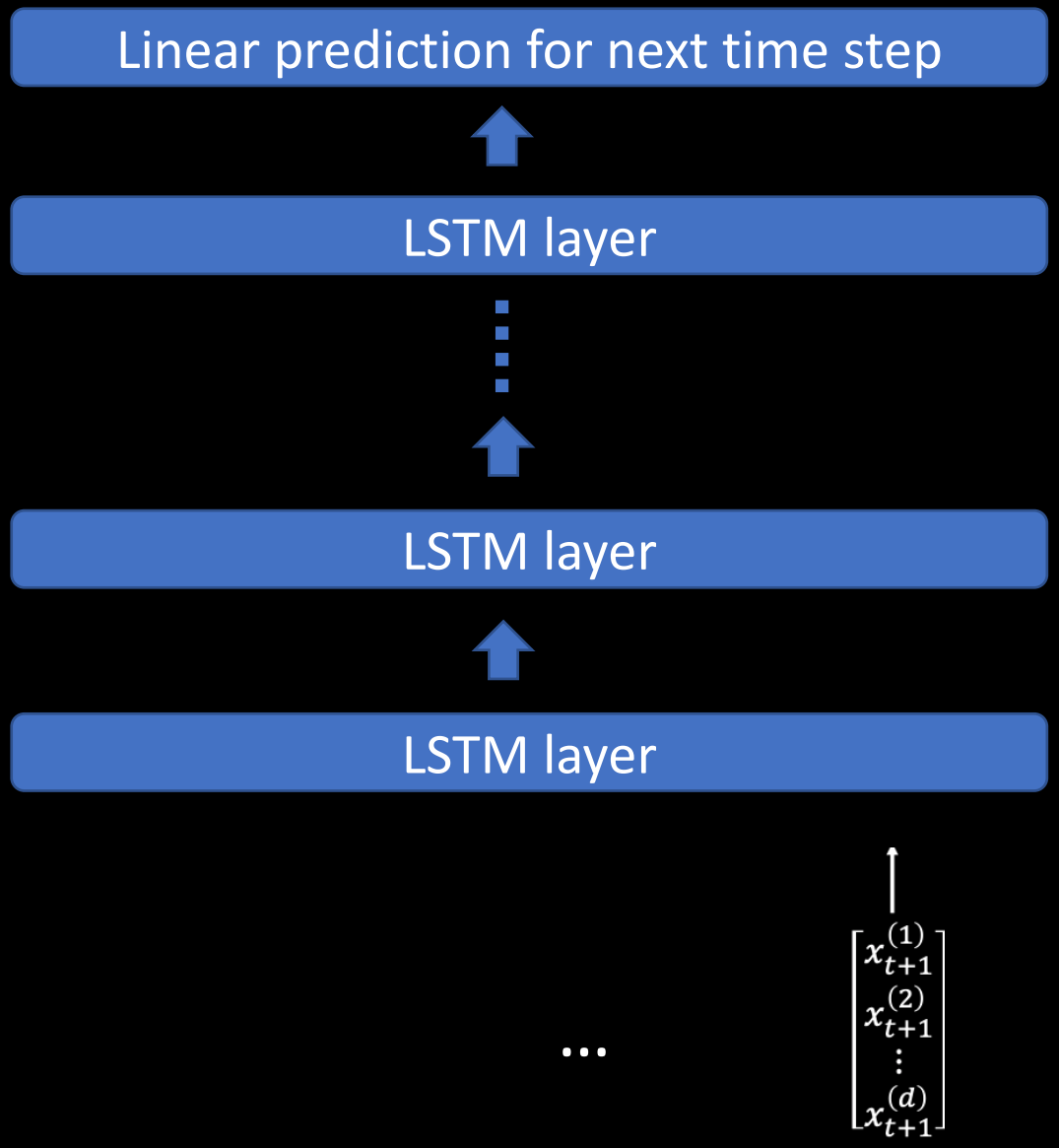
LSTMs for linear regression



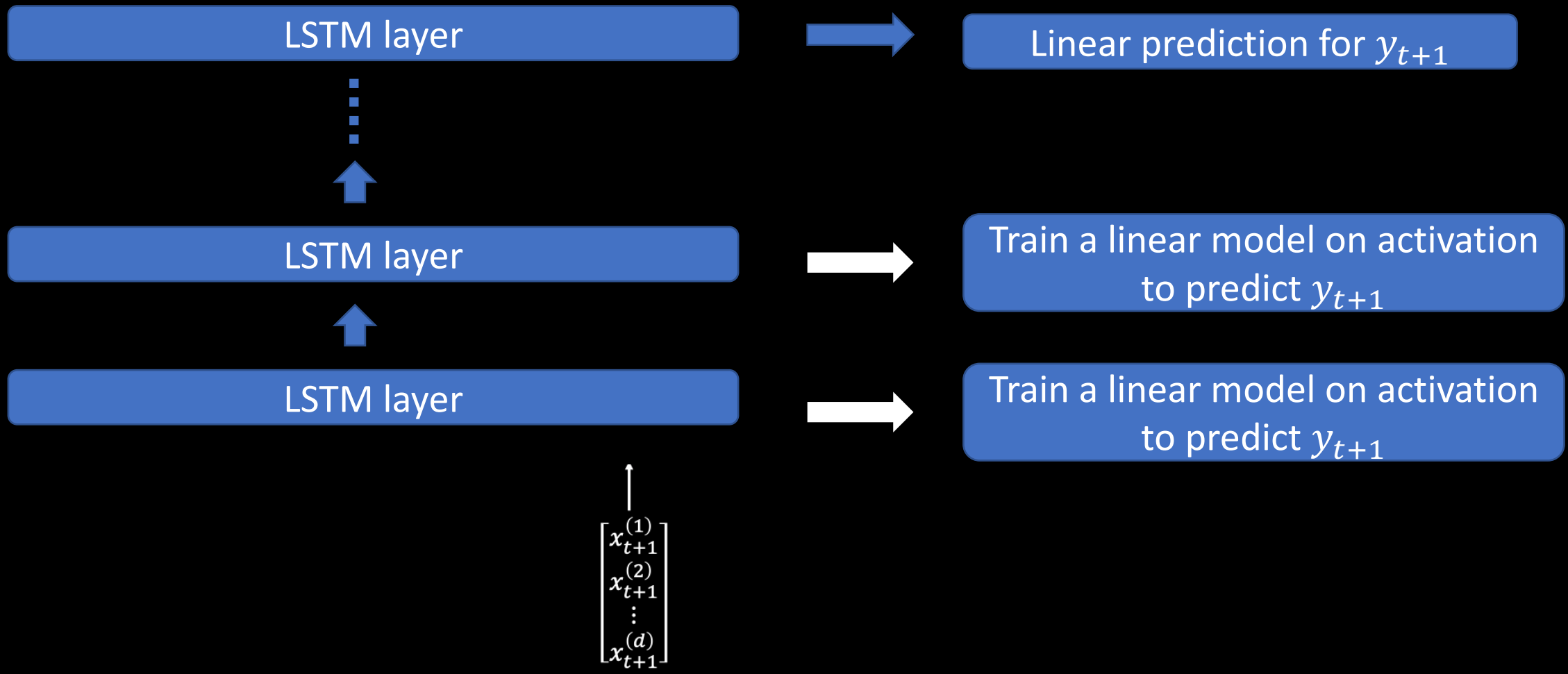
LSTMs for linear regression



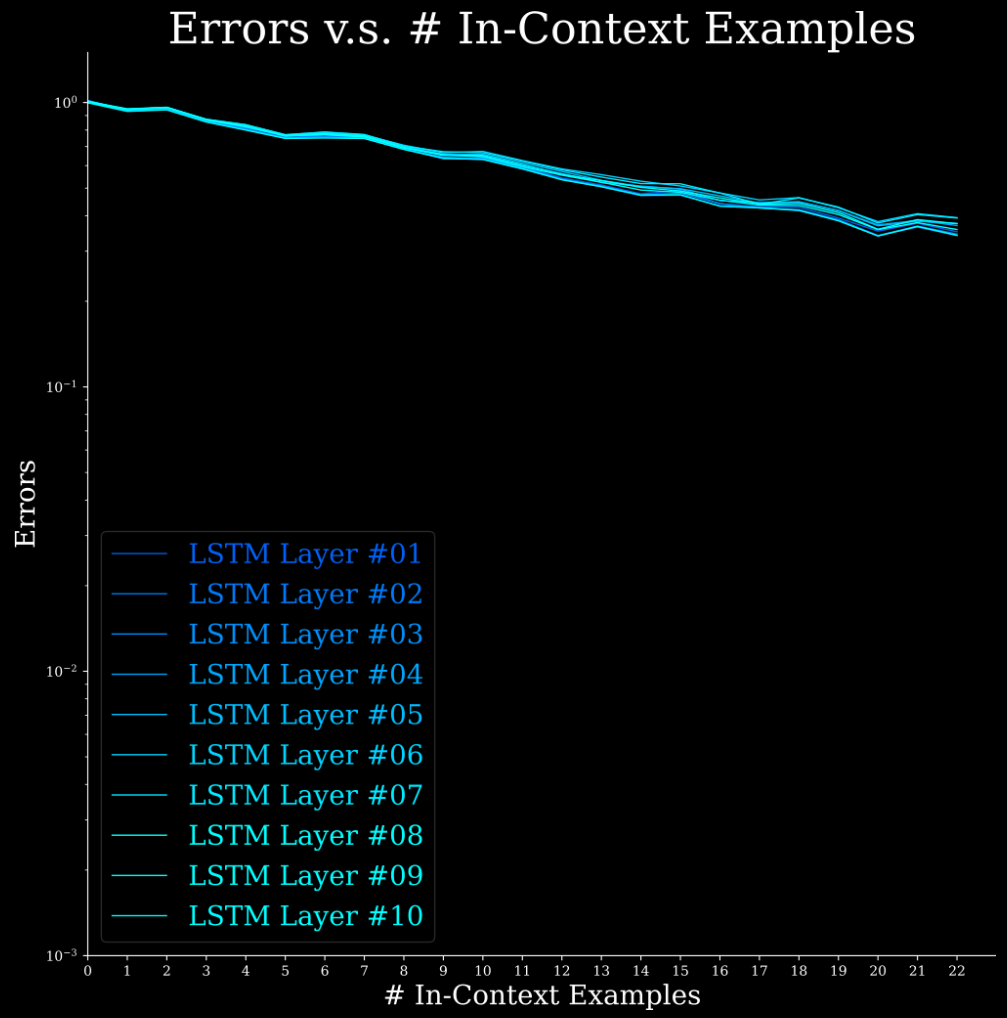
LSTMs for linear regression

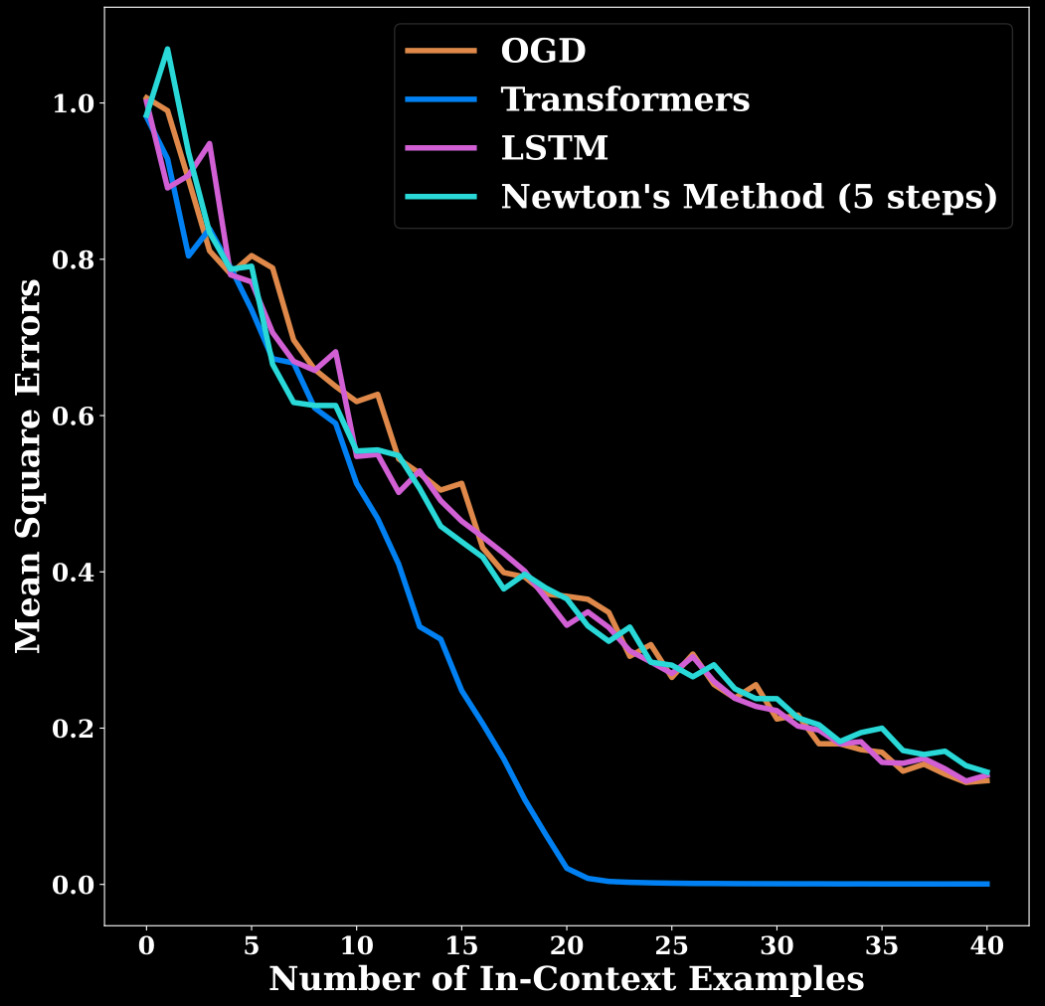


LSTMs as an iterative algorithm: probing layers



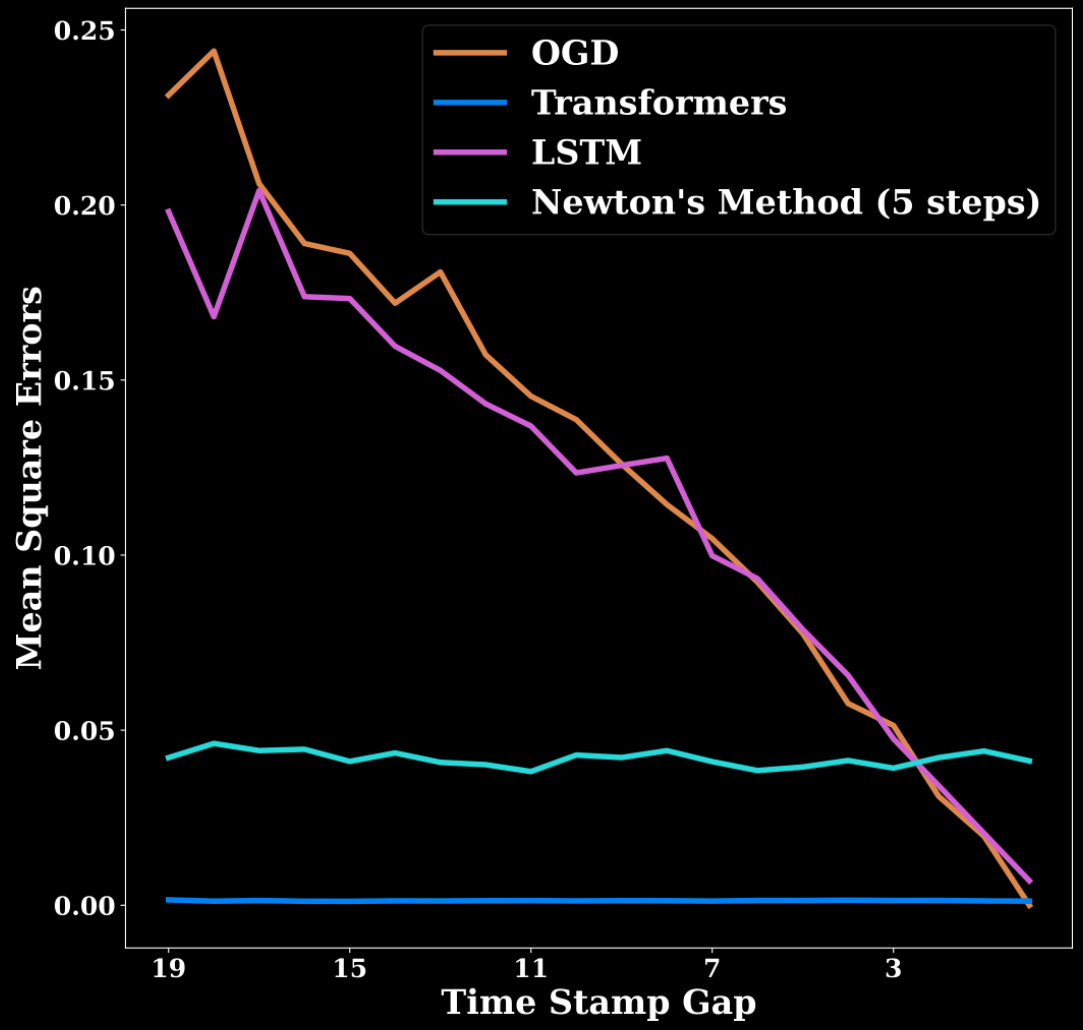
What do LSTMs implement?





LSTMs seem similar to online gradient descent

Like OGD, LSTMs ‘forget’ previous examples

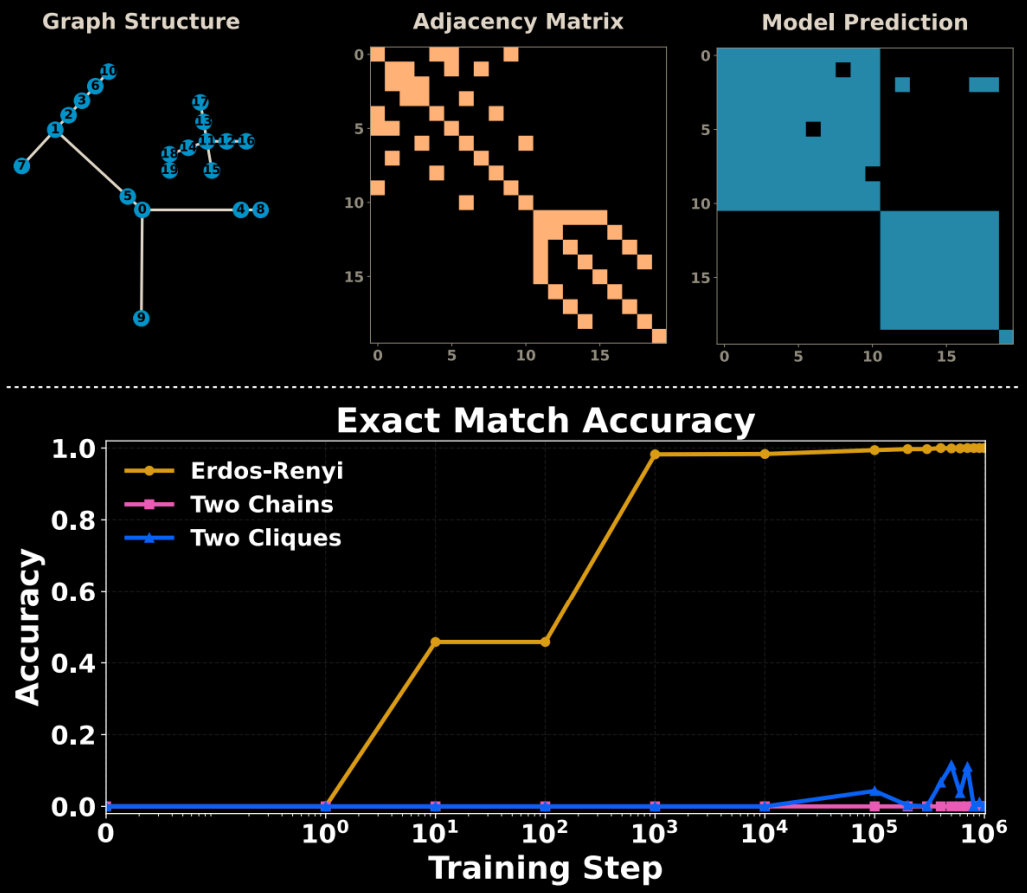


Error when input from t time steps ago is given as query point

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

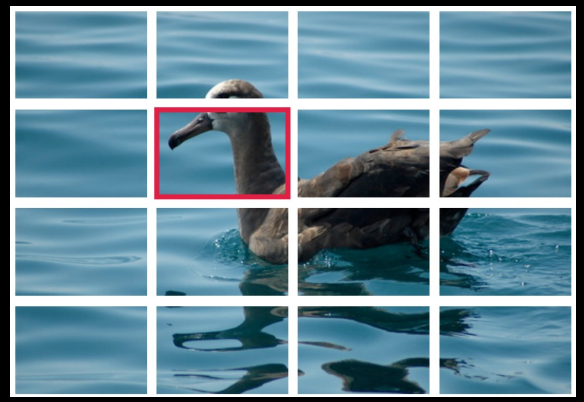


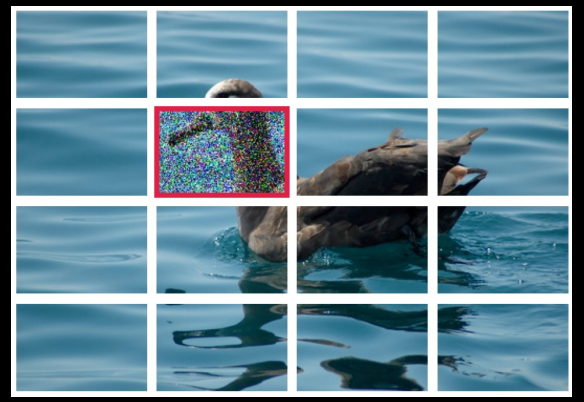
Algorithms vs. Data-driven Heuristics

