LookupFFN

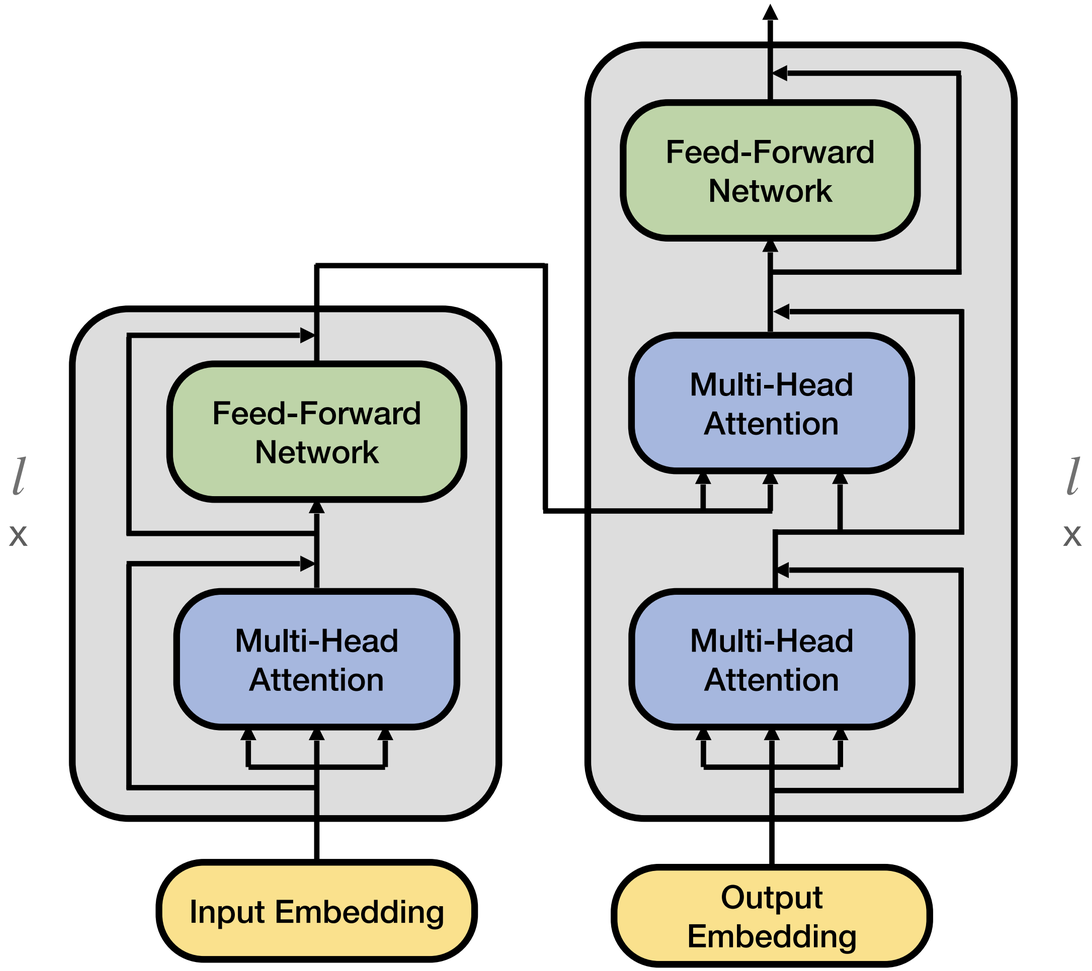
Making Transformers Compute-lite for CPU inference

Zhanpeng Zeng Michael Davies Pranav Pulijala Karthikeyan Sankaralingam Vikas Singh

Motivation

Transformer

Transformers are extremely effective on LLM and visions, but very compute intensive.

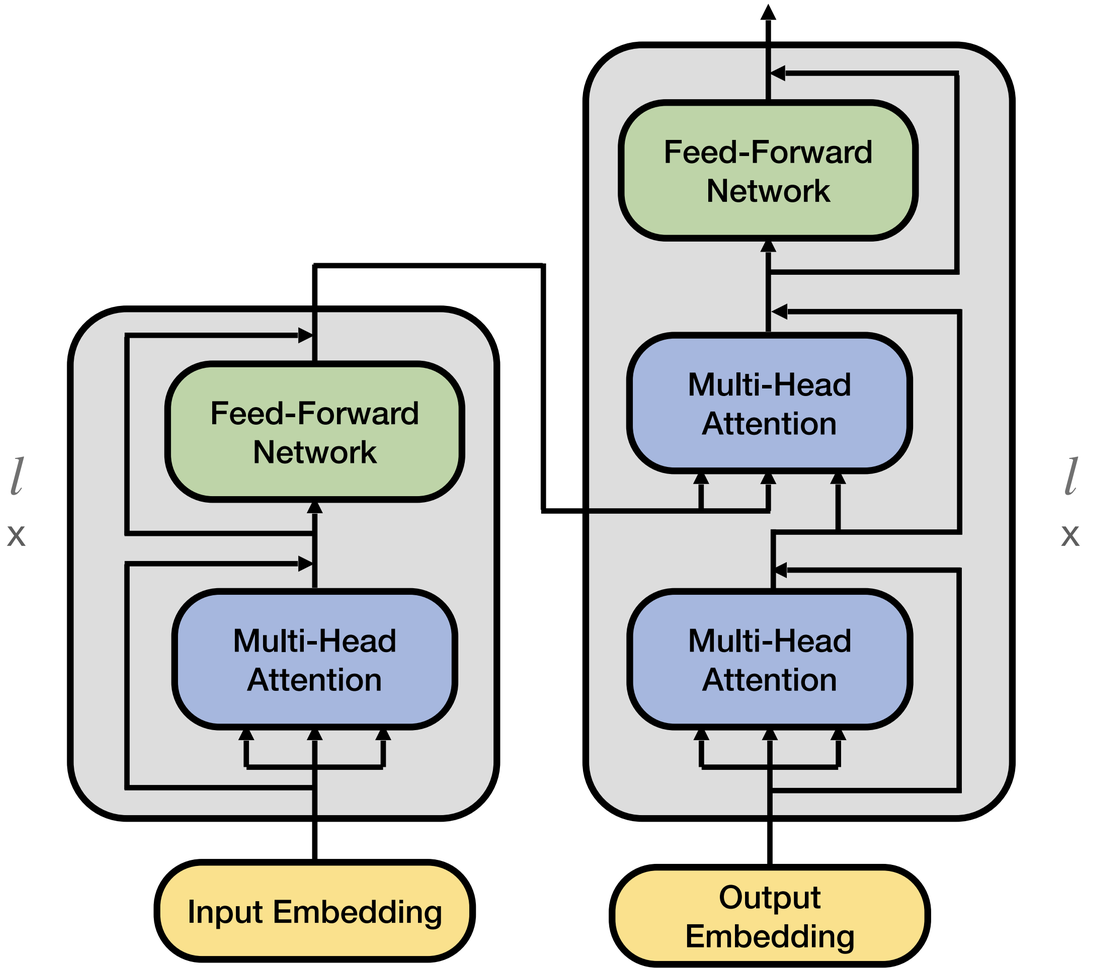


Motivation

Transformer

Transformers are extremely effective on LLM and visions, but very compute intensive.

How can we make it compute-lite?



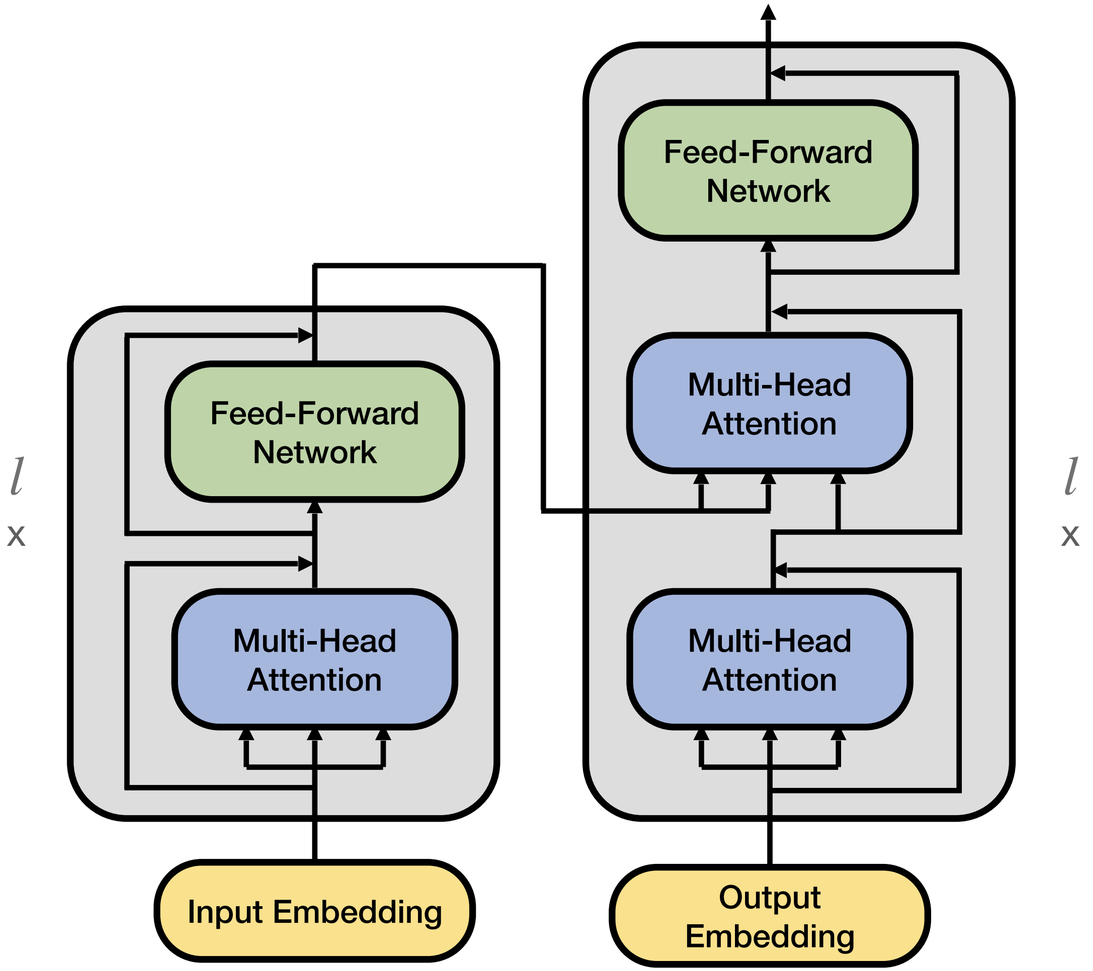
Motivation

Transformer

Transformers are extremely effective on LLM and visions, but very compute intensive.

How can we make it compute-lite?

Can we almost eliminate matrix multiplications?

