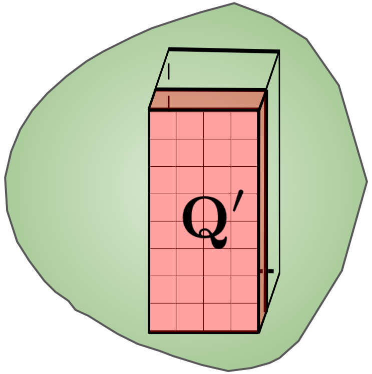
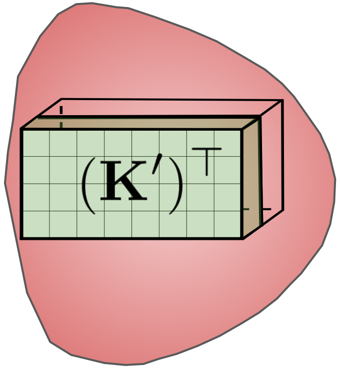
softmax attention



Performers



Towards New Transformers' Revolution

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, Adrian Weller, Vikas Sindhwani

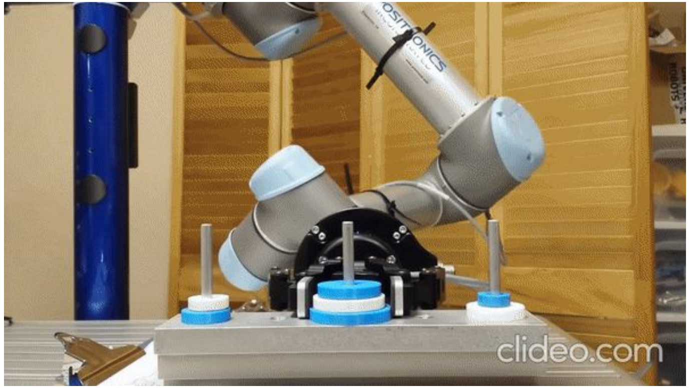
"Rethinking Attention with Performers" Google AI Blog Post code

to appear @ ICLR 2021, oral presentation,
top 40 accepted papers

Why we need better Memorization & Attention in ML?

temporal/spatial attention

Fig. 1 DeepMind policy navigating simply by "sight".



temporal attention

Fig. 2 Robotic arm "solving" Hanoi towers.

- *Lifelong Learning* requires going beyond purely-reactive Robotics tasks:

  "Developmental Robotics: A Complex Dynamical System with Several Spatiotemporal scales."

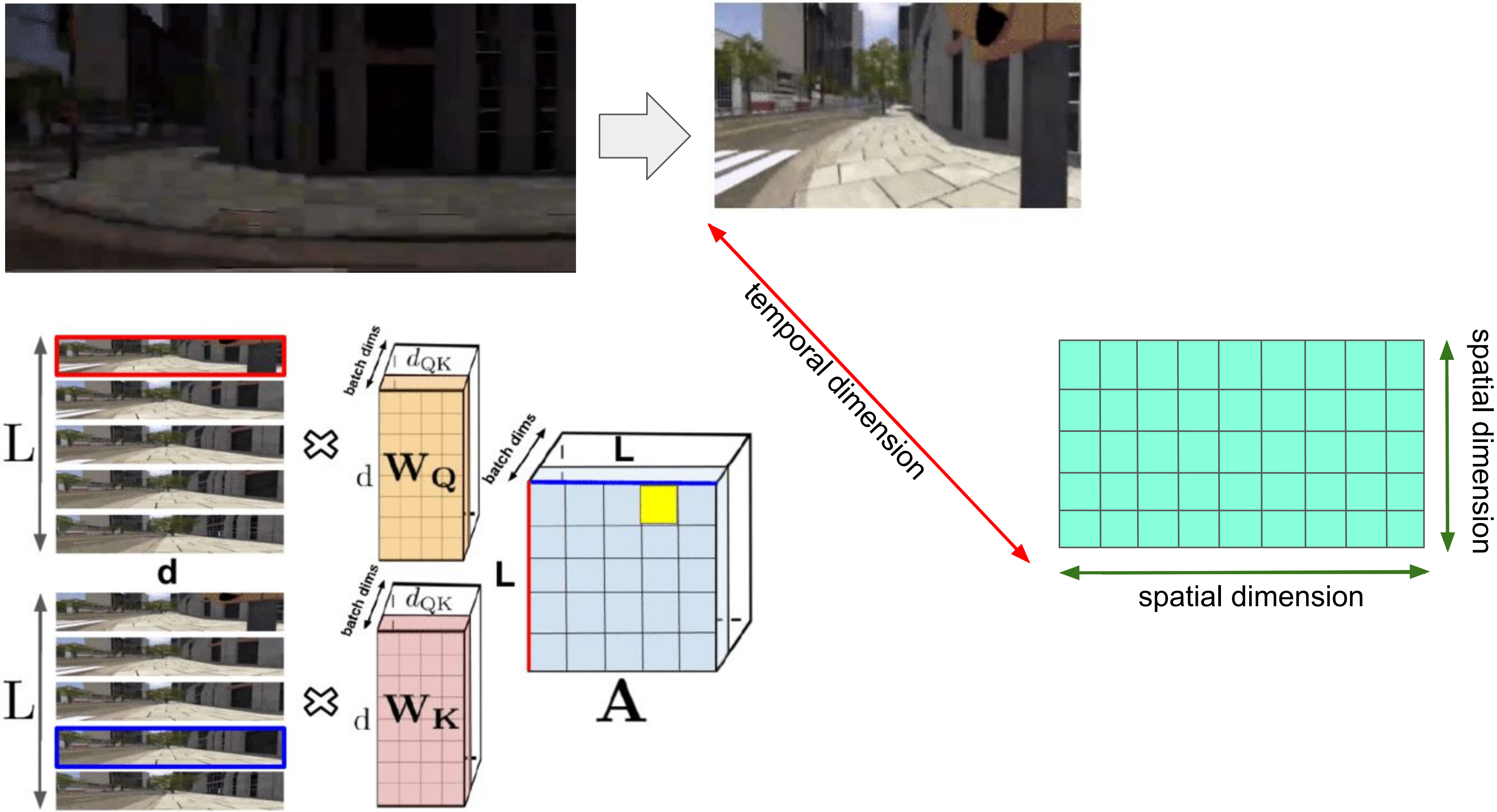
- Memory is a key to AI and currently existing sequential recurrent architectures fail to memorize well.

- Lets learn how to attend to the world - is Attention all you need ? Different attention dimensions: **spatial** & **temporal**.

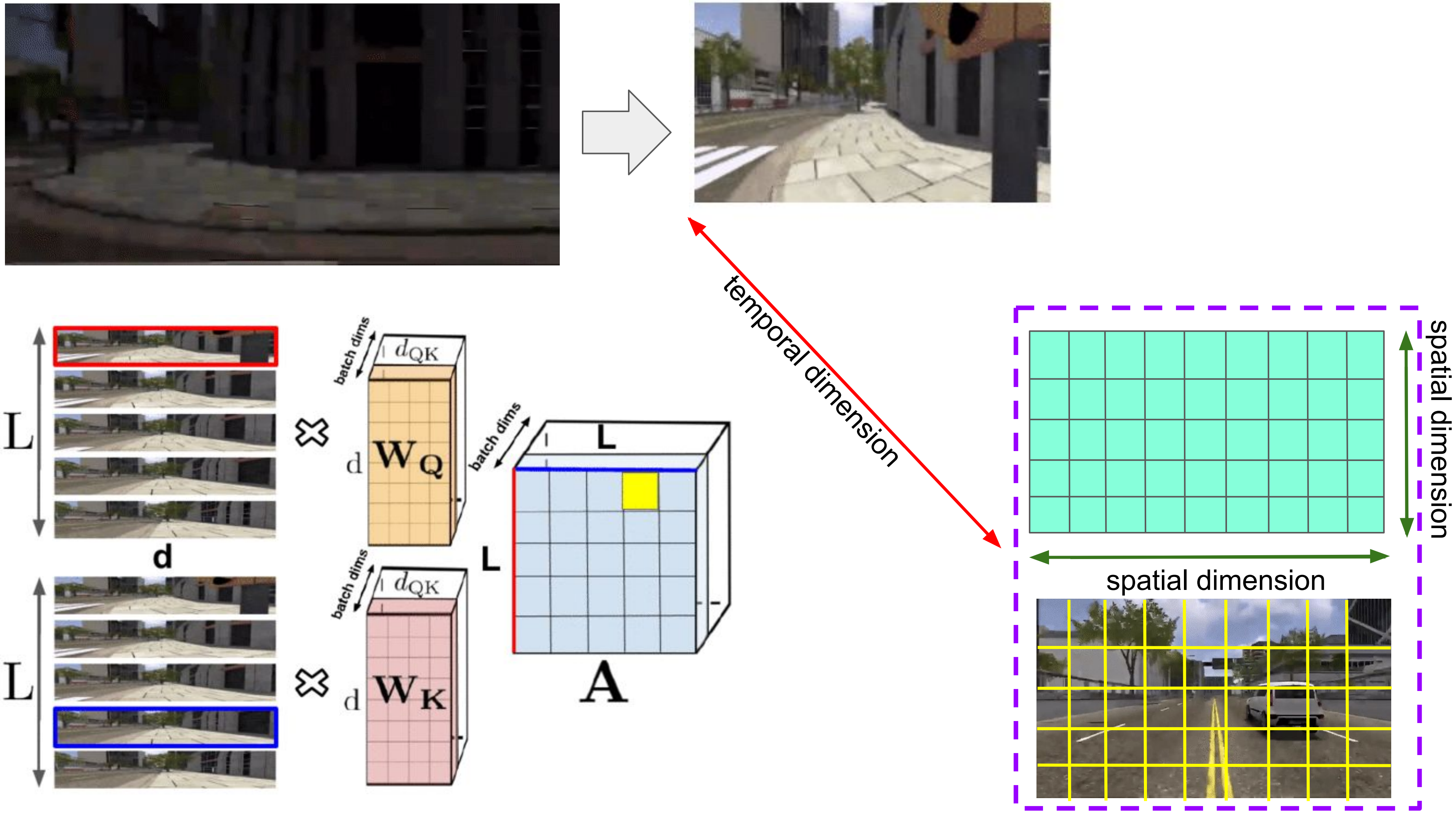
- Standard attention mechanisms are effectively parallelizable and avoid catastrophic forgetting, but are not scalable.

  "It uses more memory and more computation per real interaction…" [DeepMind nav by sight]

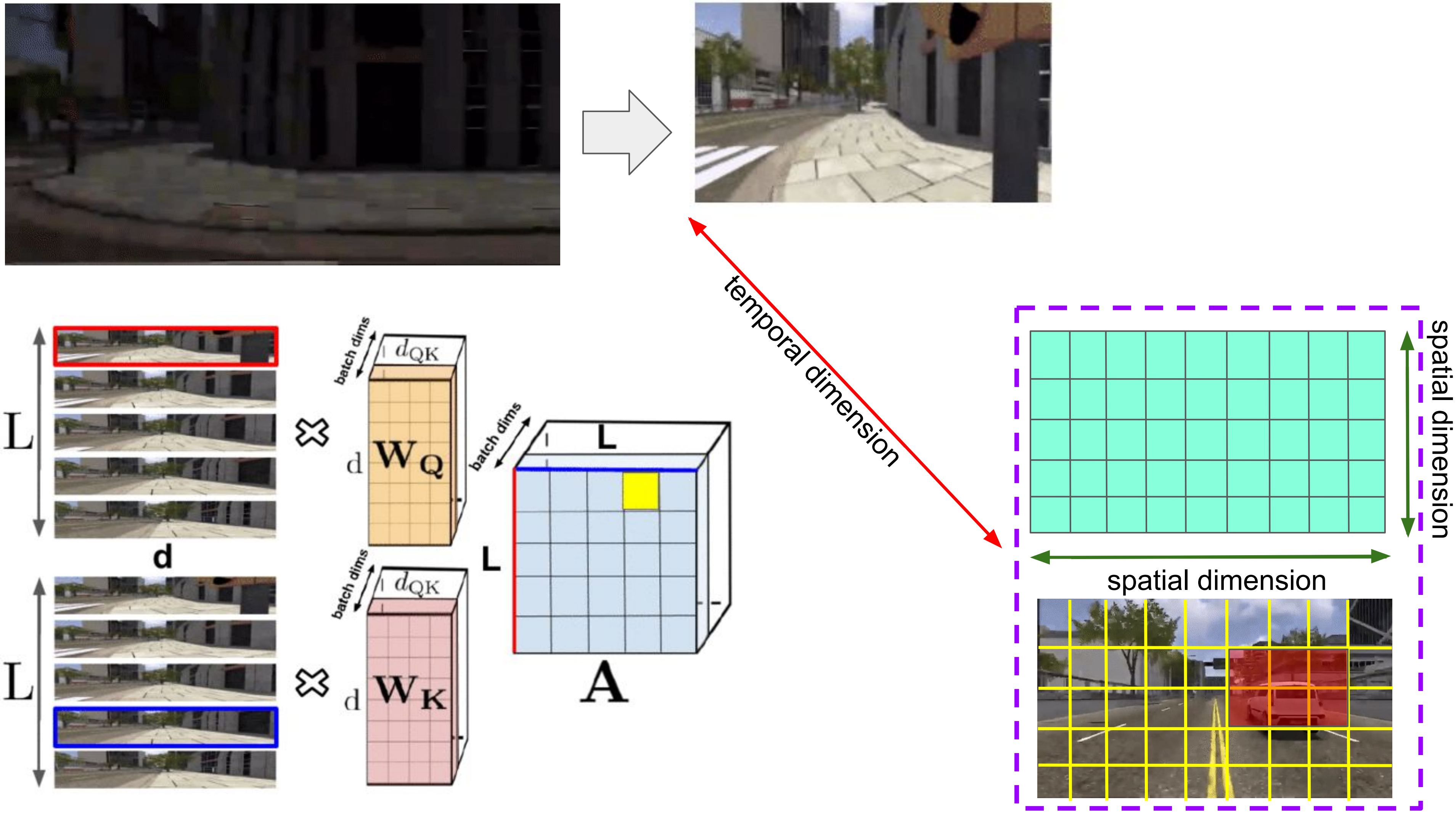
- Lifelong Learning Robotics requires **long range contexts** with **no attention priors**.