- 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?





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

Evaluate model on original input

Evaluate model on perturbation to random token

Across data modalities, transformers learn lower sensitivity functions

Transformers Learn Low Sensitivity Functions: Investigations and Implications,
Bhavya Vasudeva, Deqing Fu, Tianyi Zhou, Elliott Kau, Youqi Huang, Vatsal Sharan, ICLR, 2025

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!

Slide from Josh Alman from yesterday

Can we use Transformers to discover data structures from scratch?

Omar Salemohamed
(Universite de Montreal/MILA)

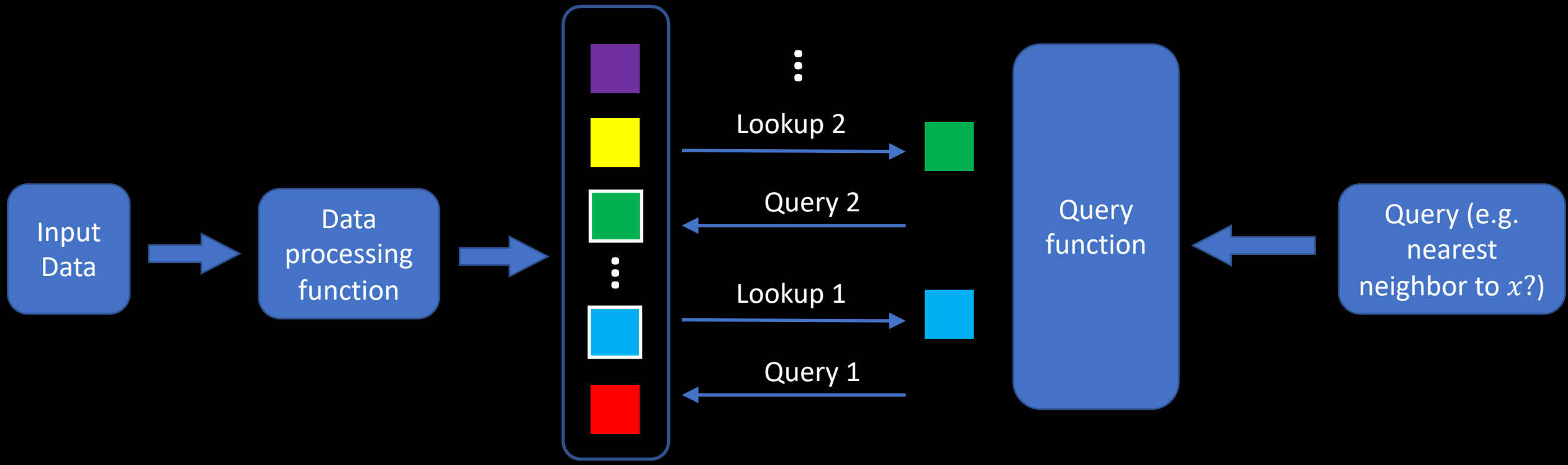
Laurent Charlin
(HEC Montreal/MILA)

Shivam Garg
(MSR NYC)

Greg Valiant
(Stanford)

