Fermilab

ML Part 4: Permutation Invariance

Abhijith Gandrakota

CODAS-HEP 2023

Princeton University, NJ

Lecture adapted from J. Ngadiuba's and M. Kagan's courses

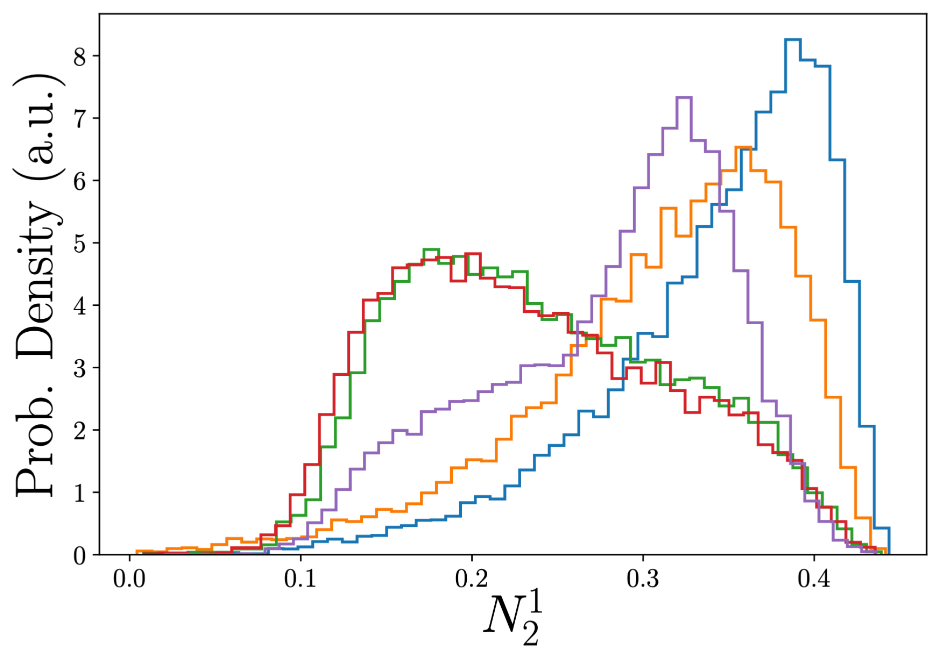
Range of ML Algorithms

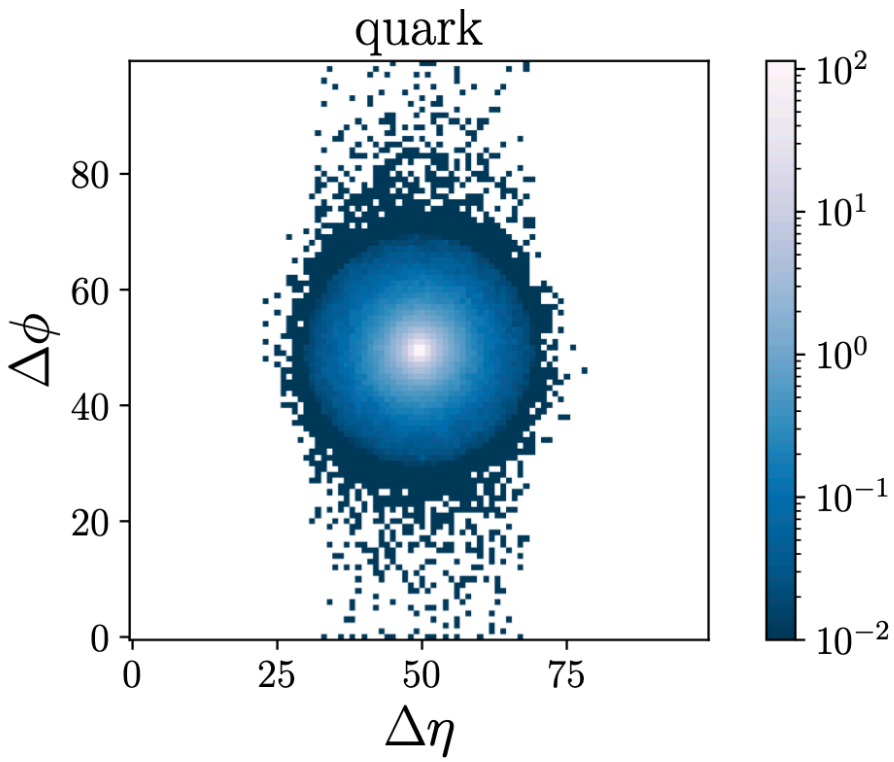
Linear regression

Transformers
Graphs, etc . . . .

Finally here !

So far we had





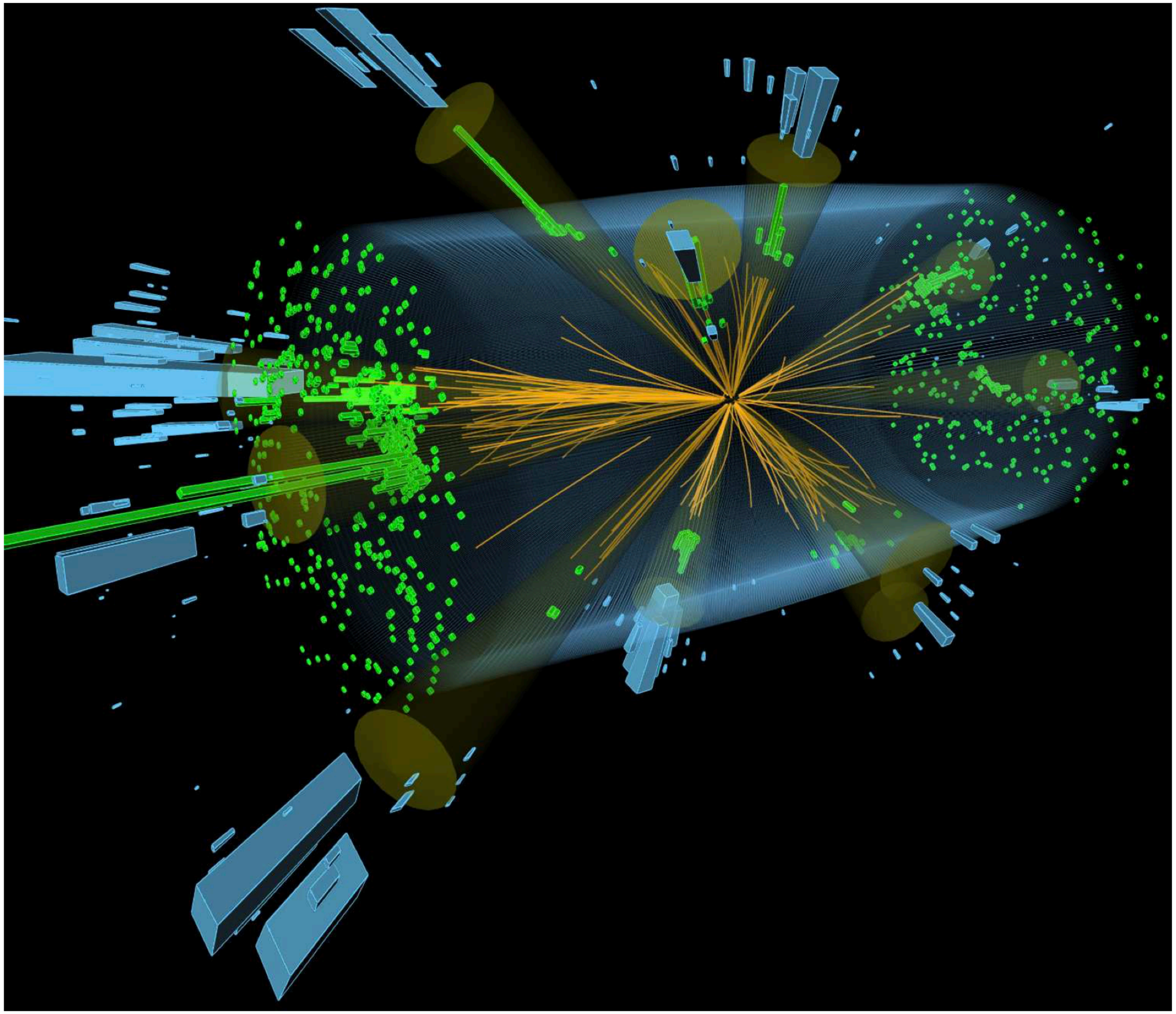
High-Level / Analytical features

- Best to use MLP
  - Lightweight and faster
- Typically uses engineered features
  - Miss out local structures / patterns
- Might not give best of the performance

Images

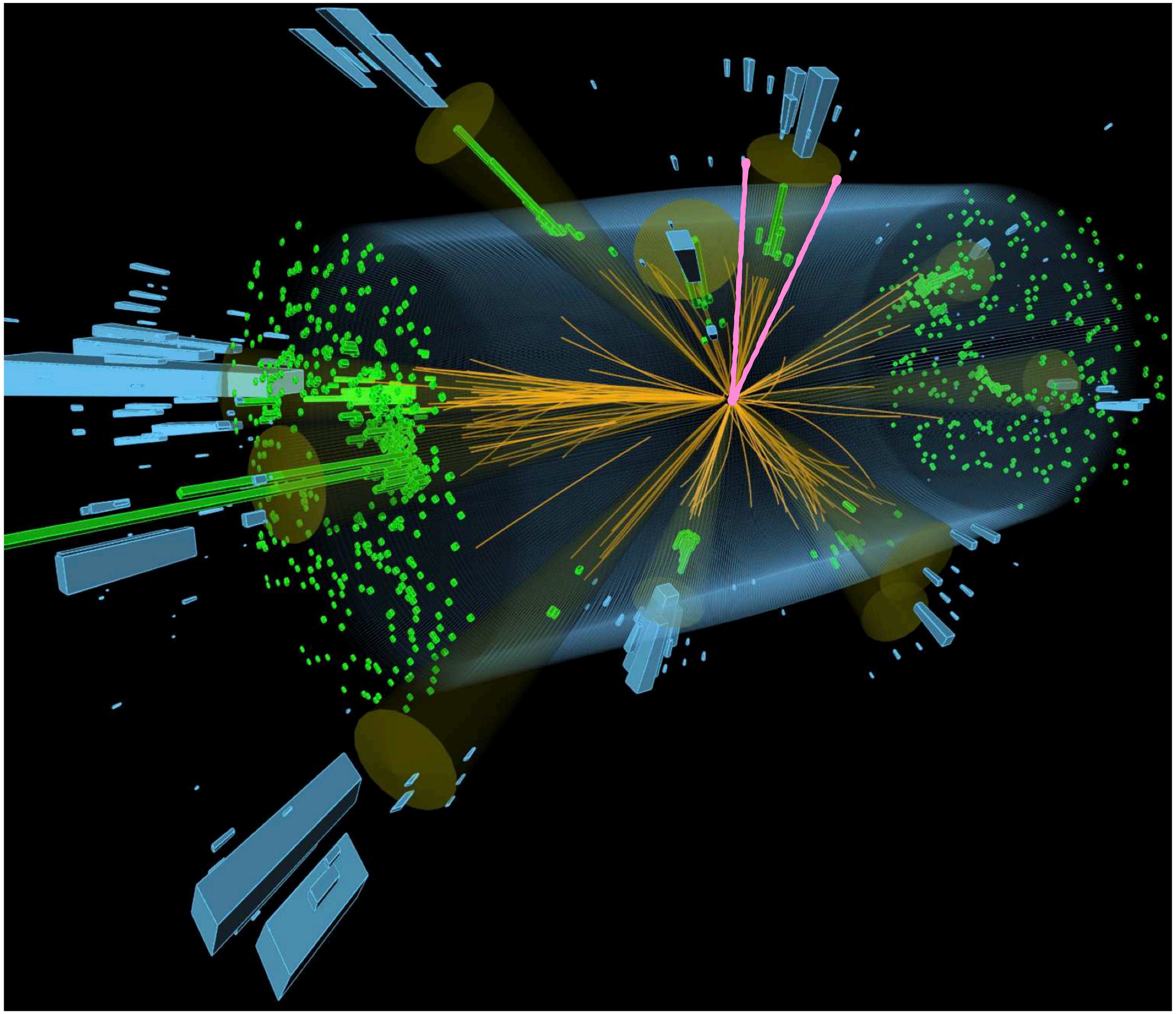
- Data from calorimeters / telescopes
- Best to use CNNs
  - Translational invariance
- Information contained is ~ 2D
  - Image in HEP are very sparse