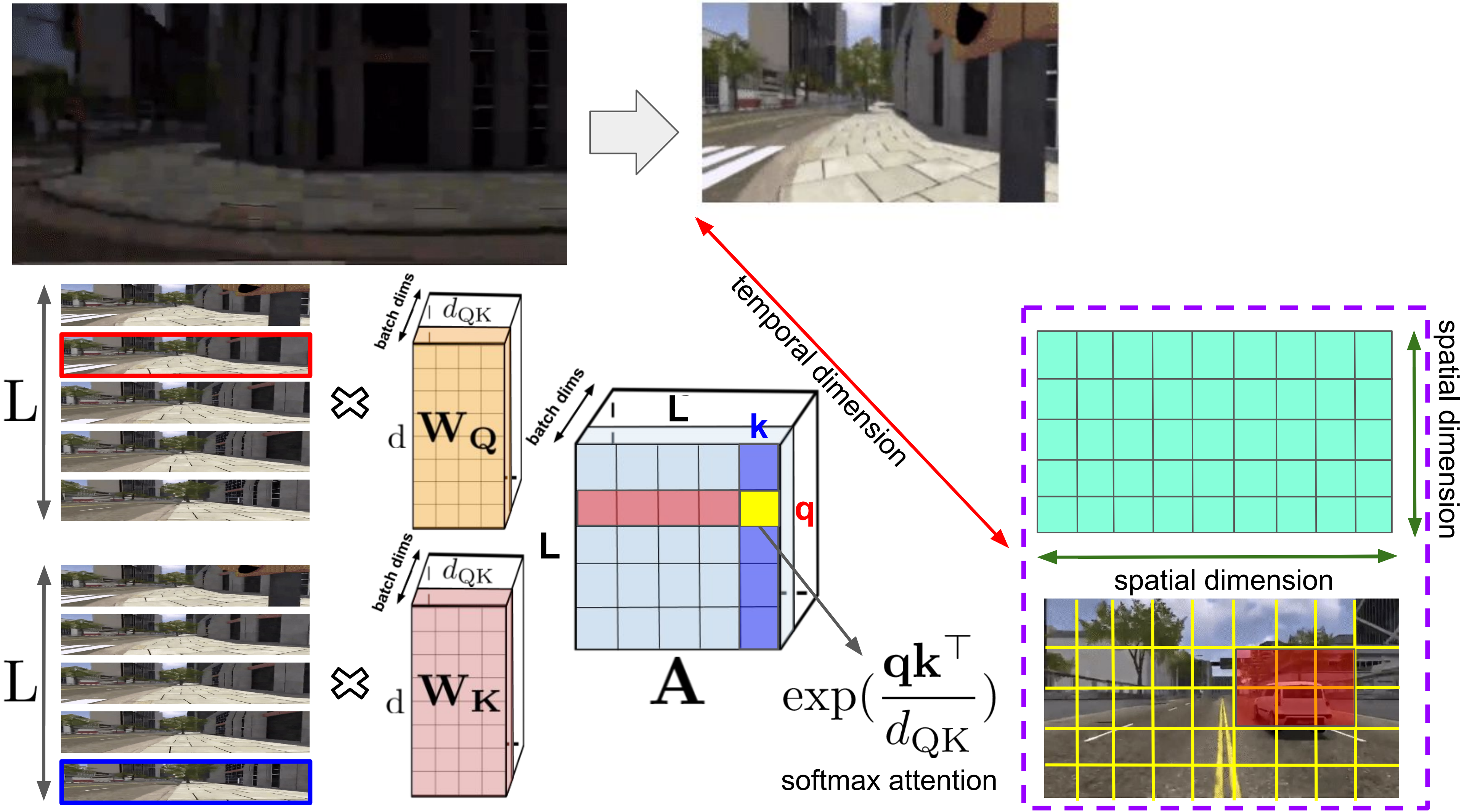
temporal dimension
spatial dimension
spatial dimension
L
d
L
batch dims
d_{QK}
W_Q
d
W_K
L
L
A



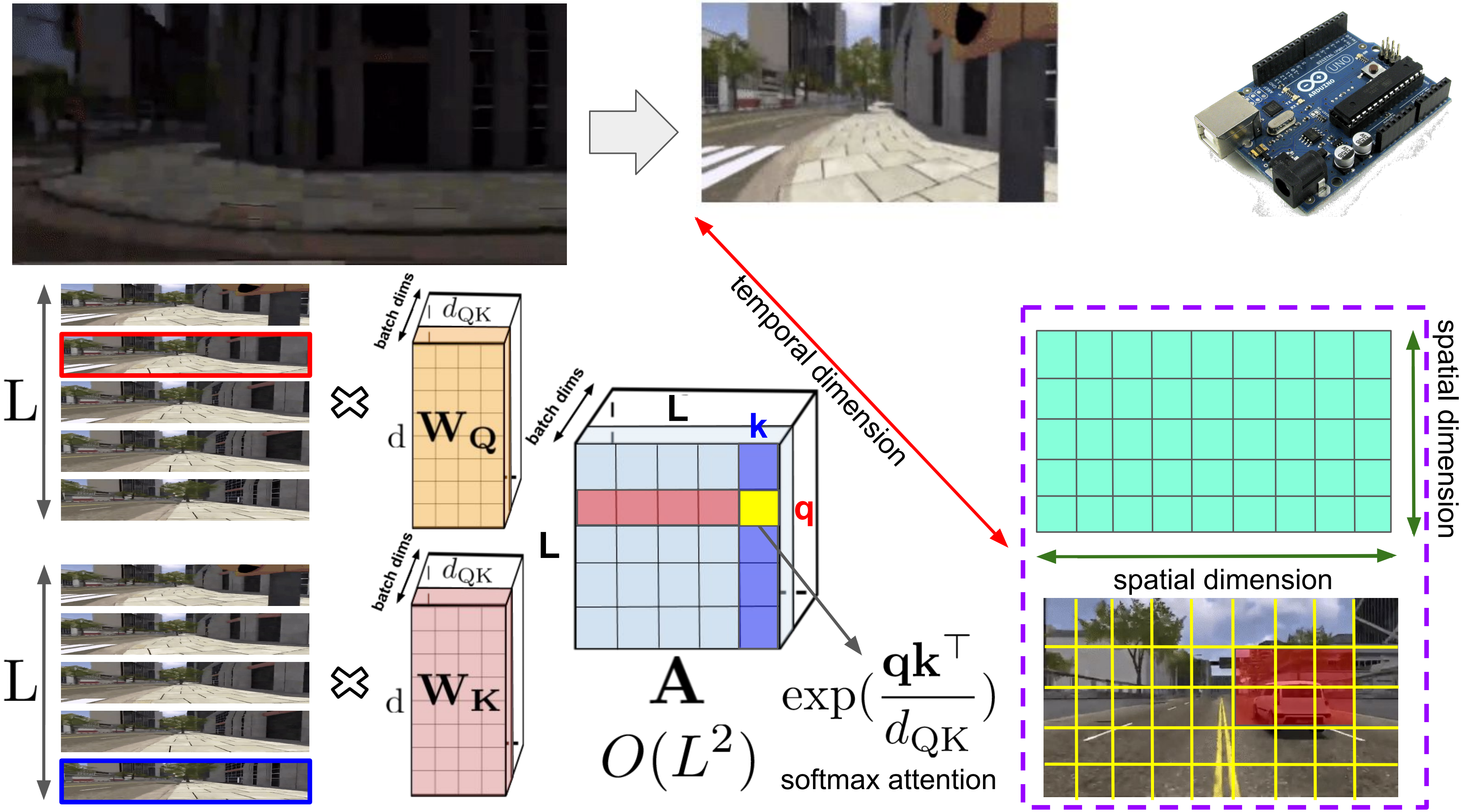
temporal dimension
spatial dimension
spatial dimension
L
d
L
batch dims
d_{QK}
d
W_Q
W_K
L
L
A

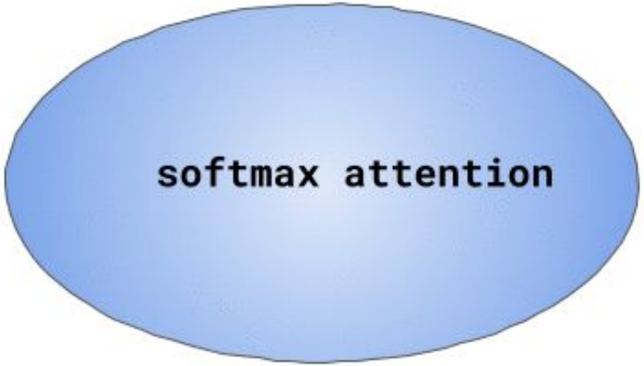


L
d
L
d
batch dims
d_{QK}
W_Q
d
batch dims
d_{QK}
W_K
batch dims
L
L
A
temporal dimension
spatial dimension
spatial dimension



L
L
batch dims
d_{QK}
d
$\mathbf{W_Q}$
$\mathbf{W_K}$
L
k
q
$\mathbf{A}$
temporal dimension
spatial dimension
spatial dimension
$\exp\left(\frac{\mathbf{q}\mathbf{k}^\top}{d_{QK}}\right)$
softmax attention


L
L
batch dims
d_QK
d
W_Q
batch dims
d_QK
d
W_K
batch dims
L
L
k
q
A
O(L^2)
exp(qk^⊤ / d_QK)
softmax attention
temporal dimension
spatial dimension
spatial dimension



Performers = **Linear** Attention Modules Capable of Modeling Different Attention Kernels (Also **SOFTMAX** !)

Performers performing on the Long Range Arena

Long Range Arena Paper

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	FAIL	<u>54.39</u>
Local Attention	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Sparse Trans.	17.07	63.58	**59.59**	**44.24**	71.71	FAIL	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
Linformer	35.70	53.94	52.27	38.56	<u>76.34</u>	FAIL	51.36
Reformer	**37.27**	56.10	53.40	38.07	68.50	FAIL	50.67
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	FAIL	51.39
Synthesizer	<u>36.99</u>	61.68	54.67	41.61	69.45	FAIL	52.88
BigBird	36.05	64.02	<u>59.29</u>	40.83	74.87	FAIL	**55.01**
Linear Trans.	16.13	**65.90**	53.09	42.34	75.30	FAIL	50.55
Performer	18.01	<u>65.40</u>	53.82	<u>42.77</u>	**77.05**	FAIL	51.41
Task Avg (Std)	29 (9.7)	61 (4.6)	55 (2.6)	41 (1.8)	72 (3.7)	FAIL	52 (2.4)

Table 1: Experimental results on Long-Range Arena benchmark. Best model is in boldface and second best is underlined. All models do not learn anything on Path-X task, contrary to the Pathfinder task and this is denoted by FAIL. This shows that increasing the sequence length can cause seriously difficulties for model training. We leave Path-X on this benchmark for future challengers but do not include it on the Average score as it has no impact on relative performance.

Model	Steps per second 1K	2K	3K	4K	Peak Memory Usage (GB) 1K	2K	3K	4K
Transformer	8.1	4.9	2.3	1.4	0.85	2.65	5.51	9.48
Local Attention	9.2 (1.1x)	8.4 (1.7x)	7.4 (3.2x)	7.4 (5.3x)	0.42	0.76	1.06	1.37
Linformer	9.3 (1.2x)	9.1 (1.9x)	8.5 (3.7x)	7.7 (5.5x)	**0.37**	**0.55**	0.99	**0.99**
Reformer	4.4 (0.5x)	2.2 (0.4x)	1.5 (0.7x)	1.1 (0.8x)	0.48	0.99	1.53	2.28
Sinkhorn Trans	9.1 (1.1x)	7.9 (1.6x)	6.6 (2.9x)	5.3 (3.8x)	0.47	0.83	1.13	1.48
Synthesizer	8.7 (1.1x)	5.7 (1.2x)	6.6 (2.9x)	1.9 (1.4x)	0.65	1.98	4.09	6.99
BigBird	7.4 (0.9x)	3.9 (0.8x)	2.7 (1.2x)	1.5 (1.1x)	0.77	1.49	2.18	2.88
Linear Trans.	9.1 (1.1x)	9.3 (1.9x)	8.6 (3.7x)	7.8 (5.6x)	**0.37**	0.57	**0.80**	1.03
Performer	**9.5** (1.2x)	**9.4** (1.9x)	**8.7** (3.8x)	**8.0** (5.7x)	**0.37**	0.59	0.82	1.06

Table 2: Benchmark results of all Xformer models with a consistent batch size of 32 across all models. We report relative speed increase/decrease in comparison with the vanilla Transformer in brackets besides the steps per second. Memory usage refers to per device memory usage across each TPU device. Benchmarks are run on 4x4 TPU V3 Chips.

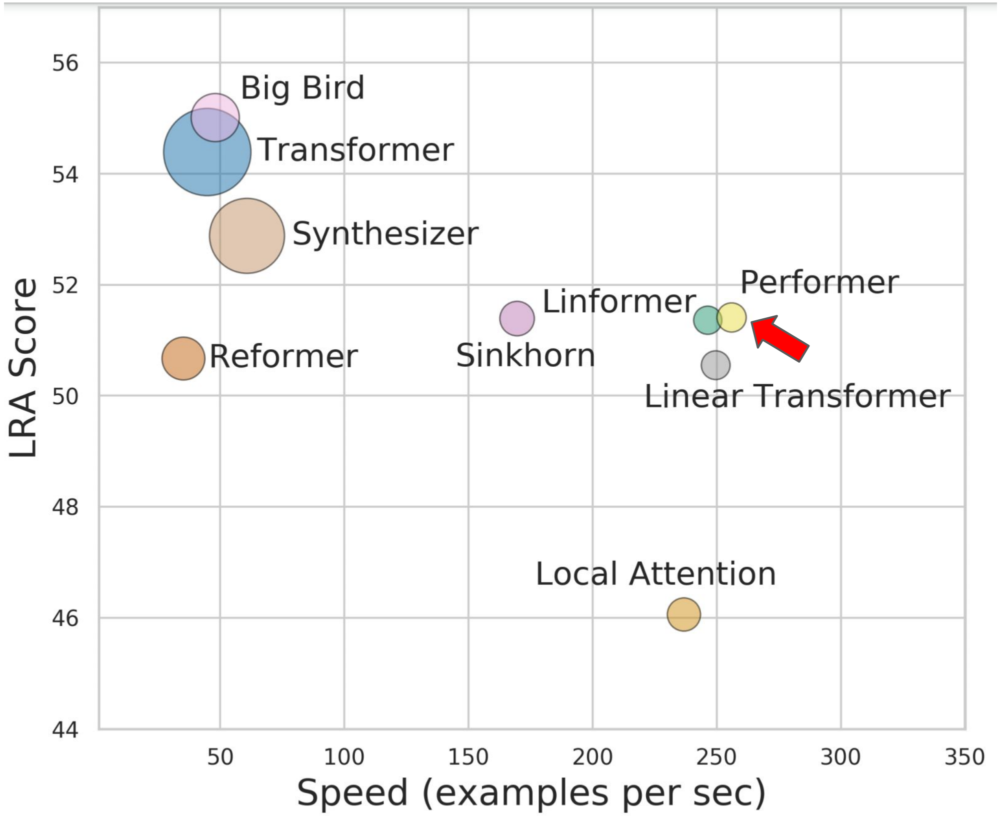


Figure 3: Performance (y axis), speed (x axis), and memory footprint (size of the circles) of different models.

Performers in Vision

Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet

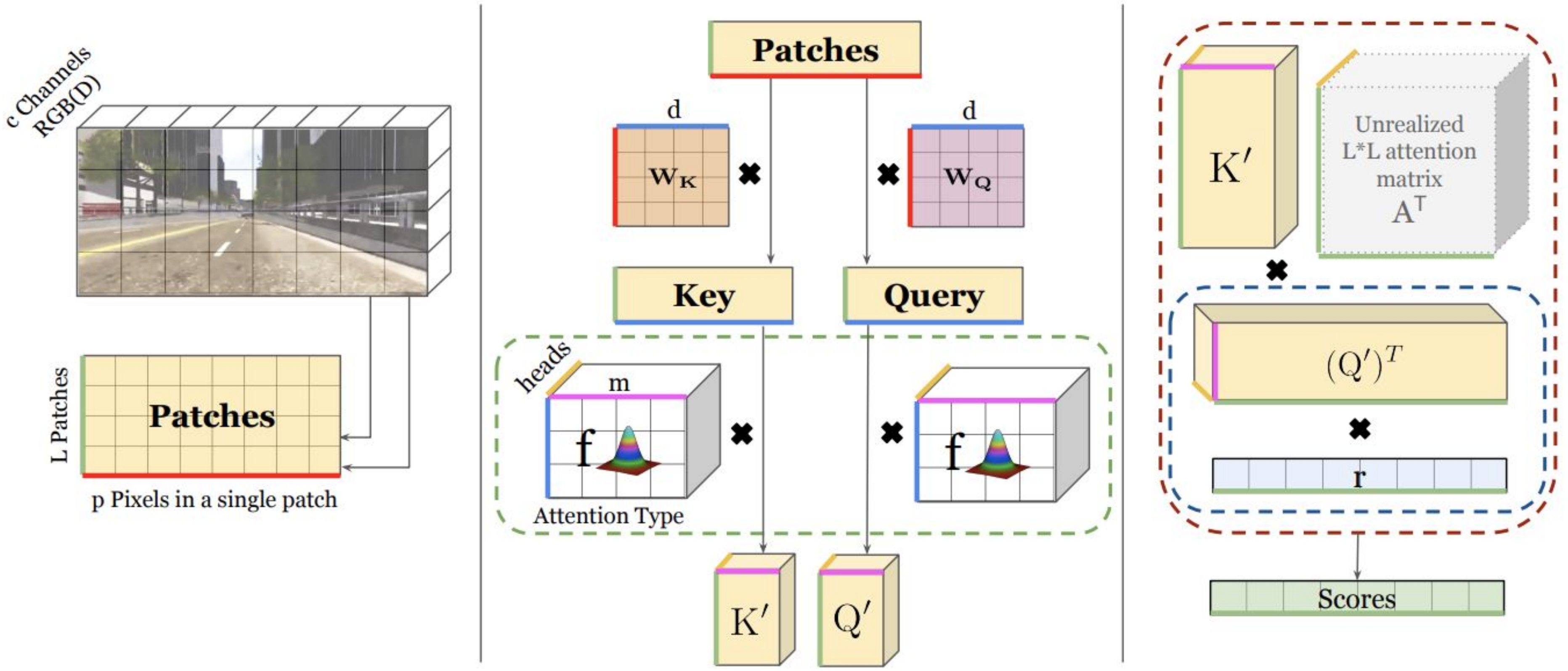
Yuan et al.