Discovering Data Structures: Nearest Neighbor Search and Beyond, Neurips 2025

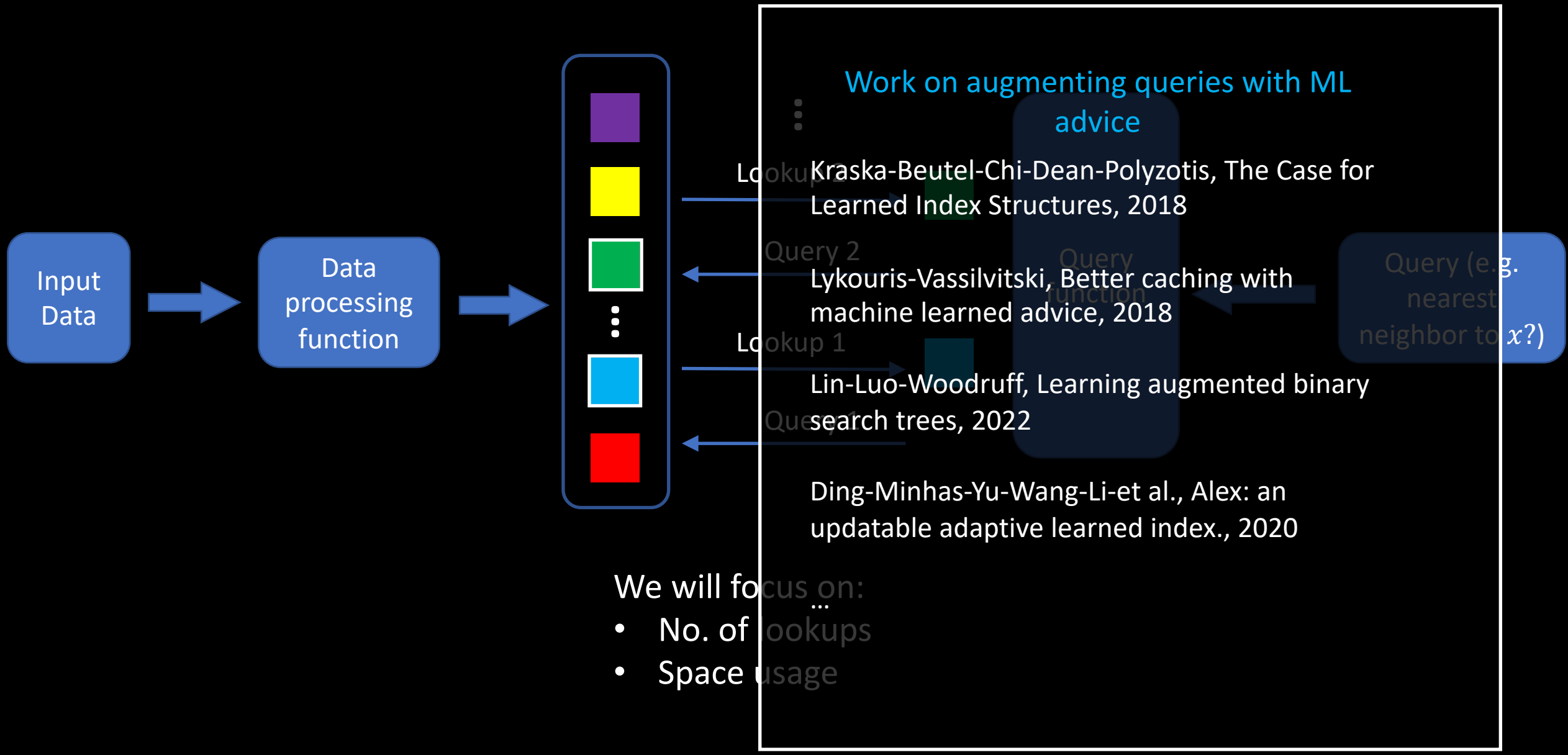
Data structures (think nearest neighbor lookup in 1D)



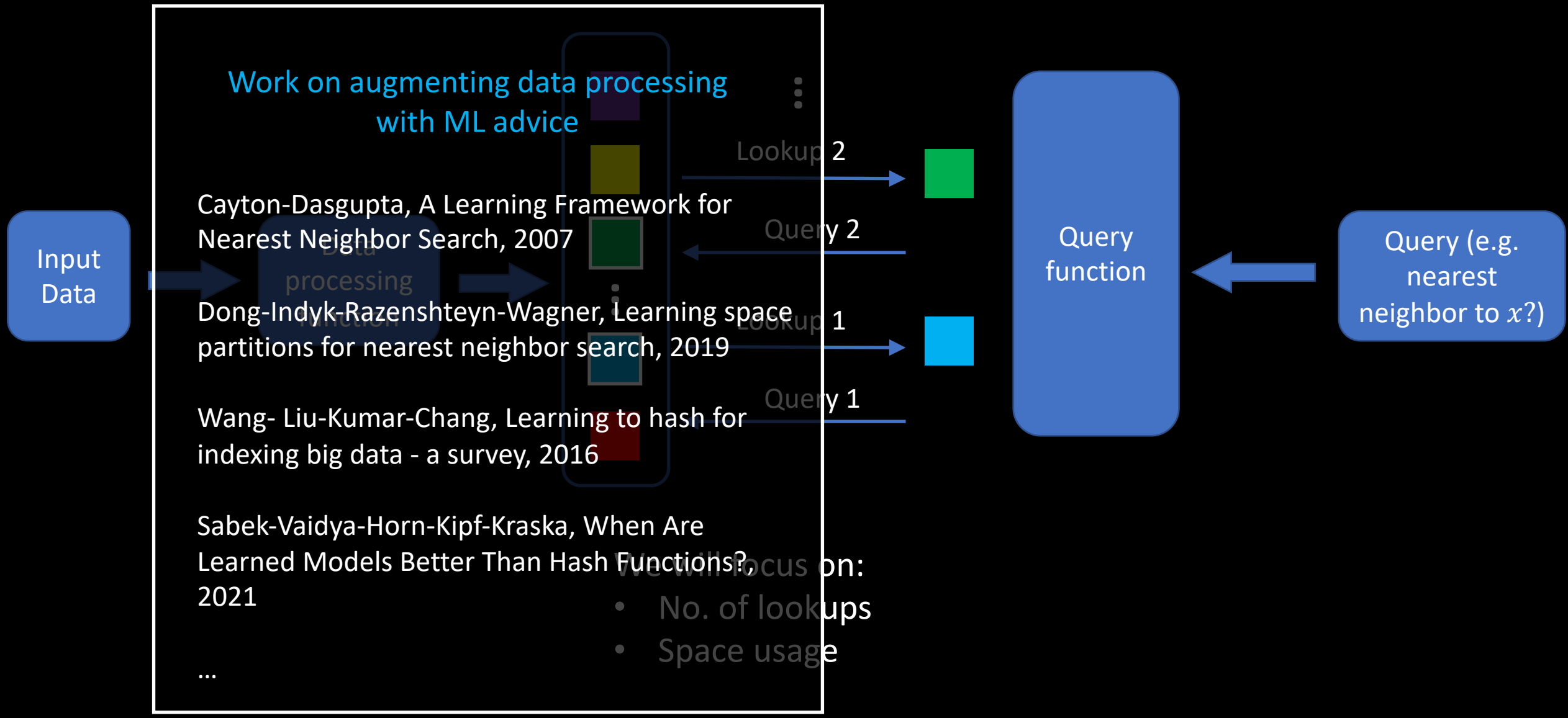
We will focus on:

- No. of lookups
- Space usage

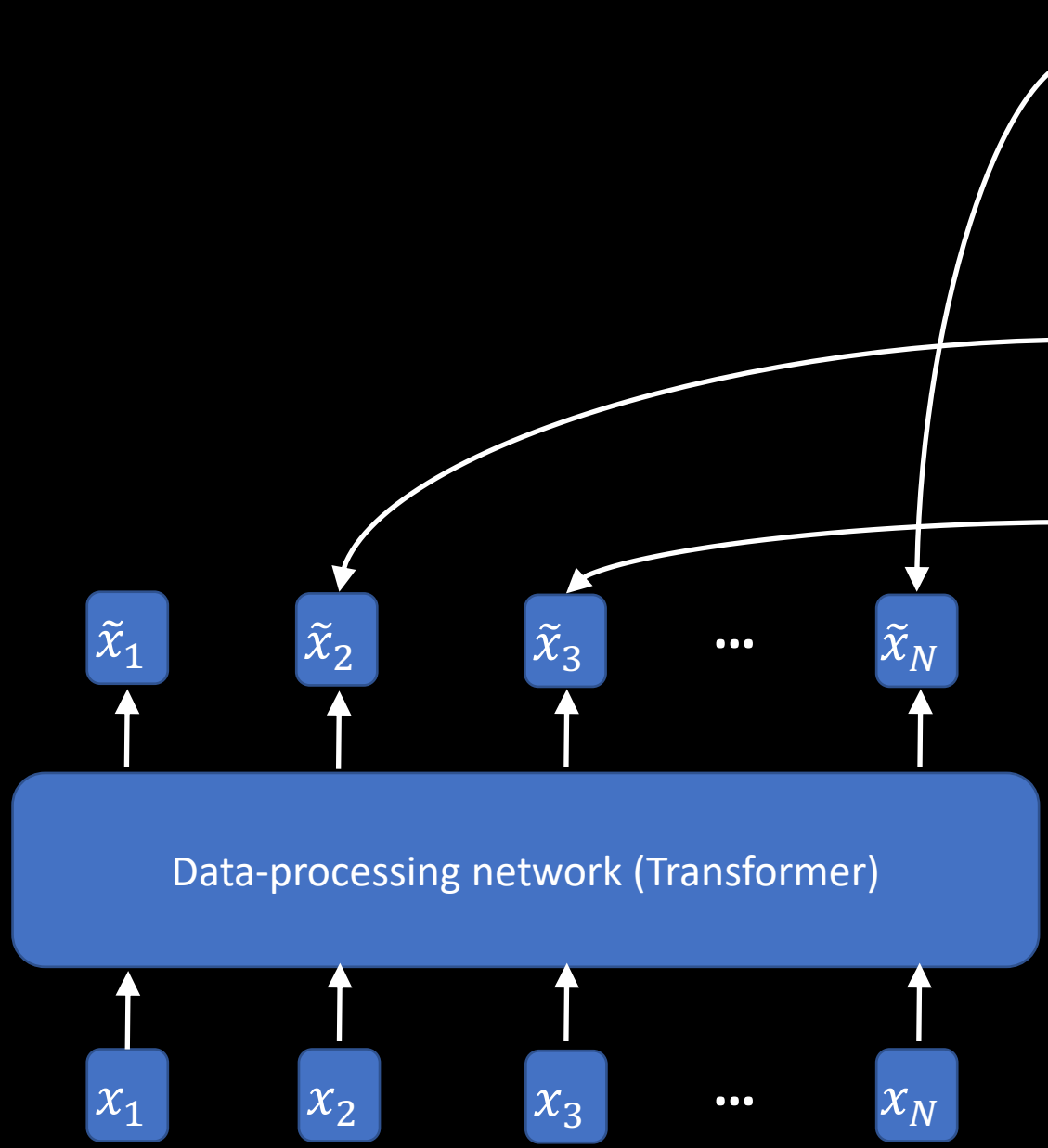
Recent work has tried to augment data structures with ML



Recent work has tried to augment data structures with ML



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



Query q,
all previous queries
& lookups

Query q,
query & lookup of MLP_1

Query q

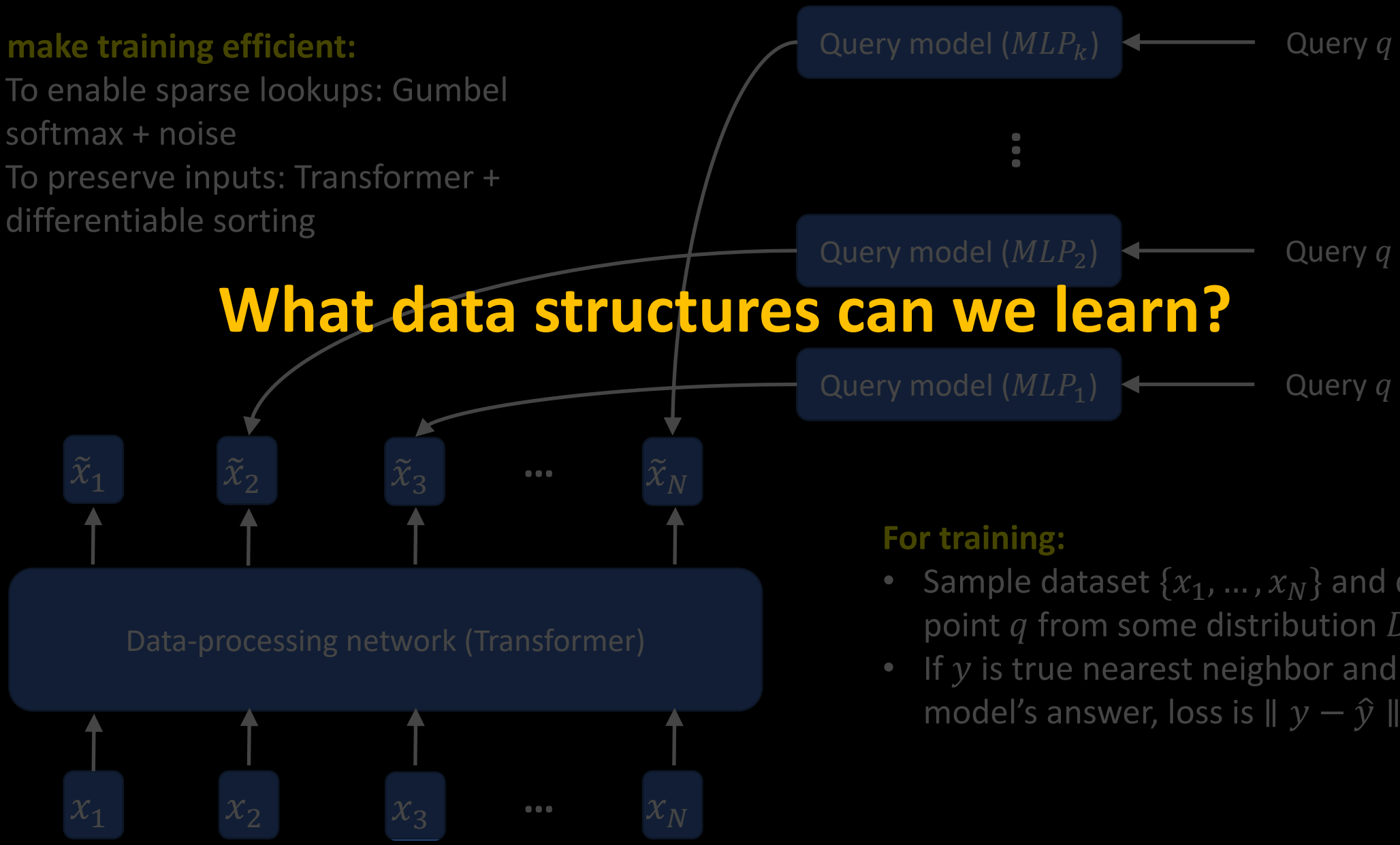
For training:

- 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

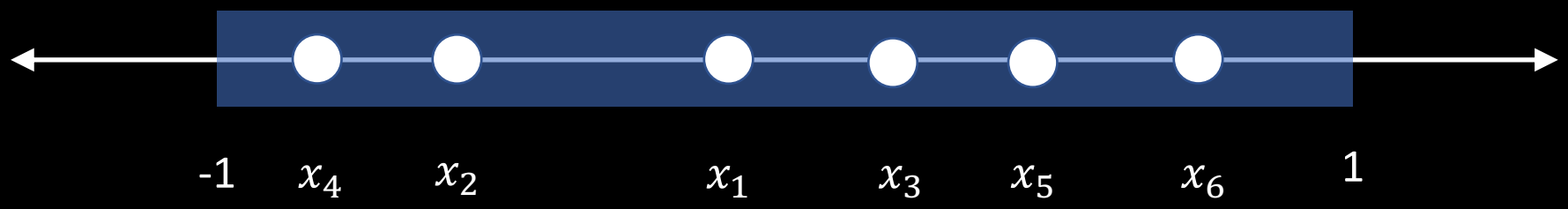


What data structures can we learn?

For training:

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

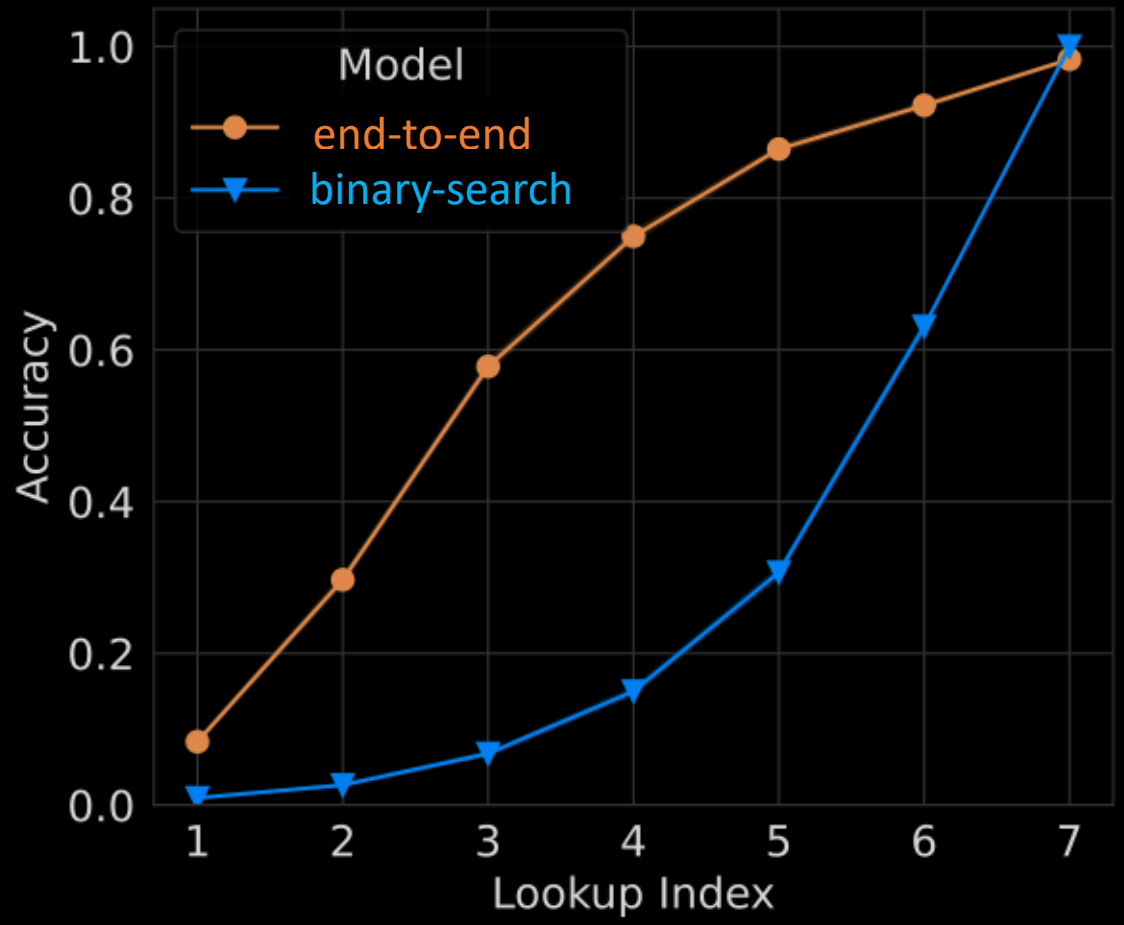
Uniform distribution in 1D

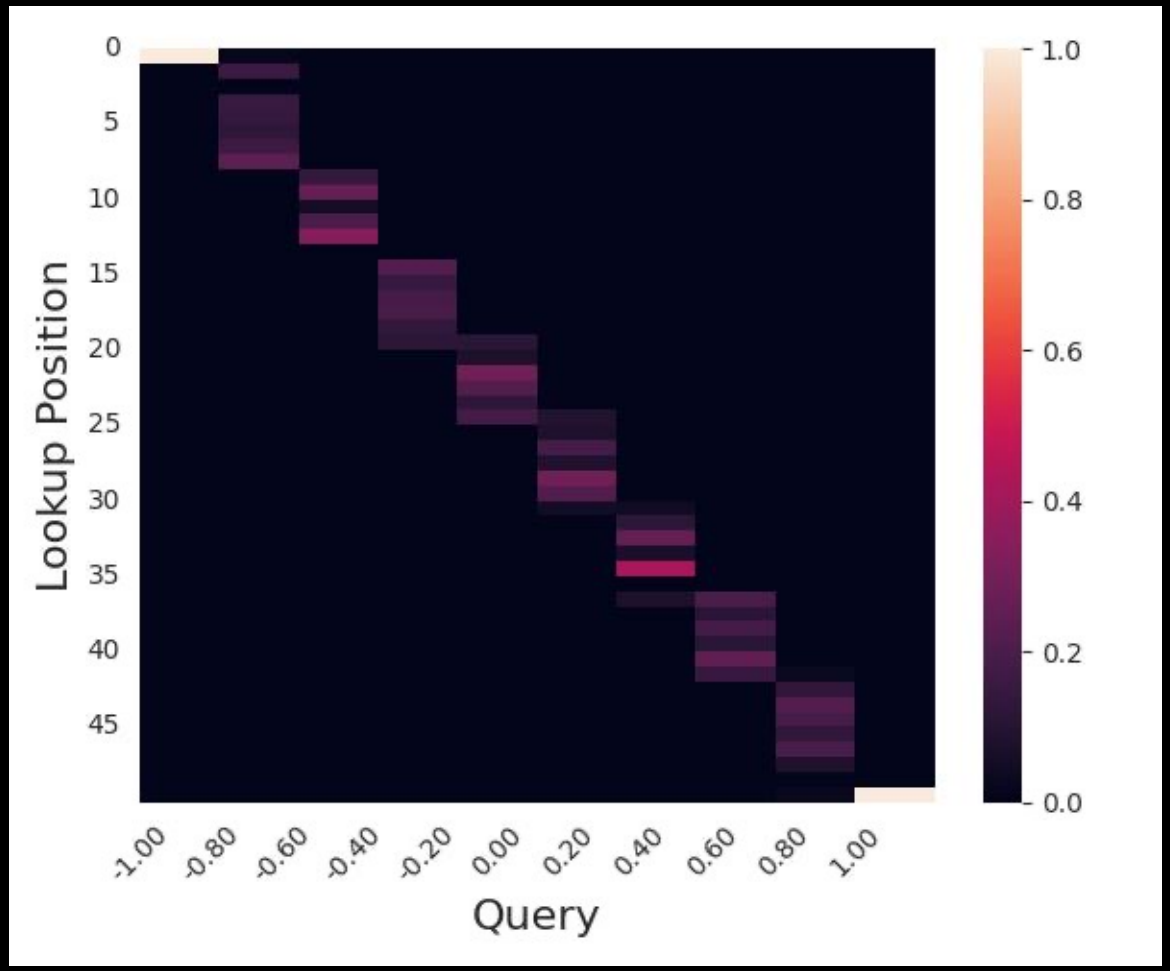


Model trained on this distribution:

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

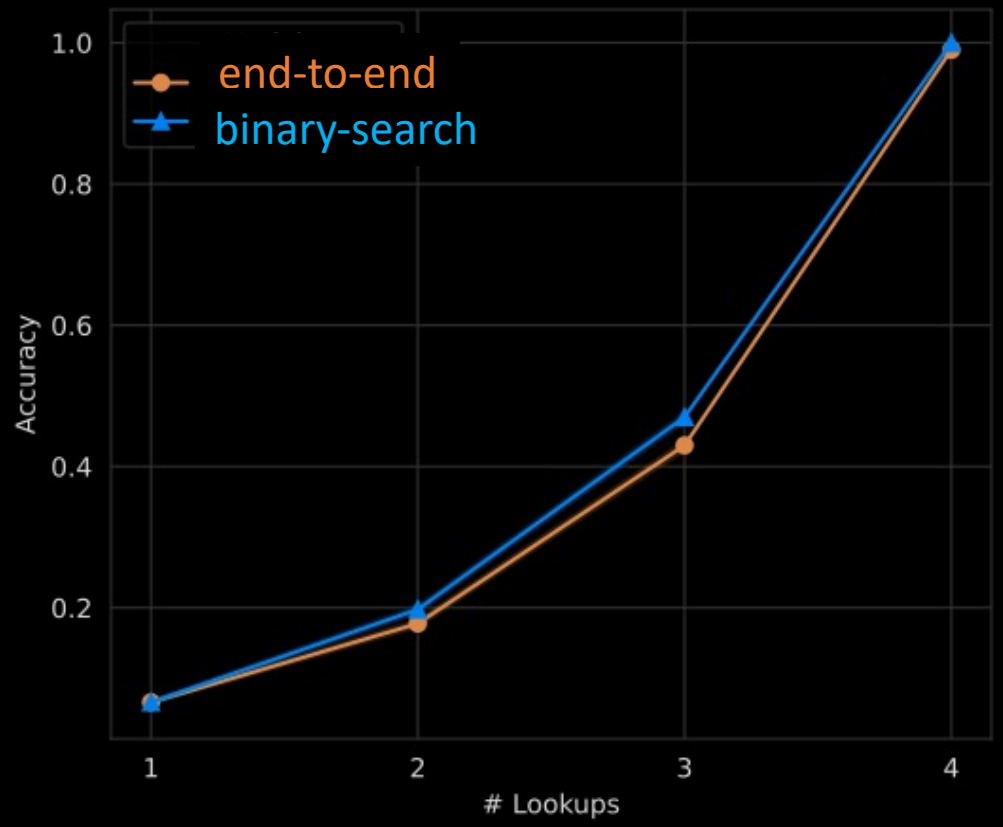
Model outperforms binary search

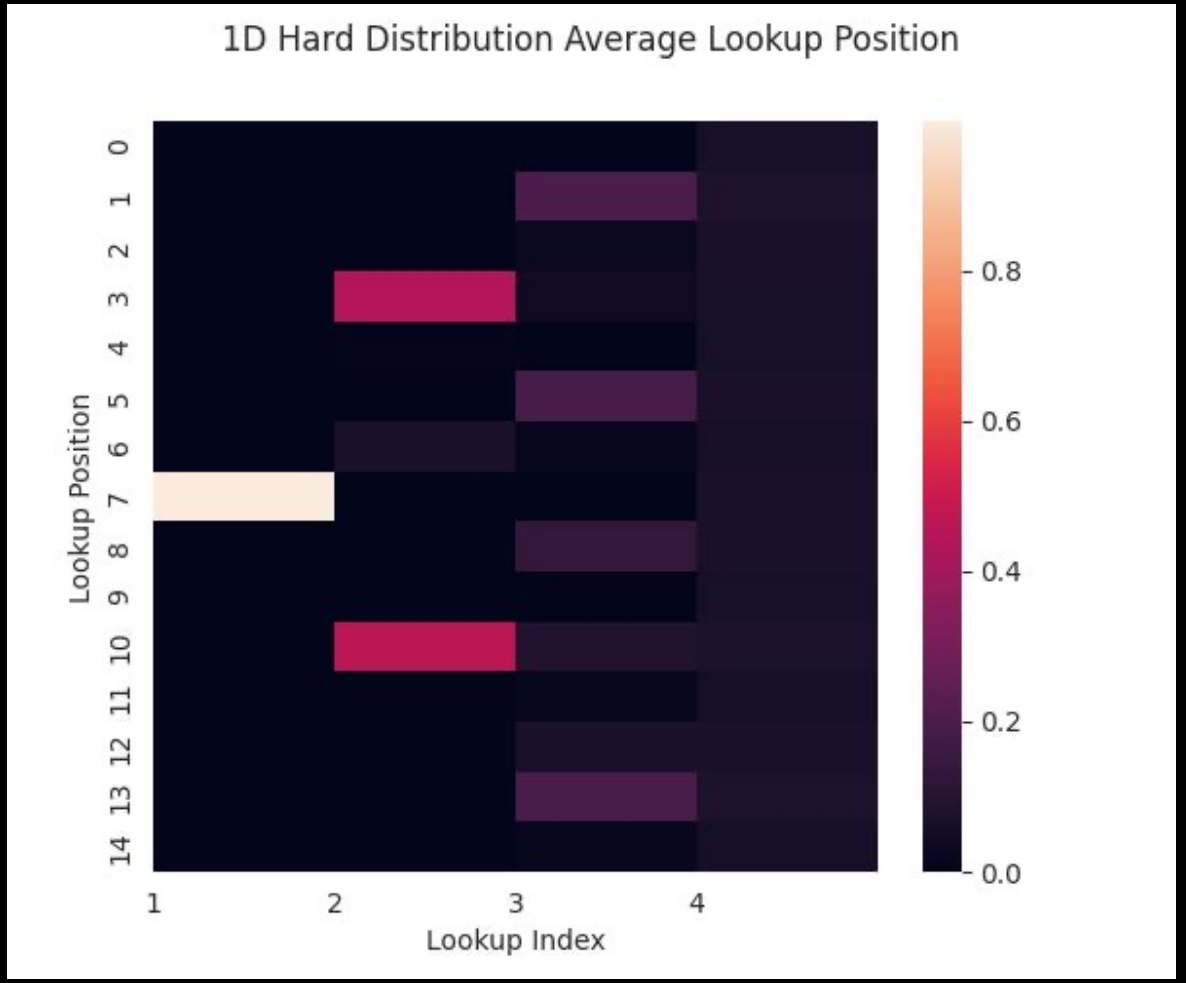




Query model begins search not far from nearest neighbor

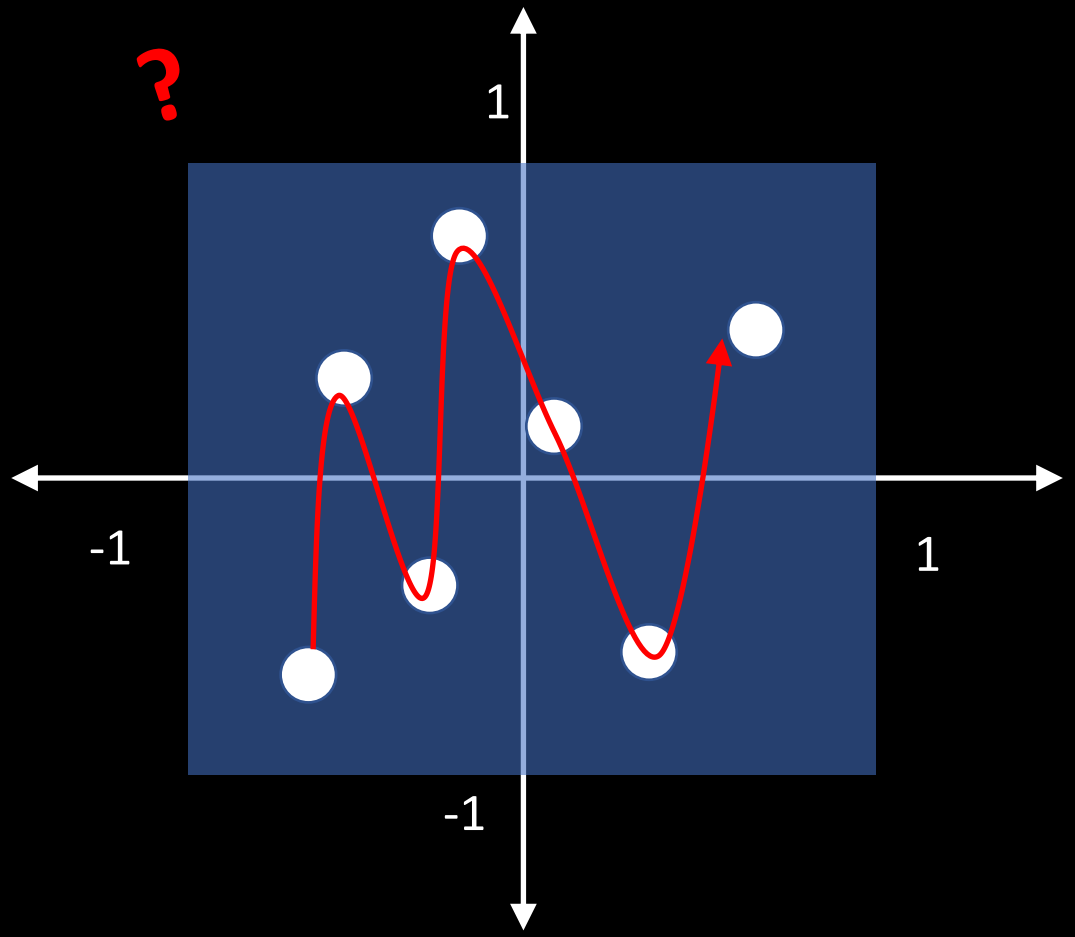
Harder 1D distribution where quantiles don't concentrate



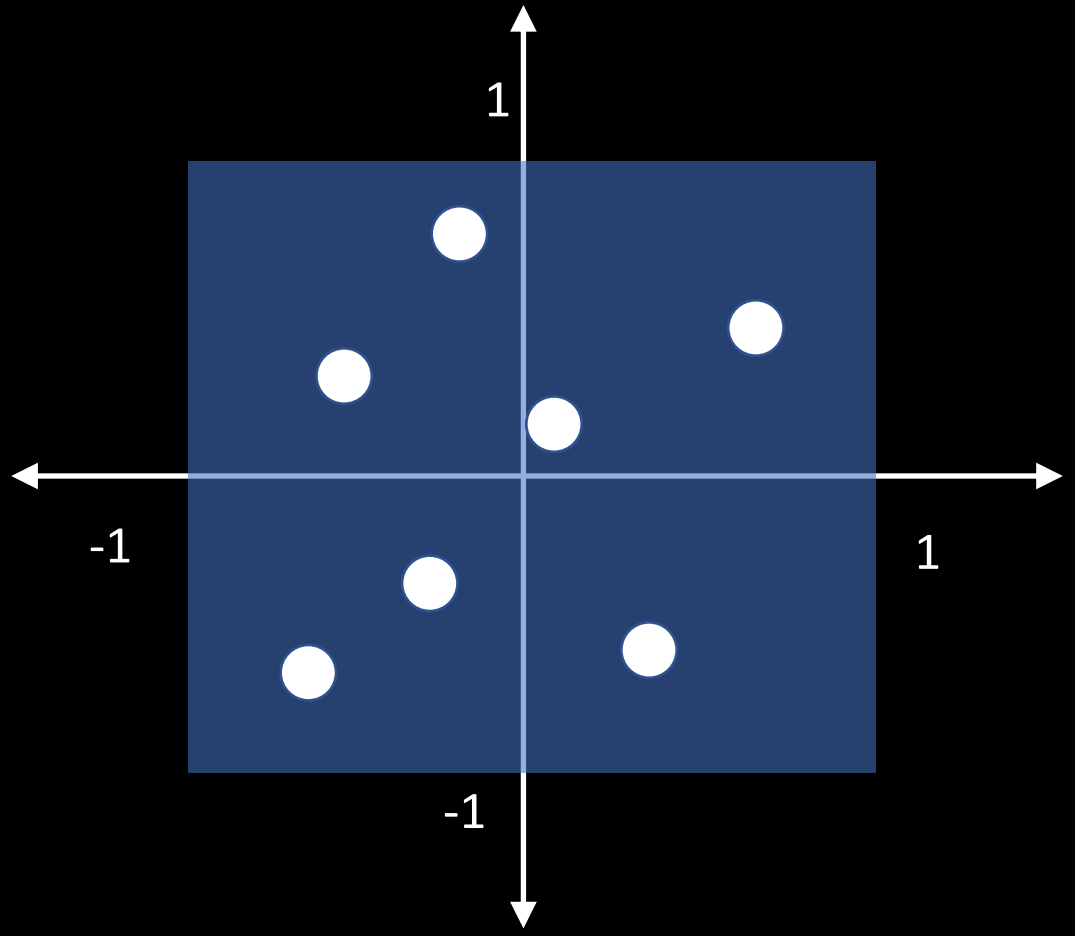


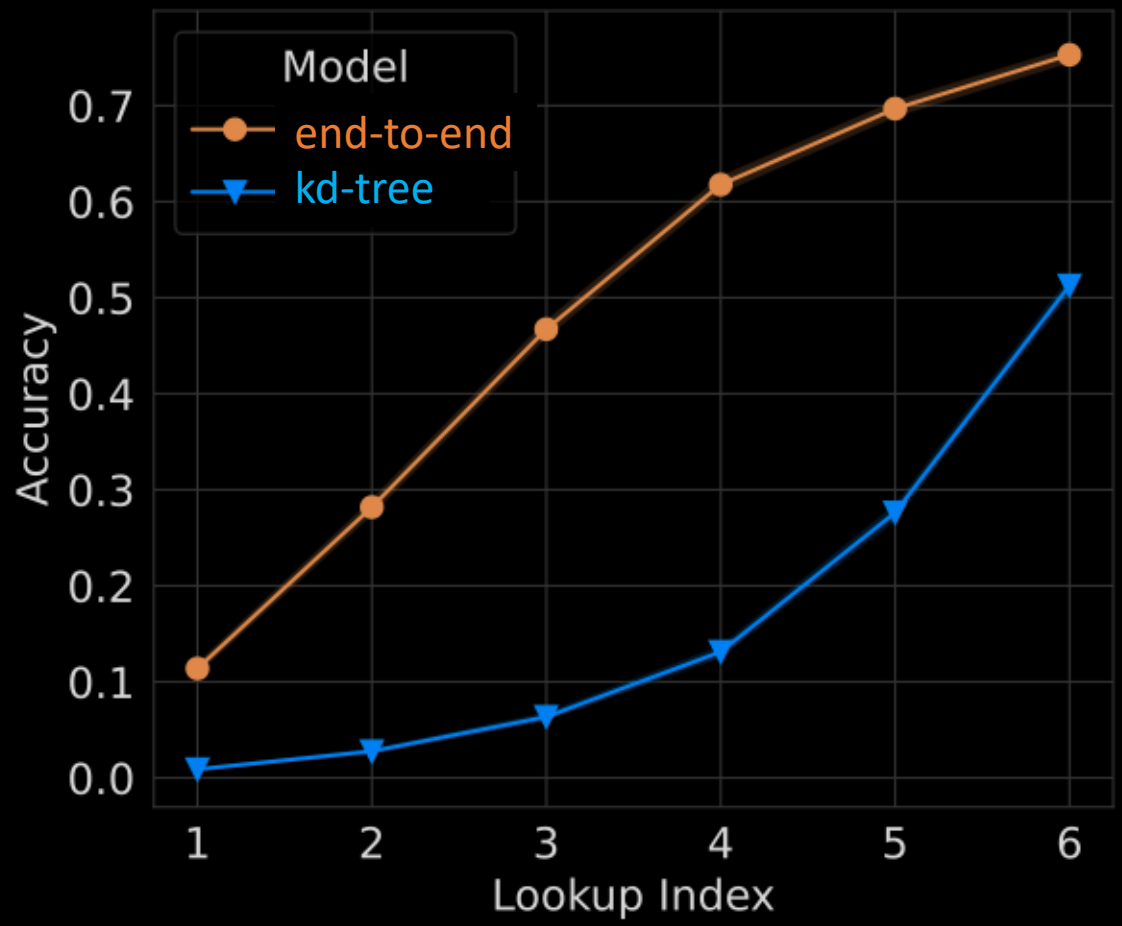
Model learns binary search!