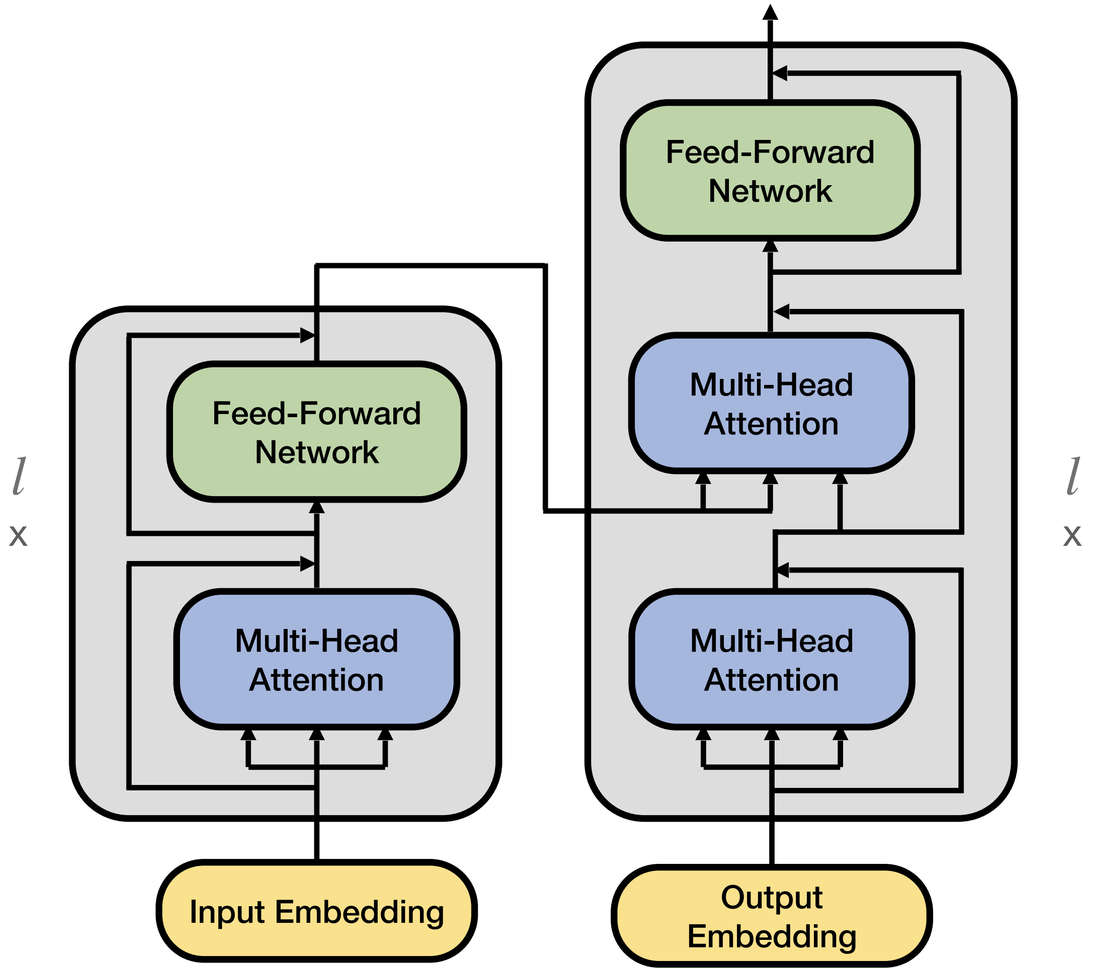


Motivation

Transformer

$$Y = \mathscr{F}(A) + A$$

$$A = \mathscr{G}(X) + X$$



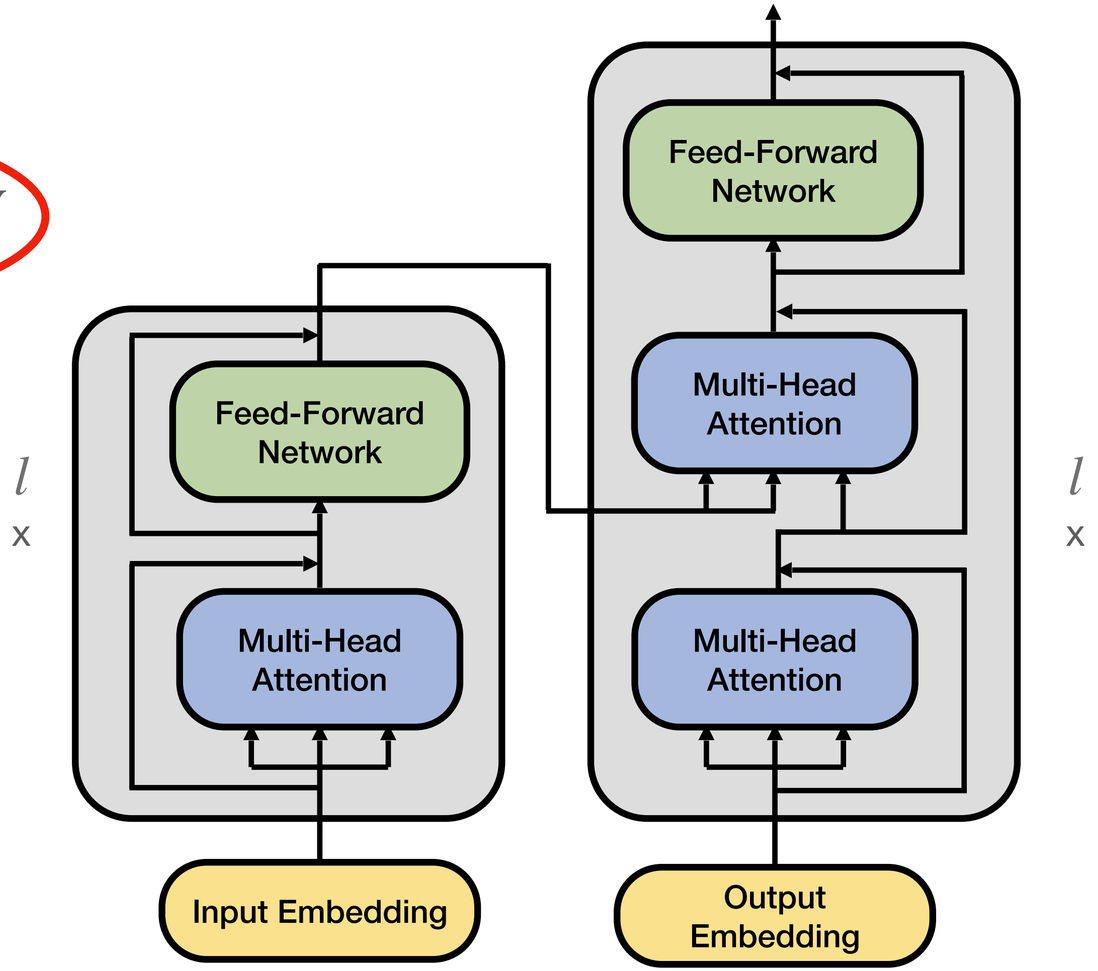
Motivation

Transformer

$$\mathscr{F}(x) = \sigma(xW^\top)V$$

$$Y = \mathscr{F}(A) + A$$

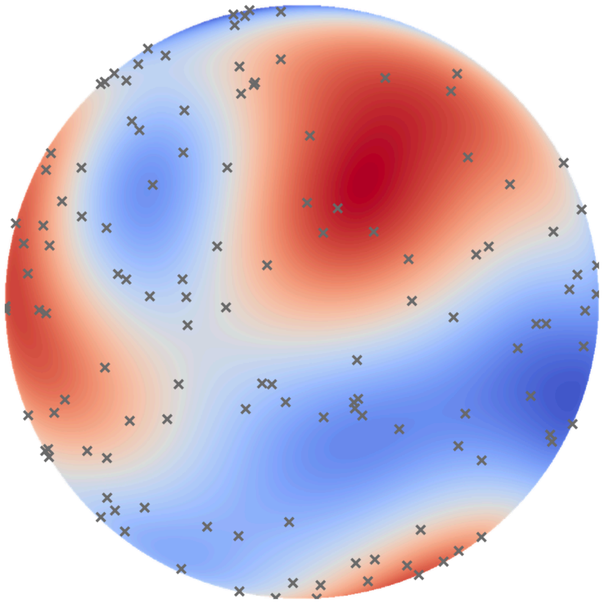
$$A = \mathscr{G}(X) + X$$



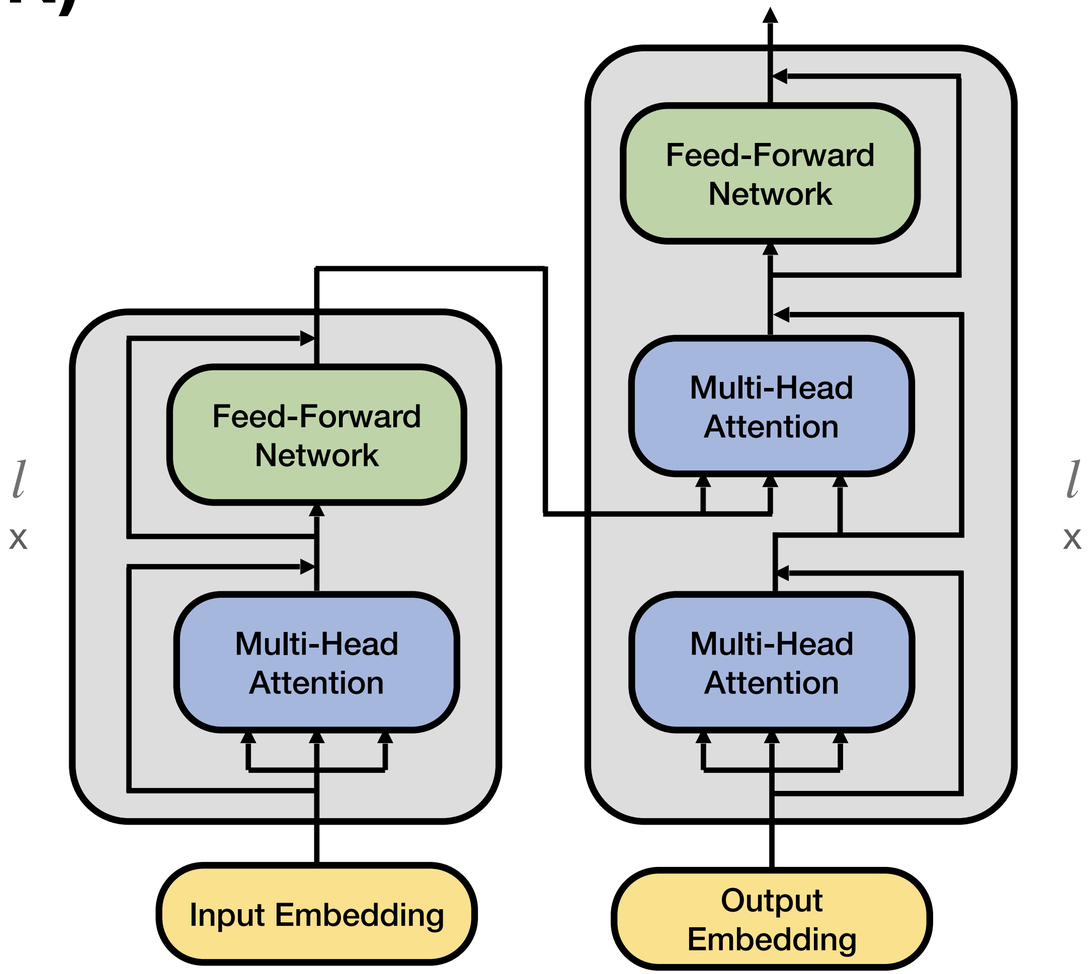
Motivation

Two Layer Feed-Forward Network (FFN)

$$\mathcal{F}(\cdot) = \sigma(\cdot W^\top)V = \sum_{i=1}^{t} \sigma(\langle \cdot, [W]_i \rangle)[V]_i$$



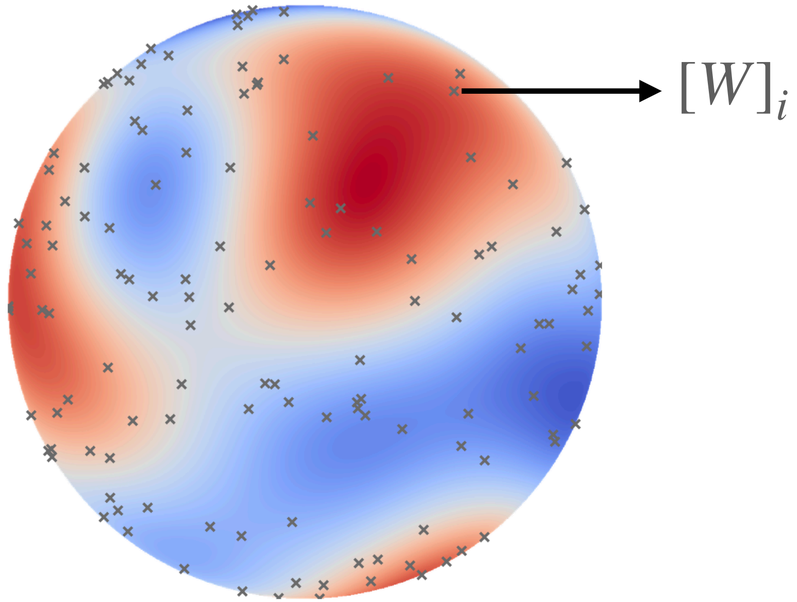
True $\mathcal{F}$



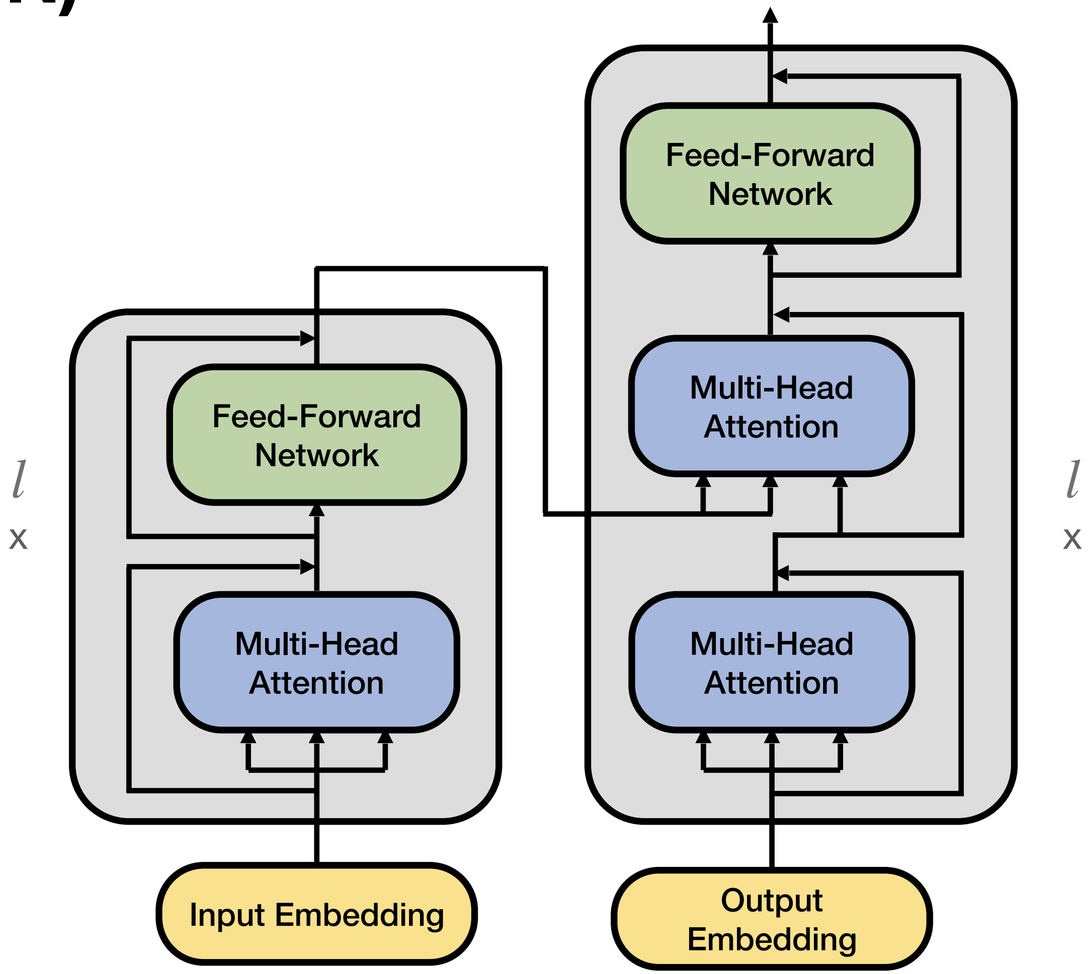
Motivation

Two Layer Feed-Forward Network (FFN)

$$\mathcal{F}(\cdot) = \sigma(\cdot W^\top)V = \sum_{i=1}^{t} \sigma(\langle \cdot, [W]_i \rangle)[V]_i$$



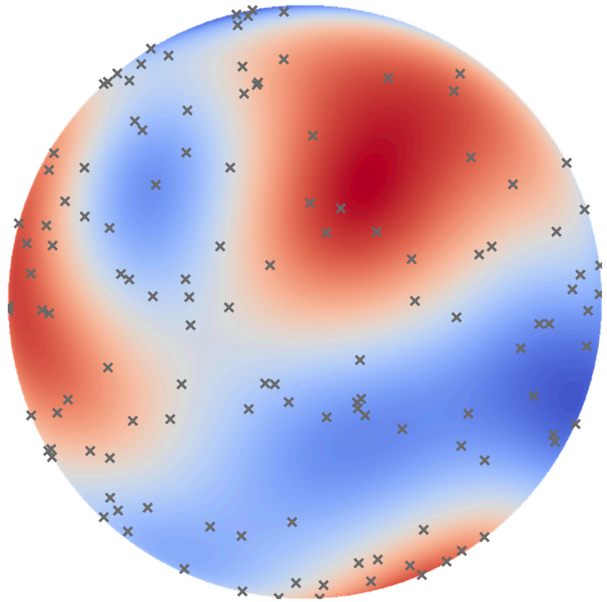
True $\mathcal{F}$



Making FFN Efficient

Revisiting LSH for approximating FFN

$$\mathcal{F}(\cdot) = \sum_{i=1}^{t} \sigma(\langle \cdot, [W]_i \rangle)[V]_i$$