Models	Top1-Acc (%)	Params (M)	MACs (G)
ResNet50 [15]	76.2	25.5	4.3
ResNet50*	79.1	25.5	4.3
T2T-ViT-14	**80.6**	21.4	4.8
T2T-ViT$_t$-14	**80.7**	21.5	5.2
ResNet101 [15]	77.4	44.6	7.9
ResNet101*	79.9	44.6	7.9
T2T-ViT-19	**81.2**	39.0	8.0
T2T-ViT$_t$-19	**81.4**	39.0	8.4
ResNet152 [15]	78.3	60.2	11.6
ResNet152*	80.8	60.2	11.6
T2T-ViT-24	**81.8**	63.9	12.6
T2T-ViT$_t$-24	**82.2**	64.1	13.2

Models	Tokens-to-Token module				T2T-ViT backbone			Model size	
	T2T Transformer	Depth	Hidden dim	MLP size	Depth	Hidden dim	MLP size	Params (M)	MACs (G)
ViT-S/16 [12]	-	-	-	-	8	786	2358	48.6	10.1
ViT-B/16 [12]	-	-	-	-	12	786	3072	86.8	17.6
ViT-L/16 [12]	-	-	-	-	24	1024	4096	304.3	63.6
T2T-ViT$_t$-14	Transformer	2	64	64	14	384	1152	21.5	5.2
T2T-ViT$_t$-19	Transformer	2	64	64	19	448	1344	39.0	8.4
T2T-ViT$_t$-24	Transformer	2	64	64	24	512	1536	64.1	13.2
T2T-ViT-14	Performer	2	64	64	14	384	1152	21.4	4.8
T2T-ViT-19	Performer	2	64	64	19	448	1344	39.0	8.0
T2T-ViT-24	Performer	2	64	64	24	512	1536	63.9	12.6
T2T-ViT-7	Performer	2	64	64	8	256	512	4.2	0.9
T2T-ViT-10	Performer	2	64	64	10	256	512	5.6	1.2
T2T-ViT-12	Performer	2	64	64	12	256	512	6.8	1.4

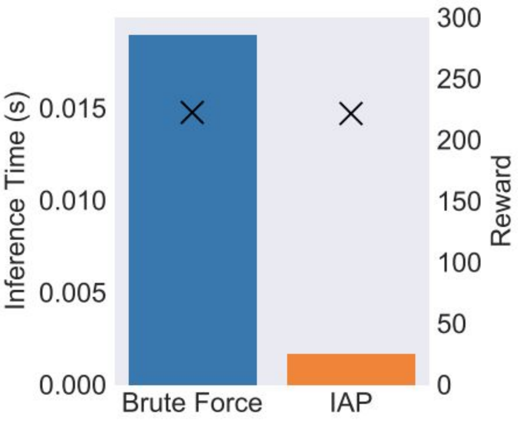
Performers for Vision in Reinforcement Learning (IAP)

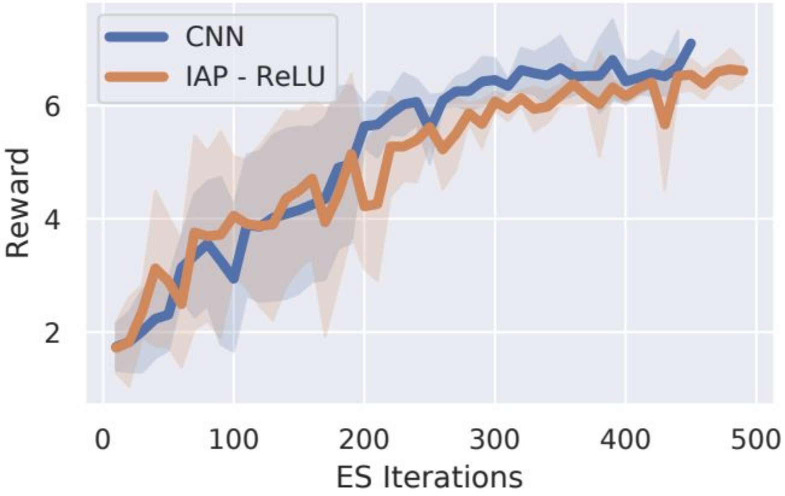
Unlocking Pixels for Reinforcement Learning via Implicit Attention Choromanski et al.

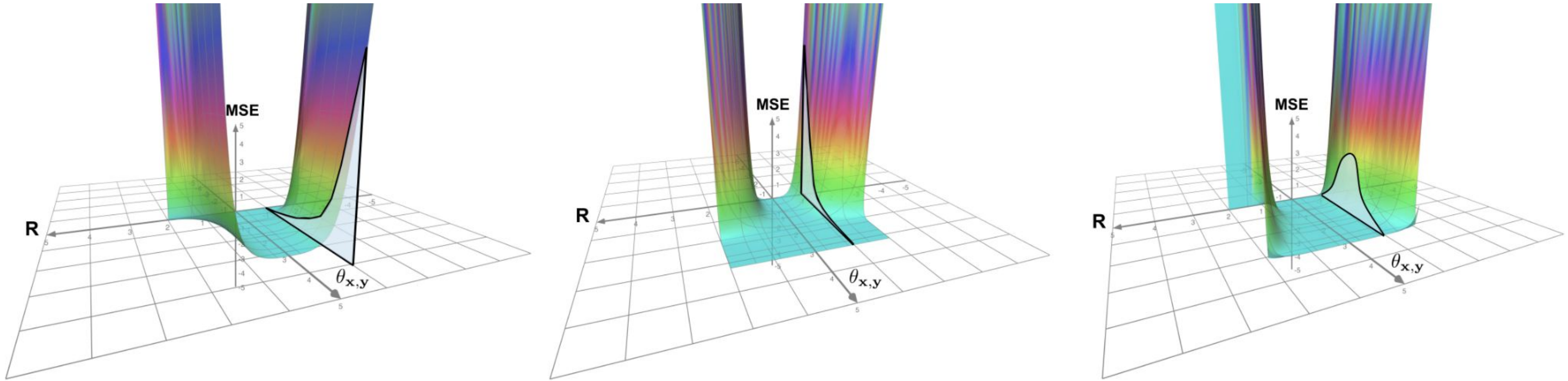


Performers for Vision in Reinforcement Learning (**IAP**)

Unlocking Pixels for Reinforcement Learning via Implicit Attention Choromanski et al.

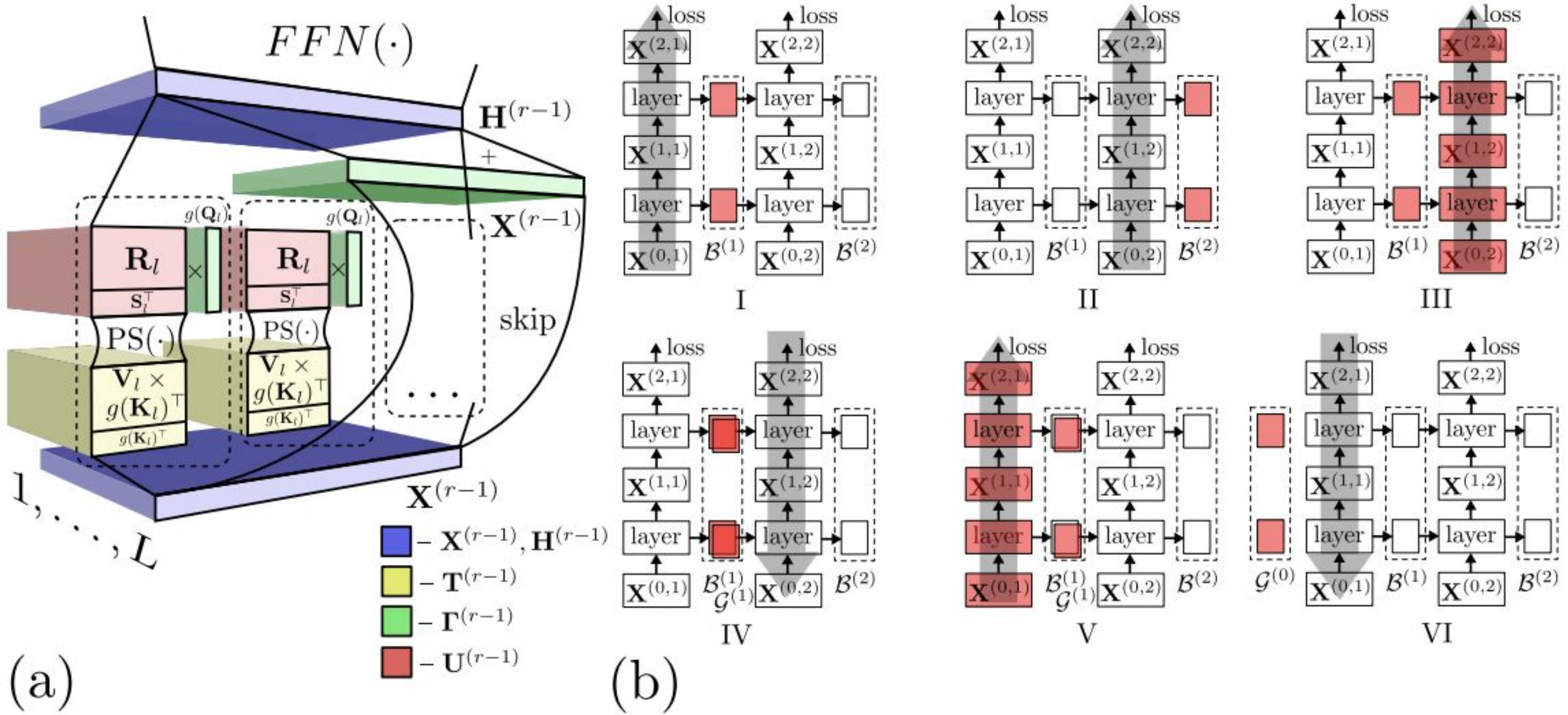




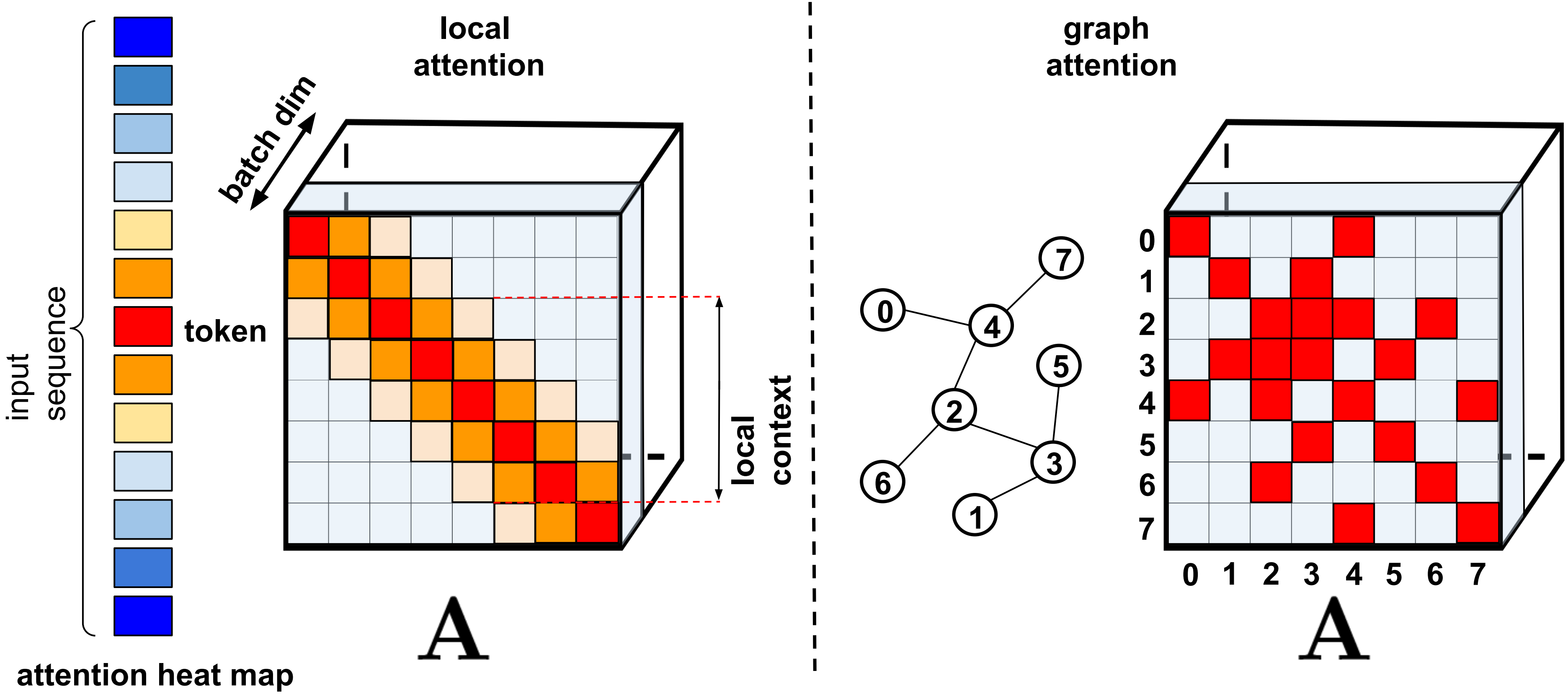


Sub-Linear Memory: How To Make Performers Slim

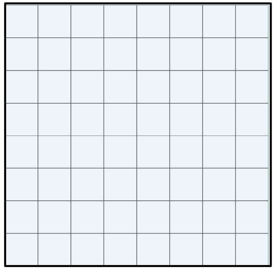
Likhosherstov et al.