A Collision event at LHC



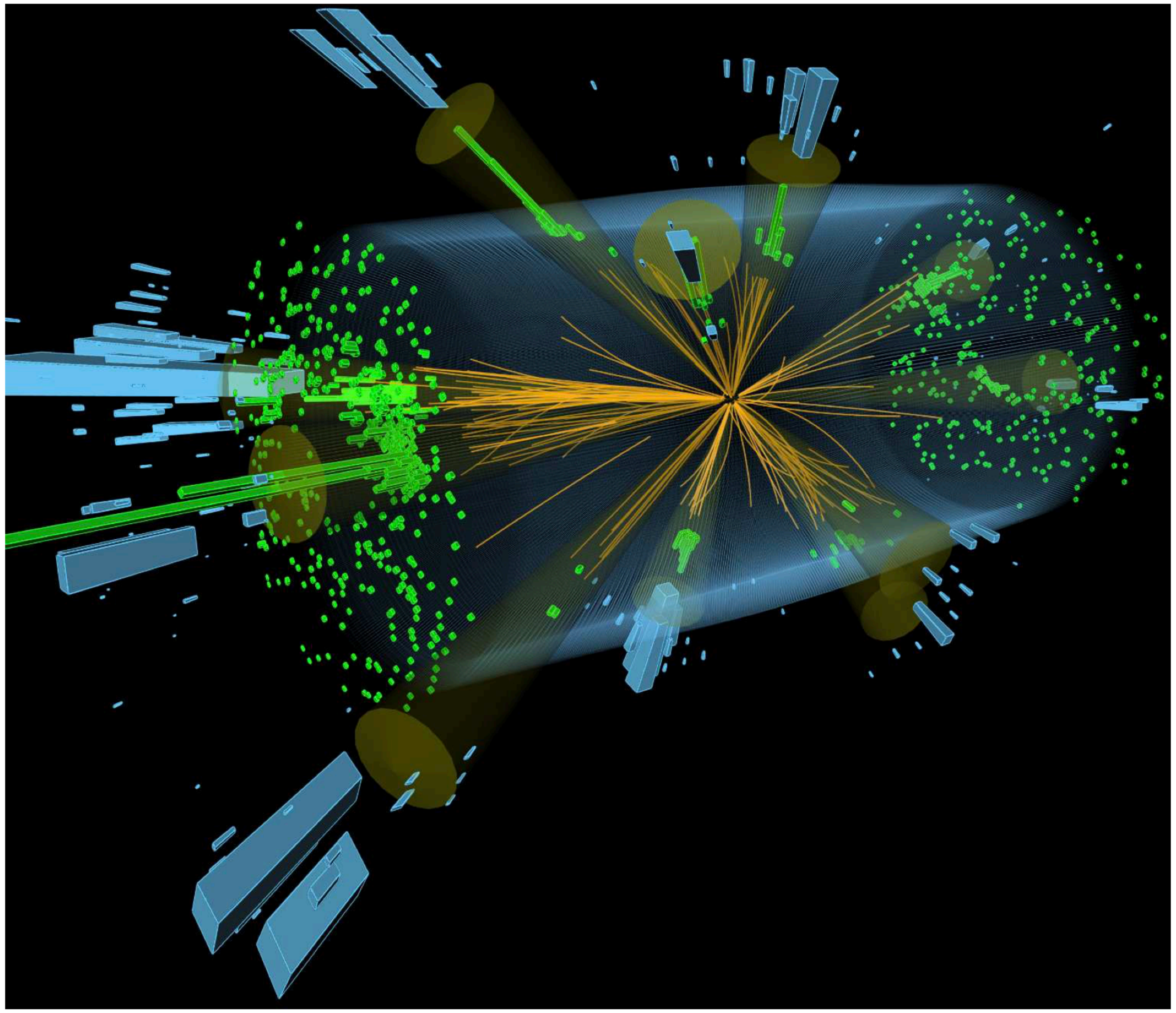
A Collision event at LHC

- High granularity particle tracks; Sparse detector images



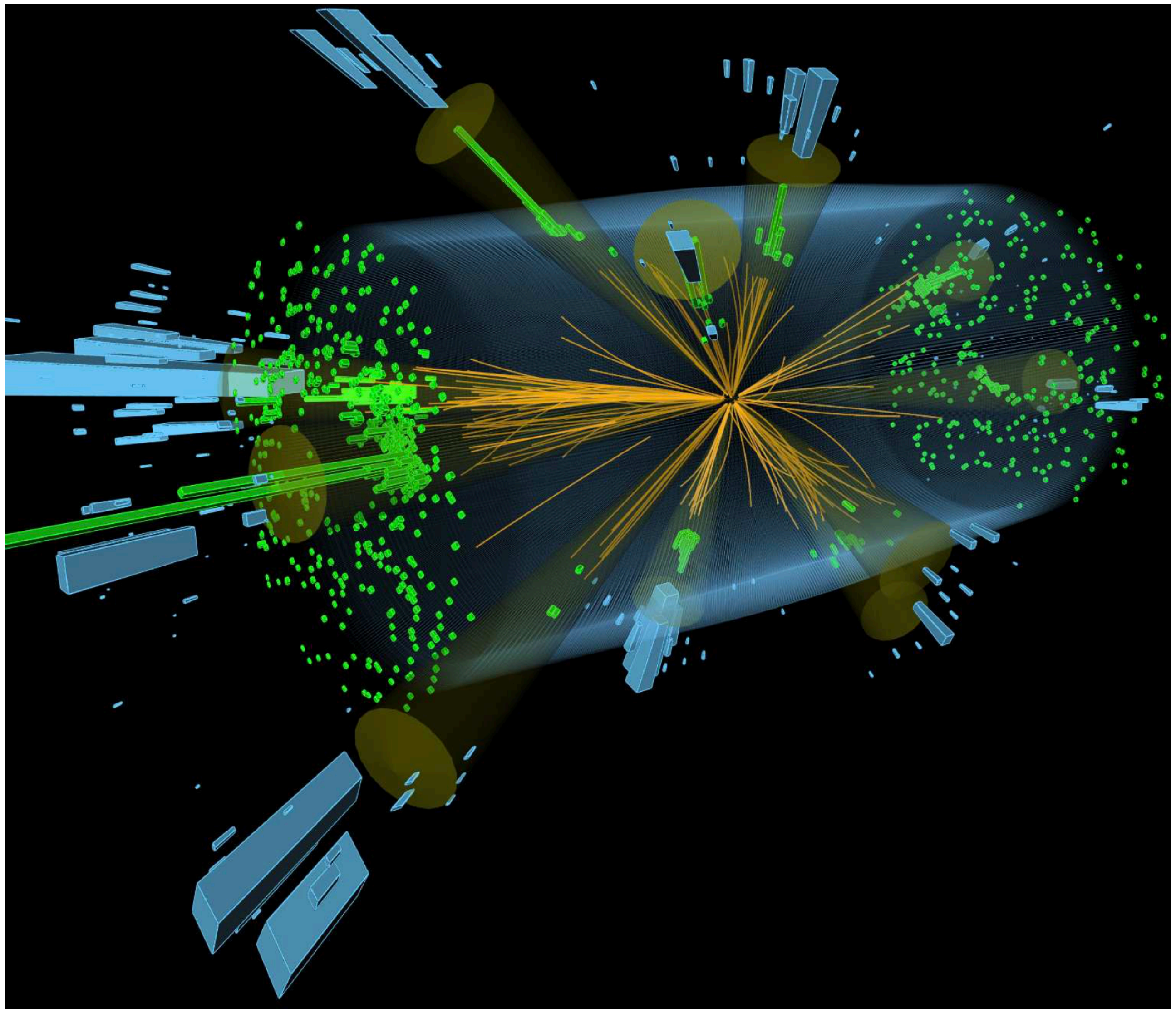
A Collision event at LHC

- High granularity particle tracks; Sparse detector images
- How do we tackle this data structure ?



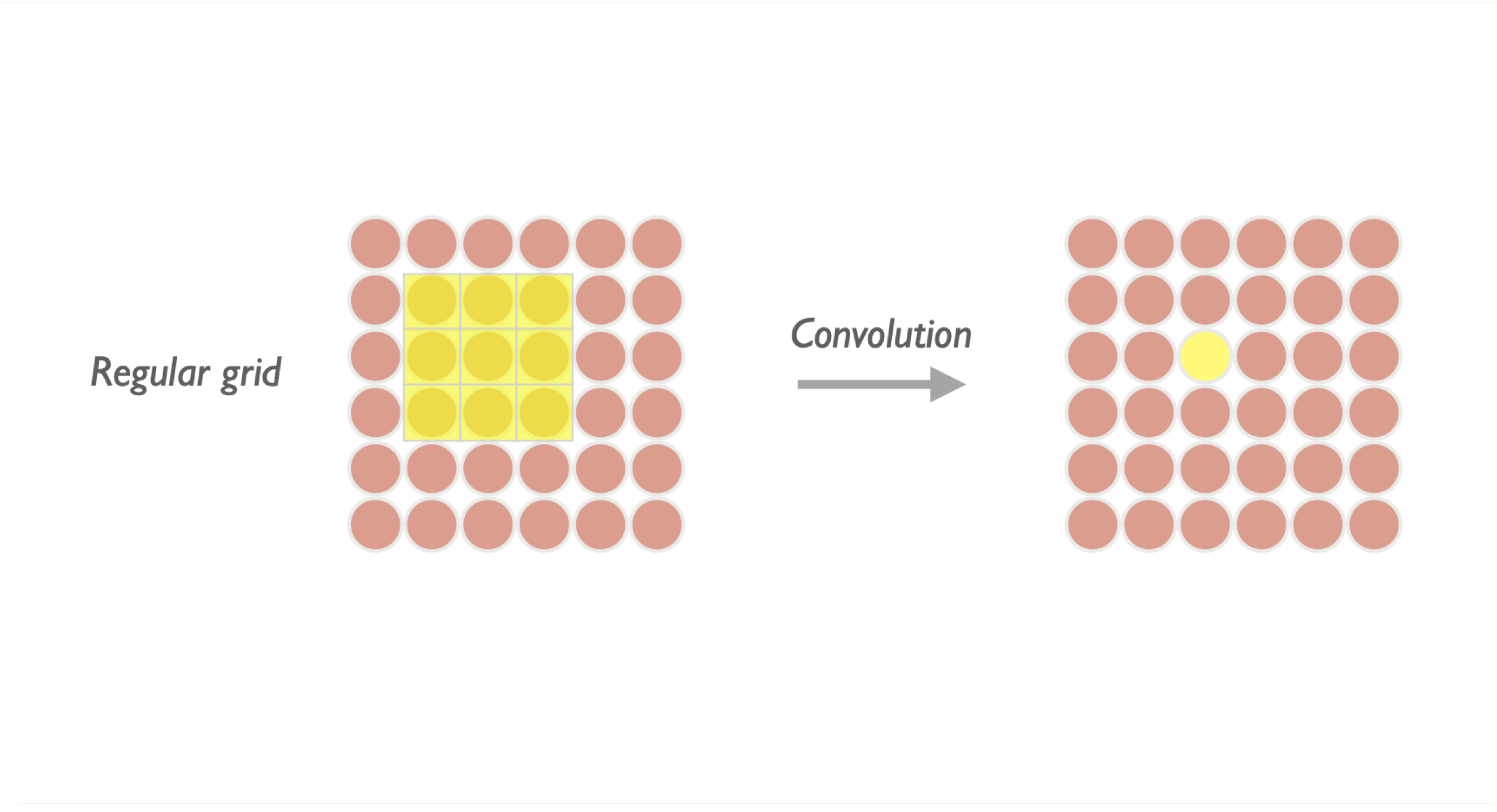
A Collision event at LHC

- High granularity particle tracks; Sparse detector images
- How do we tackle this data structure ?
- Feed it to CNN ?
  - How about encoding momentum ?
  - We will loose granularity of tracks
  - Detector images mostly contain 0s