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

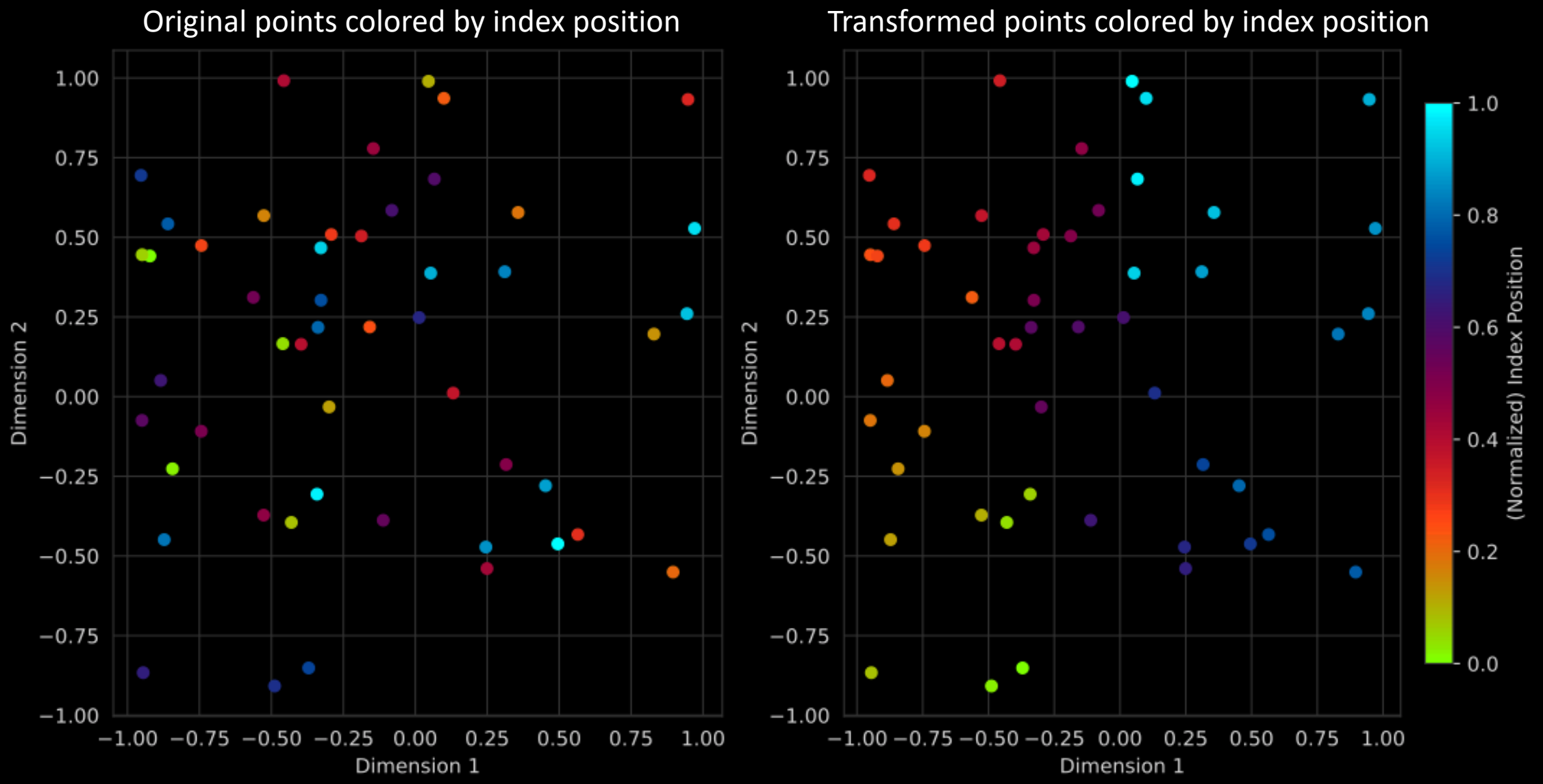


Uniform distribution in 2D: Outperforms kd-trees

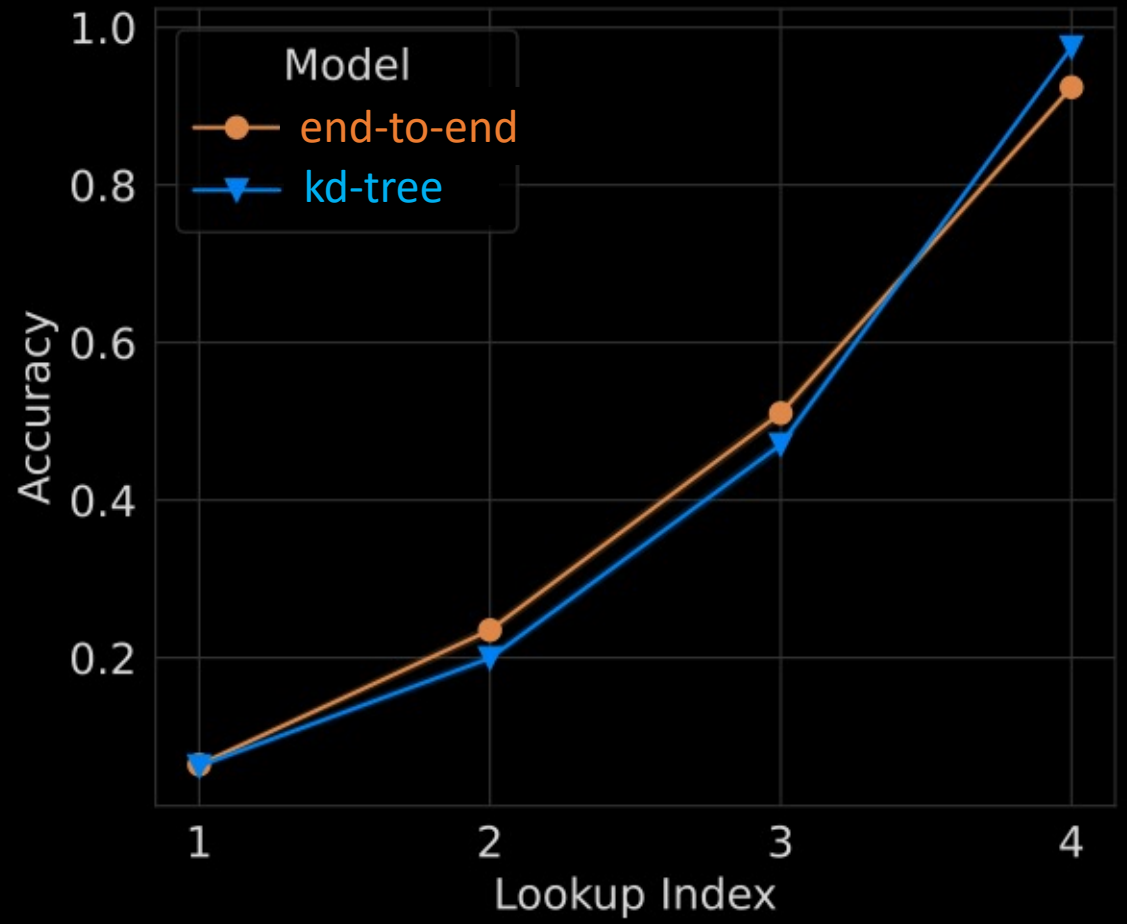




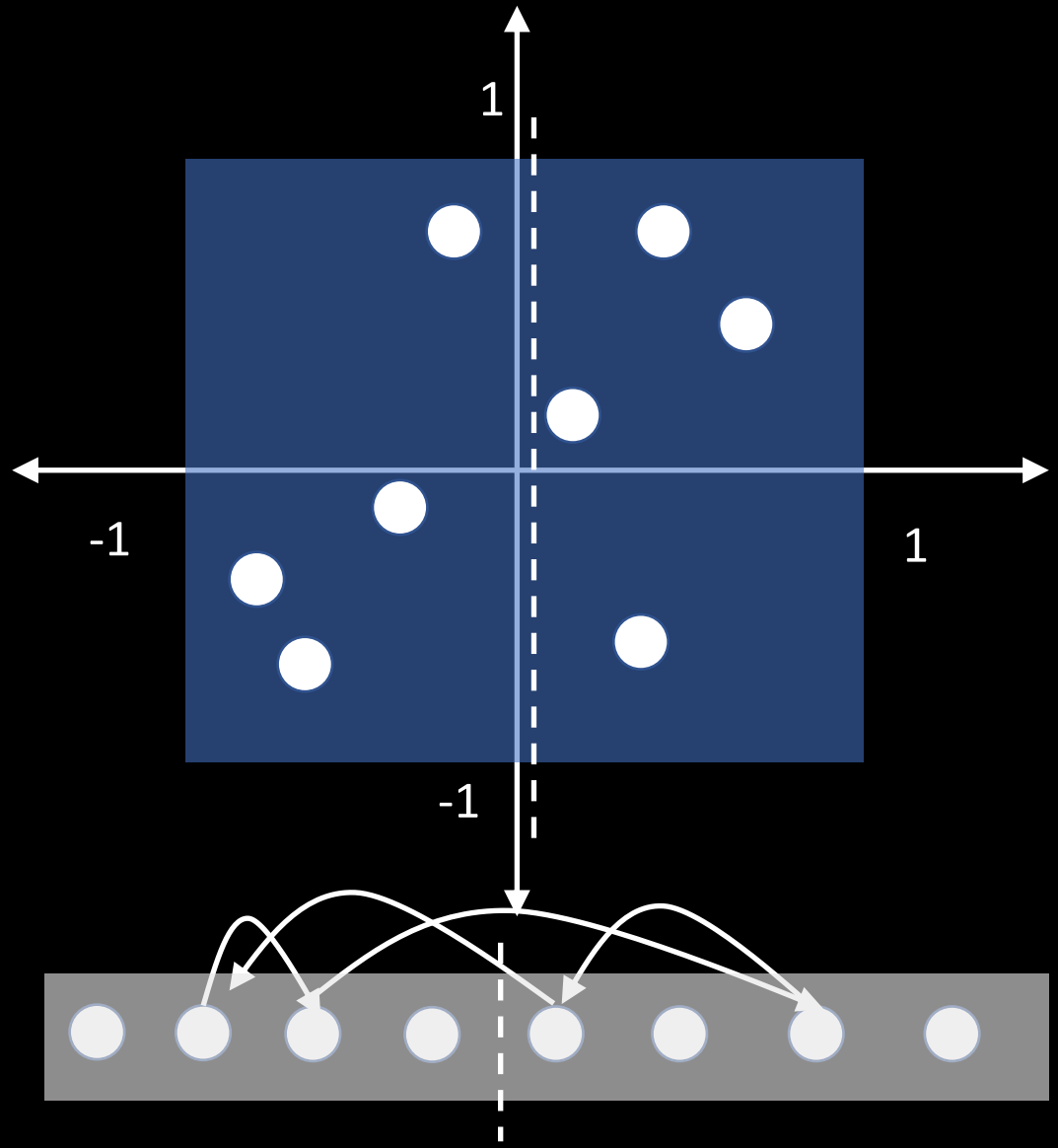
Model learns to index nearby points together



Hard distribution in 2D: Matches kd-trees

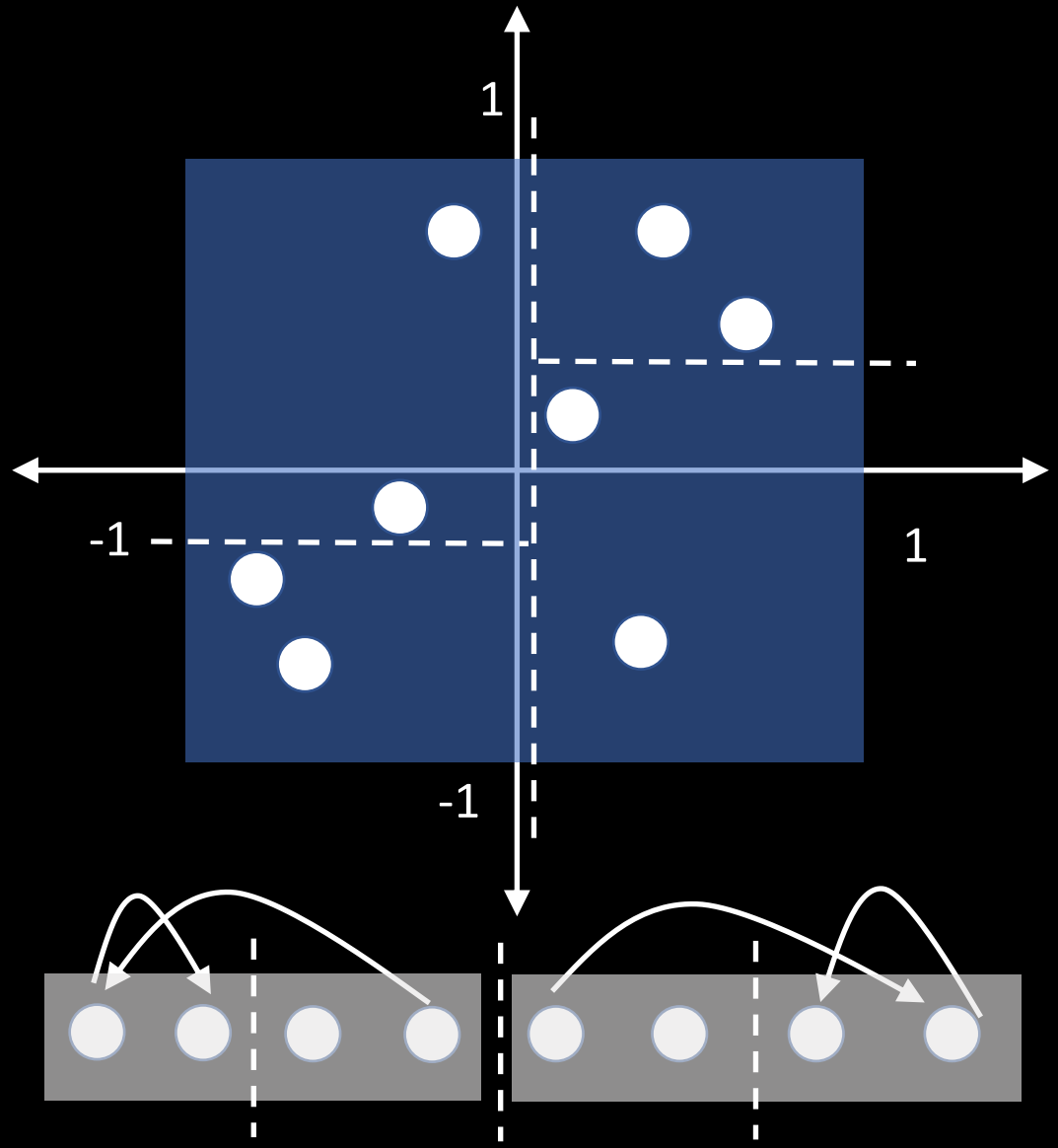


Can see that the model is essentially recovering a kd-tree!