What we Do NOT DO: Sparsifying Attention

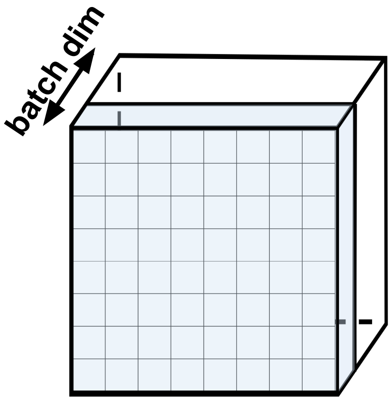


Attention is Kernelizable - A Tale of Random Maps



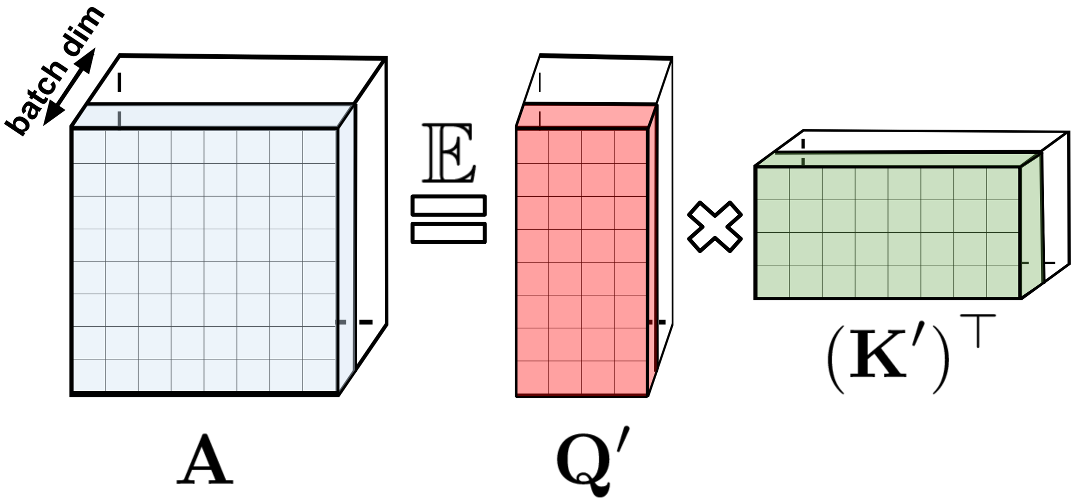
$\mathbf{A}$

Attention is Kernelizable - A Tale of Random Maps

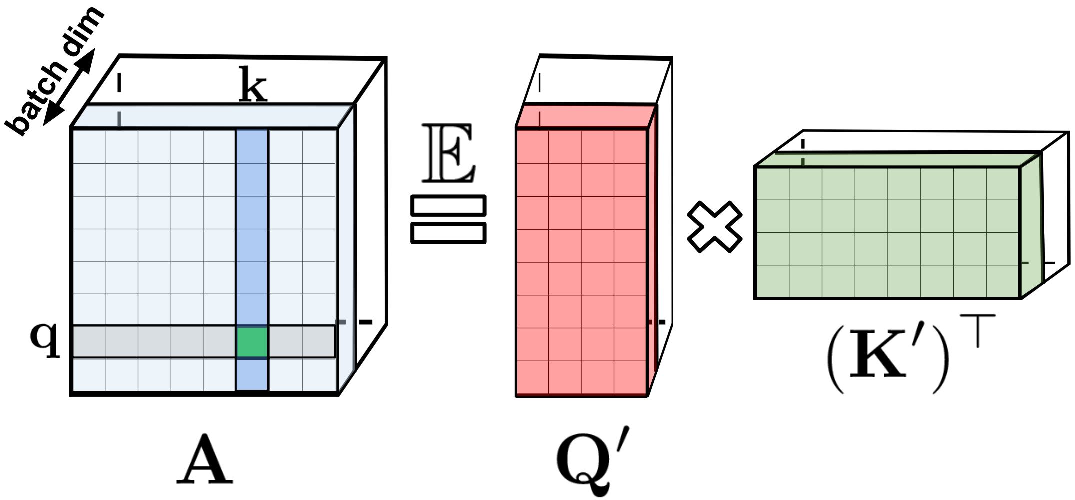


$\mathbf{A}$

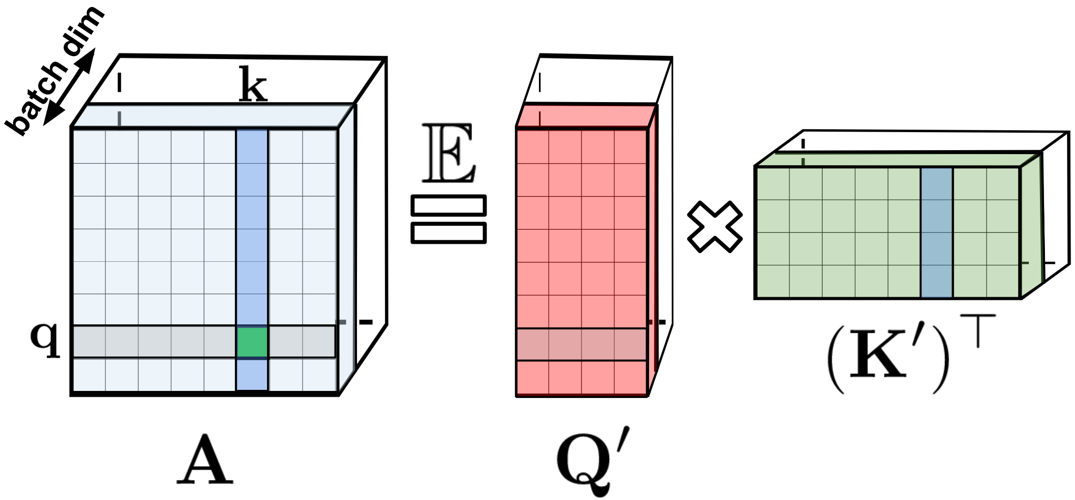
Attention is Kernelizable - A Tale of Random Maps



Attention is Kernelizable - A Tale of Random Maps

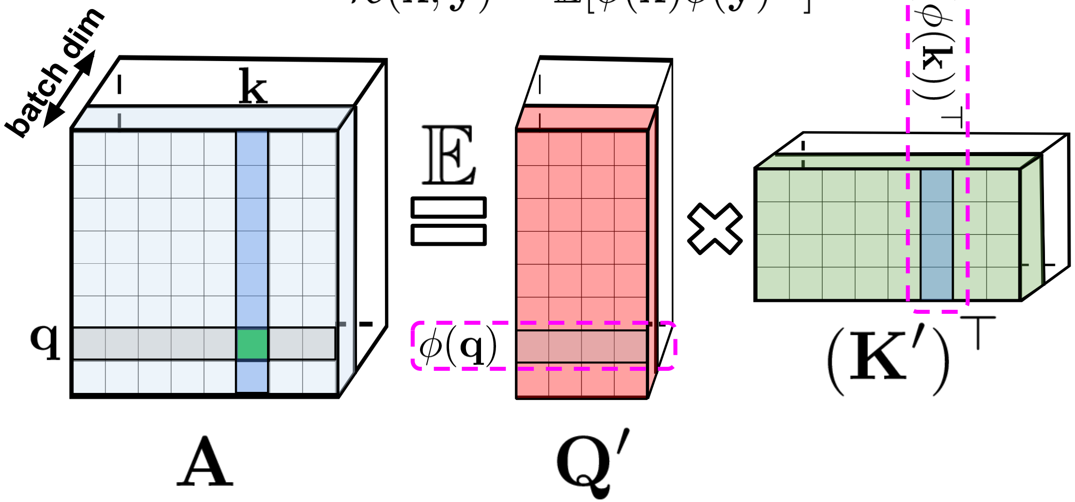


Attention is Kernelizable - A Tale of Random Maps



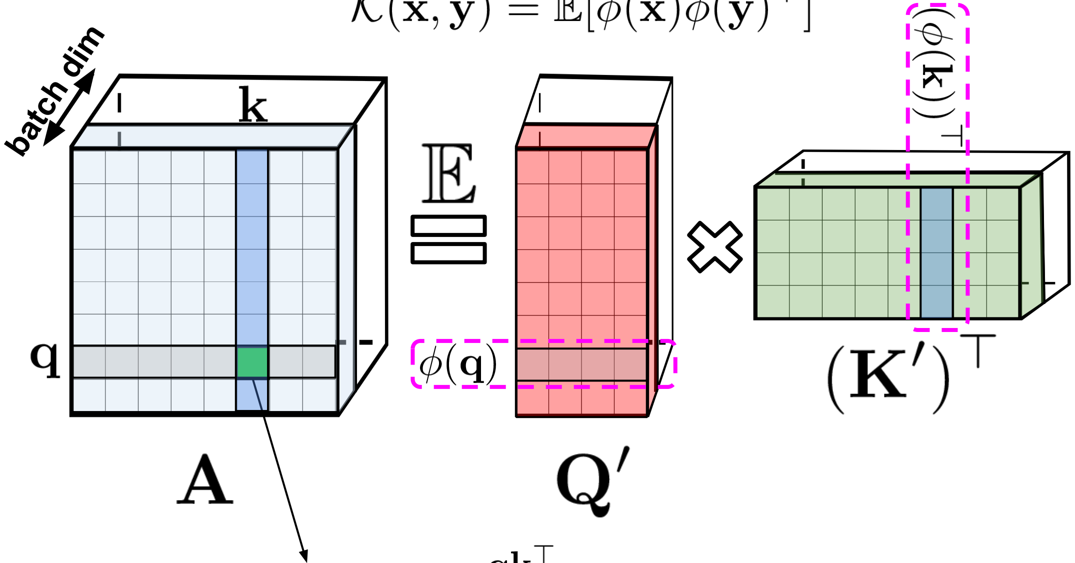
Attention is Kernelizable - A Tale of Random Maps

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})\phi(\mathbf{y})^\top]$$



Attention is Kernelizable - A Tale of Random Maps

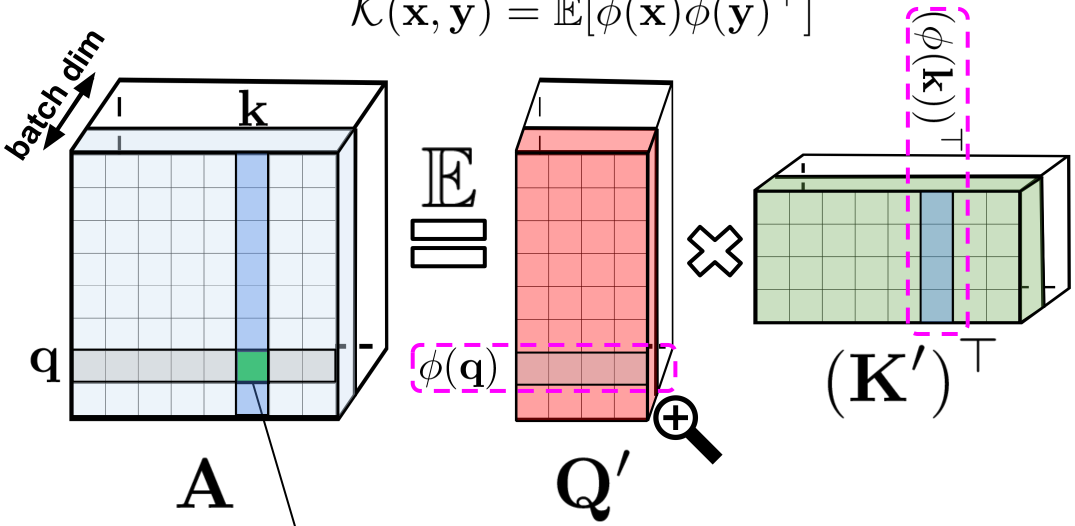
$$\mathcal{K}(\mathbf{x},\mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})\phi(\mathbf{y})^\top]$$



e.g. $\mathcal{K}(\mathbf{q},\mathbf{k}) = \exp(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d}})$

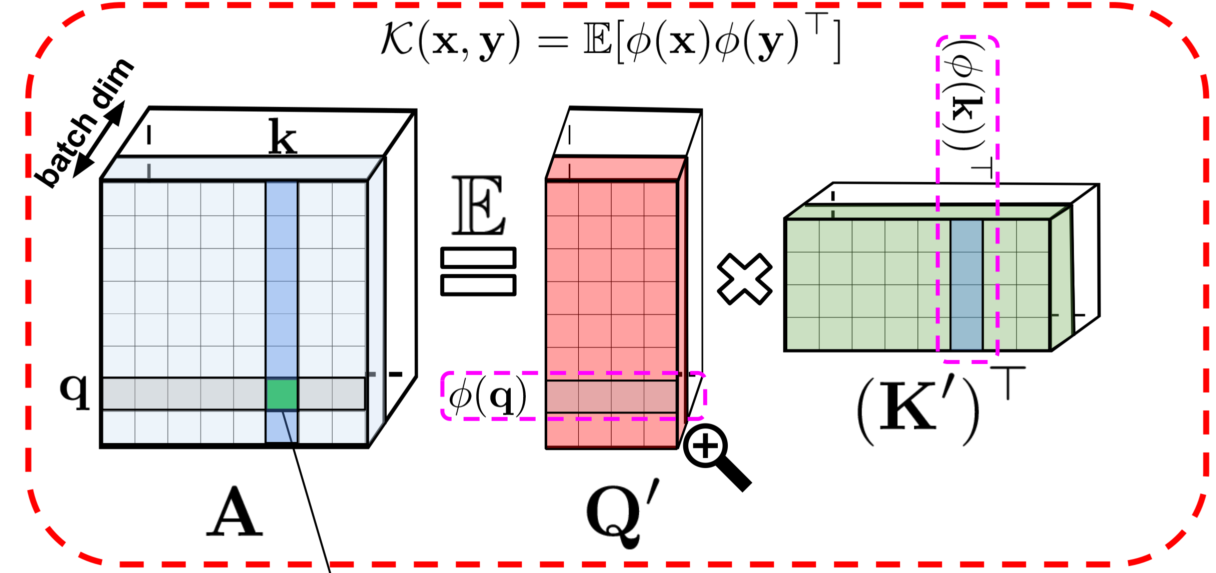
Attention is Kernelizable - A Tale of Random Maps

$$\mathcal{K}(\mathbf{x},\mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})\phi(\mathbf{y})^\top]$$



e.g. $\mathcal{K}(\mathbf{q},\mathbf{k}) = \exp(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d}})$

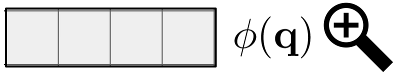
Attention is Kernelizable - A Tale of Random Maps



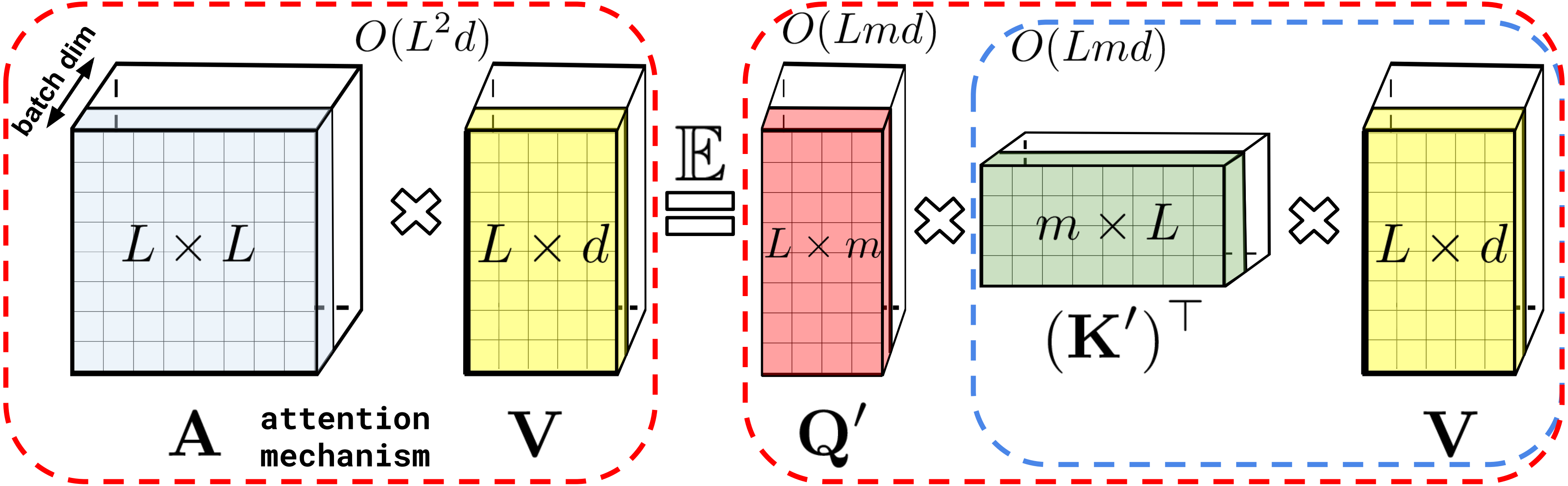
e.g. $\mathcal{K}(\mathbf{q},\mathbf{k}) = \exp(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d}})$

$$\begin{cases} \phi(\mathbf{x}) = \frac{1}{\sqrt{m}} \underbrace{\exp(\frac{\|\mathbf{x}\|_2^2}{2\sqrt{d}})}_{\text{deterministic feature}} \cdot (\underbrace{\sin(\frac{\mathbf{x}}{d^{\frac{1}{4}}}\omega_1^\top)}_{\text{random feature}}, \dots, \sin(\frac{\mathbf{x}}{d^{\frac{1}{4}}}\omega_m^\top), \cos(\frac{\mathbf{x}}{d^{\frac{1}{4}}}\omega_1^\top), \dots, \cos(\frac{\mathbf{x}}{d^{\frac{1}{4}}}\omega_m^\top)) \\ \omega \sim \mathcal{N}(0, \mathbf{I}_d) \end{cases}$$