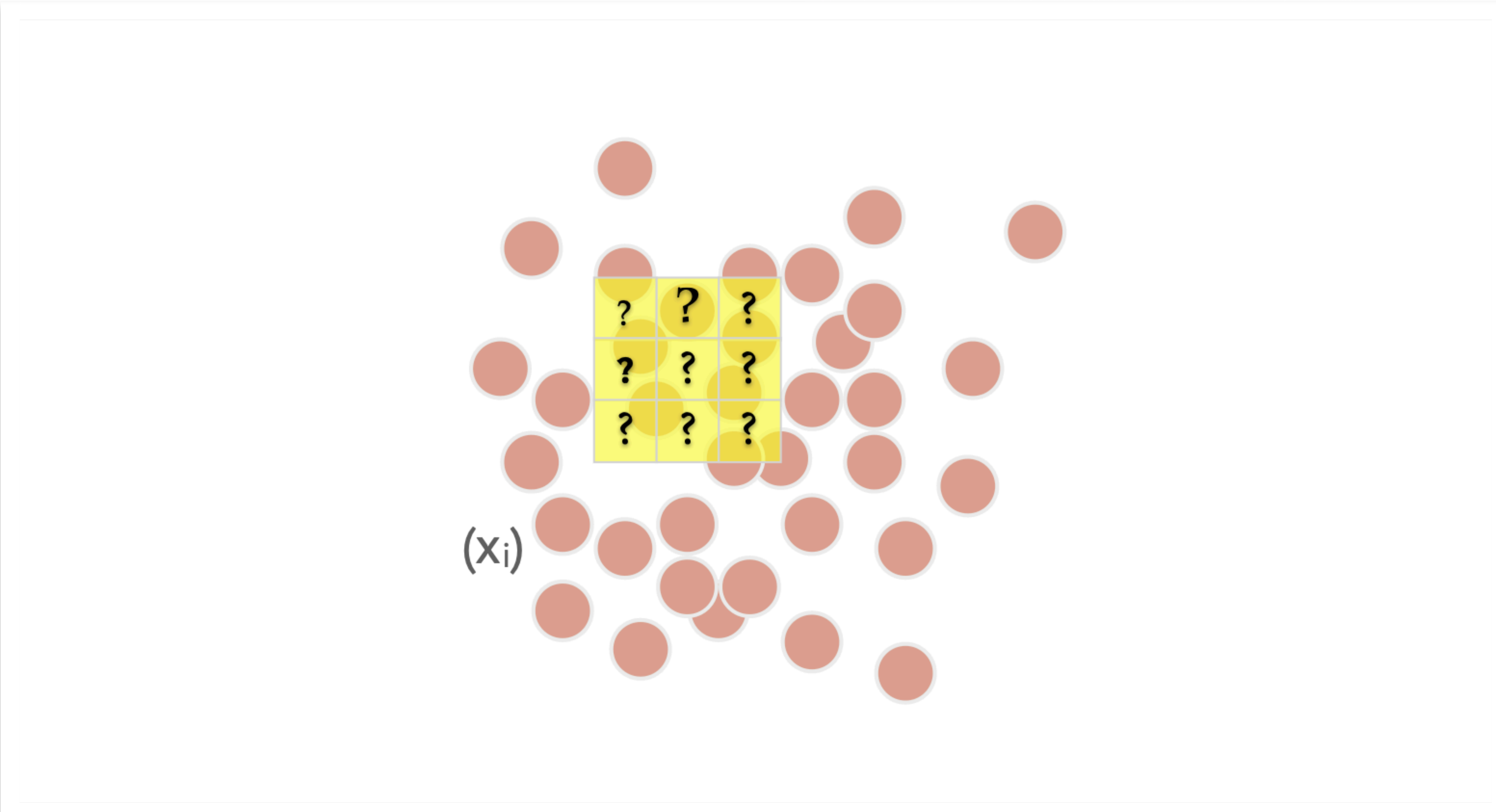
A Collision event at LHC

- High granularity particle tracks; Sparse detector images



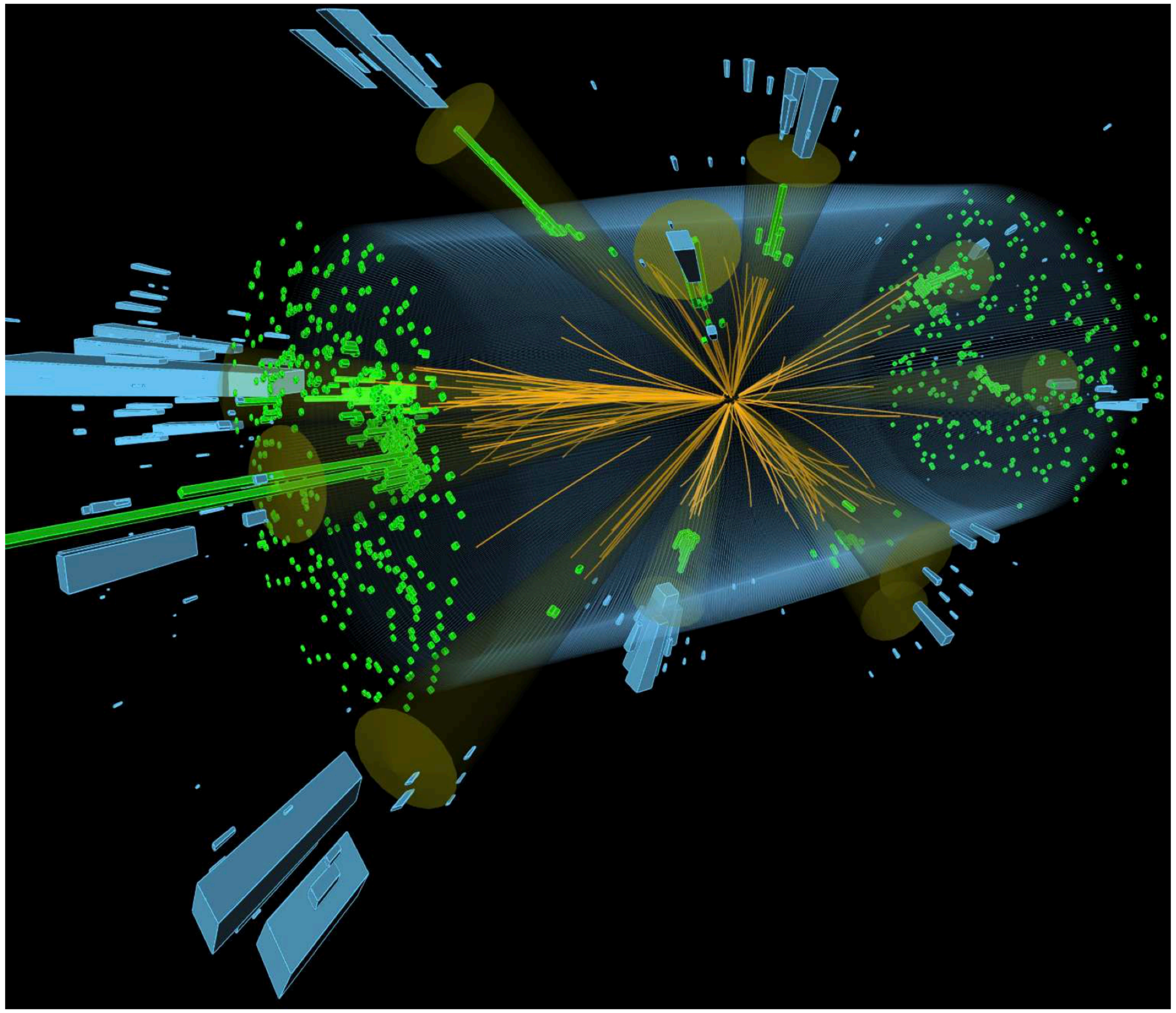
A Collision event at LHC

- High granularity particle tracks; Sparse detector images



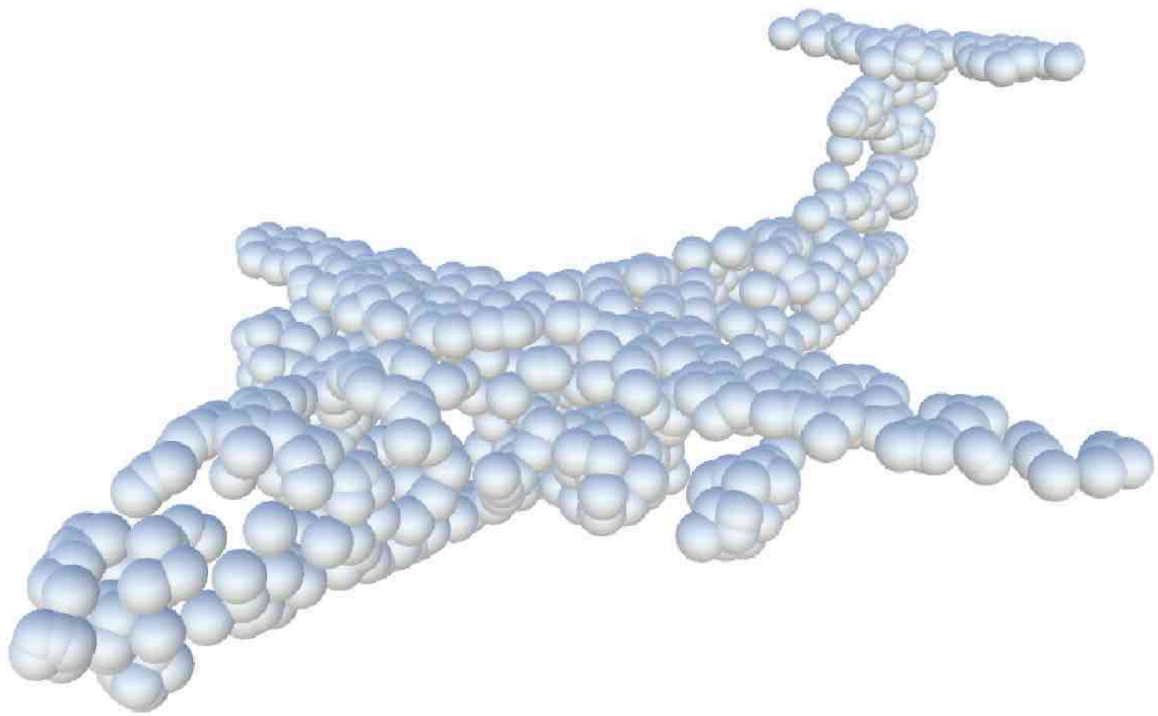
A Collision event at LHC

- High granularity particle tracks; Sparse detector images
- How do we tackle this data structure ?
- Feed it to CNN ?
  - How about encoding momentum ?
  - We will loose granularity of tracks
  - Detector images mostly contain 0s
- May be a MLP ?
  - Flatten all particle momentum, feed it as input ?
  - Which particles goes first ? Why ?
  - Huge number of parameters for first layer