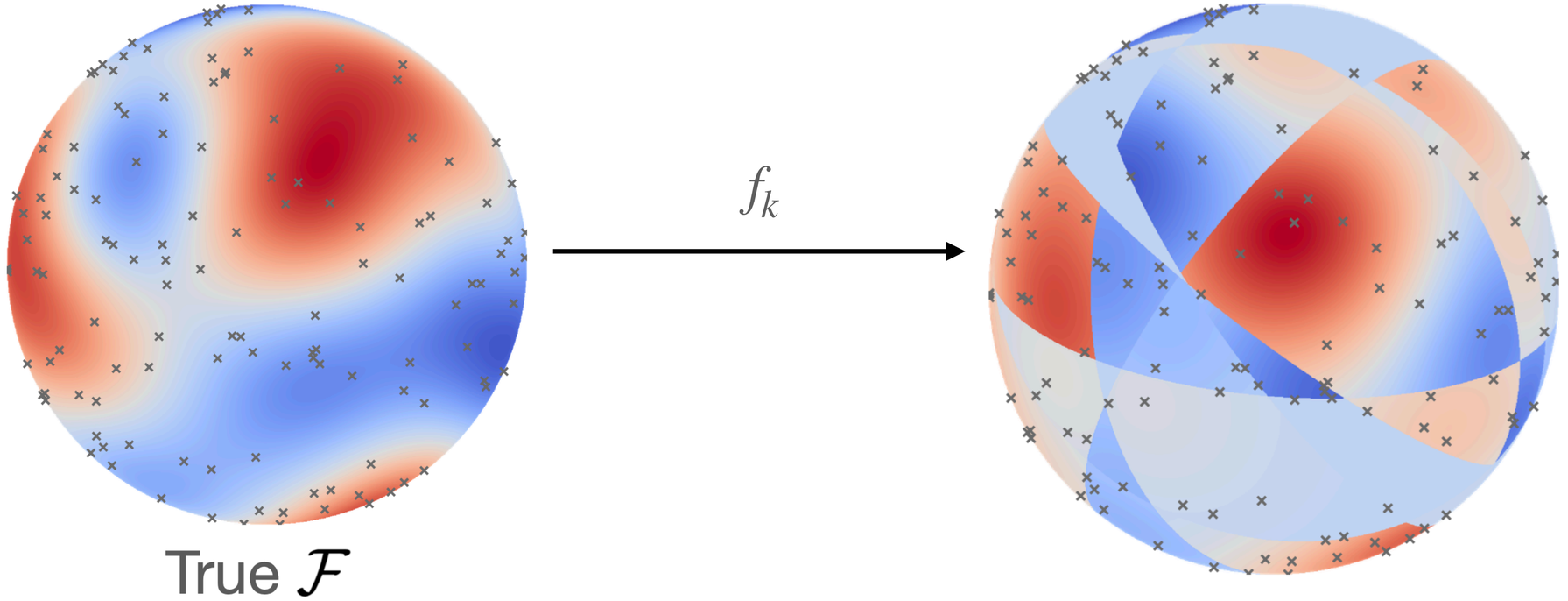
True $\mathcal{F}$

Making FFN Efficient

Revisiting LSH for approximating FFN

$$\mathcal{F}(\cdot) = \sum_{i=1}^{t} \sigma(\langle \cdot, [W]_i \rangle)[V]_i$$

$$\mathcal{F}(\cdot) = \sum_{i \in S(\cdot)} \sigma(\langle \cdot, [W]_i \rangle)[V]_i$$

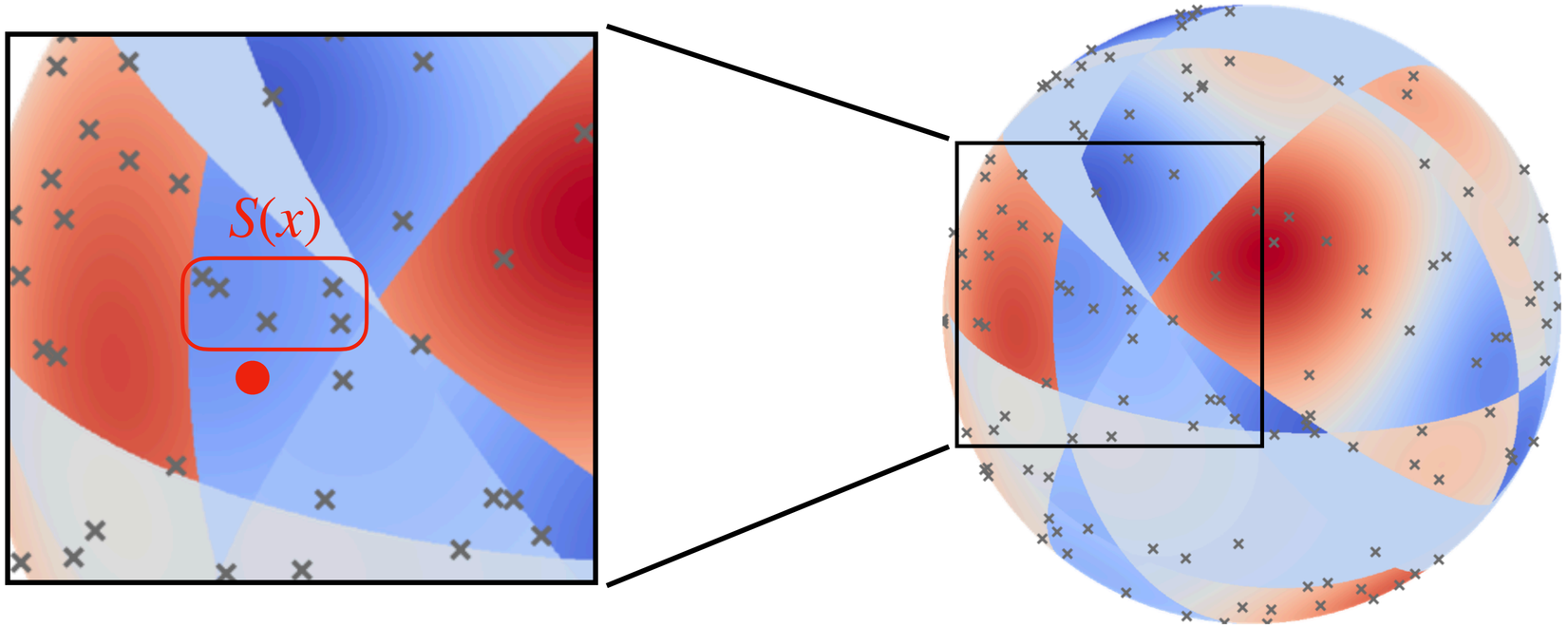


Chen, B., Medini, T., Farwell, J., gobriel, s., Tai, C., and Shrivastava, A. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. MLSys 2020

Making FFN Efficient

Revisiting LSH for approximating FFN

$$\mathcal{F}(x) = \sum_{i \in S(x)} \sigma(\langle x, [W]_i \rangle)[V]_i \qquad \mathcal{F}(\cdot) = \sum_{i \in S(\cdot)} \sigma(\langle \cdot, [W]_i \rangle)[V]_i$$

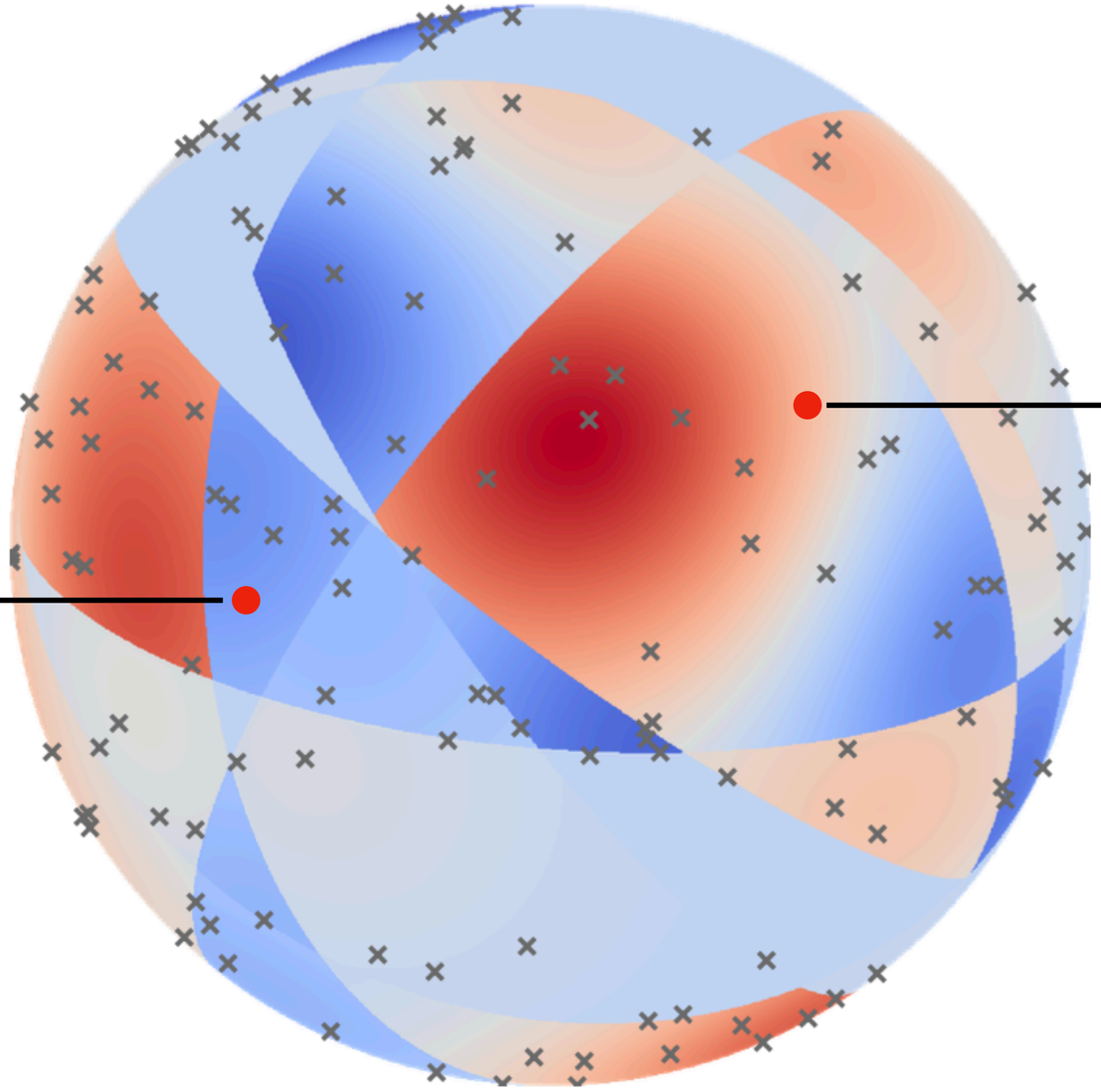


Chen, B., Medini, T., Farwell, J., gobriel, s., Tai, C., and Shrivastava, A. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. MLSys 2020