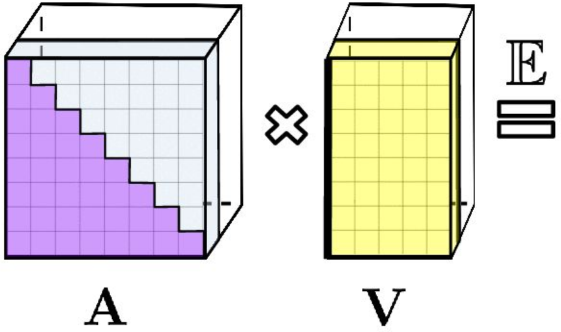
e.g. softmax kernel features

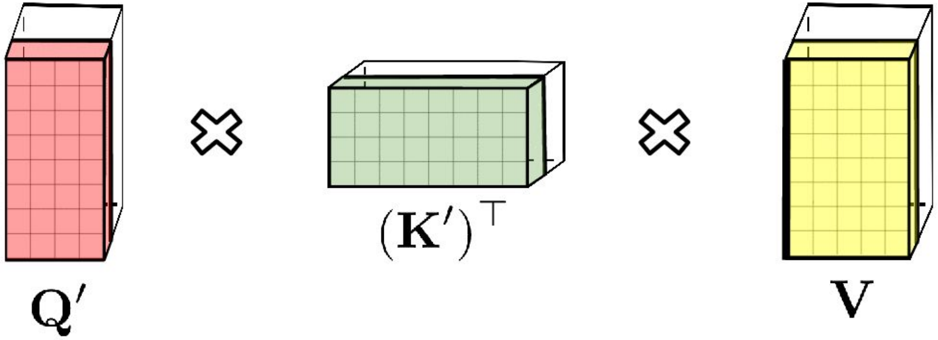


Associativity for Speedups and Space Compression

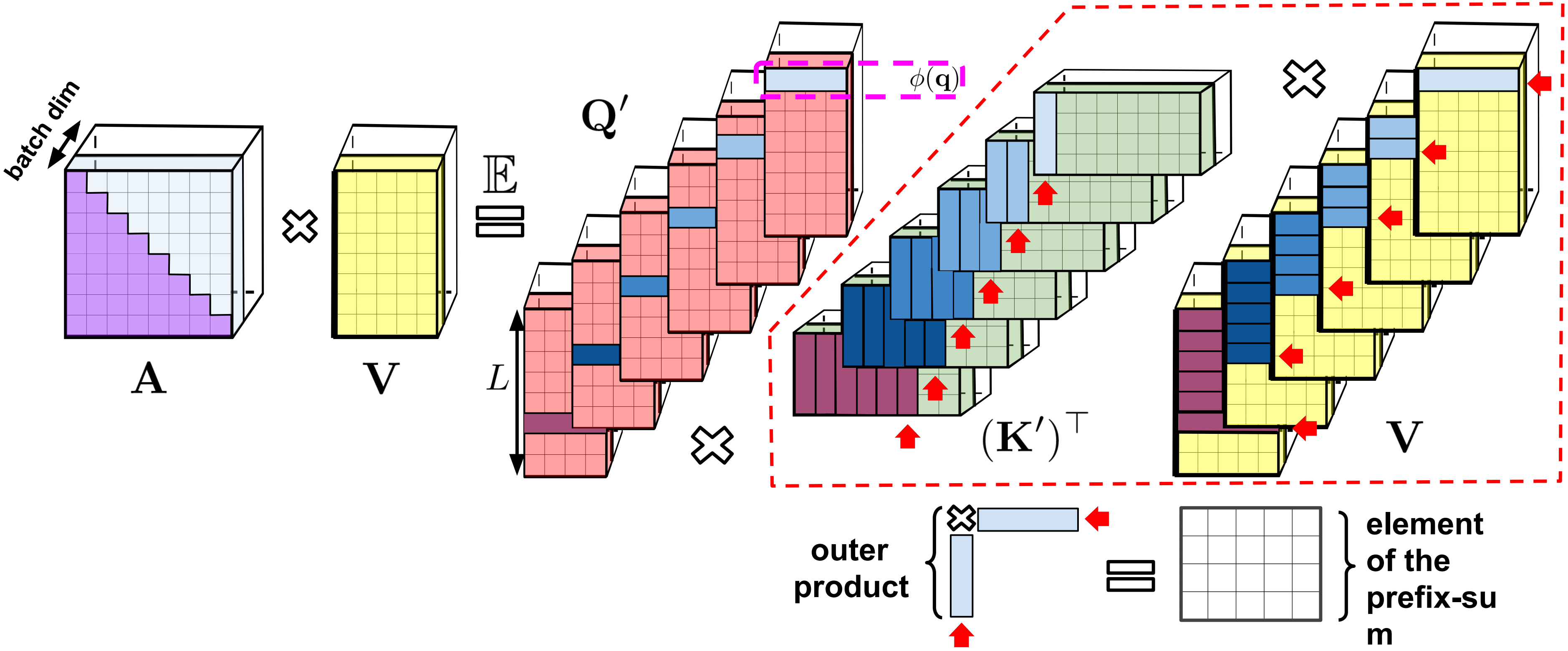


Causal Transformers as Prefix-Sums Calculators





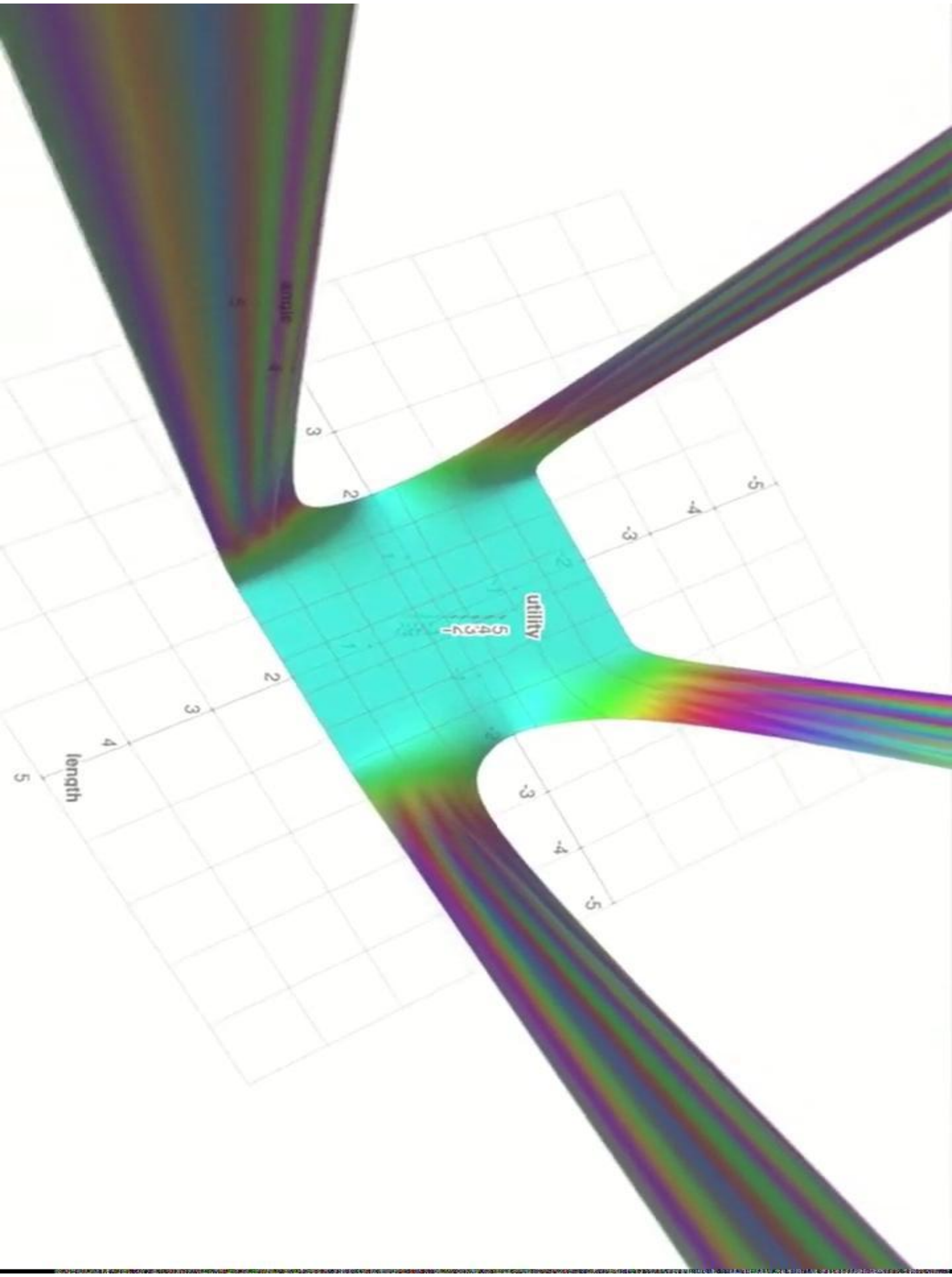
Causal Transformers as Prefix-Sums Calculators



Why this is not the end of the story ?

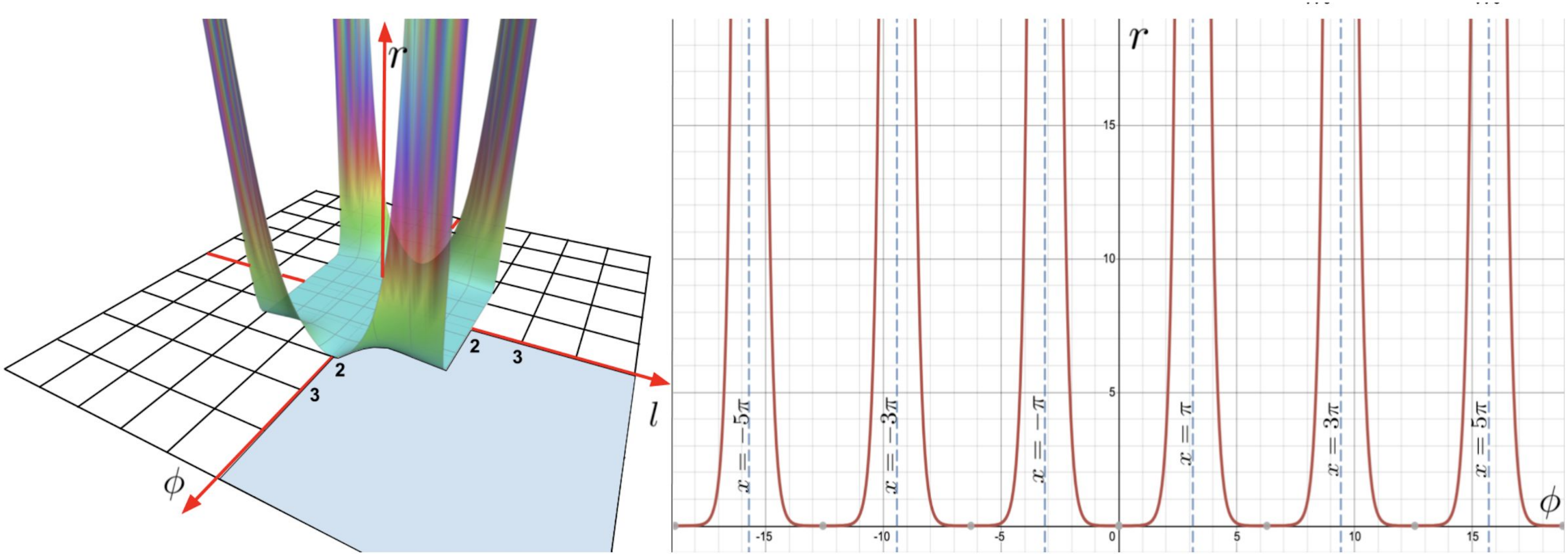
FAVOR+

$$\text{utility} = \frac{\mathrm{MSE}(\mathrm{SM}^{\mathrm{trig}}(\mathbf{x}, \mathbf{y}))}{\mathrm{MSE}(\mathrm{SM}^{+}(\mathbf{x}, \mathbf{y}))}$$



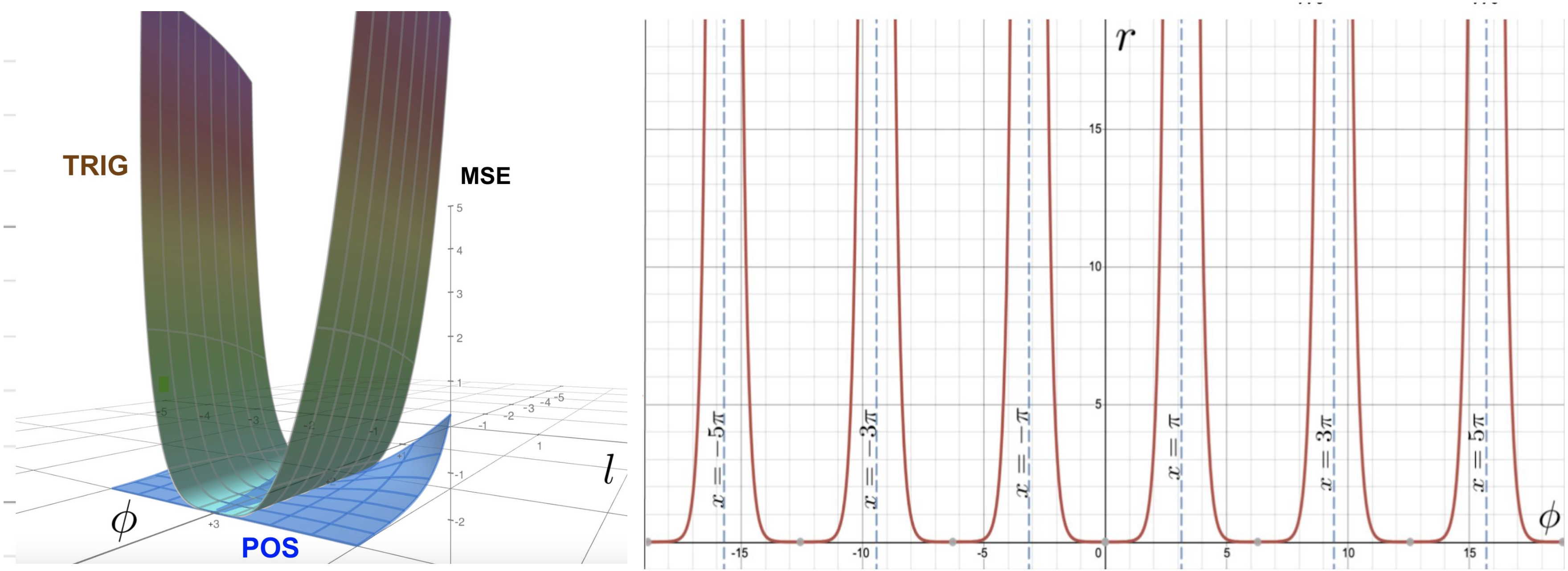
$$\text{length} = \|\mathbf{x}\| = \|\mathbf{y}\|$$

Do we need to go beyond trigonometric features ?