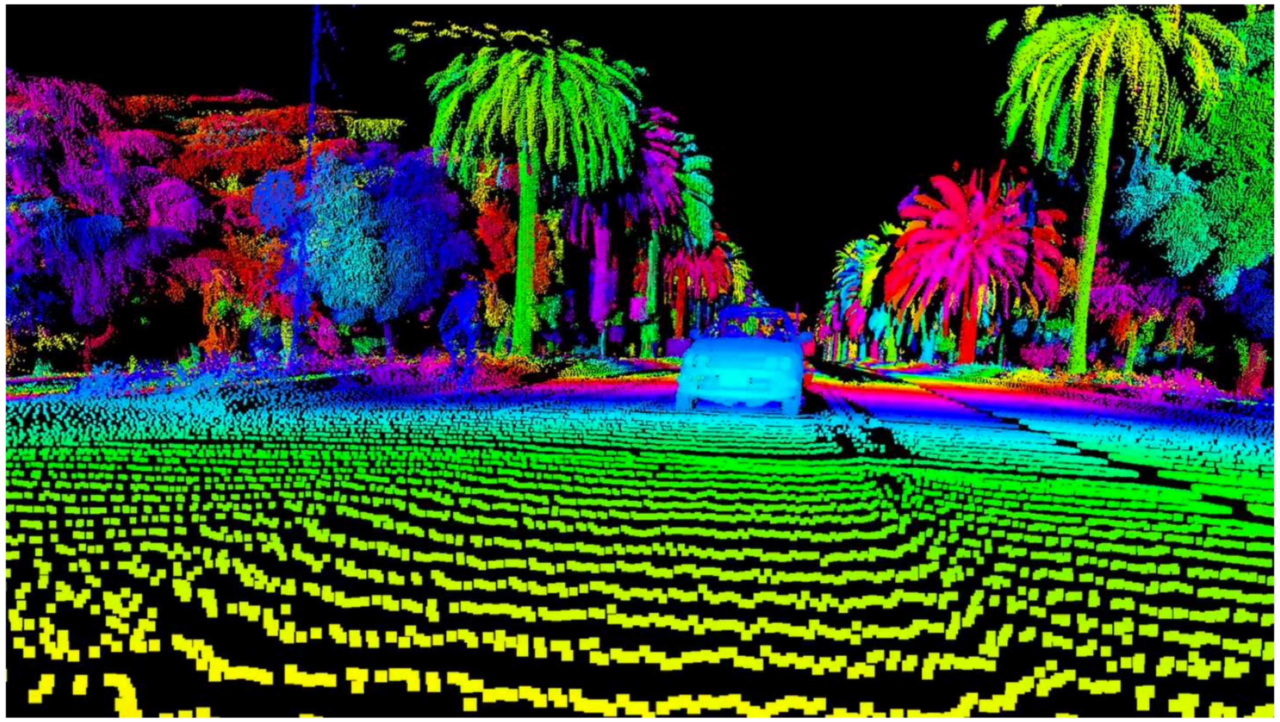


Point clouds

- Describes 3D objects and used in tasks such as self-driving w/ LIDAR



LIDAR data from self-driving car sensor



Point cloud

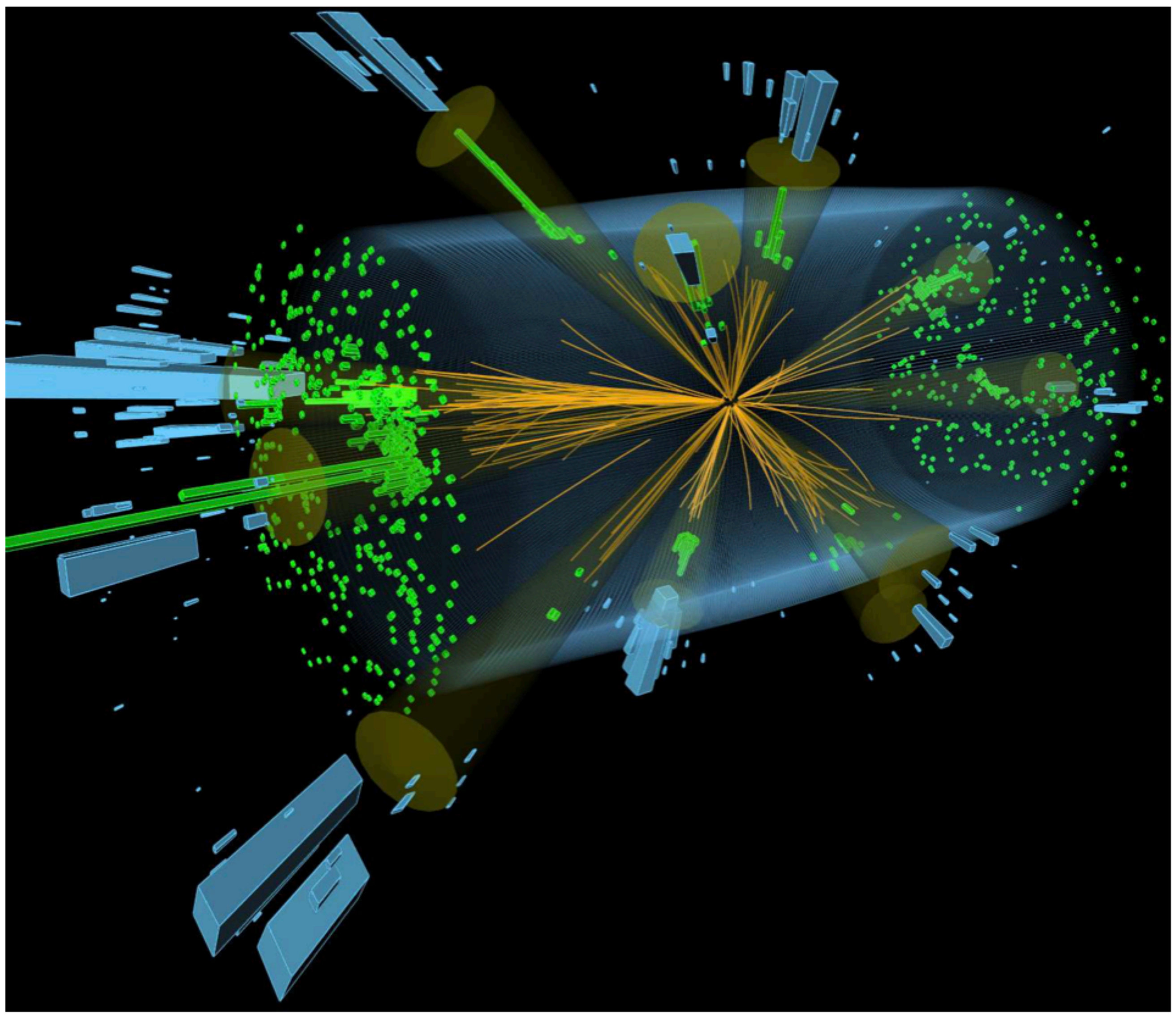
From Wikipedia, the free encyclopedia

A **point cloud** is a set of data points in space.

Particle Cloud

Point cloud: "A set of data points in space" –Wikipedia

Jet/event　　Particles　　Feature



Jet (Particle cloud)

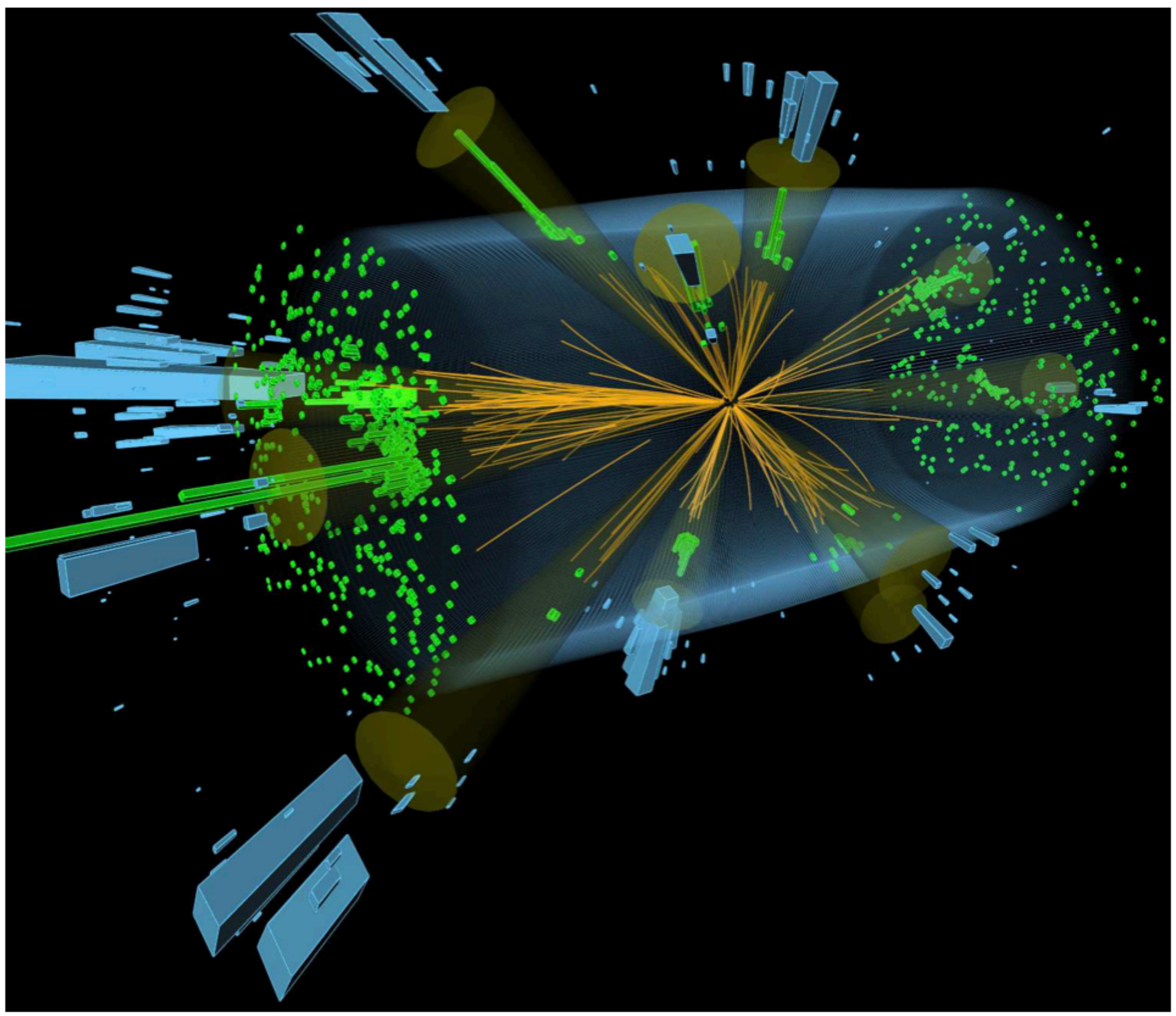
From Wikipedia, the free encyclopedia

A **jet (particle cloud)** is a set of particles in space. Particle clouds are generally created by clustering a large number of particles measured by particle detectors, e.g., ATLAS EXPERIMENT and CMS.

Particle Cloud

Point cloud: "A set of data points in space" –Wikipedia

Jet/event Particles Feature



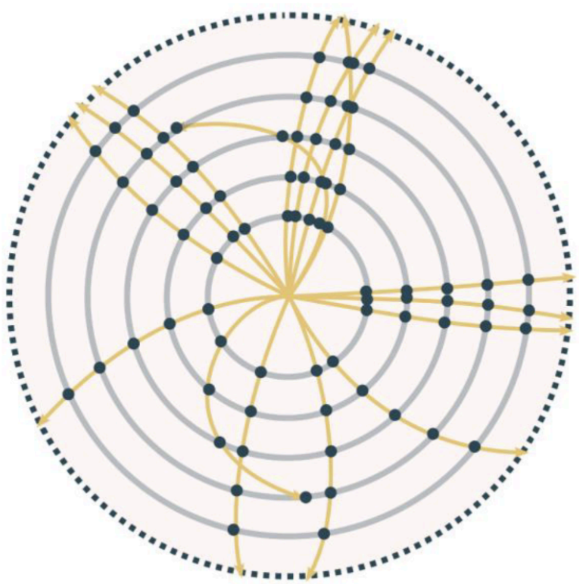
Jet (Particle cloud)

From Wikipedia, the free encyclopedia

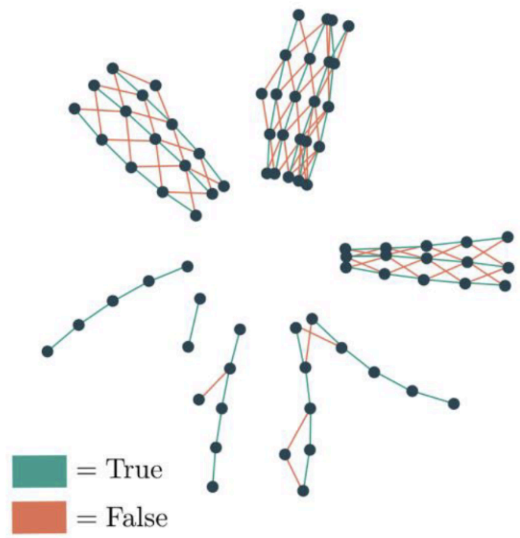
A **jet (particle cloud)** is a set of particles in space. Particle clouds are generally created by clustering a large number of particles measured by particle detectors, e.g., ATLAS EXPERIMENT and .