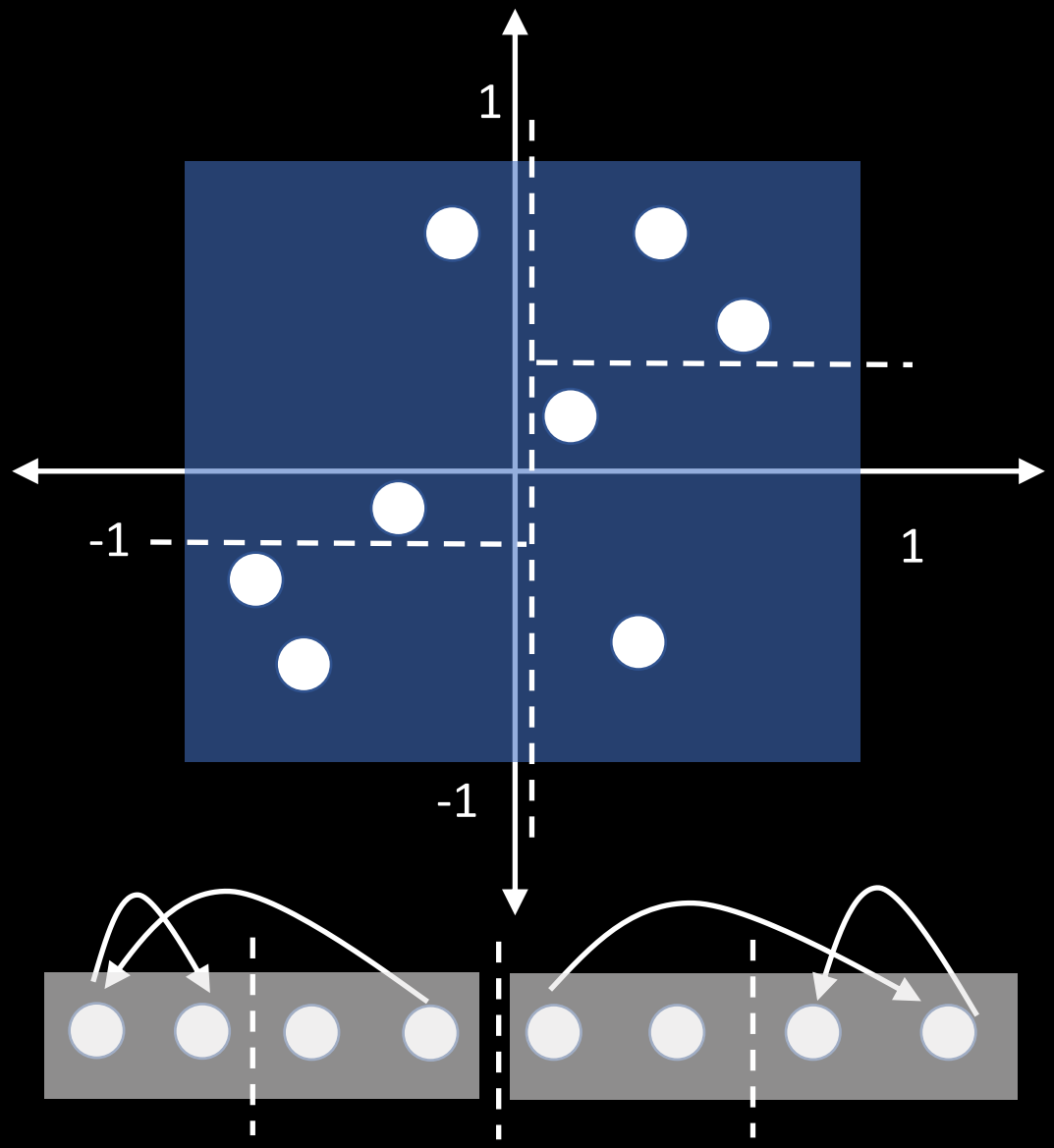
Sort by x-axis

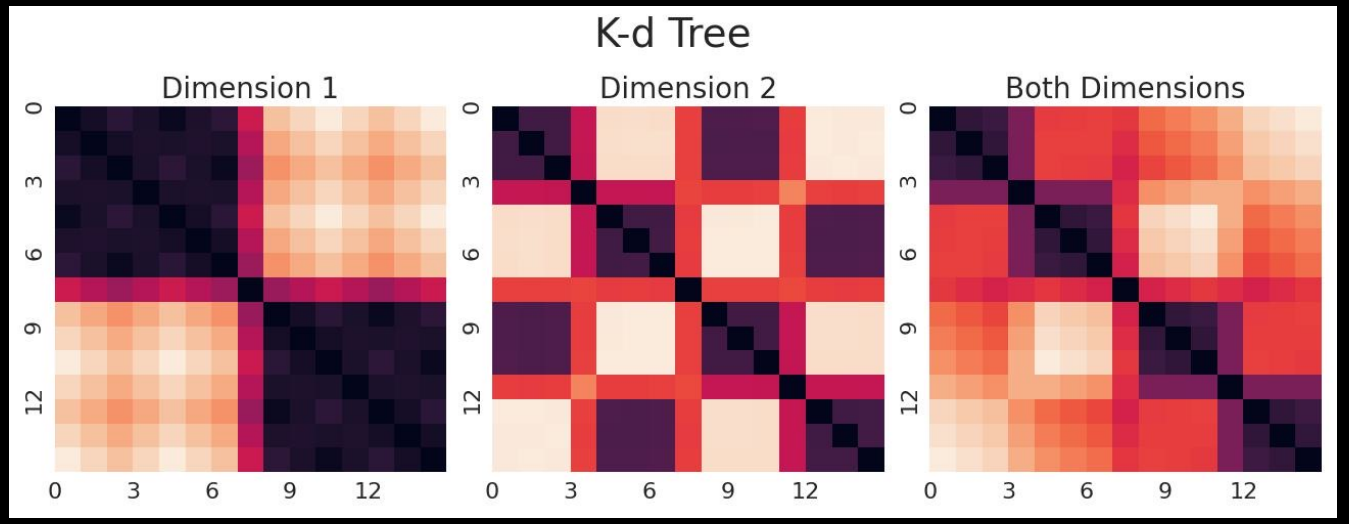
Can see that the model is essentially recovering a kd-tree!