Making FFN Efficient

Revisiting LSH for approximating FFN: Challenge

$$\mathcal{F}(x) = \sum_{i \in S(x)} \sigma(\langle x, [W]_i \rangle)[V]_i$$

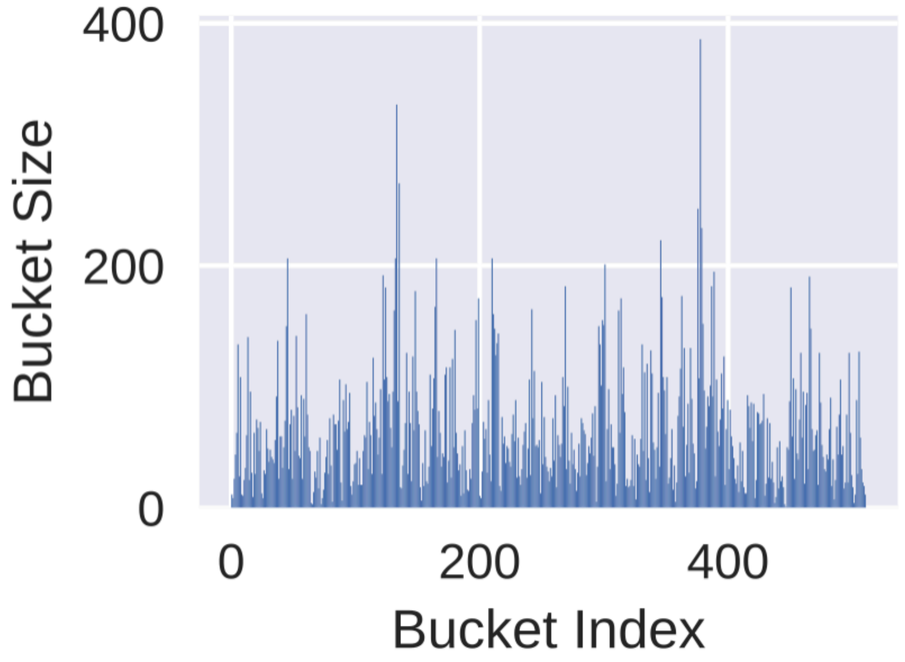


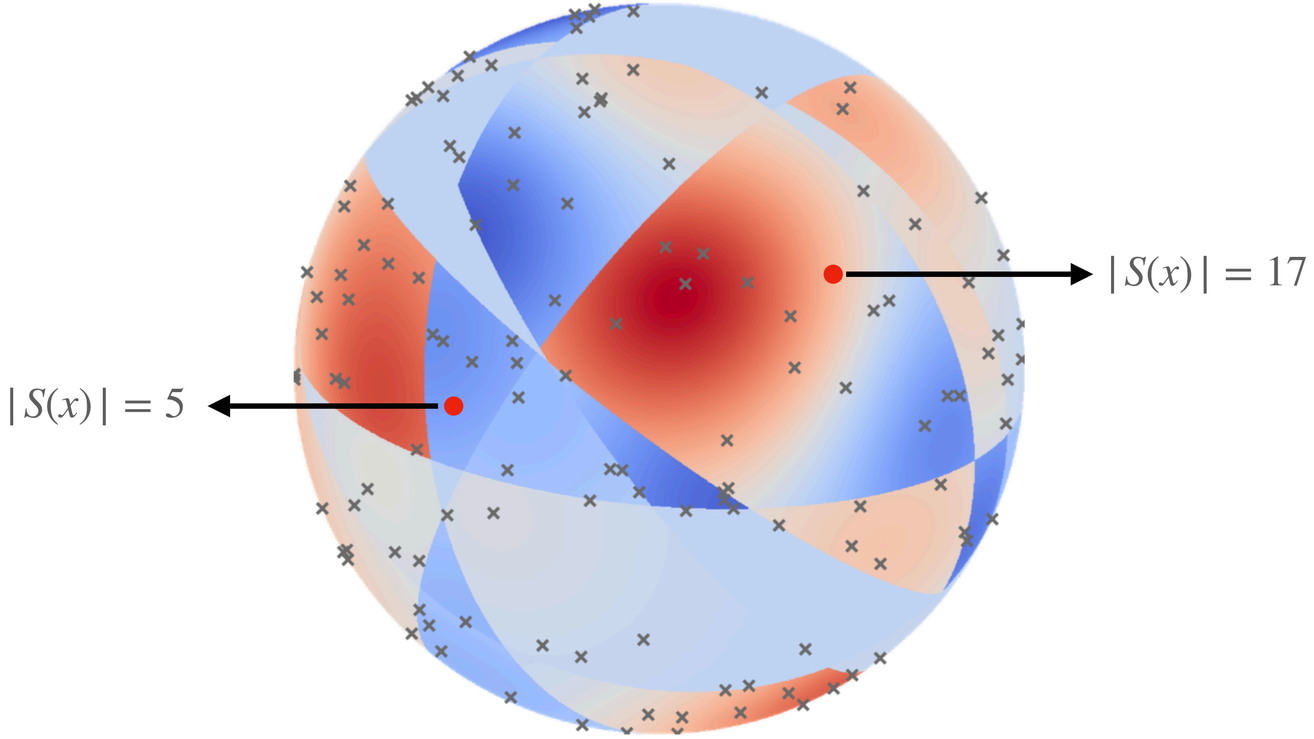
$|S(x)| = 17$

$|S(x)| = 5$

Making FFN Efficient

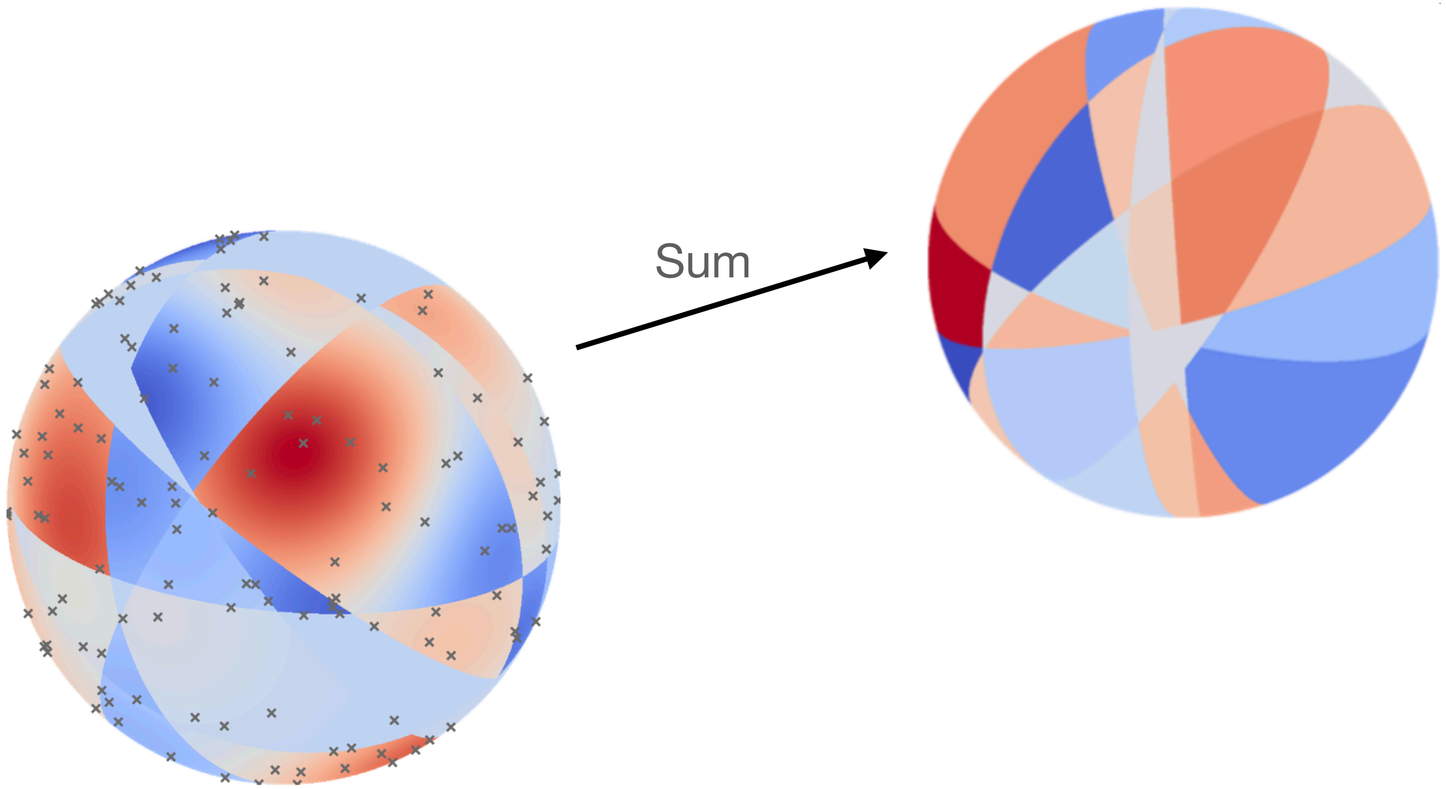
Revisiting LSH for approximating FFN: Challenge





Making FFN Efficient

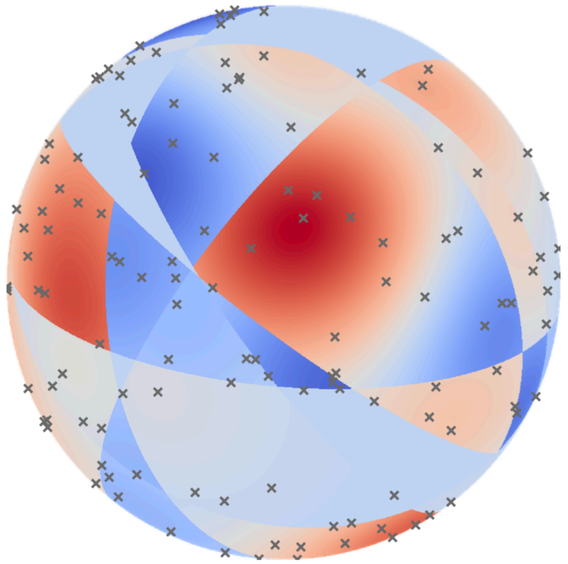
Revisiting LSH for approximating FFN: Potential Solution



Zeng, Z., Xiong, Y., Ravi, S., Acharya, S., Fung, G. M., and Singh, V. You only sample (almost) once: Linear cost self-attention via bernoulli sampling. ICML 2021.

Making FFN Efficient

Revisiting LSH for approximating FFN: Potential Solution



Sum

$$[T_k]_j = \sum_{f_k([W]_i)=j} [V]_i$$

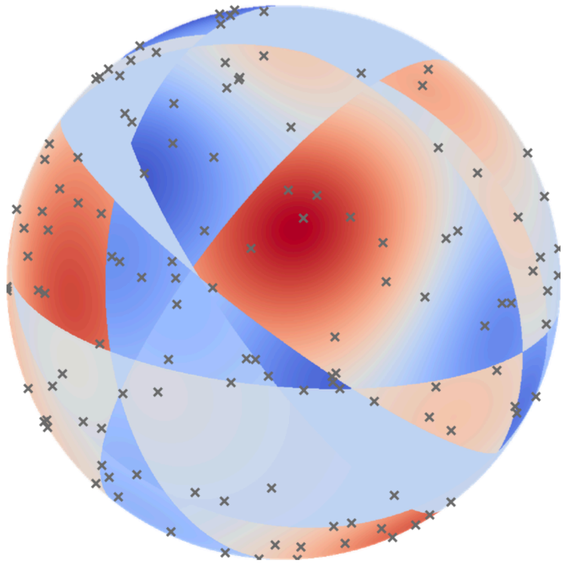
Code	Hash Table
0	v4 + v6
1	v2 + v7
2	v3
3	v0 + v1 + v5

V	W	Code
v0	w0	3
v1	w1	3
v2	w2	1
v3	w3	2
v4	w4	0
v5	w5	3
v6	w6	0
v7	w7	1

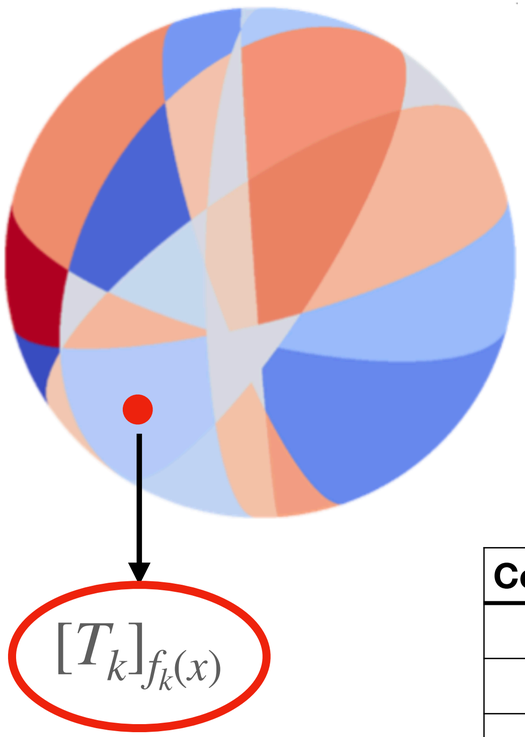
Zeng, Z., Xiong, Y., Ravi, S., Acharya, S., Fung, G. M., and Singh, V. You only sample (almost) once: Linear cost self-attention via bernoulli sampling. ICML 2021.

Making FFN Efficient

Revisiting LSH for approximating FFN: Potential Solution



Sum



$$[T_k]_j = \sum_{f_k([W]_i)=j} [V]_i$$

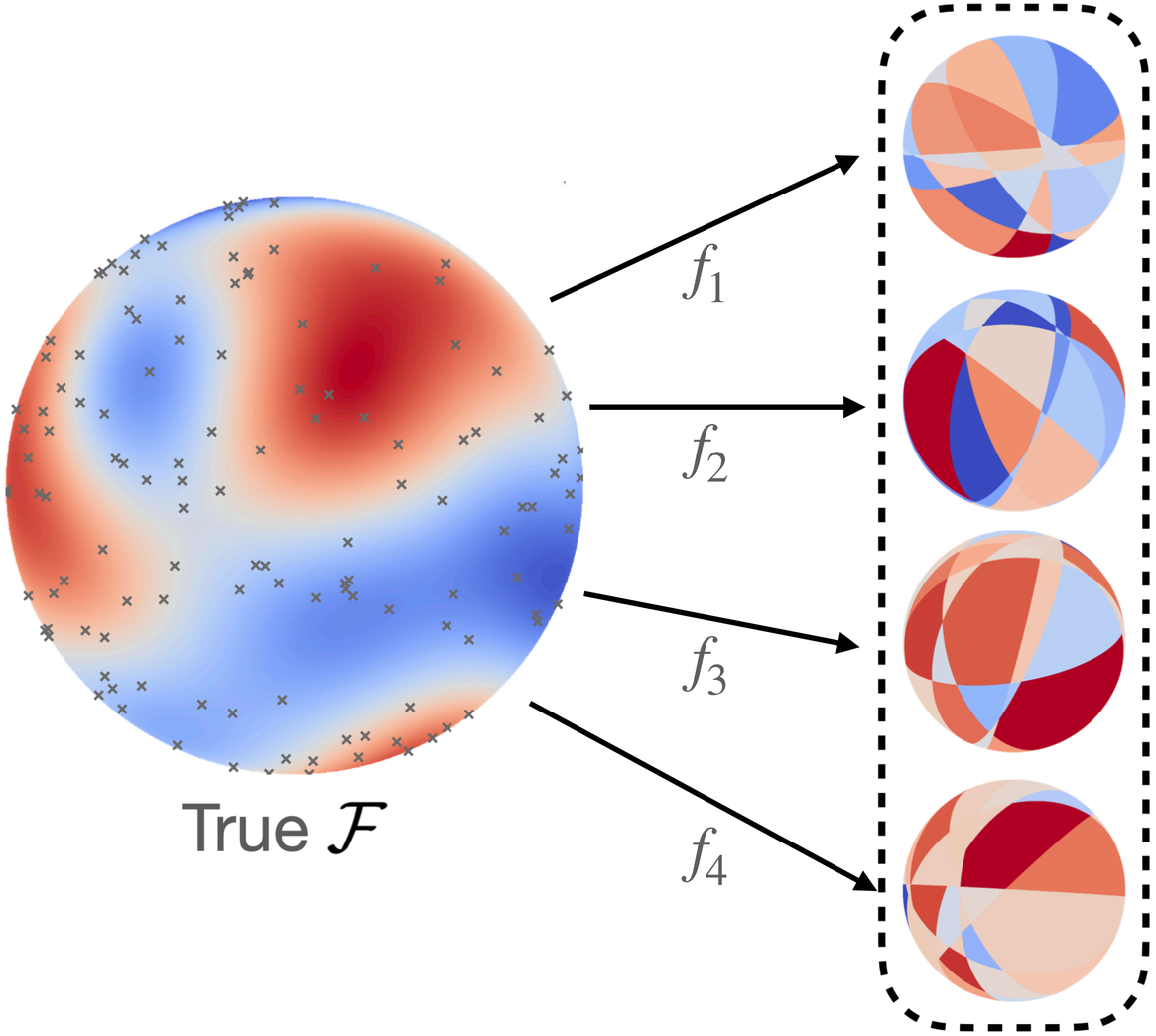
Code	Hash Table
0	v4 + v6
1	v2 + v7
2	v3
3	v0 + v1 + v5

V	W	Code
v0	w0	3
v1	w1	3
v2	w2	1
v3	w3	2
v4	w4	0
v5	w5	3
v6	w6	0
v7	w7	1

Zeng, Z., Xiong, Y., Ravi, S., Acharya, S., Fung, G. M., and Singh, V. You only sample (almost) once: Linear cost self-attention via bernoulli sampling. ICML 2021.