- particles are intrinsically unordered
- primary information:
  - 2D coordinates in the η-ϕ space
- Also additional "features":
  - energy/momenta
  - charge/particle type
  - track quality/ impact parameters/etc.

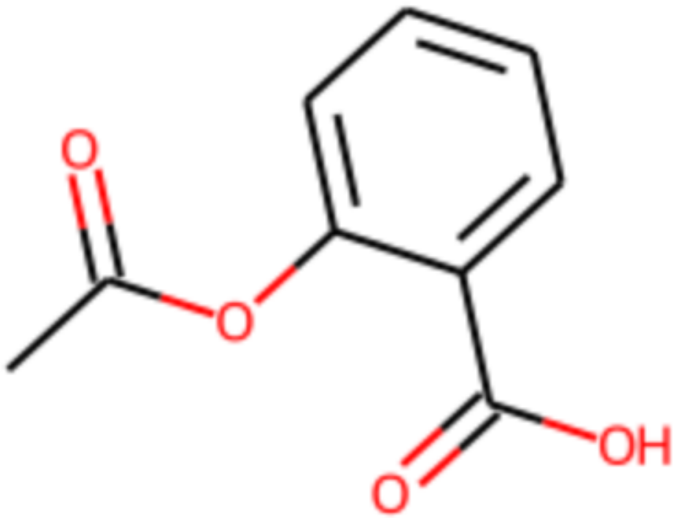
Beyond MLP and CNN: Graphs



Input data is a 3D *point cloud*

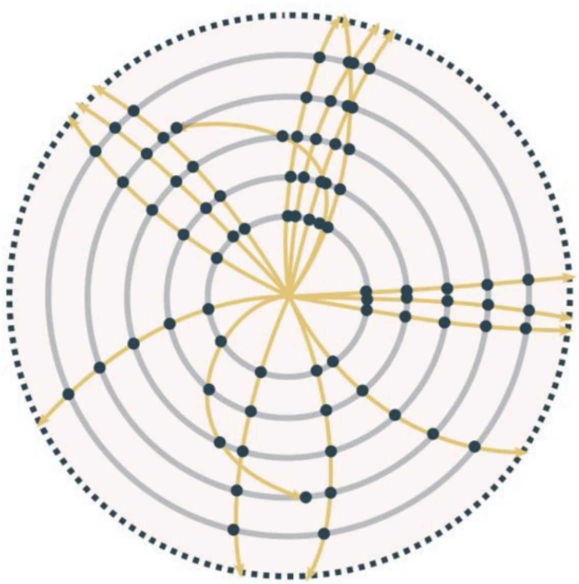


Train on graphs with edge truth labels

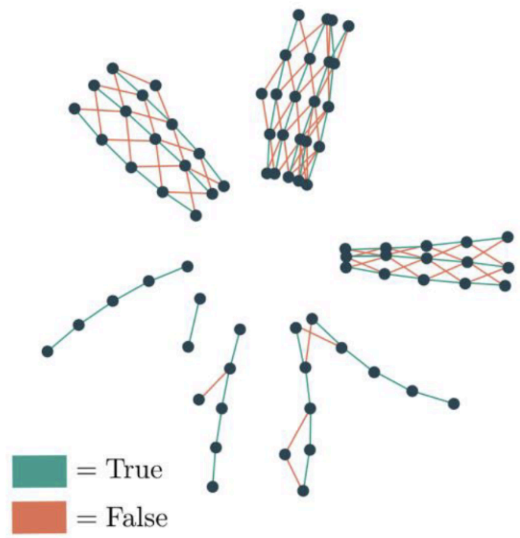


- Input has a inherent structure
  - Trajectory of particles
  - Inherent symmetry in data
  - Etc . . .
- Example: Molecular structure
  - Used in classification
  - Drug Molecule Generation
  - Etc . . .

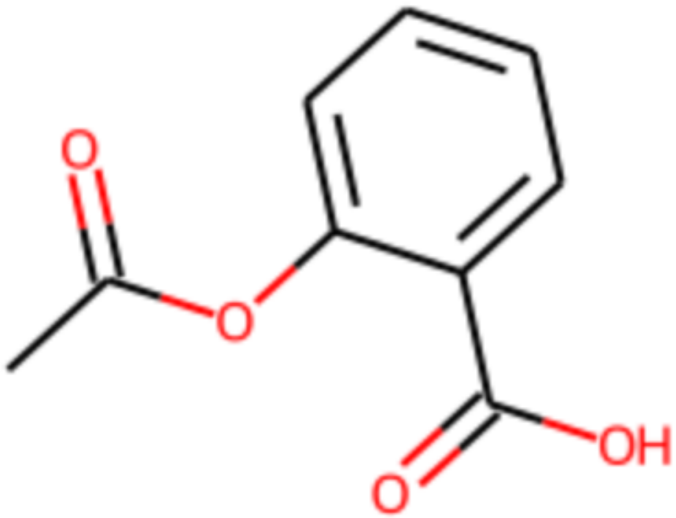
Beyond MLP and CNN: Graphs



Input data is a 3D *point cloud*



Train on graphs with edge truth labels

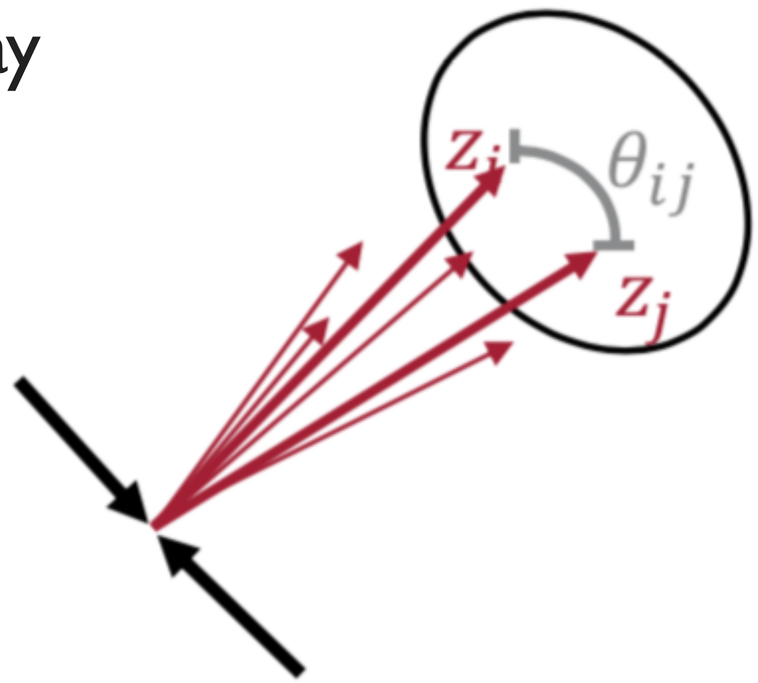


- Input has a inherent structure
  - Trajectory of particles
  - Inherent symmetry in data
  - Etc . . .
- Example: Molecular structure
  - Used in classification
  - Drug Molecule Generation
  - Etc . . .

We will not go into much details about Graph Neural Networks

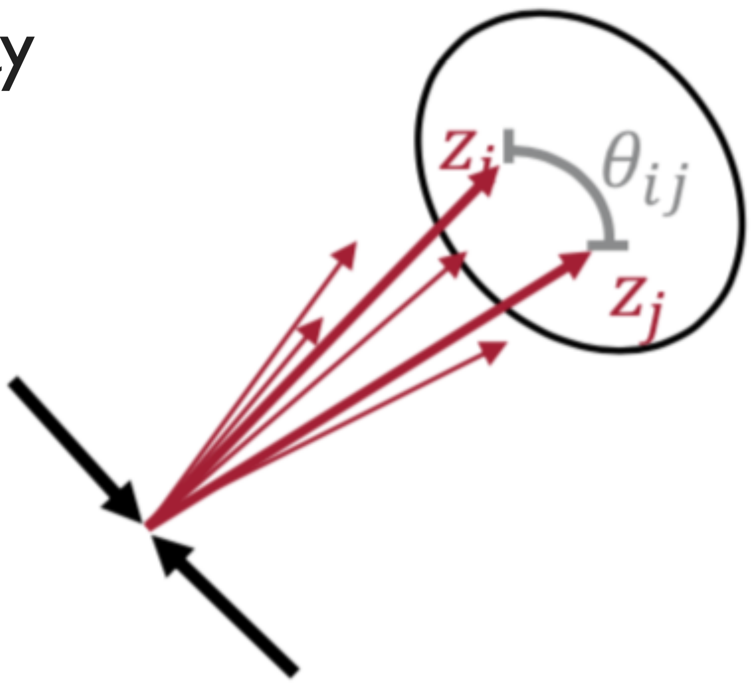
Permutation *Invariance*

- Lets say our input is set of particles (points) from decay
    - Each Input, $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
    - Where $X_i = \{P_\mu,$ particle type (PID), Other features (d_0, d_z) $\}$
- Input Structure: $N_{jets} \times (N_{particles} \times N_{features})$



Permutation *Invariance*

- Lets say our input is set of particles (points) from decay

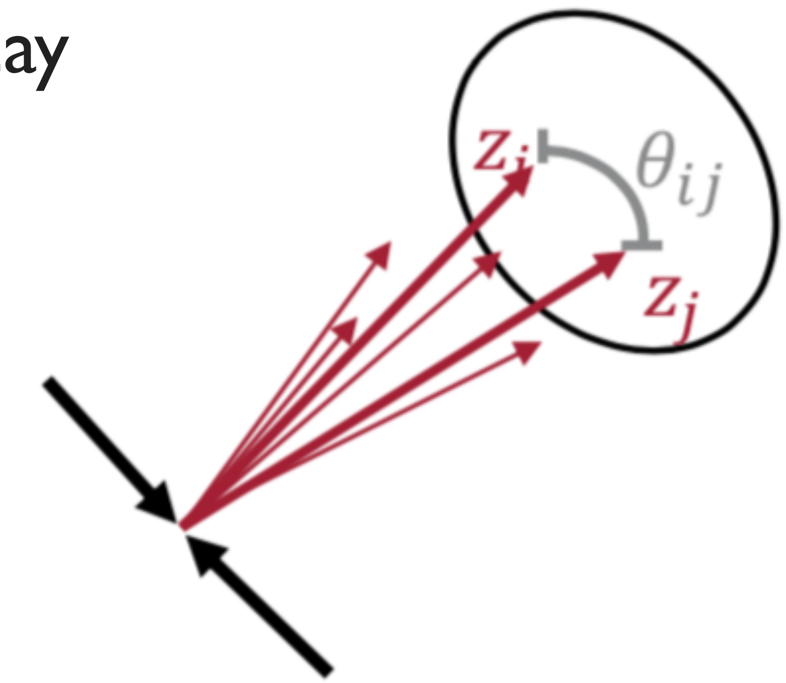


  - Each Input, $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu$, particle type (PID), Other features (d_0, d_z) $\}$

- Input Structure: $N_{jets} \times (N_{particles} \times N_{features})$

Permutation *Invariance*

- Lets say our input is set of particles (points) from decay



  - Each Input is $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu,$ particle type (PID), Other features (d_0, d_z) $\}$