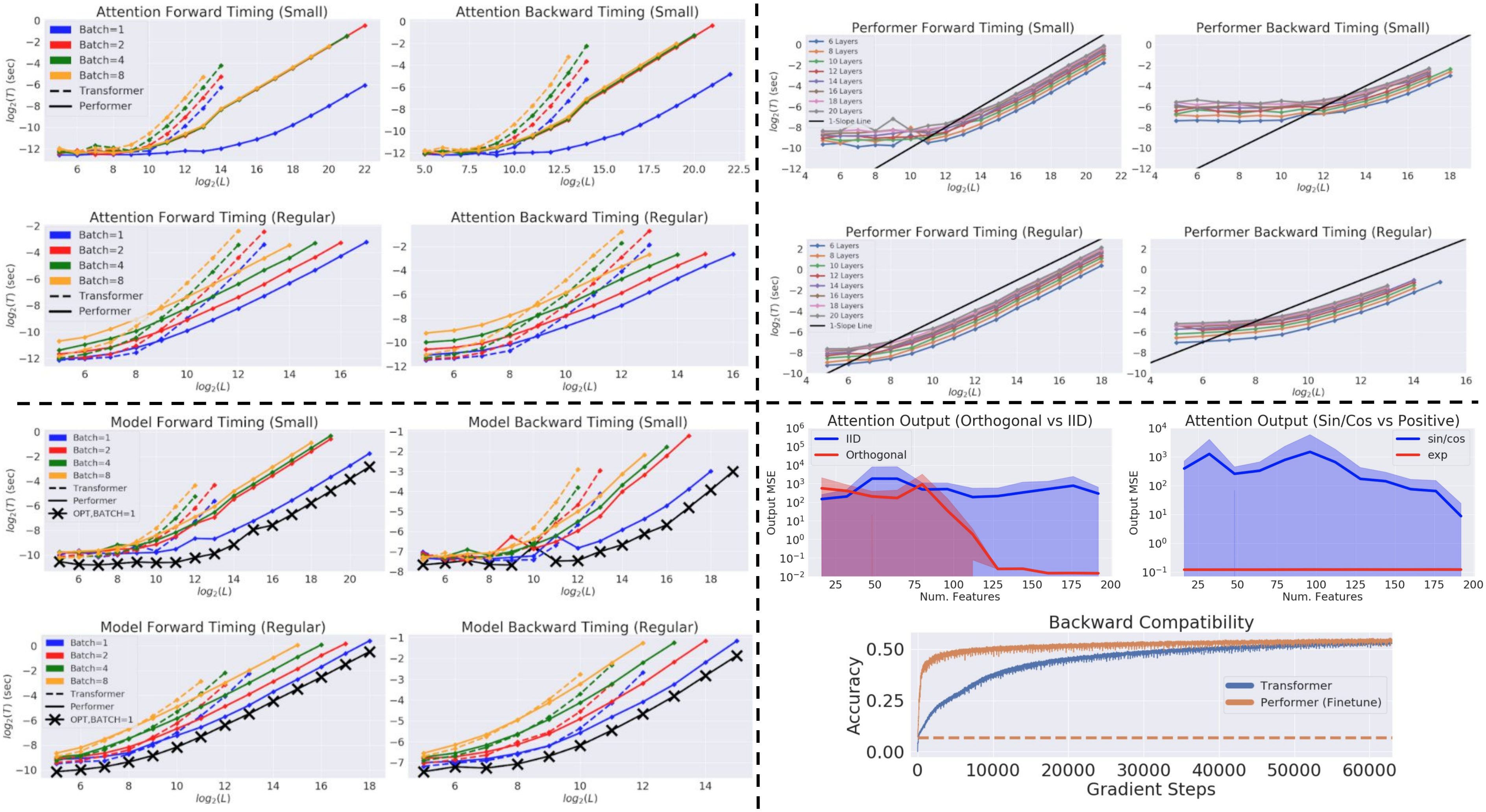


Left: Symmetrized (around origin) utility function r (defined as a ratio of the mean squared errors of estimators built on: trigonometric and positive random features) as a function of the angle φ (in radians) between input feature vectors and their lengths l. Larger values indicate regions of (φ, l)-space with better performance of positive random features. We see that for critical regions with φ large enough (small enough softmax-kernel values) our method is arbitrarily more accurate than trigonometric random features. Plot presented for domain [−π, π] × [−2, 2]. **Right:** The slice of function r for fixed l = 1 and varying angle φ.

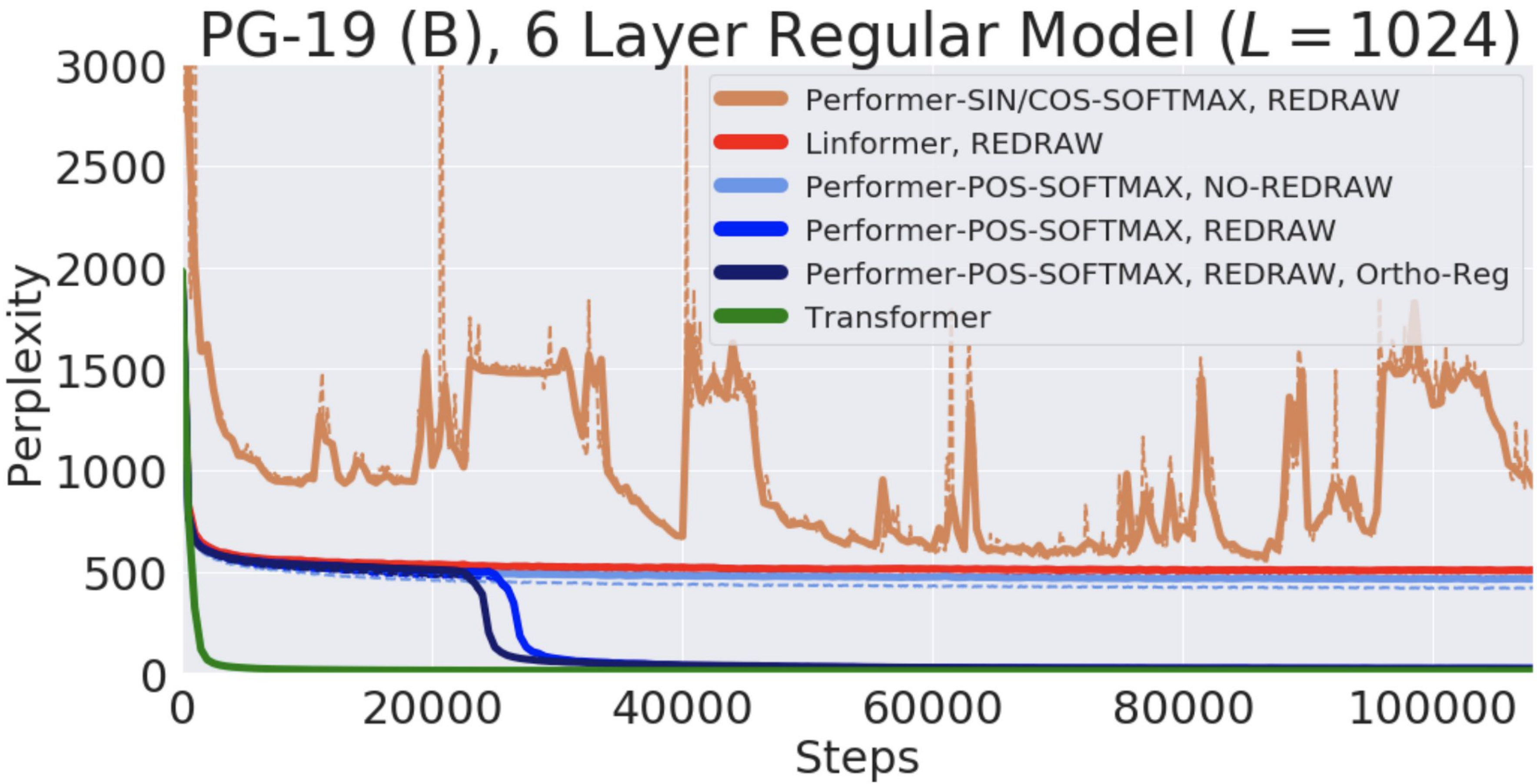
Do we need to go beyond trigonometric features ?



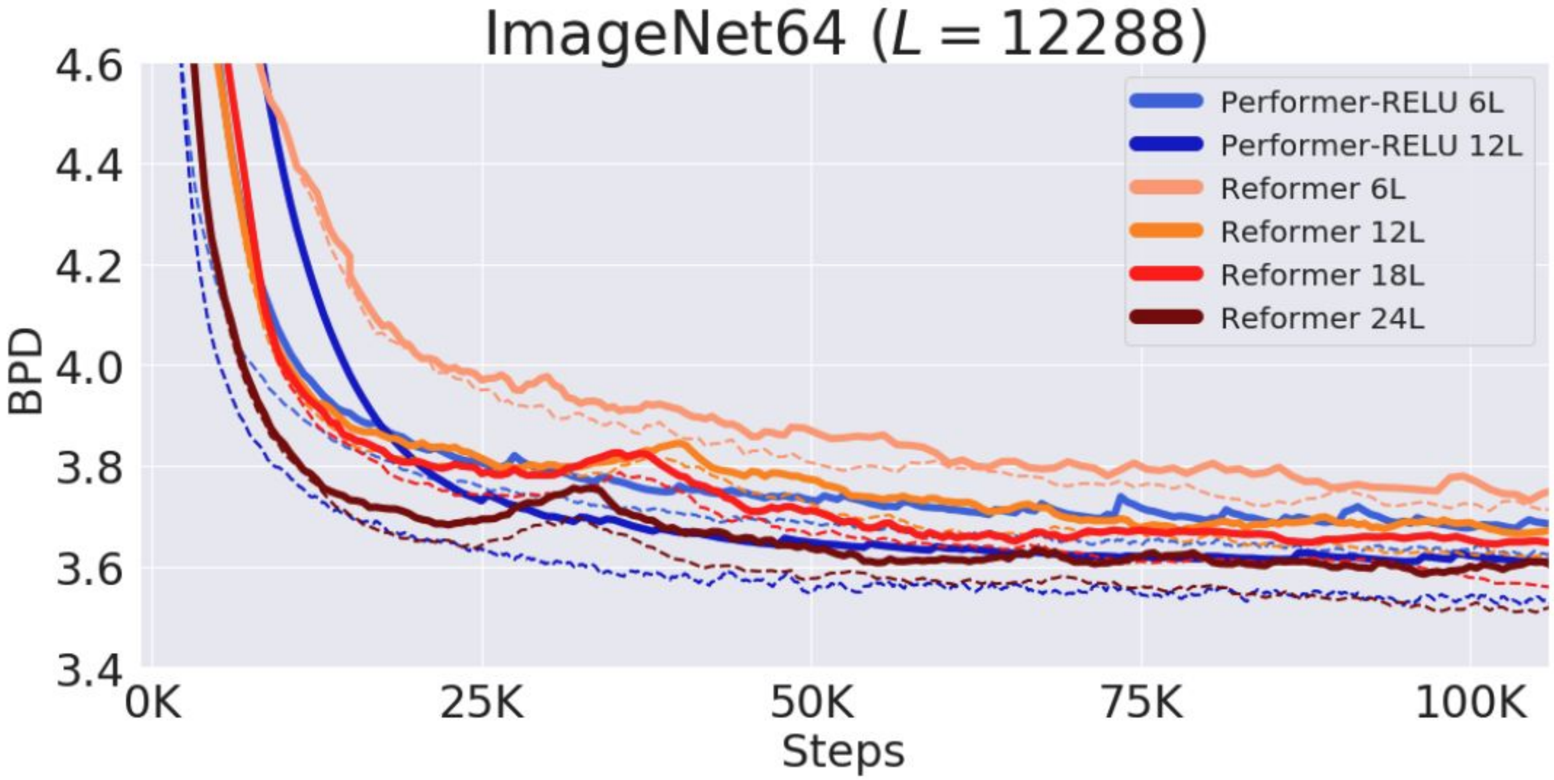
Left: Comparison of the mean squared errors (MSEs) of the estimators applying trigonometric random features (TRIG) and the one leveraging the mechanism of positive random features (POS) in the region of small softmax-kernel values.
Right: The slice of function r for fixed l = 1 and varying angle φ.


Attention Forward Timing (Small)
Attention Backward Timing (Small)
Attention Forward Timing (Regular)
Attention Backward Timing (Regular)
Batch=1
Batch=2
Batch=4
Batch=8
Transformer
Performer
$log_2(T)$ (sec)
$log_2(L)$
Performer Forward Timing (Small)
Performer Backward Timing (Small)
Performer Forward Timing (Regular)
Performer Backward Timing (Regular)
6 Layers
8 Layers
10 Layers
12 Layers
14 Layers
16 Layers
18 Layers
20 Layers
1-Slope Line
Model Forward Timing (Small)
Model Backward Timing (Small)
Model Forward Timing (Regular)
Model Backward Timing (Regular)
OPT,BATCH=1
Attention Output (Orthogonal vs IID)
IID
Orthogonal
Attention Output (Sin/Cos vs Positive)
sin/cos
exp
Output MSE
Num. Features
Backward Compatibility
Transformer
Performer (Finetune)
Accuracy
Gradient Steps

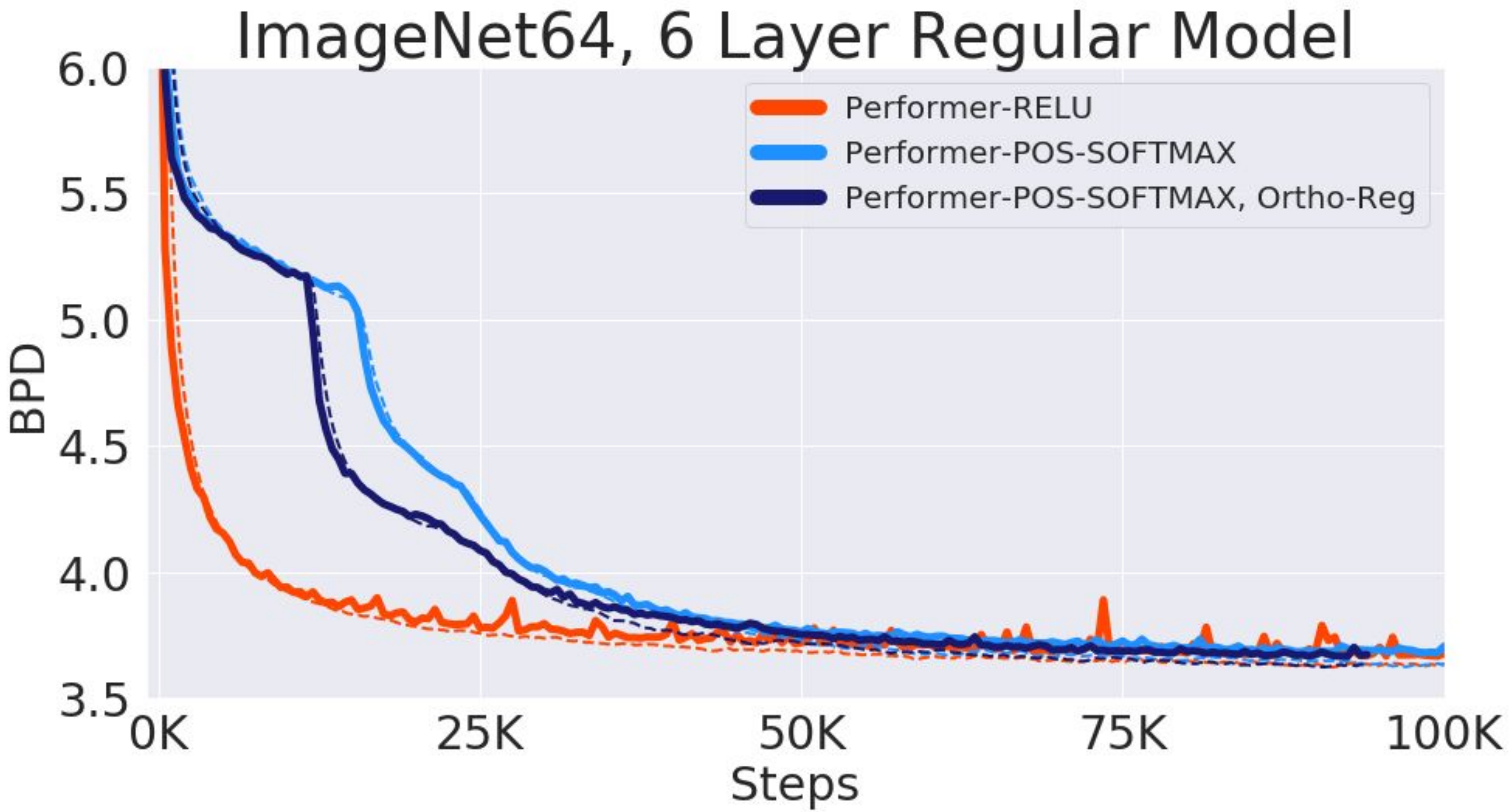
Positive vs trigonometric random features in practice



Performers on ImageNet64 - pixel predictions models

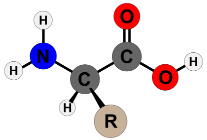
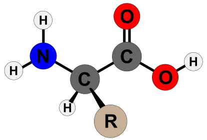
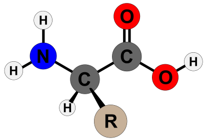
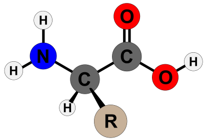
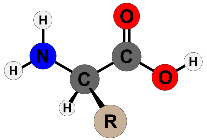
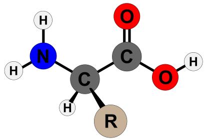
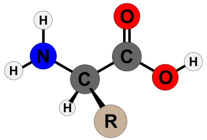
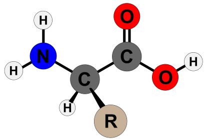
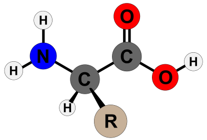