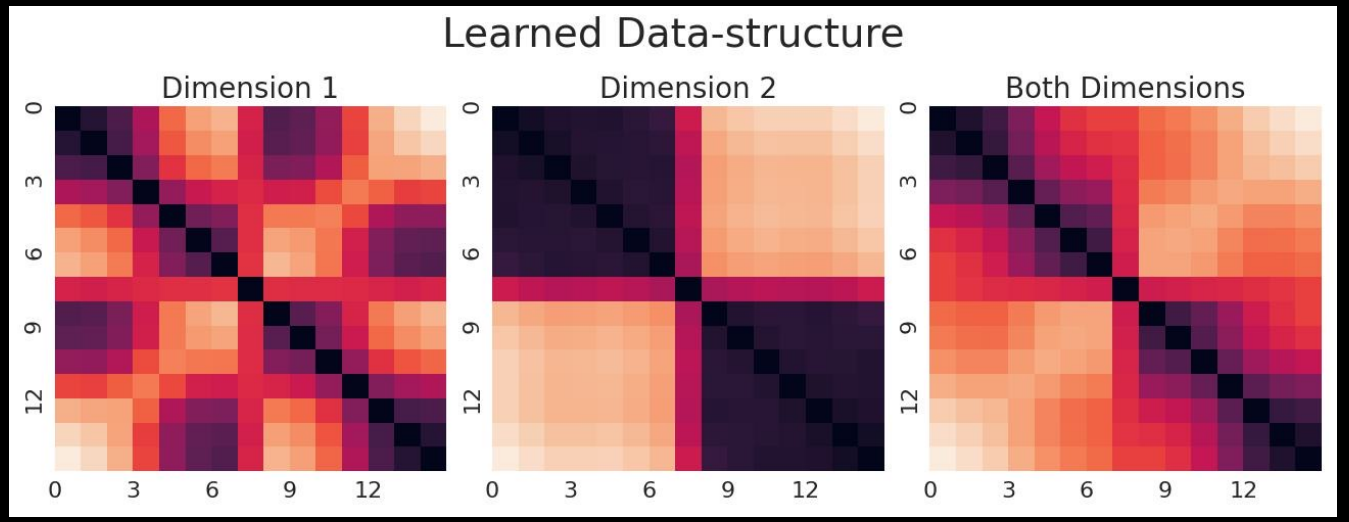


Sort each half
by y-axis

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

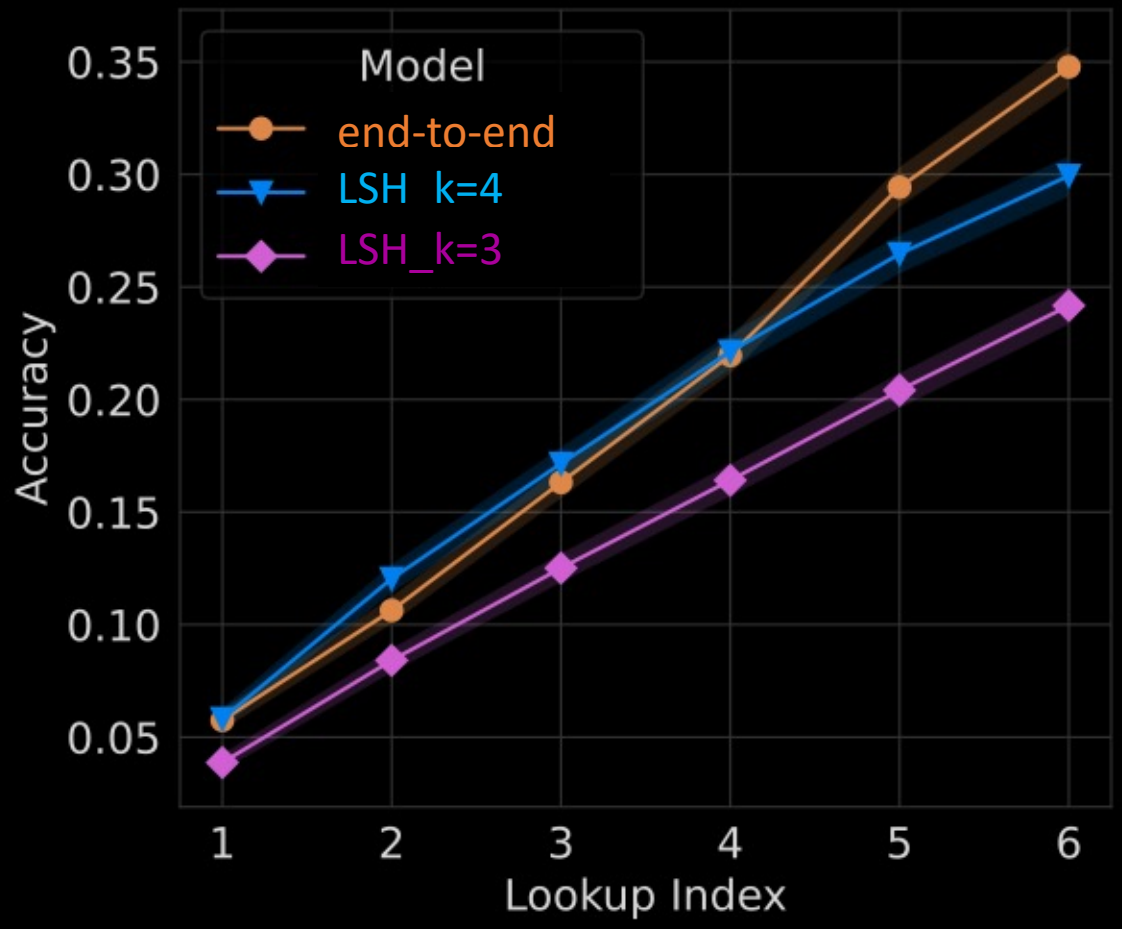




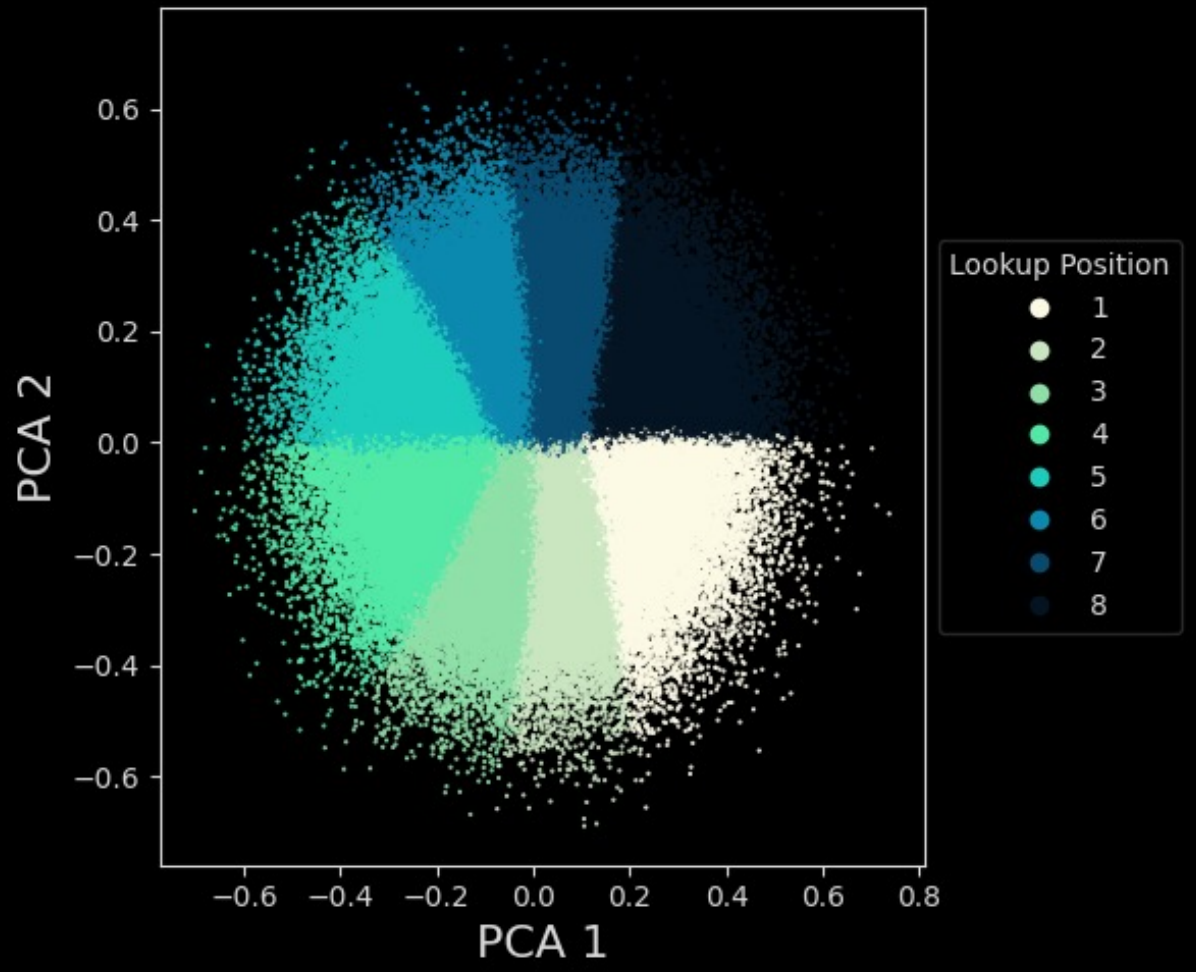


Uniform distribution in 30D: Matches LSH

- 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

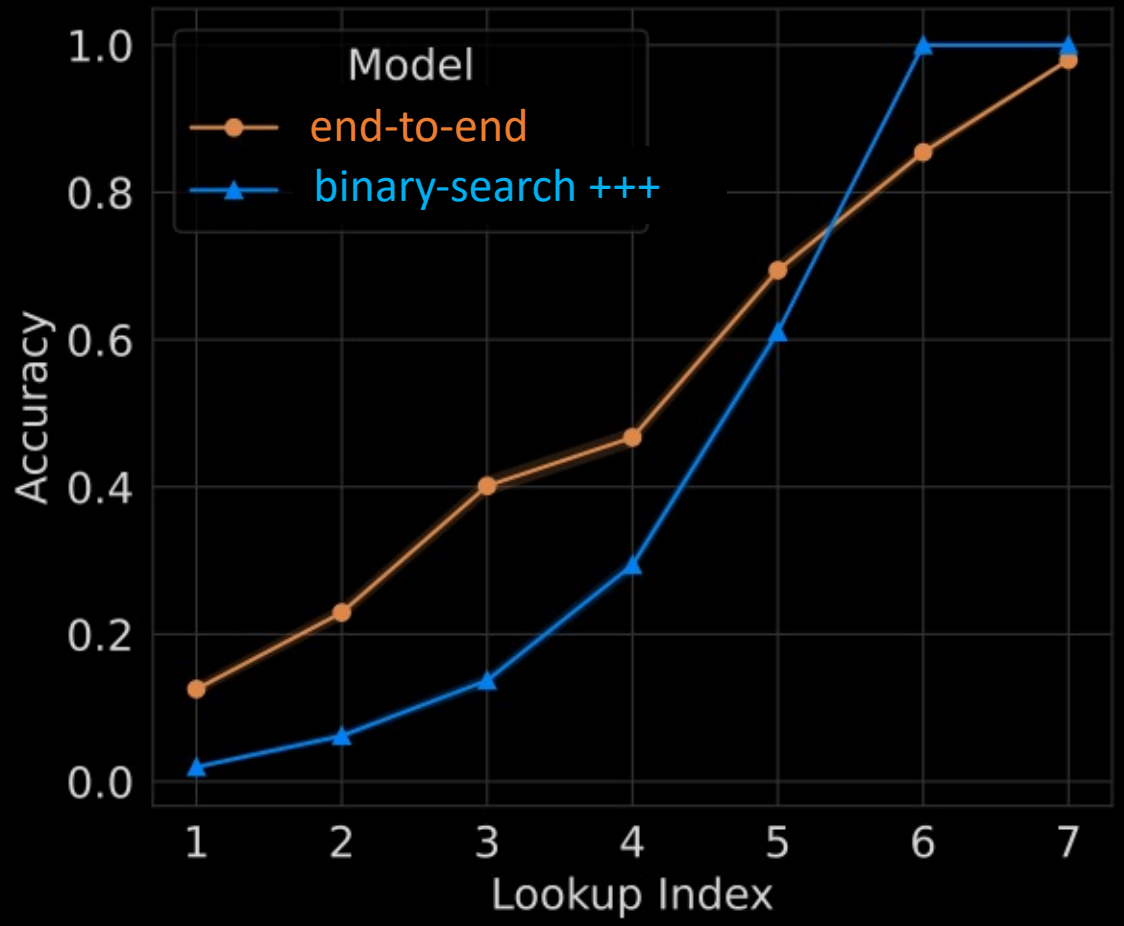
Input: 50 images of numbers uniformly drawn from [0,200]

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

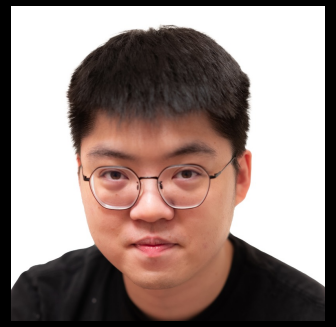
- Model also learns to use extra space
- We also show we can learn data structures for frequency estimation in a data stream, recovering/outperforming count-sketch

Models outperform data-independent baselines

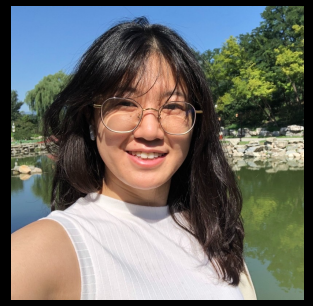
- Also consider settings with power-law distributions etc.

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

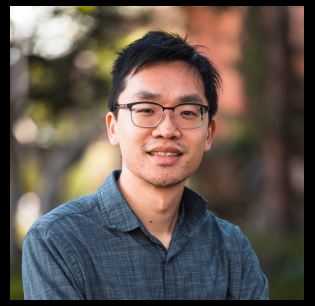
- ***Can we use these to understand tradeoffs in theory, build better strategies for high-dimensional NN search and other data structure problems?***
- ***Can we use a similar methodology for algorithm discovery under resource constraints more generally, for example to discover new optimization techniques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



Deqing Fu



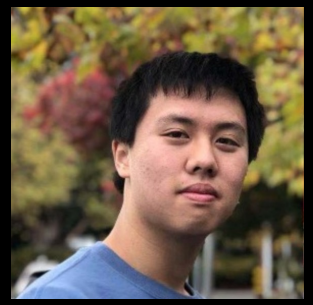
Tianqi Chen



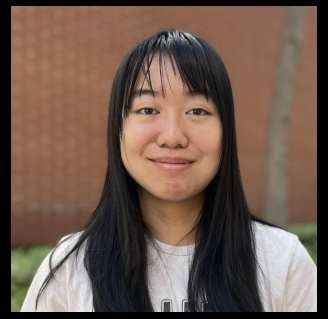
Robin Jia

Tianyi Zhou

Bhavya Vasudeva



Elliot Kau



You-Qi Huang

Omar Salemohamed

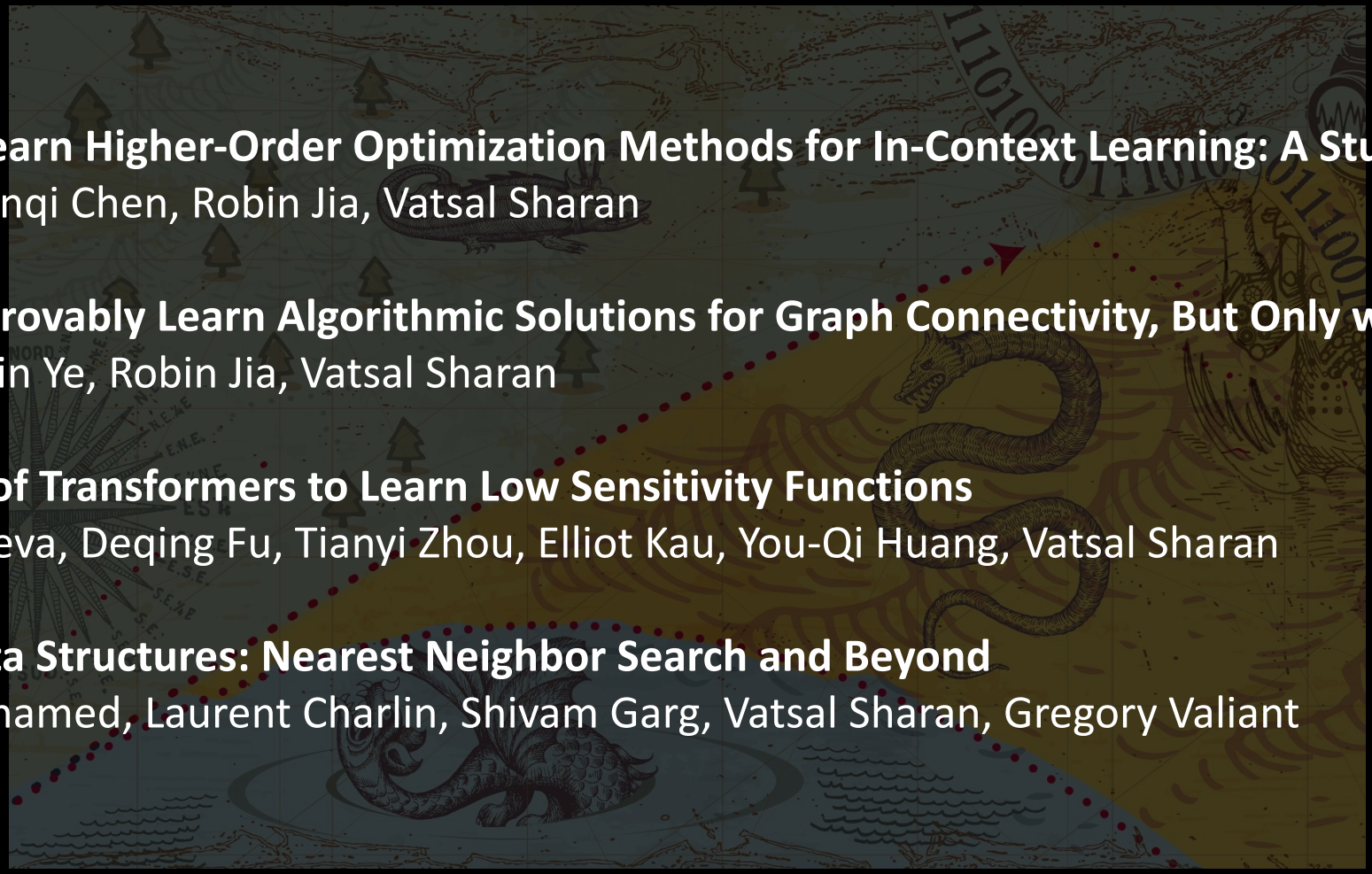
Laurent Charlin

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

- **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