- $$\mathbb{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$$

OR

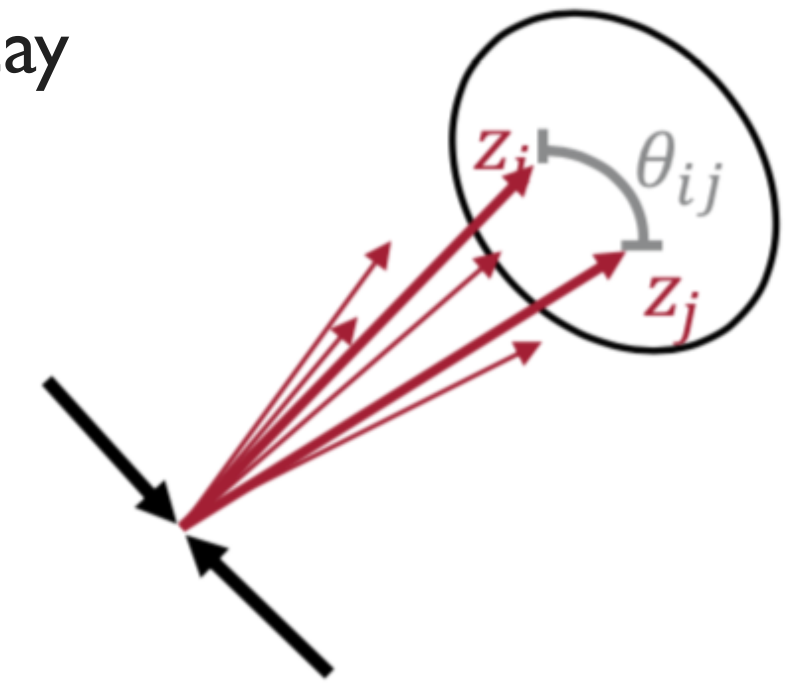
$$\mathrm{P}\mathbb{X} = \begin{bmatrix} X_5 \\ X_{n-1} \\ \vdots \\ X_1 \end{bmatrix}$$

Neural Network

$\mathbb{Y}$ Output

Permutation *Invariance*

- Lets say our input is set of particles (points) from decay



  - Each Input is $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu,$ particle type (PID), Other features (d_0, d_z) $\}$

- $$\mathbb{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$$

OR

$$\mathrm{P}\mathbb{X} = \begin{bmatrix} X_5 \\ X_{n-1} \\ \vdots \\ X_1 \end{bmatrix}$$

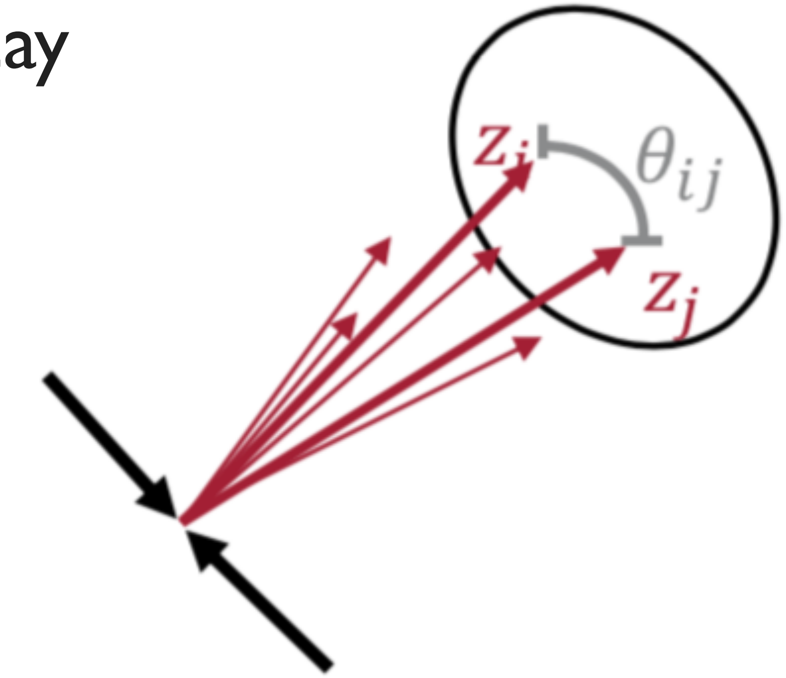
Neural Network

$\longrightarrow \mathbb{Y}$

Same Output

Permutation *Invariance*

- Lets say our input is set of particles (points) from decay



  - Each Input, $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu,$ particle type (PID),
    Other features (d_0, d_z) $\}$

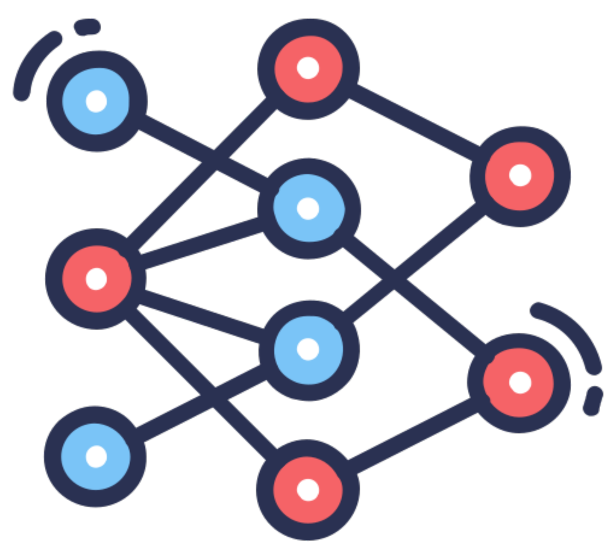
- We need NN: $f \mid f(\mathrm{P}\mathbb{X}) = f(\mathbb{X})$
  - MLPs need ordering and are out of question

Flower 1

Length

n Petals

Color

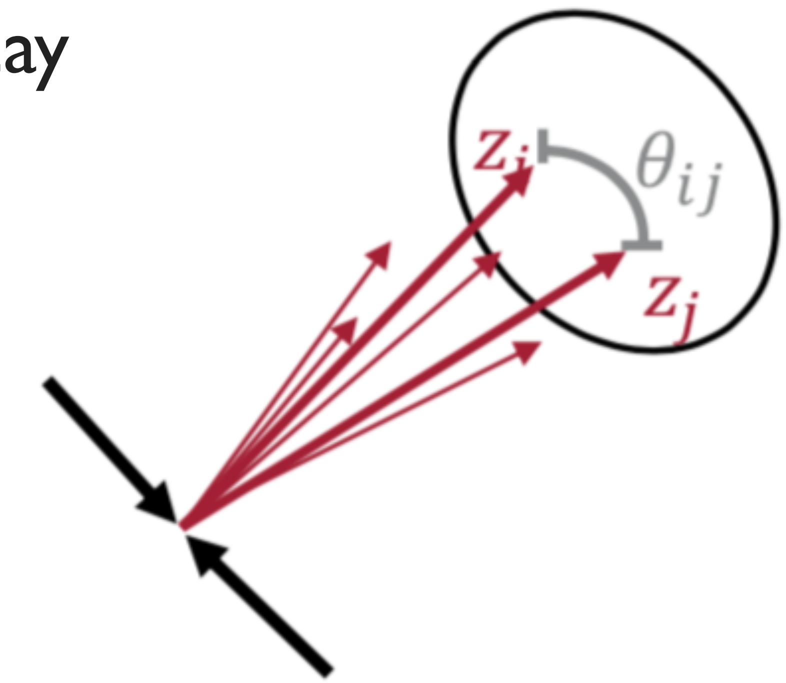


$\mathbb{Y}$

Permutation *Invariance*

Fermilab

- Lets say our input is set of particles (points) from decay



  - Each Input, $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu,$ particle type (PID),
    Other features (d_0, d_z) $\}$

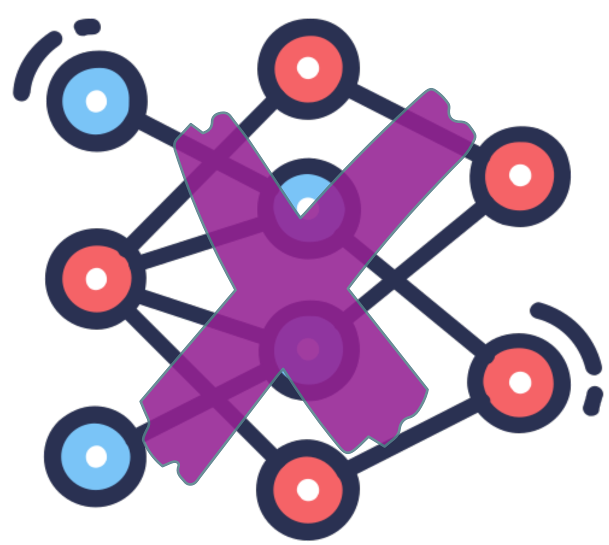
- We need NN: $f \mid \; f(\mathrm{P}\mathbb{X}) = f(\mathbb{X})$
  - MLPs need ordering and are out of question

Flower 2

Length

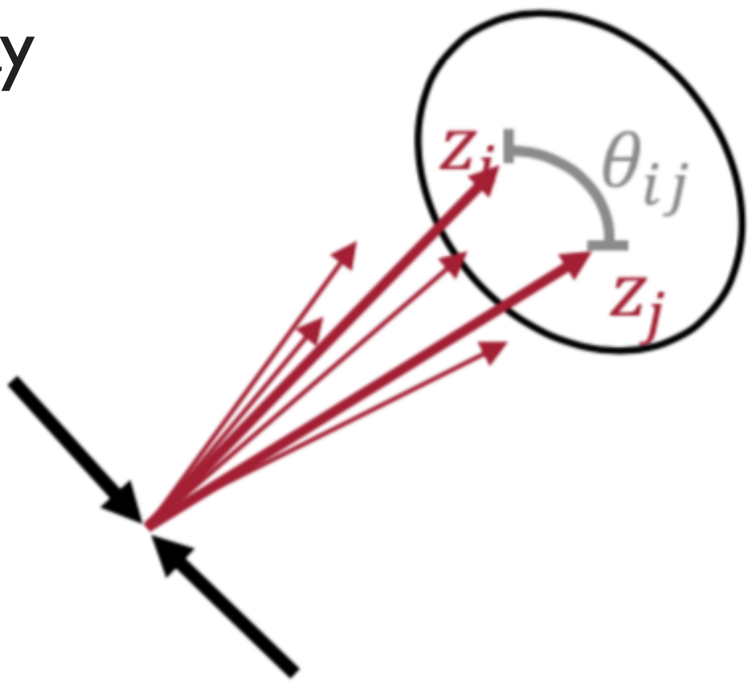
Color

n Petals



$\mathbb{Y}$

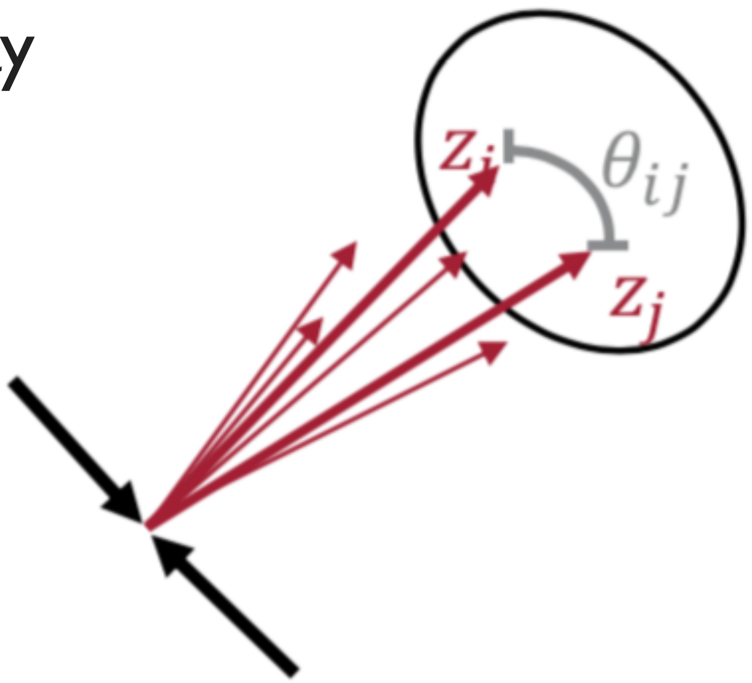
Permutation *Invariance*



- Lets say our input is set of particles (points) from decay
  - Each Input, $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu,$ particle type (PID), Other features (d_0, d_z) $\}$
- We need NN: $f \mid f(\mathrm{P}\mathbb{X}) = f(\mathbb{X})$
  - ~~MLPs need ordering and are out of question~~
  - CNNs aren't perfect for particle *sets*
    - Imposed structure and inefficient

Permutation *Invariance*

- Lets say our input is set of particles (points) from decay

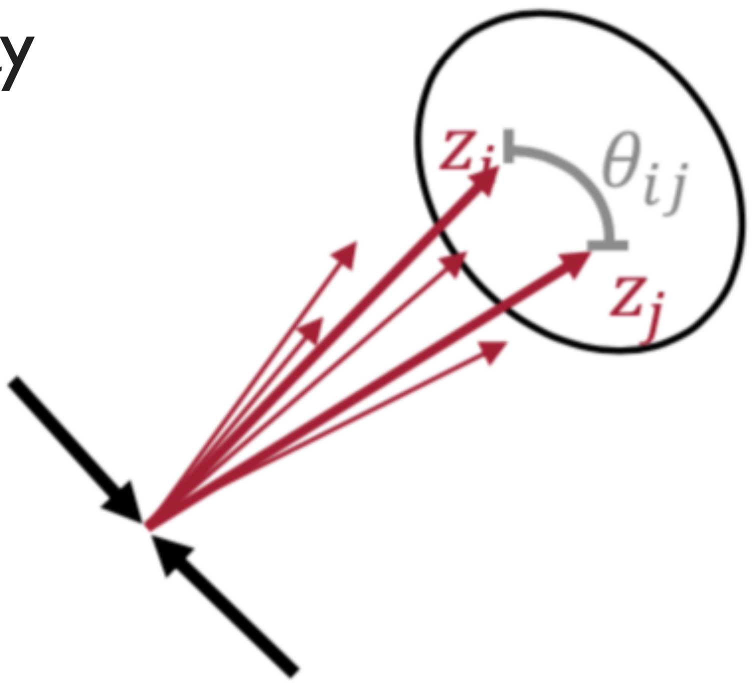


  - Each Input, $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu,$ particle type (PID),
    Other features (d_0, d_z) $\}$

- We need NN: $f \mid f(\mathrm{P}\mathbb{X}) = f(\mathbb{X})$
  - ~~MLPs need ordering and are out of question~~
  - ~~CNNs aren't perfect for particle sets~~
    - ~~Imposed structure and inefficient~~
    - Can we work directly on the sets ?