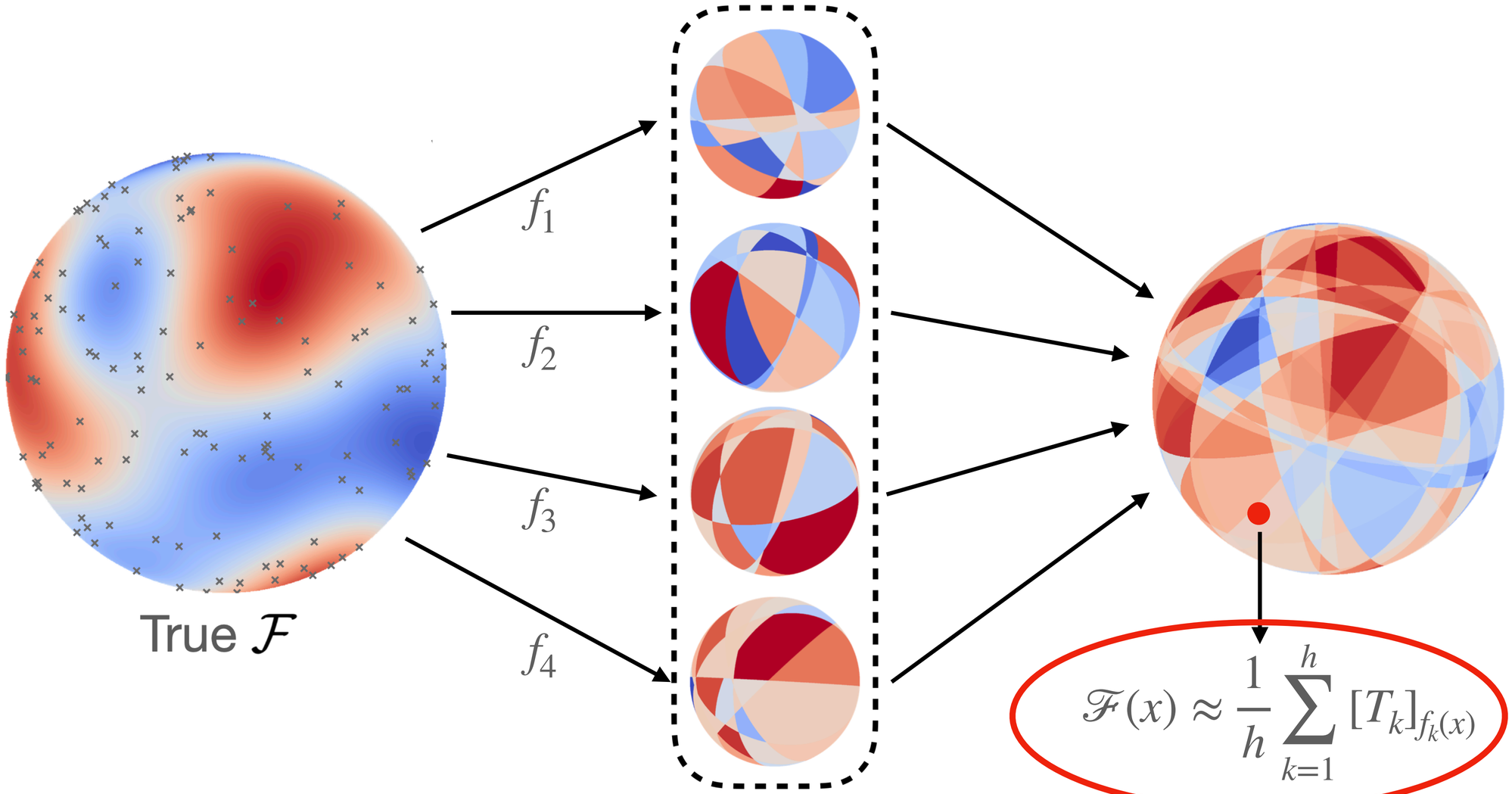
Making FFN Efficient

Revisiting LSH for approximating FFN: Potential Solution



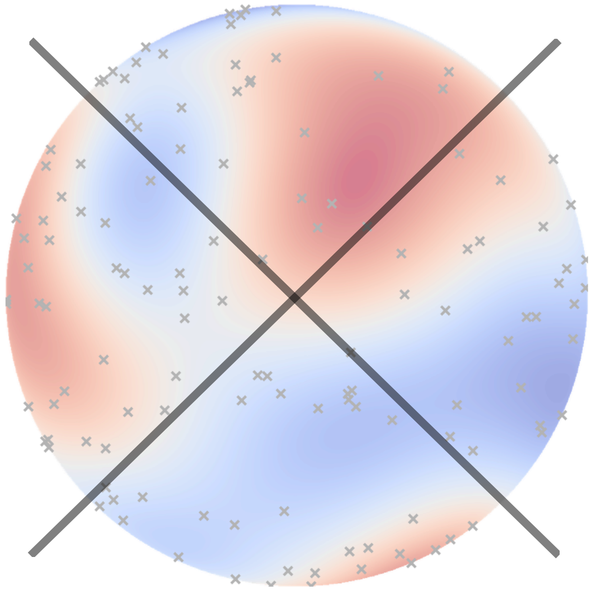
Making FFN Efficient

Revisiting LSH for approximating FFN: Potential Solution

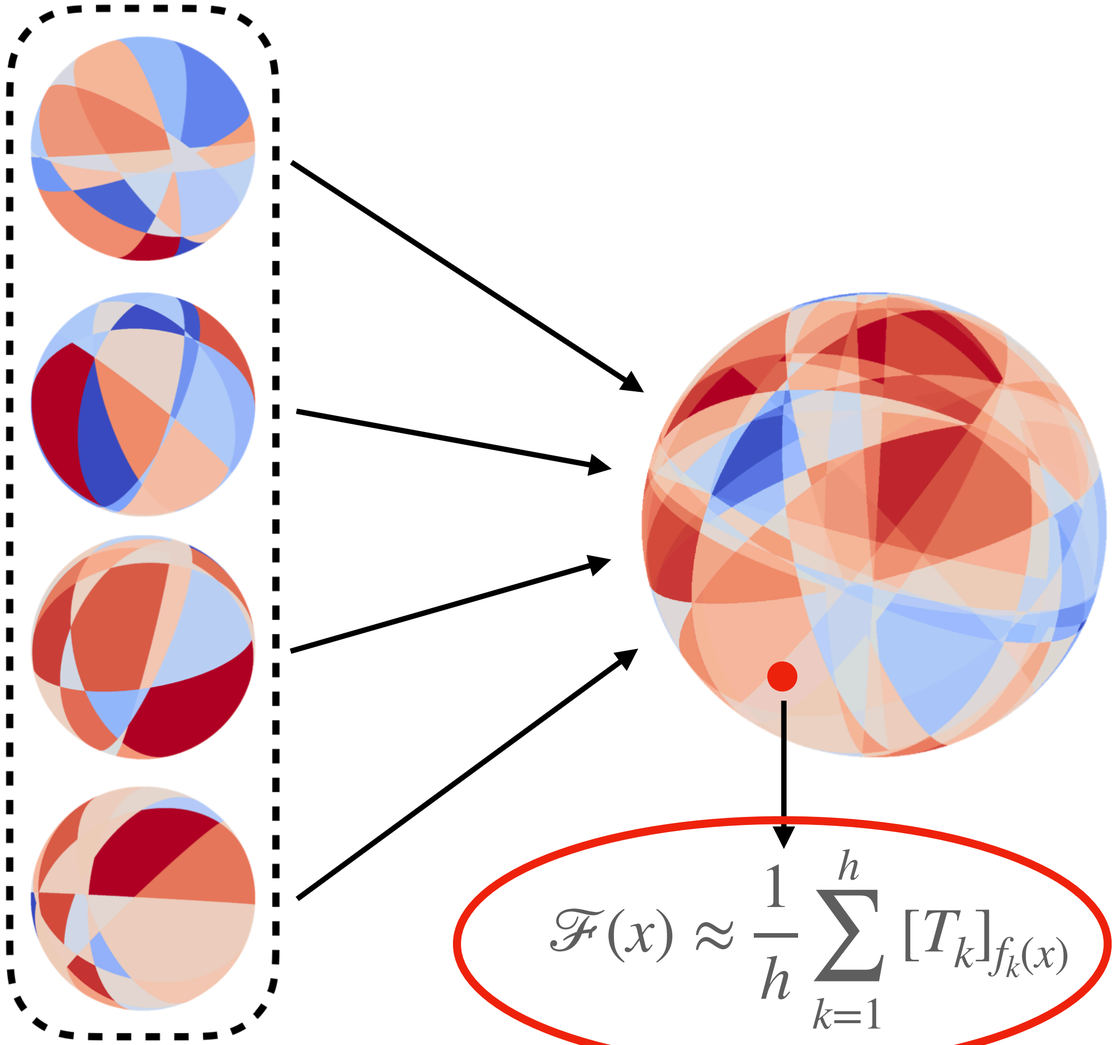


Making FFN Efficient

LookupFFN: Breaking dependency on True F



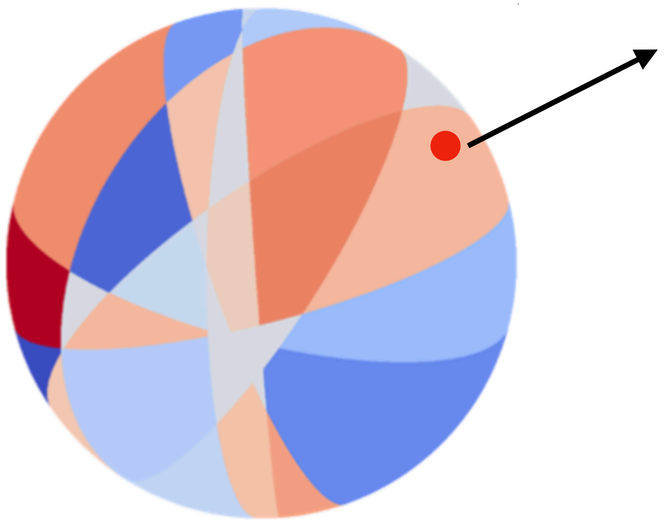
True $\mathcal{F}$



$$\mathcal{F}(x) \approx \frac{1}{h} \sum_{k=1}^{h} [T_k]_{f_k(x)}$$

Making FFN Efficient

LookupFFN: Make it differentiable



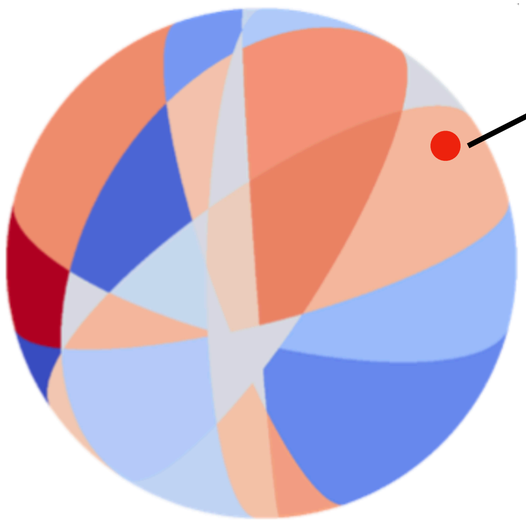
$[T_k]_{f_k(x)}$

$z_k = xR_k$

$f_k(x) = \text{decimal}(\text{sign}(z_k))$

Making FFN Efficient

LookupFFN: Make it differentiable



$[T_k]_{f_k(x)}$

$z_k = xR_k$

$$f_k(x) = \text{decimal}(\text{sign}(z_k))$$

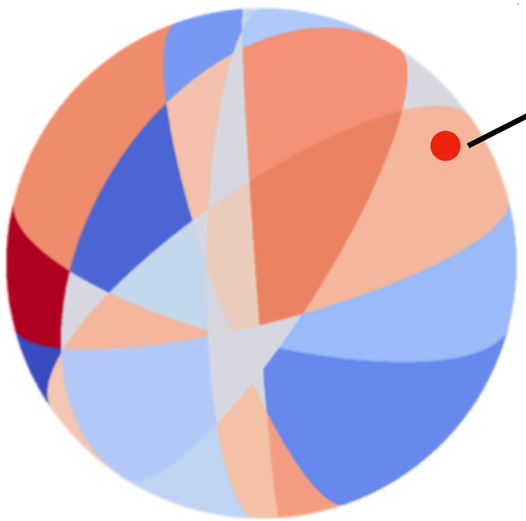
Equivalent

$$f_k(x) = \arg\max_i(\langle z_k, [S]_i \rangle)$$

-1	-1	-1
-1	-1	1
-1	1	-1
-1	1	1
1	-1	-1
1	-1	1
1	1	-1
1	1	1

Making FFN Efficient

LookupFFN: Make it differentiable



$[T_k]_{f_k(x)}$

$z_k = xR_k$

$$f_k(x) = \text{decimal}(\text{sign}(z_k))$$

Equivalent

$$f_k(x) = \arg\max_i(\langle z_k, [S]_i \rangle)$$

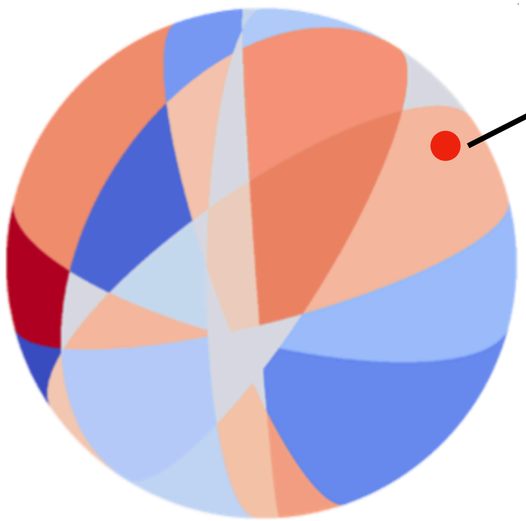
Differentiable Relaxation

$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

-1	-1	-1
-1	-1	1
-1	1	-1
-1	1	1
1	-1	-1
1	-1	1
1	1	-1
1	1	1

Making FFN Efficient

LookupFFN: Make it differentiable



$[T_k]_{f_k(x)}$

$$z_k = xR_k$$

$$f_k(x) = \text{decimal}(\text{sign}(z_k))$$

Equivalent

$$f_k(x) = \arg\max_i(\langle z_k, [S]_i \rangle)$$

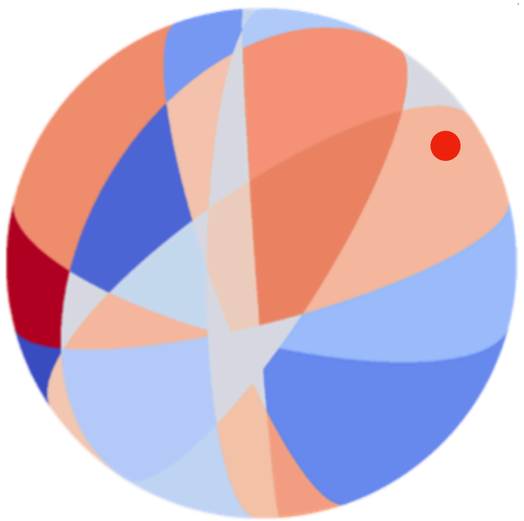
Differentiable Relaxation

$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

-1	-1	-1
-1	-1	1
-1	1	-1
-1	1	1
1	-1	-1
1	-1	1
1	1	-1
1	1	1

Making FFN Efficient

LookupFFN: Efficient Approximation



$$z_k = xR_k$$

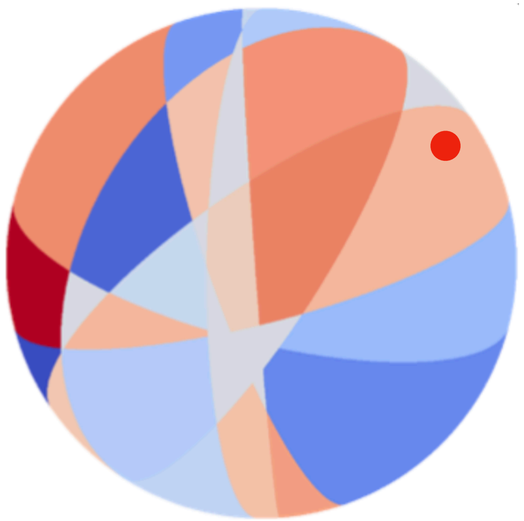
$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