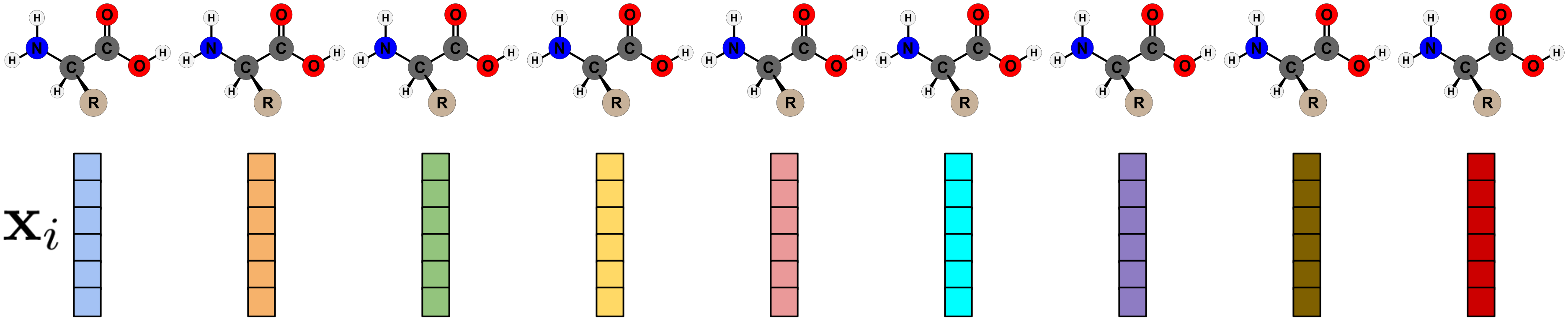
Performers on ImageNet64 - Approx. Softmax vs ReLU

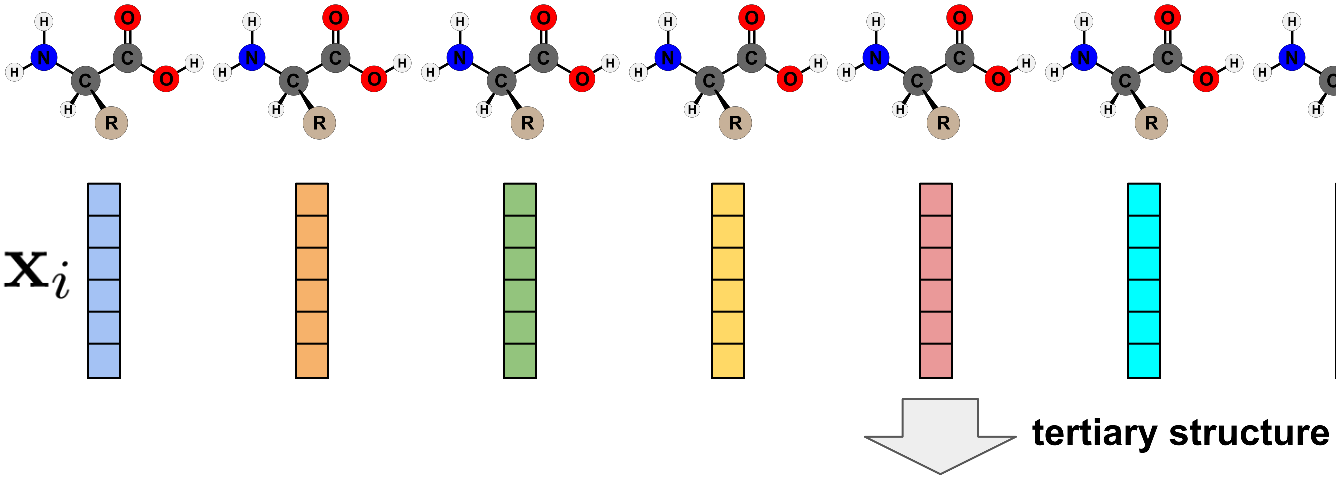


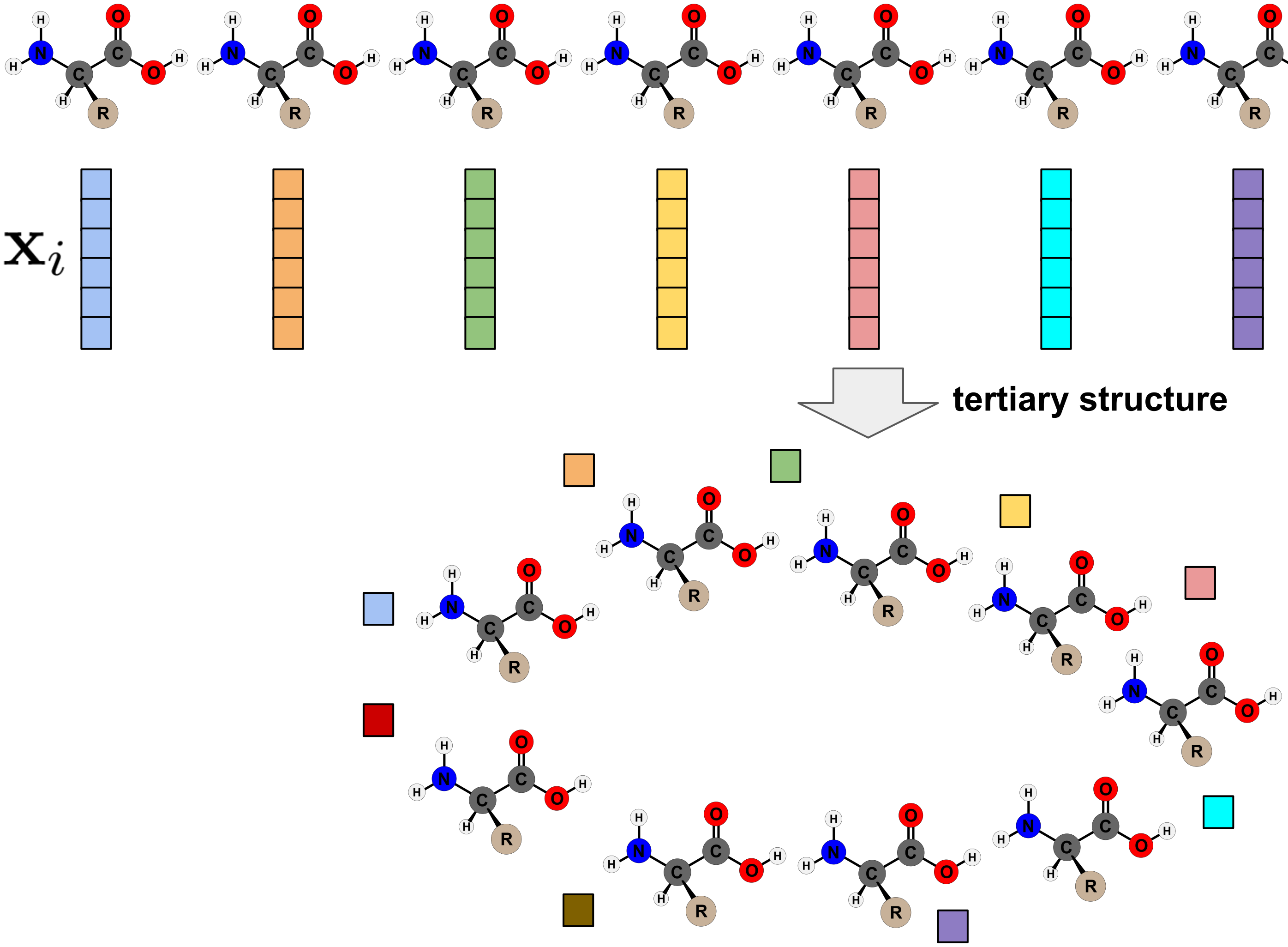
Performers
for Protein Design

Performers for Protein Design

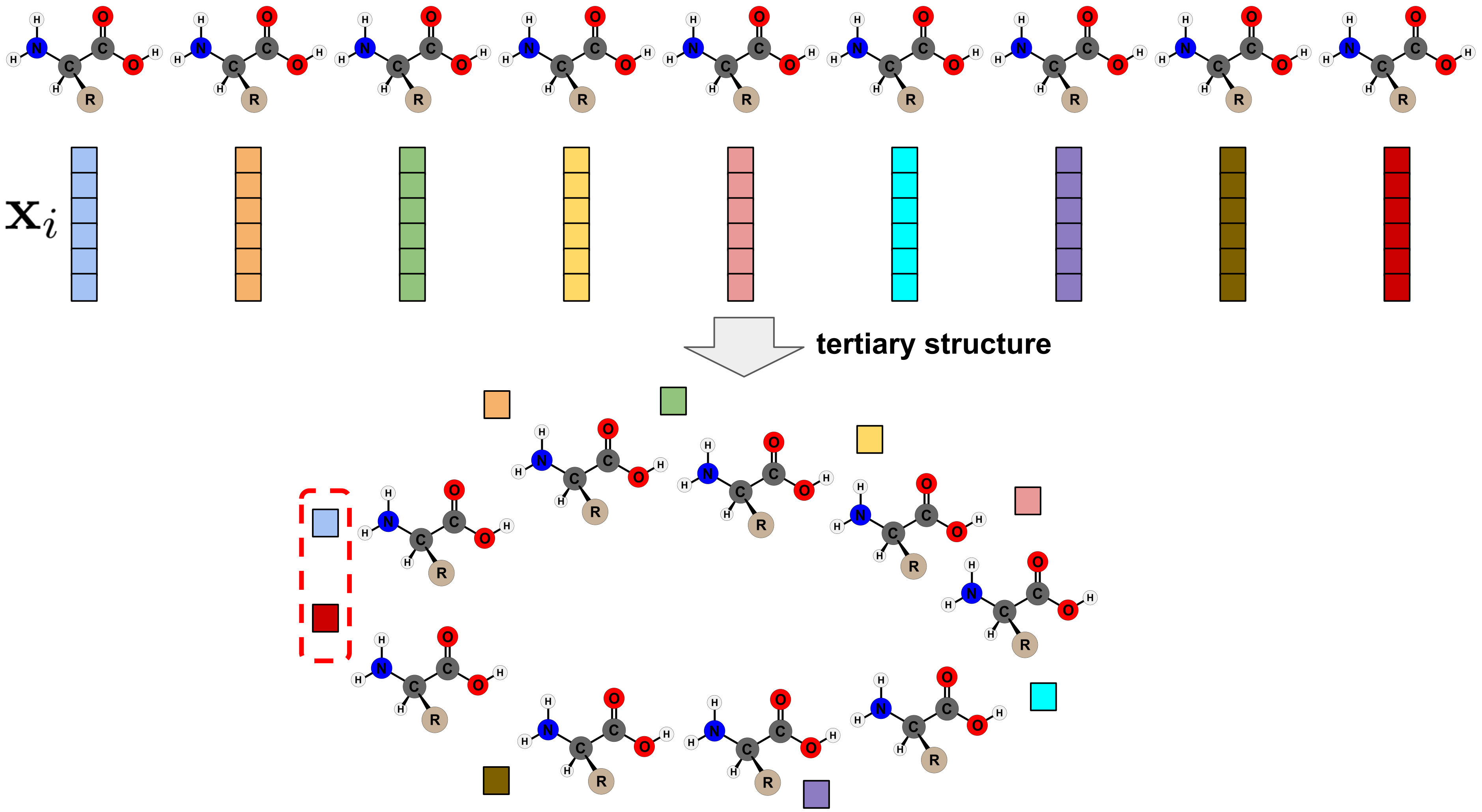











$\mathbf{x}_i$


$\mathbf{x}_i$
tertiary structure



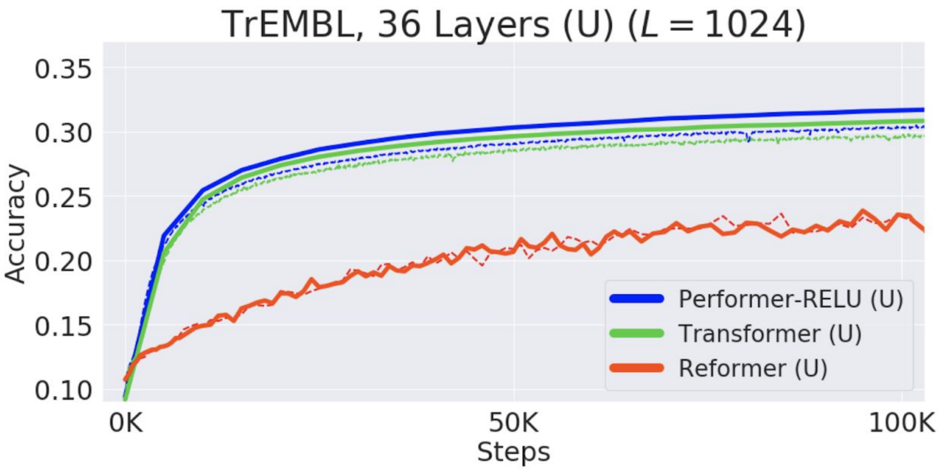
$\mathbf{x}_i$
tertiary structure

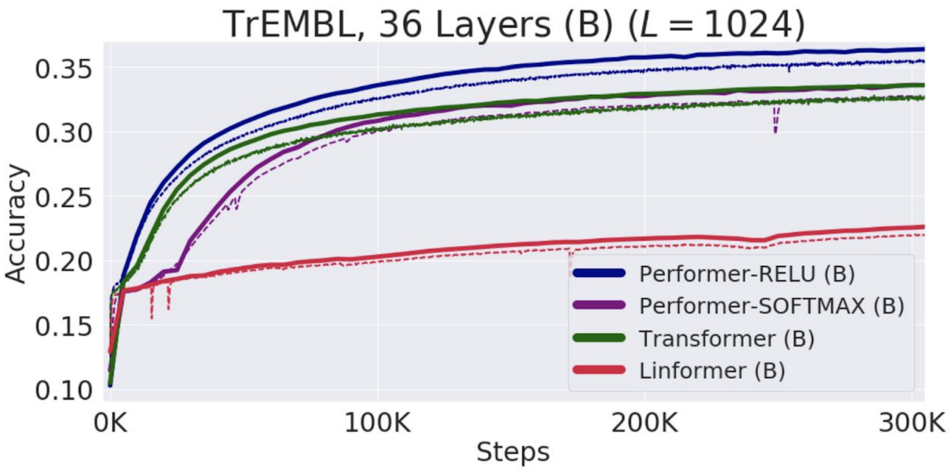


$\mathbf{x}_i$
tertiary structure

What we have already done...