Permutation *Invariance*

- Lets say our input is set of particles (points) from decay



  - Each Input, $\mathbb{X} = (X_1, X_2, \ldots\ldots, X_n)^T$
  - Where $X_i = \{P_\mu,$ particle type (PID), Other features (d_0, d_z) $\}$
- We need NN: $f \mid f(\mathrm{P}\mathbb{X}) = f(\mathbb{X})$
  - ~~MLPs need ordering and are out of question~~
  - ~~CNNs aren't perfect for particle sets~~
    - ~~Imposed structure and inefficient~~
    - Can we work directly on the sets ?

Yes!
We can use DeepSets !

DeepSet Theorem

Deep Sets

[1703.06114]

Manzil Zaheer[1,2], **Satwik Kottur**[1], **Siamak Ravanbhakhsh**[1],
Barnabás Póczos[1], **Ruslan Salakhutdinov**[1], **Alexander J Smola**[1,2]
[1] Carnegie Mellon University [2] Amazon Web Services

Deep Sets Theorem [63]. *Let $\mathfrak{X} \subset \mathbb{R}^d$ be compact, $X \subset 2^{\mathfrak{X}}$ be the space of sets with bounded cardinality of elements in $\mathfrak{X}$, and $Y \subset \mathbb{R}$ be a bounded interval. Consider a continuous function $f : X \to Y$ that is invariant under permutations of its inputs, i.e. $f(x_1, \ldots, x_M) = f(x_{\pi(1)}, \ldots, x_{\pi(M)})$ for all $x_i \in \mathfrak{X}$ and $\pi \in S_M$. Then there exists a sufficiently large integer ℓ and continuous functions $\Phi : \mathfrak{X} \to \mathbb{R}^\ell$, $F : \mathbb{R}^\ell \to Y$ such that the following holds to an arbitrarily good approximation:*[1]

DeepSet Theorem

Deep Sets

[1703.06114]

Manzil Zaheer[1,2], **Satwik Kottur**[1], **Siamak Ravanbhakhsh**[1],
Barnabás Póczos[1], **Ruslan Salakhutdinov**[1], **Alexander J Smola**[1,2]
[1] Carnegie Mellon University [2] Amazon Web Services

Feature space

Variable length

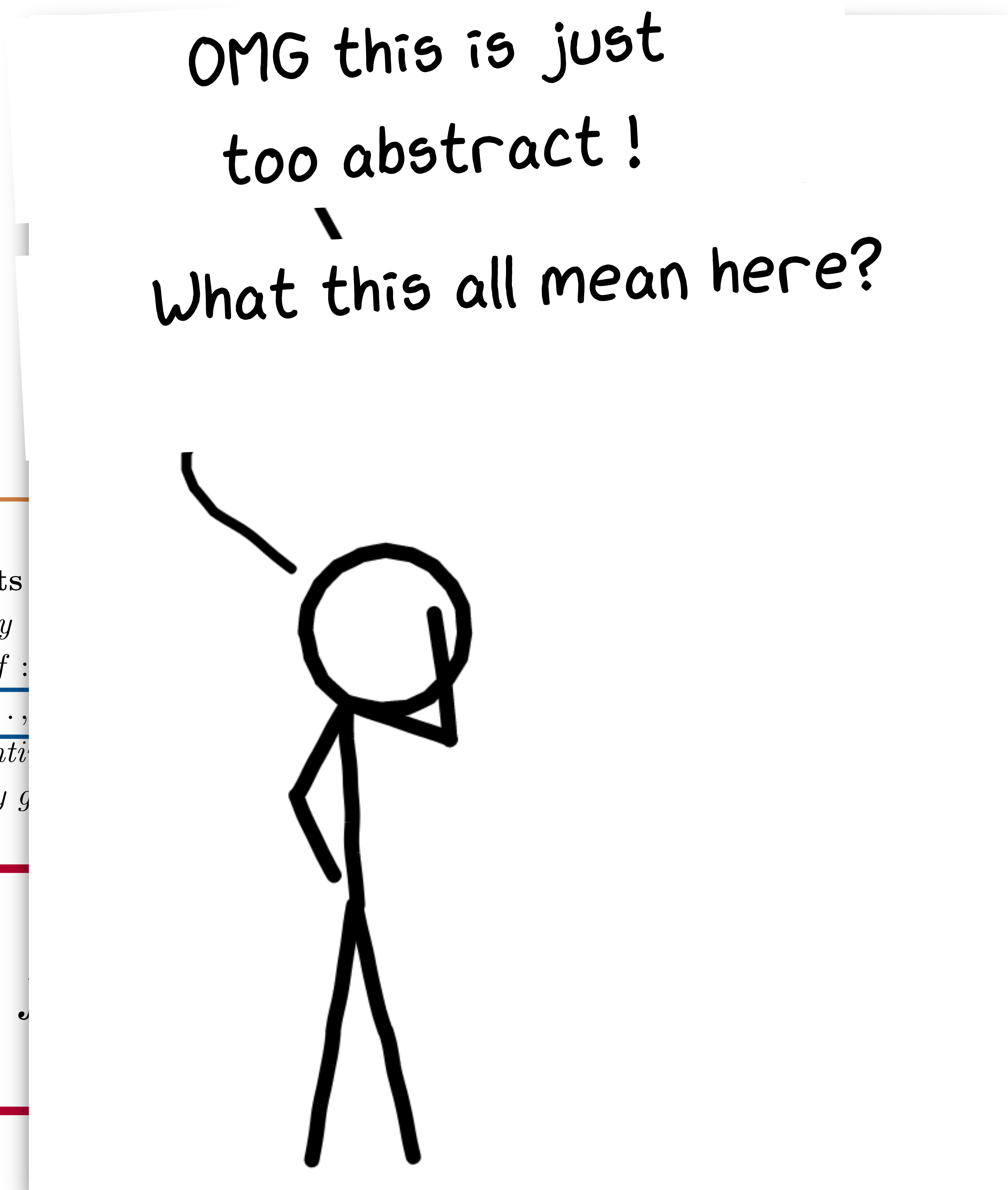
Deep Sets Theorem [63]. *Let* $\boxed{\mathfrak{X} \subset \mathbb{R}^d}$ *be compact,* $\boxed{X \subset 2^{\mathfrak{X}} \text{ be the space of sets}}$ *with bounded cardinality of elements in* $\mathfrak{X}$*, and* $Y \subset \mathbb{R}$ *be a bounded interval. Consider a continuous function* $f : X \to Y$ *that is invariant under permutations of its inputs, i.e.* $f(x_1, \ldots, x_M) = \boxed{f(x_{\pi(1)}, \ldots, x_{\pi(M)}) \text{ for all } x_i \in \mathfrak{X} \text{ and } \pi \in S_M}$*. Then there exists a sufficiently large integer* ℓ *and continuous functions* $\Phi : \mathfrak{X} \to \boxed{\mathbb{R}^\ell}$*,* $F : \boxed{\mathbb{R}^\ell} \to Y$ *such that the following holds to an arbitrarily good approximation:*[1]

Permutation invariance

Latent space

$$\boxed{f(\{x_1, \ldots, x_M\}) = F\left(\sum_{i=1}^{M} \Phi(x_i)\right)}$$

DeepSet Theorem



3.06114]

Feature space

Variable length

Permutation invariance

Latent space

Deep Sets ... *n bounded*
cardinality ... *ontinuous*
function f : ... $., x_M) =$
$\boxed{f(x_{\pi(1)}, \dots,}$ ... *ue integer*
ℓ *and conti* ... *lds to an*
arbitrarily g

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\})$$

- Physics given function f ; takes in particles set & outputs the class of the jet
  - We have no idea how to describe this function analytically!

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i \phi(X_i)\right)$$

- Physics given function f ; takes in particles set & outputs the class of the jet
- But, we can describe this function using two different functions
  - ϕ : That acts on each particle to embed it in latent space

$$N_{particles} \times N_{features} \longrightarrow N_{particles} \times N_{embeddings}$$

DeepSet Architecture

Fermilab

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i \phi(X_i)\right)$$

- Physics given function f ; takes in particles set & outputs the class of the jet
- But, we can describe this function using two different functions
  - ϕ : That acts on each particle to embed it in latent space

$$N_{particles} \times N_{features} \longrightarrow N_{particles} \times N_{embeddings}$$

  - $\sum$: The Latent of all the particles are summed over

$$N_{particles} \times N_{embeddings} \longrightarrow N_{embeddings}$$

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i \phi(X_i)\right)$$

- Physics given function f ; takes in particles set & outputs the class of the jet
- But, we can describe this function using two different functions
  - ϕ : That acts on each particle to embed it in latent space

$$N_{particles} \times N_{features} \longrightarrow N_{particles} \times N_{embeddings}$$