Equivalent

$$\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))$$

Making FFN Efficient

LookupFFN: Efficient Approximation



$$z_k = xR_k$$

Approximate

$$\sum_{i \in N(xR_k)} \exp(\langle z_k, [S]_i \rangle)[T_k]_i$$

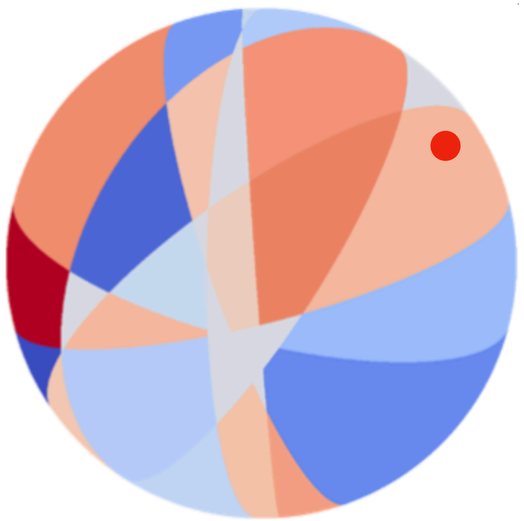
$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

Equivalent

$$\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))$$

Making FFN Efficient

LookupFFN: Efficient Approximation



$$z_k = xR_k$$

$$\arg\max_i \langle z_k, [S]_i \rangle = \text{decimal}(\text{sign}(z_k))$$

Approximate

$$\sum_{i \in N(xR_k)} \exp(\langle z_k, [S]_i \rangle)[T_k]_i$$

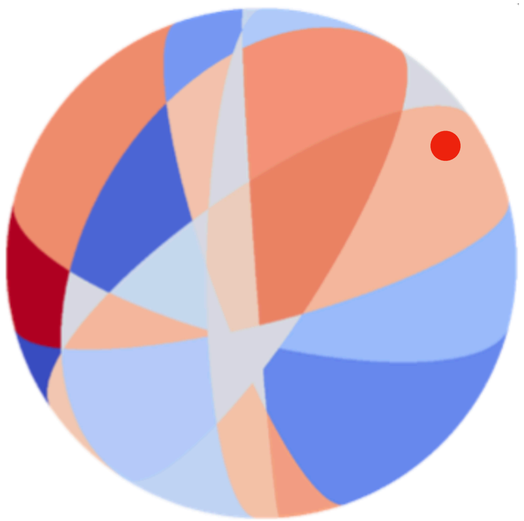
$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

Equivalent

$$\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))$$

Making FFN Efficient

LookupFFN: Put it together



$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

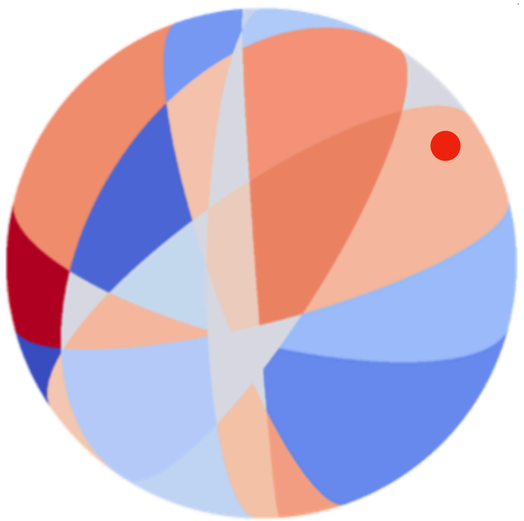
$$\downarrow$$

$$\frac{\exp(\langle z_k, [S]_{f_k(x)} \rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))}$$

$$z_k = xR_k$$

Making FFN Efficient

LookupFFN: Put it together



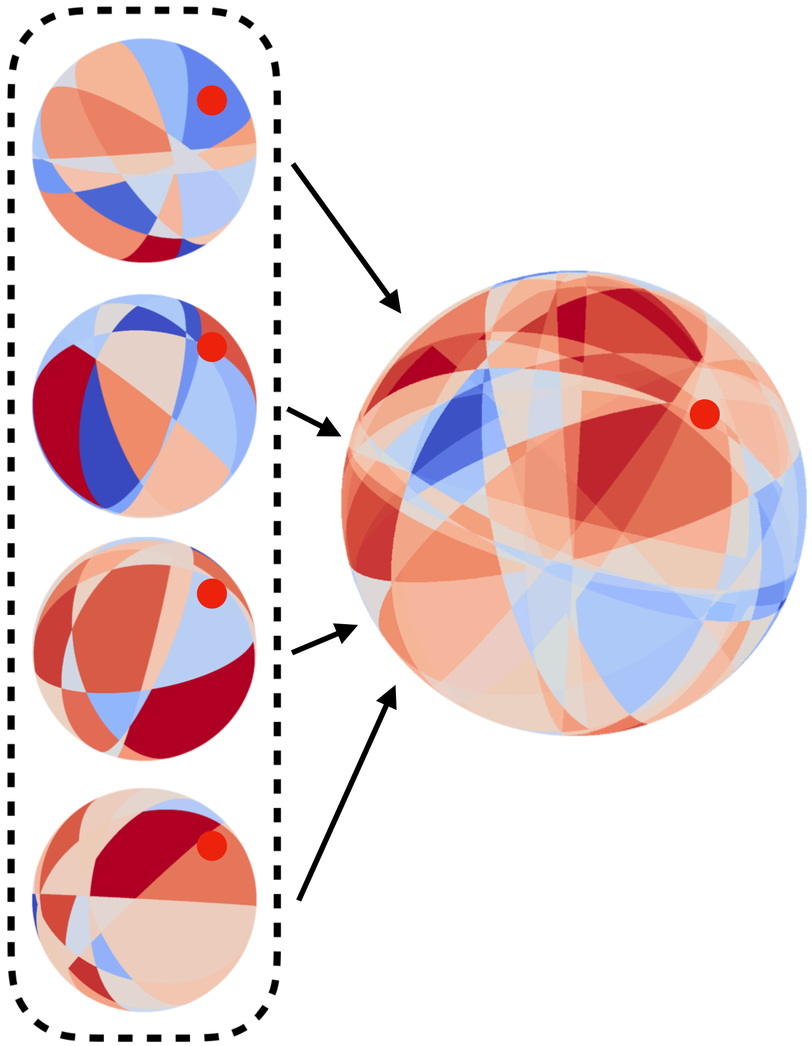
$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

$$z_k = xR_k$$

$$\frac{\exp(\langle z_k, [S]_{f_k(x)} \rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))} \longrightarrow f_k(x) = \text{decimal}(\text{sign}(z_k))$$

Making FFN Efficient

LookupFFN: Put it together



$$\frac{\sum_{i=1}^{2^\tau} \exp(\langle z_k, [S]_i \rangle)[T_k]_i}{\sum_{j=1}^{2^\tau} \exp(\langle z_k, [S]_j \rangle)}$$

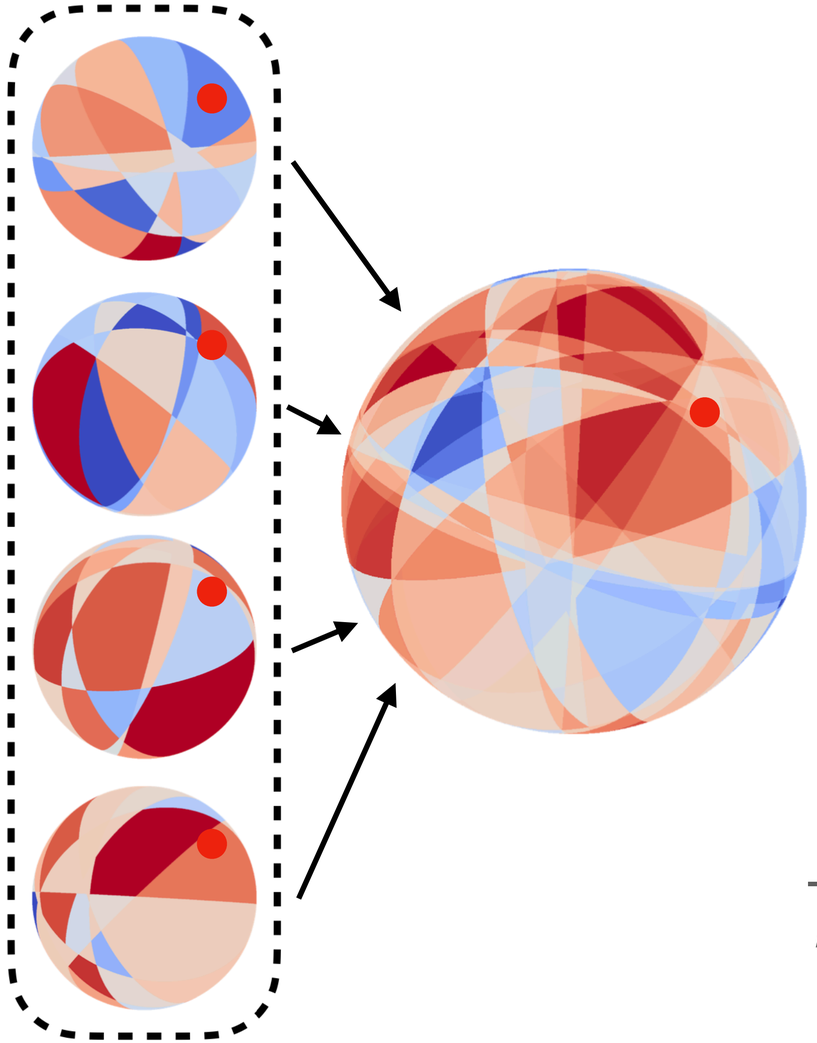
$$z_k = xR_k$$

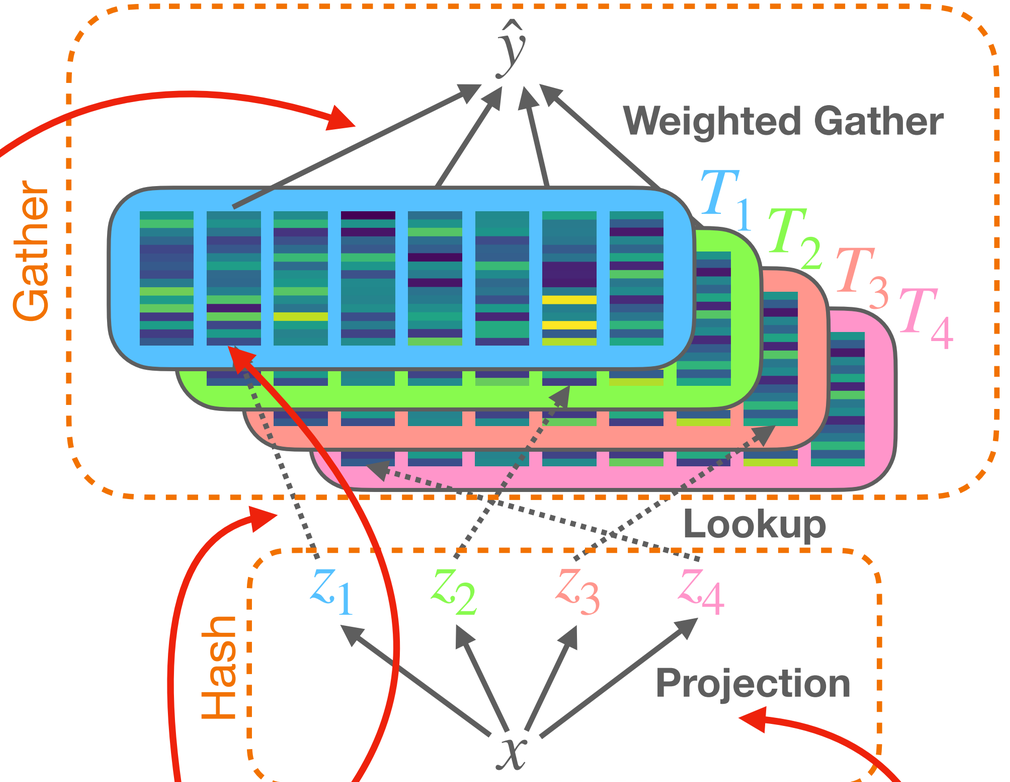
$$\frac{\exp(\langle z_k, [S]_{f_k(x)} \rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))} \longrightarrow f_k(x) = \text{decimal}(\text{sign}(z_k))$$

$$\frac{1}{h} \sum_{k=1}^{h} \frac{\exp(\langle z_k, [S]_{f_k(x)} \rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))}$$