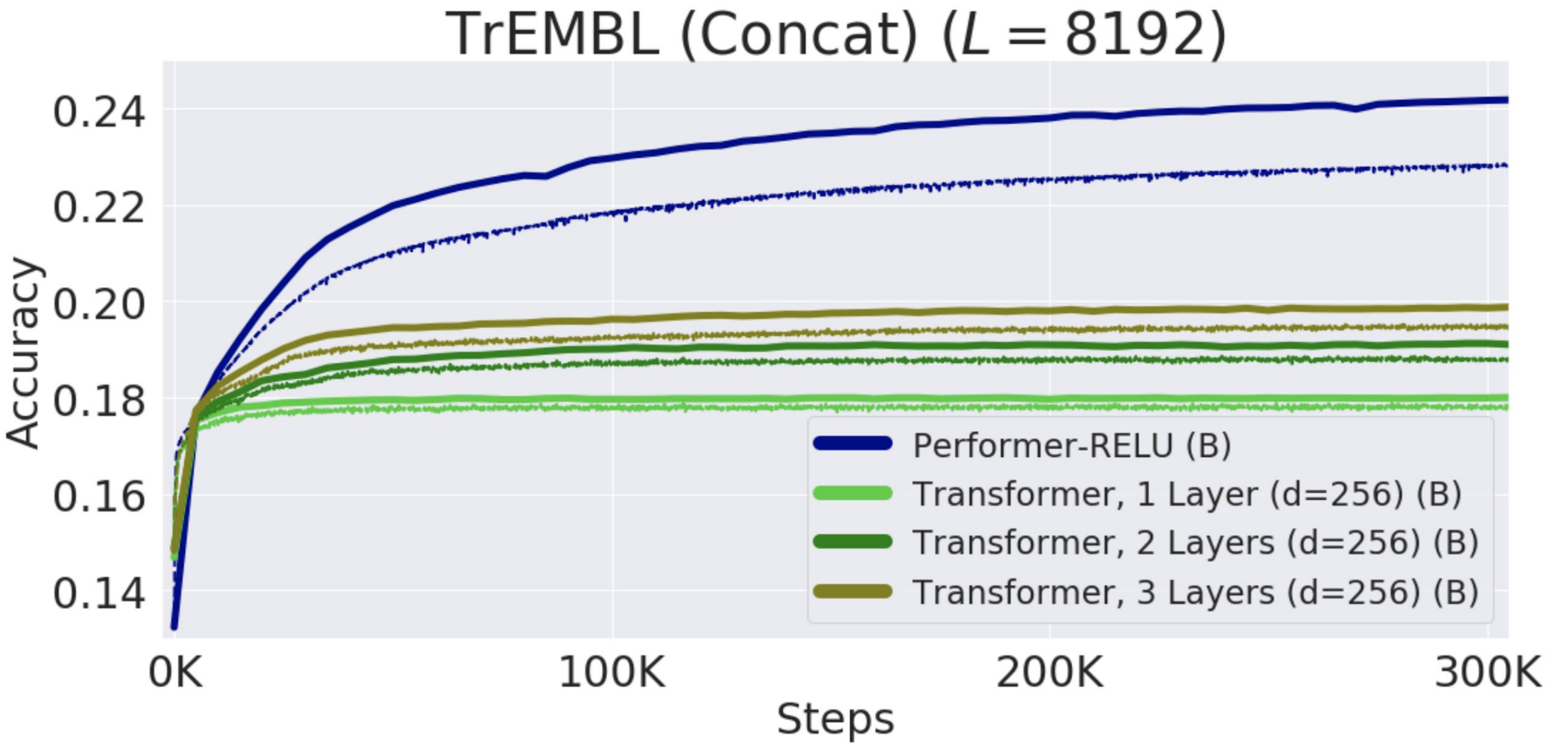
Performers on moderate-size biological sequences



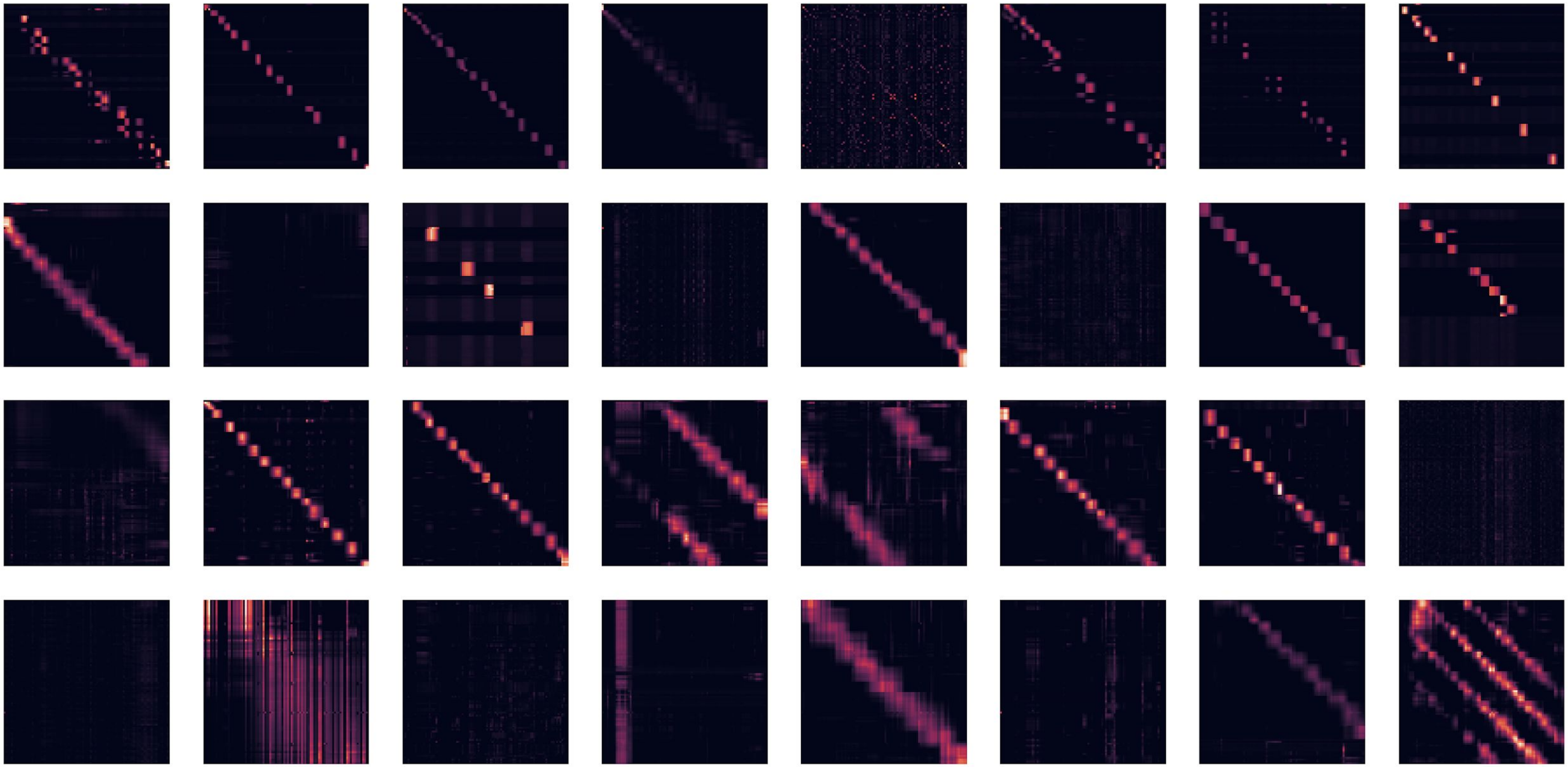


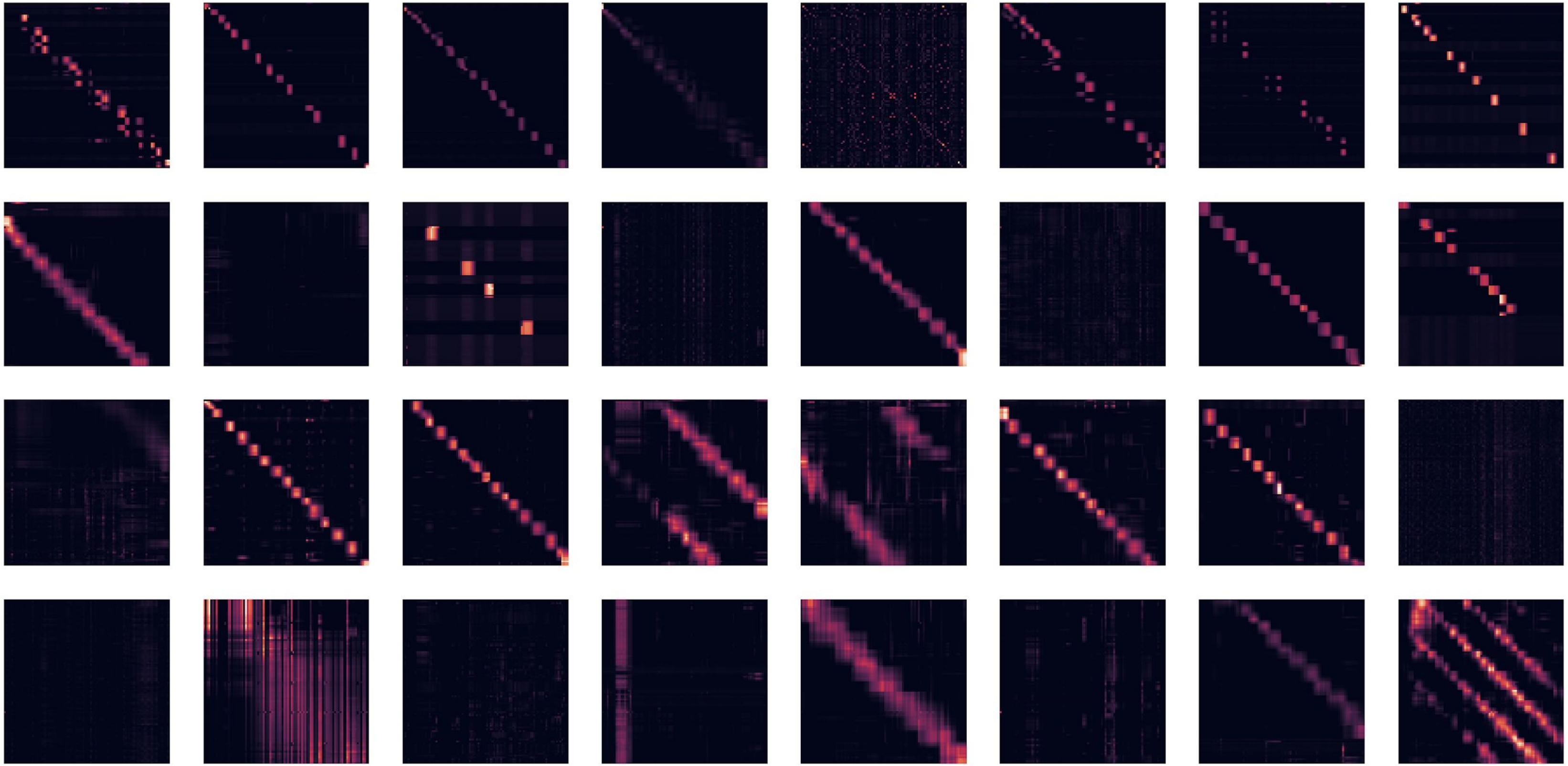
Train = Dashed, Validation = Solid, Unidirectional = (U), Bidirectional = (B). For TrEMBL, we used the exact same model parameters (nheads, nlayers, dff , d) = (8, 36, 1024, 512) from (Madani et al., 2020) for all runs. For fairness, all TrEMBL experiments used 16x16 TPU-v2's. Batch sizes were maximized for each separate run given the compute constraints. Hyperparameters & extended results including dataset statistics, out of distribution evaluations, and visualizations will appear soon in the extended version of the paper.

Performers on long biological sequences: towards modeling complexes of proteins - proof of concept

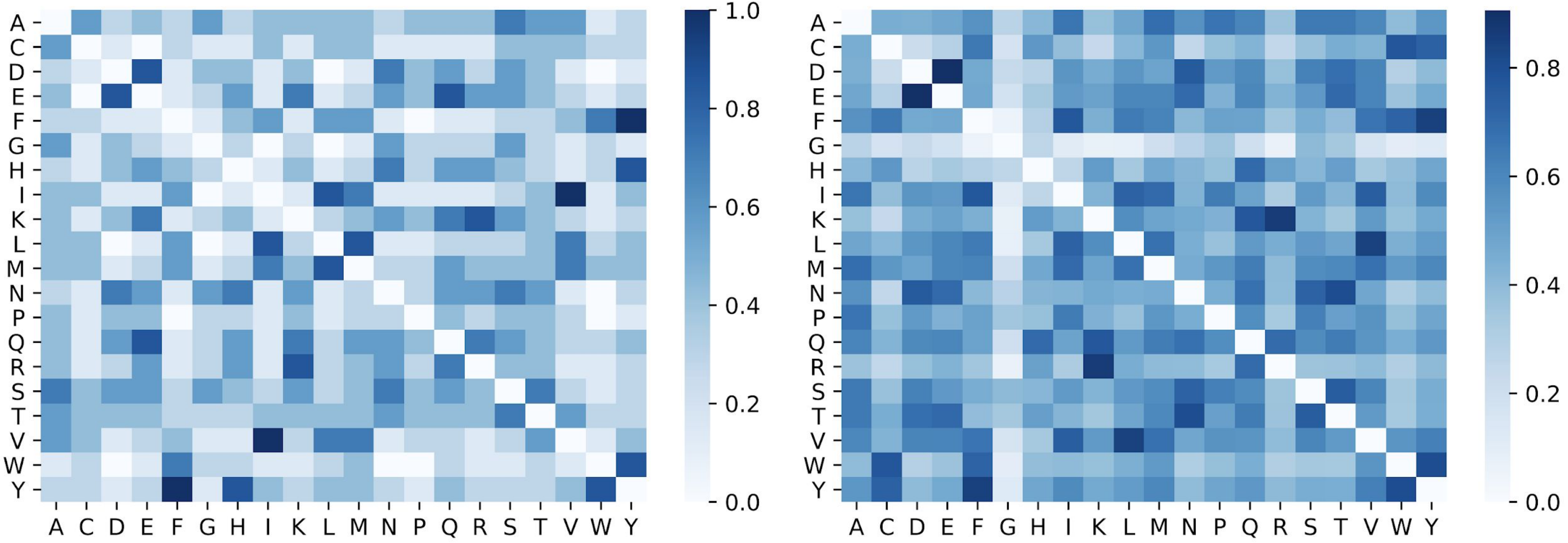


What do Performers attend to ?





We show the attention matrices for the first 4 layers and all 8 heads (each row is a layer, each column is head index, each cell contains the attention matrix across the entire BPT1_BOVIN protein sequence). Note that many heads show a diagonal pattern, where each node attends to its neighbors, and some heads show a vertical pattern, where each head attends to the same fixed positions.



Amino acid similarity matrix estimated from attention matrices aggregated across a small subset of sequences, as described in Vig et al. (Vig et al., 2020). The sub-figures correspond respectively to: (1) the normalized BLOSUM matrix, (2) the amino acid similarity estimated via a trained Performer model. Note that the Performer recognizes highly similar amino acid pairs such as (D, E) and (F, Y).

Thank you for the Attention !

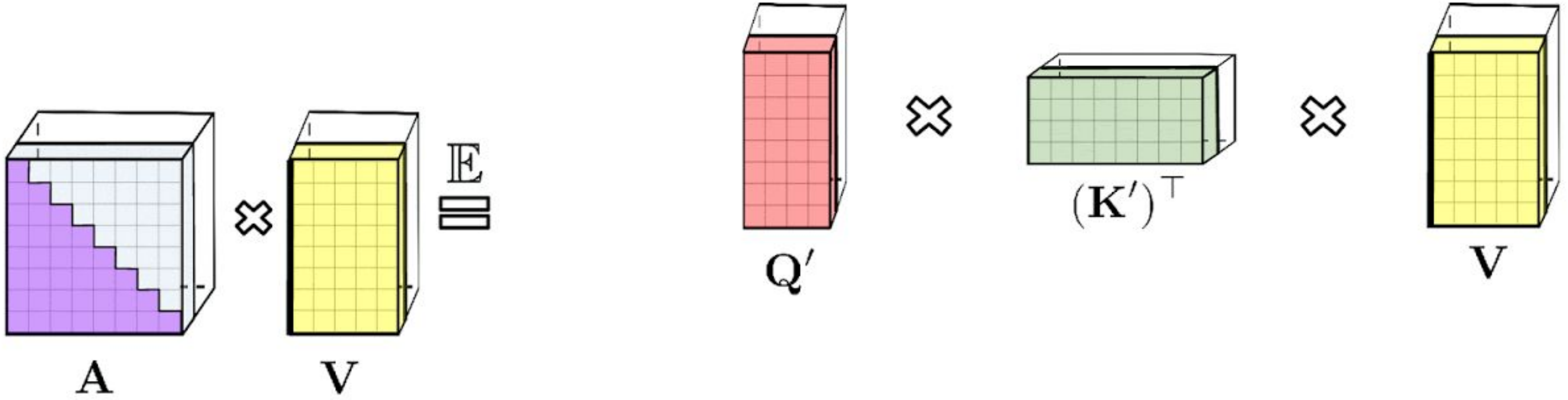


Fig. Linearized softmax causal attention as a prefix-sum computation engine.