Making FFN Efficient

LookupFFN: Put it together



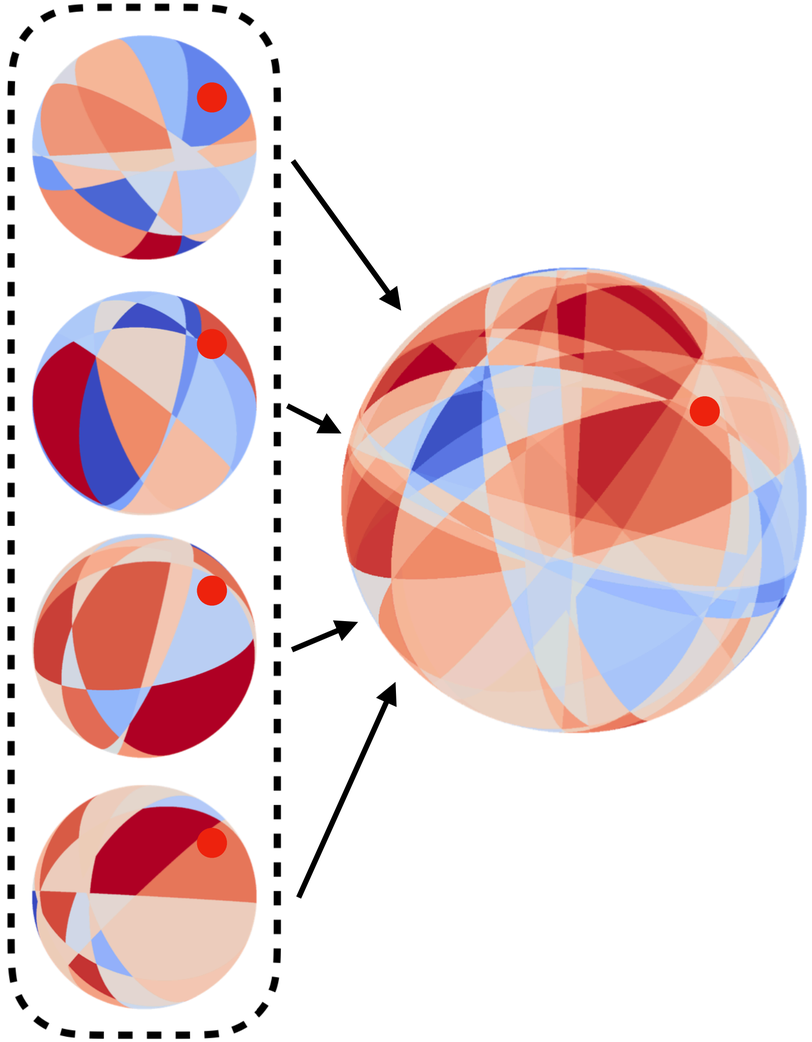


$$\frac{1}{h}\sum_{k=1}^{h}\frac{\exp(\langle z_k, [S]_{f_k(x)}\rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau}(\exp([z_k]_j)+\exp(-[z_k]_j))}$$

$$z_k = xR_k$$

Making FFN Efficient

LookupFFN: Connection to Vanilla FFN



$$\sum_{k=1}^{h} \frac{\exp(\langle z_k, [S]_{f_k(x)} \rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))} \qquad z_k = xR_k$$

Sigmoid Activation:

$$\sum_{k=1}^{h} \frac{\exp(z_k)[V]_k}{\exp(z_k) + \exp(-z_k)} \qquad z_k = 0.5x[W]_k$$

Fast Approx. of GELU:

$$\sum_{k=1}^{h} \frac{1.175 z_k \exp(z_k)[V]_k}{\exp(z_k) + \exp(-z_k} \qquad z_k = 0.851x[W]_k$$

Making FFN Efficient

Remaining Challenge: Efficient Projection

GEMM with complexity $\mathcal{O}(dh\tau)$ for all k $\quad z_k = xR_k$

$$\sum_{k=1}^{h} \frac{\exp(\langle z_k, [S]_{f_k(x)} \rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))}$$

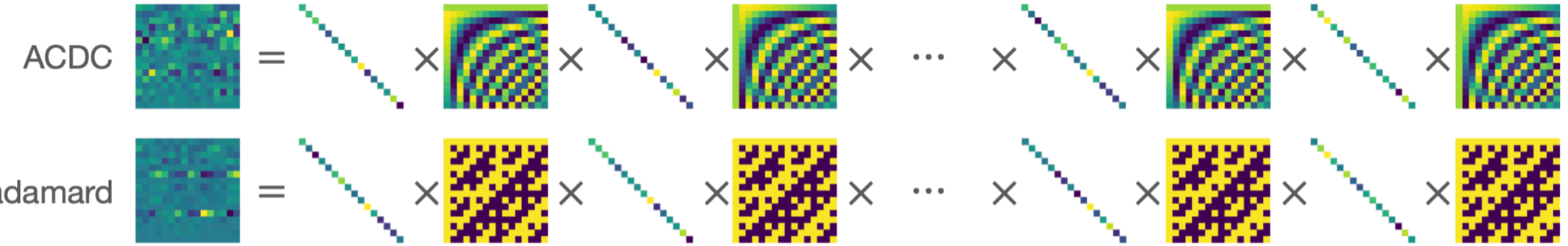
Making FFN Efficient

Remaining Challenge: Efficient Projection

GEMM with complexity $\mathcal{O}(dh\tau)$ for all k $\quad z_k = xR_k$

$$\sum_{k=1}^{h} \frac{\exp(\langle z_k, [S]_{f_k(x)} \rangle)[T_k]_{f_k(x)}}{\prod_{j=1}^{\tau} (\exp([z_k]_j) + \exp(-[z_k]_j))}$$

$$xD_1HD_2H \cdots D_{k-1}HD_kH$$



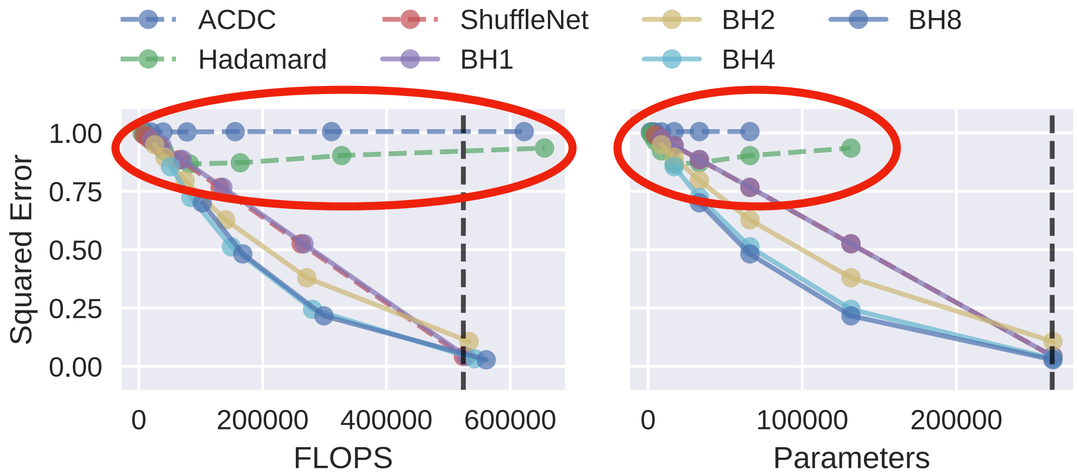
Moczulski, M., Denil, M., Appleyard, J., and de Freitas, N. ACDC: A structured efficient linear layer. In Ben- gio, Y. and LeCun, Y. (eds.), ICLR 2016

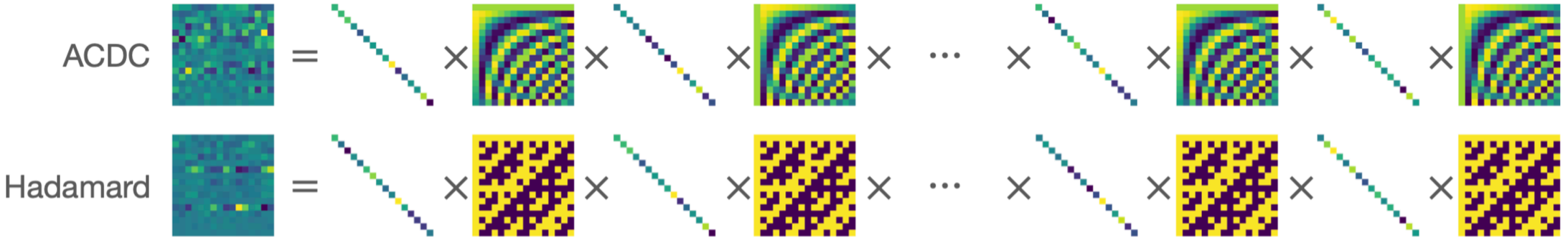
Making FFN Efficient

Remaining Challenge: Efficient Projection

GEMM with complexity $\mathcal{O}(dh\tau)$ for all k $\quad z_k = xR_k$

$$xD_1HD_2H\cdots D_{k-1}HD_kH$$





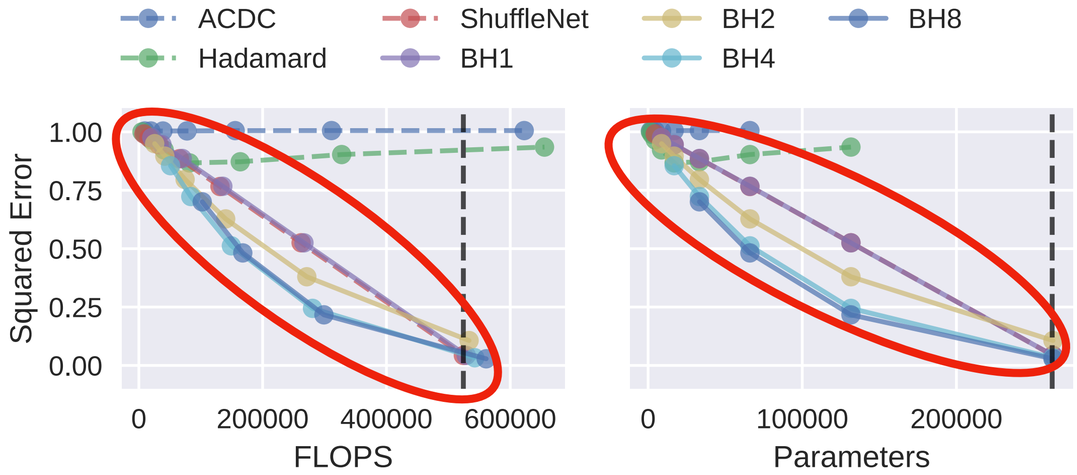
Making FFN Efficient

Efficient Projection: BH4

GEMM with complexity $\mathcal{O}(dh\tau)$ for all k $\quad z_k = xR_k$

$$xD_1HD_2H\cdots D_{k-1}HD_kH$$

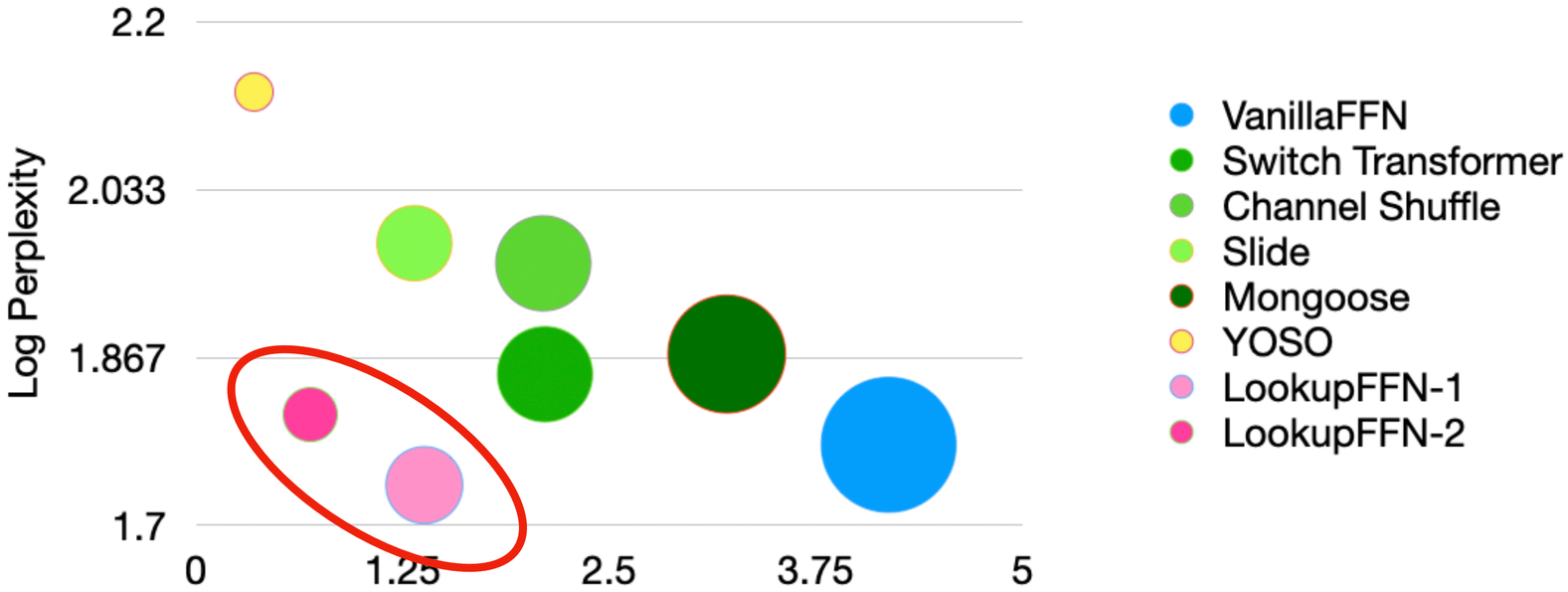
$$xB_1HB_2HB_3HB_4H$$



Evaluation

Baseline Comparison

RoBERTa pretraining (small size and base size)

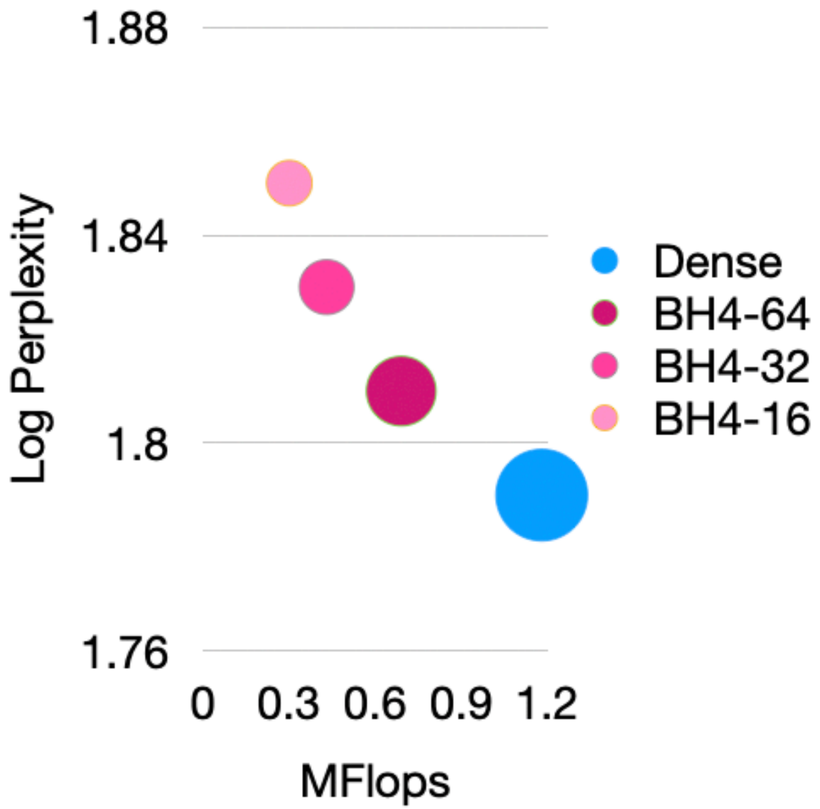


Method	h	τ	MFLOP	Log Perplexity
VanillaFFN	-	-	9.44	1.37
LookupFFN	170	9	1.39	1.41

Table 2. Log perplexity when scaling to a RoBERTa-base model.

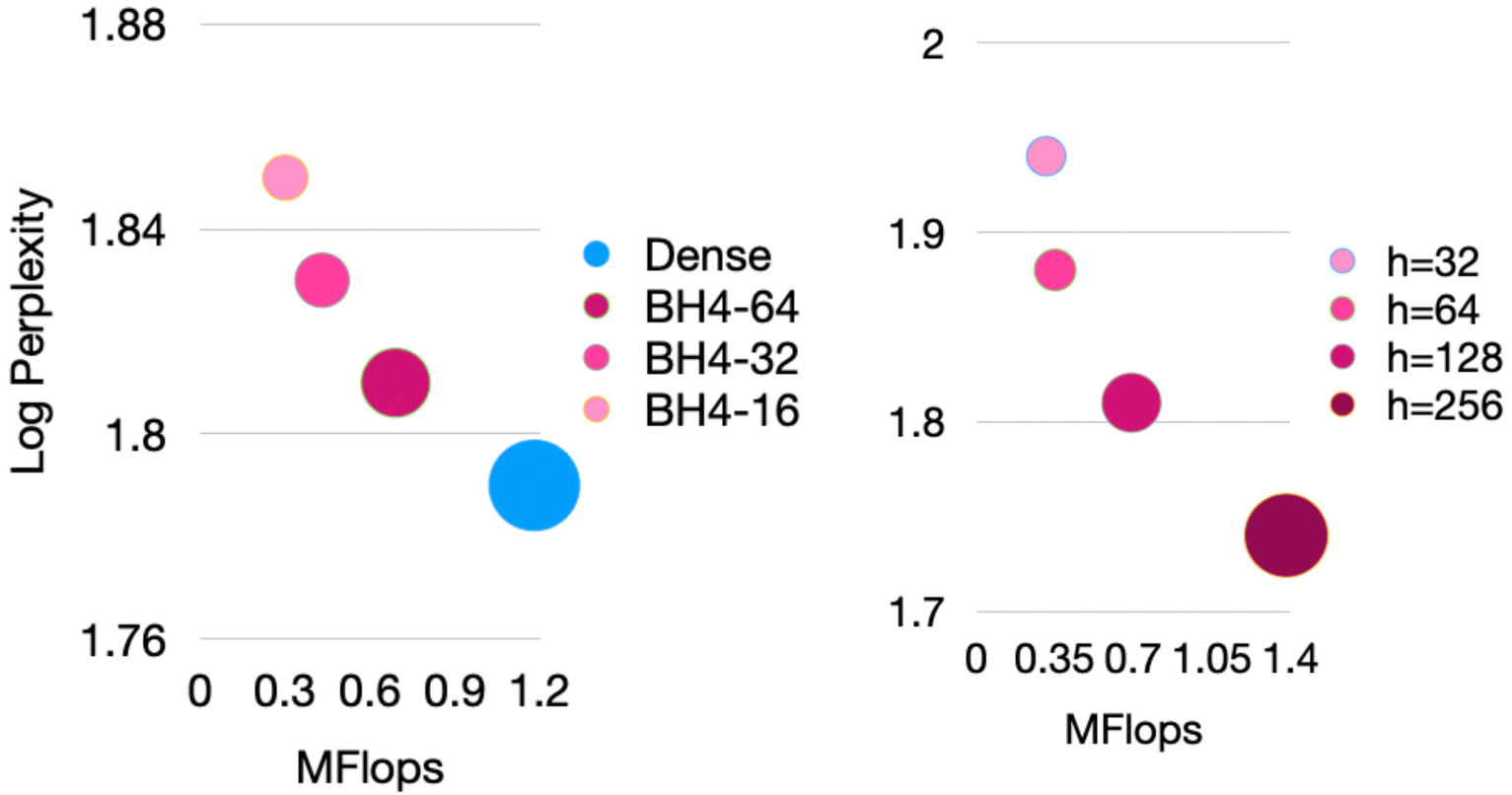
Comparing Different Configurations

RoBERTa pretraining (small size)



Comparing Different Configurations

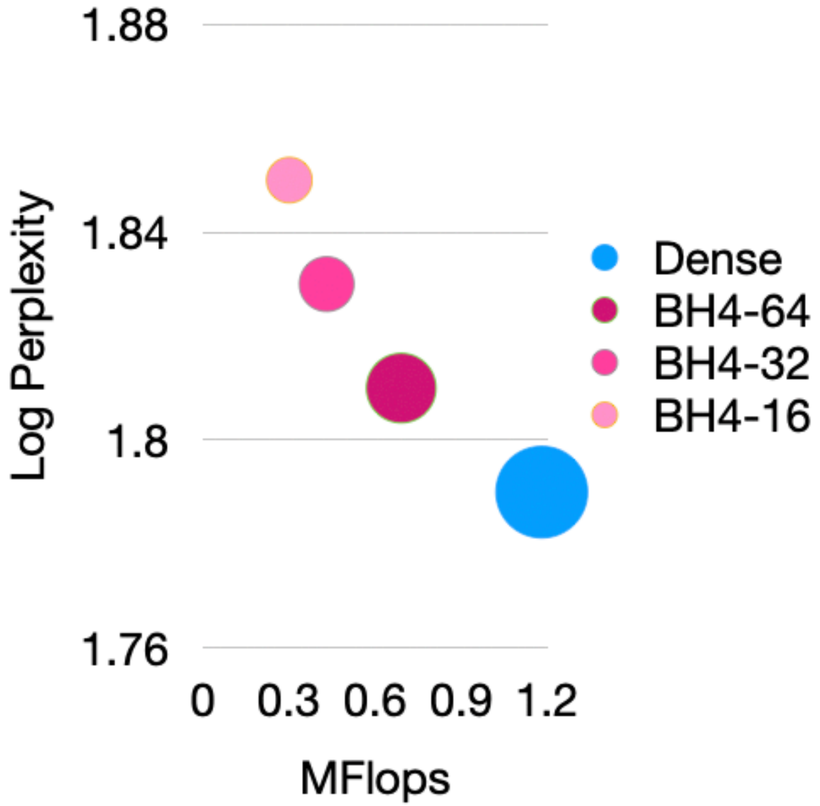
RoBERTa pretraining (small size)

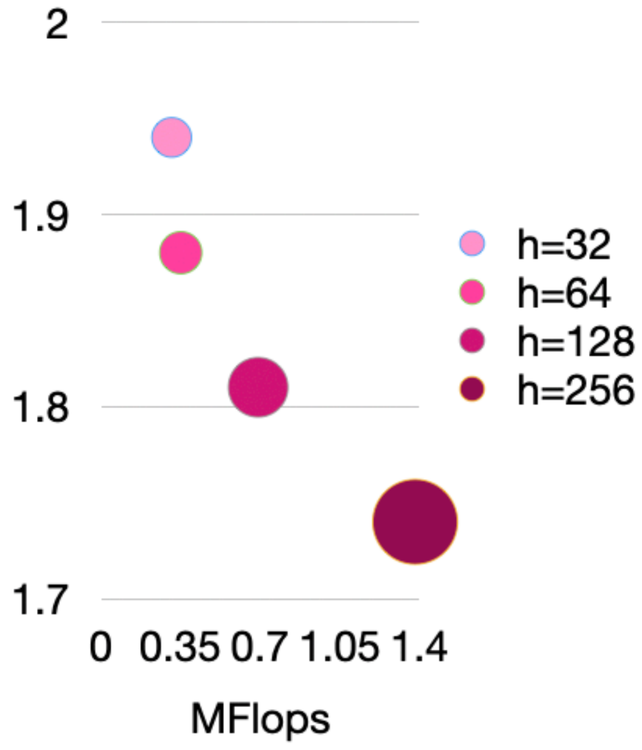


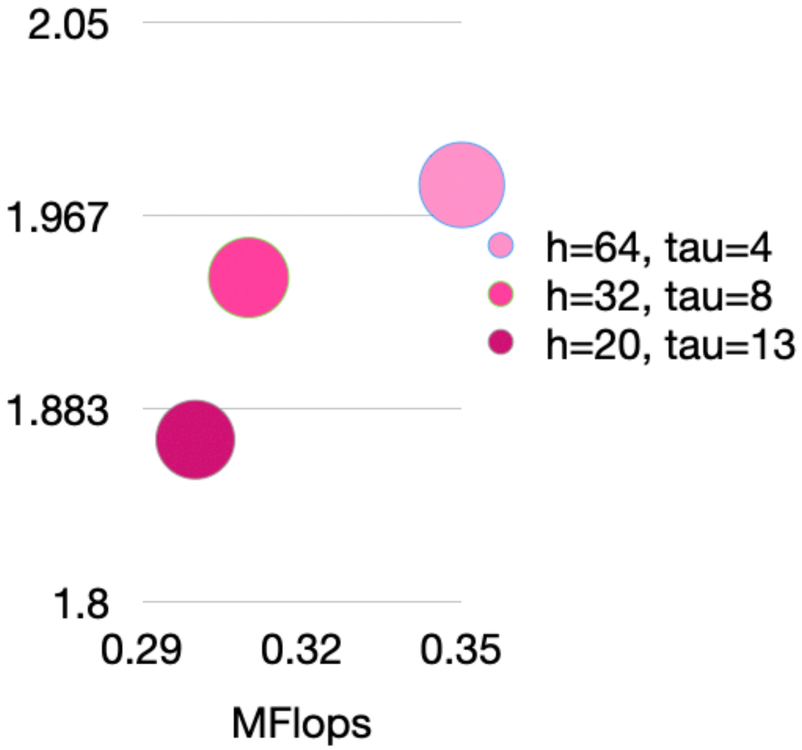
Comparing Different Configurations

RoBERTa pretraining (small size)

WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON



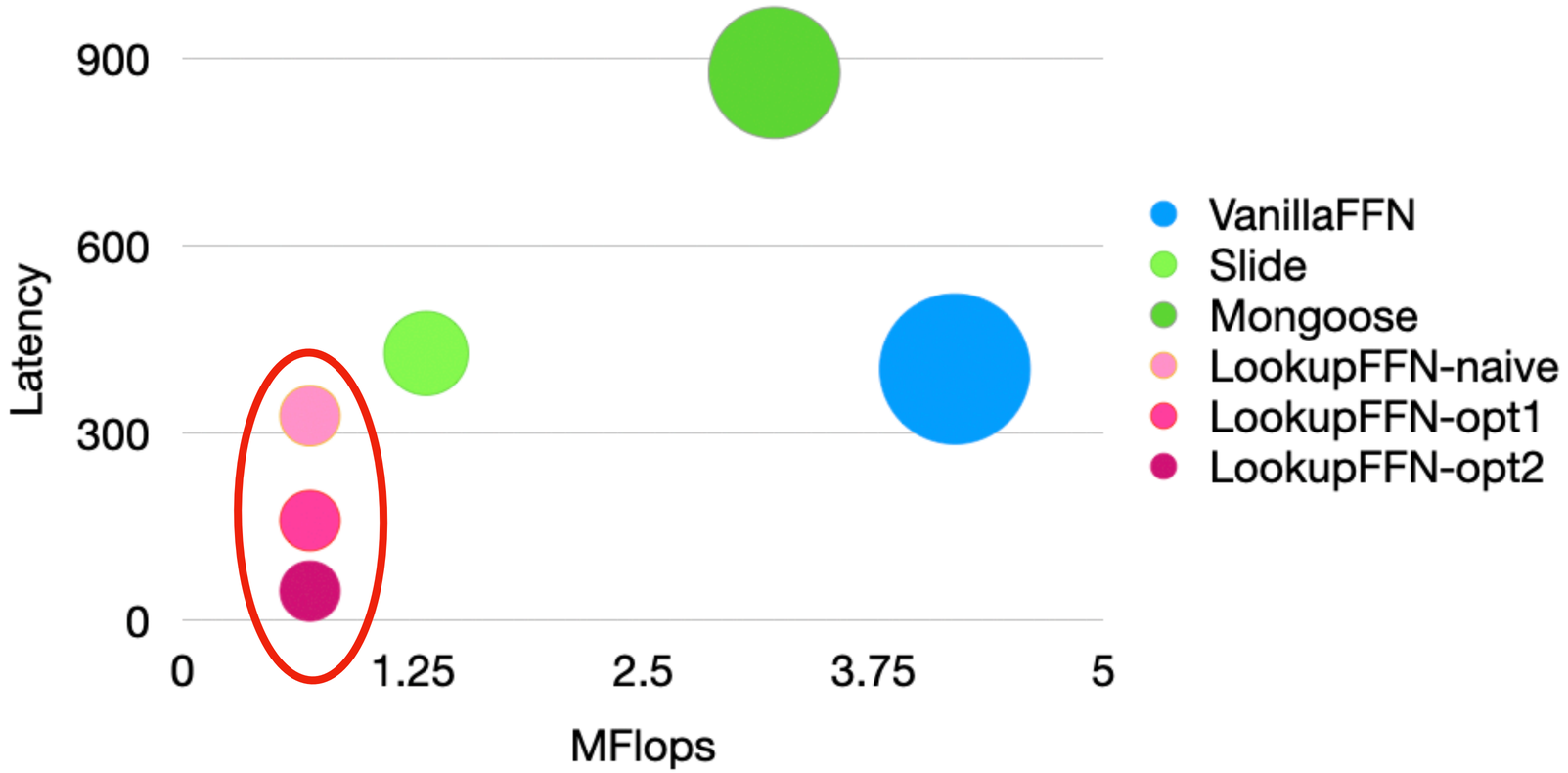




CPU Latency Comparison

RoBERTa pretraining (small size)

WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON



Takeaway

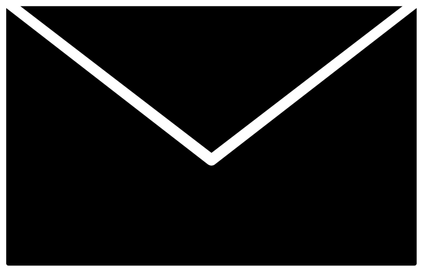
LookupFFN recasts FFNs using efficient LSH strategies.

When married with LSH based approximations of self-attention, GEMM operations within Transformer models can be nearly eliminated.

The resultant compute-lite model can allow deployment on CPU-based platforms.

Many other software/hardware opportunities exist: we have only scratched the surface.

mlpen/LookupFFN



zzeng38@wisc.edu

End