  - $\sum$: The Latent of all the particles are summed over

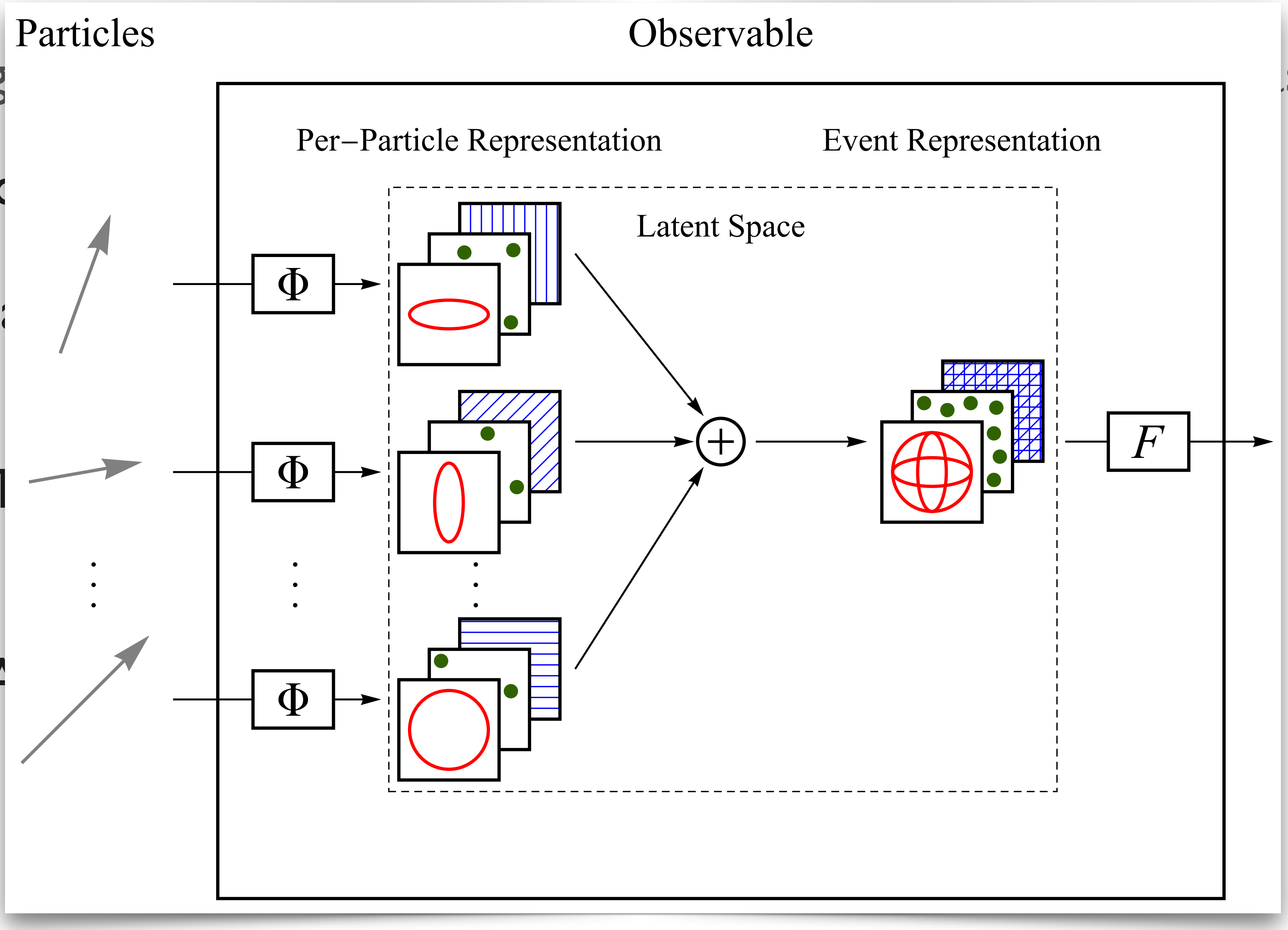
$$N_{particles} \times N_{embeddings} \longrightarrow N_{embeddings}$$

  - F : Acts on aggregated latent space to reproduce f

$$N_{embeddings} \longrightarrow N_{classes}$$

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i \phi(X_i)\right)$$



- Physics g ... ss of the jet
- But, we c
    - ϕ : Th
        - $\sum$: T
        - F : A

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i \phi(X_i)\right)$$

- Physics given function f ; takes in particles set & outputs the class of the jet
- But, we can describe this function using two different functions
  - ϕ : That acts on each particle to embed it in latent space

    $$N_{particles} \times N_{features} \longrightarrow N_{particles} \times N_{embeddings}$$

    - $\sum$: The Latent of all the particles are summed over

    $$N_{particles} \times N_{embeddings} \longrightarrow N_{embeddings}$$

    - F : Acts on aggregated latent space to reproduce f

    $$N_{embeddings} \longrightarrow N_{classes}$$

What is so special about this? How did we get Permutation Invariance?

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\boxed{\sum_i}\phi(X_i)\right)$$

- Physics given function f ; takes in particles set & outputs the class of the jet
- But, we can describe this function using two different functions
  - ϕ : That acts on each particle to embed it in latent space

    $$N_{particles} \times N_{features} \longrightarrow N_{particles} \times N_{embeddings}$$

  - $\sum$: The Latent of all the particles are summed over

    $$N_{particles} \times N_{embeddings} \longrightarrow N_{embeddings}$$

  - F : Acts on aggregated latent space to reproduce f

    $$N_{embeddings} \longrightarrow N_{classes}$$

Can we get Permutation Invariance with other operations?

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\langle \phi(X_i) \rangle\right)$$

- Physics given function f ; takes in particles set & outputs the class of the jet
- But, we can describe this function using two different functions
  - ϕ : That acts on each particle to embed it in latent space

$$N_{particles} \times N_{features} \longrightarrow N_{particles} \times N_{embeddings}$$

    - $\langle\rangle$: The Latent of all the particles are averaged over

$$N_{particles} \times N_{embeddings} \longrightarrow N_{embeddings}$$

    - F : Acts on aggregated latent space to reproduce f

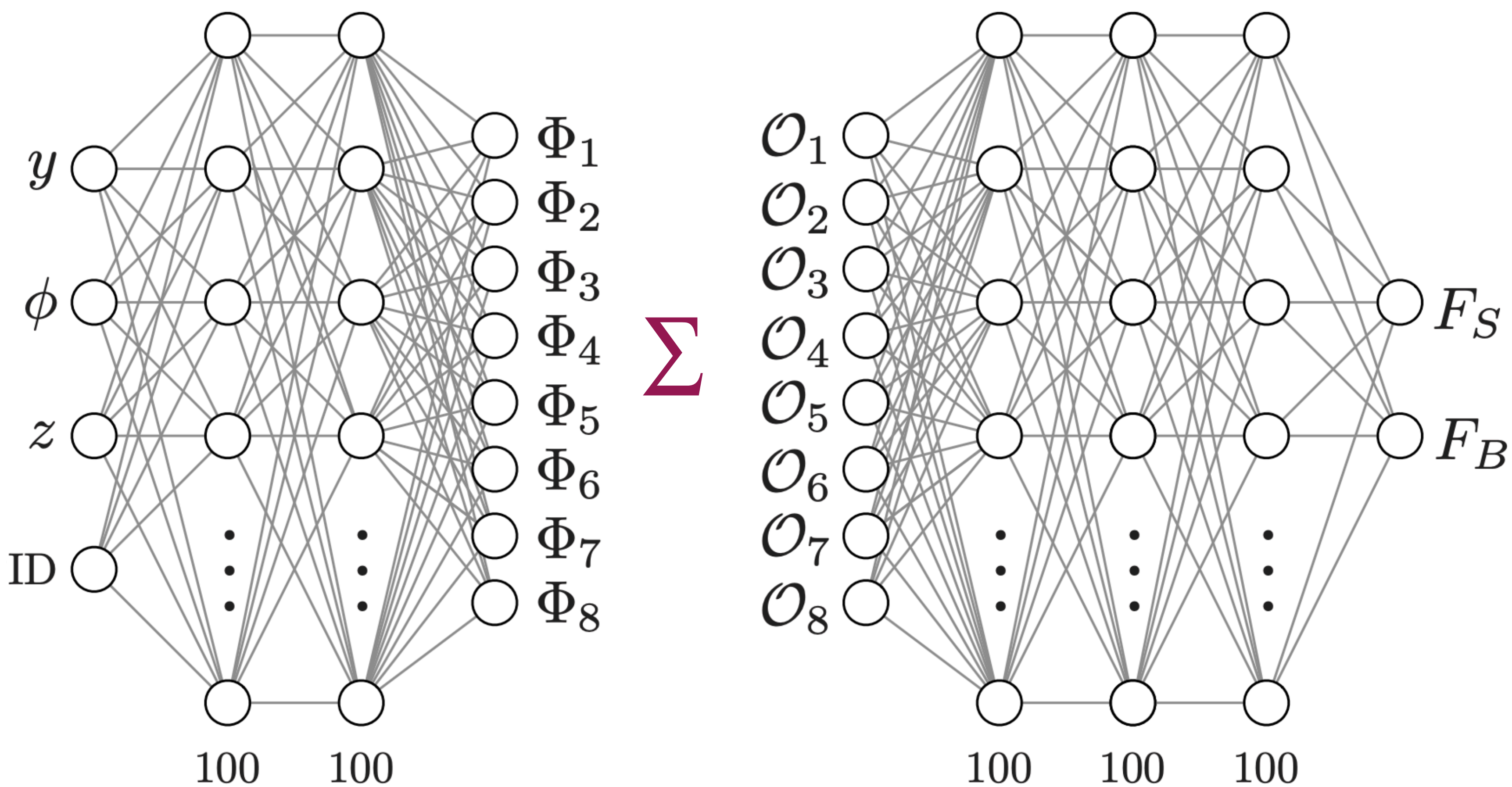
$$N_{embeddings} \longrightarrow N_{classes}$$

Anything else ?

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i \phi(X_i)\right)$$

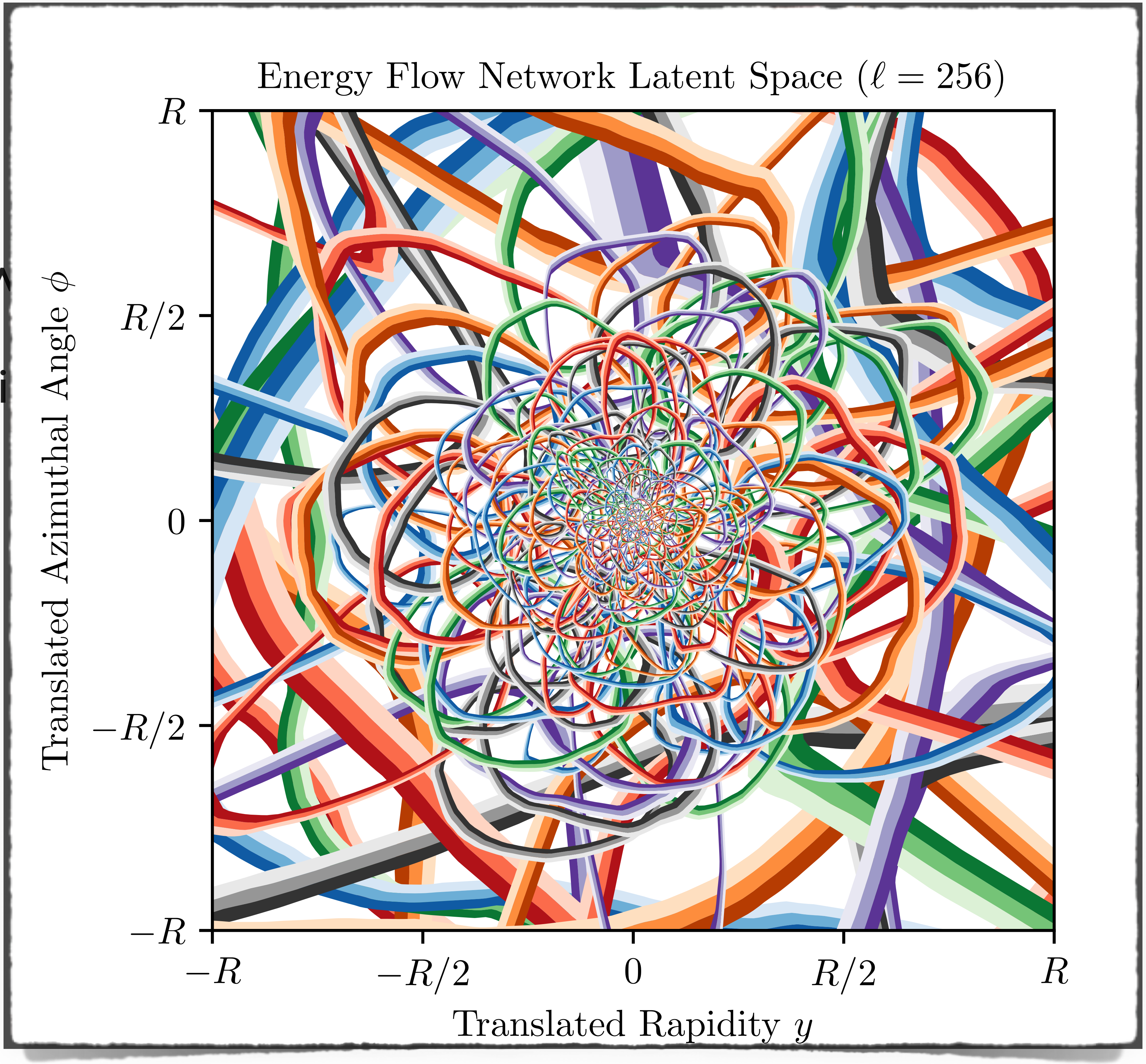
- How do we get the functions F and ϕ ?
    - Approximate them using the neural networks!



[2]

DeepSet Architecture

- How do w
  - Approxi



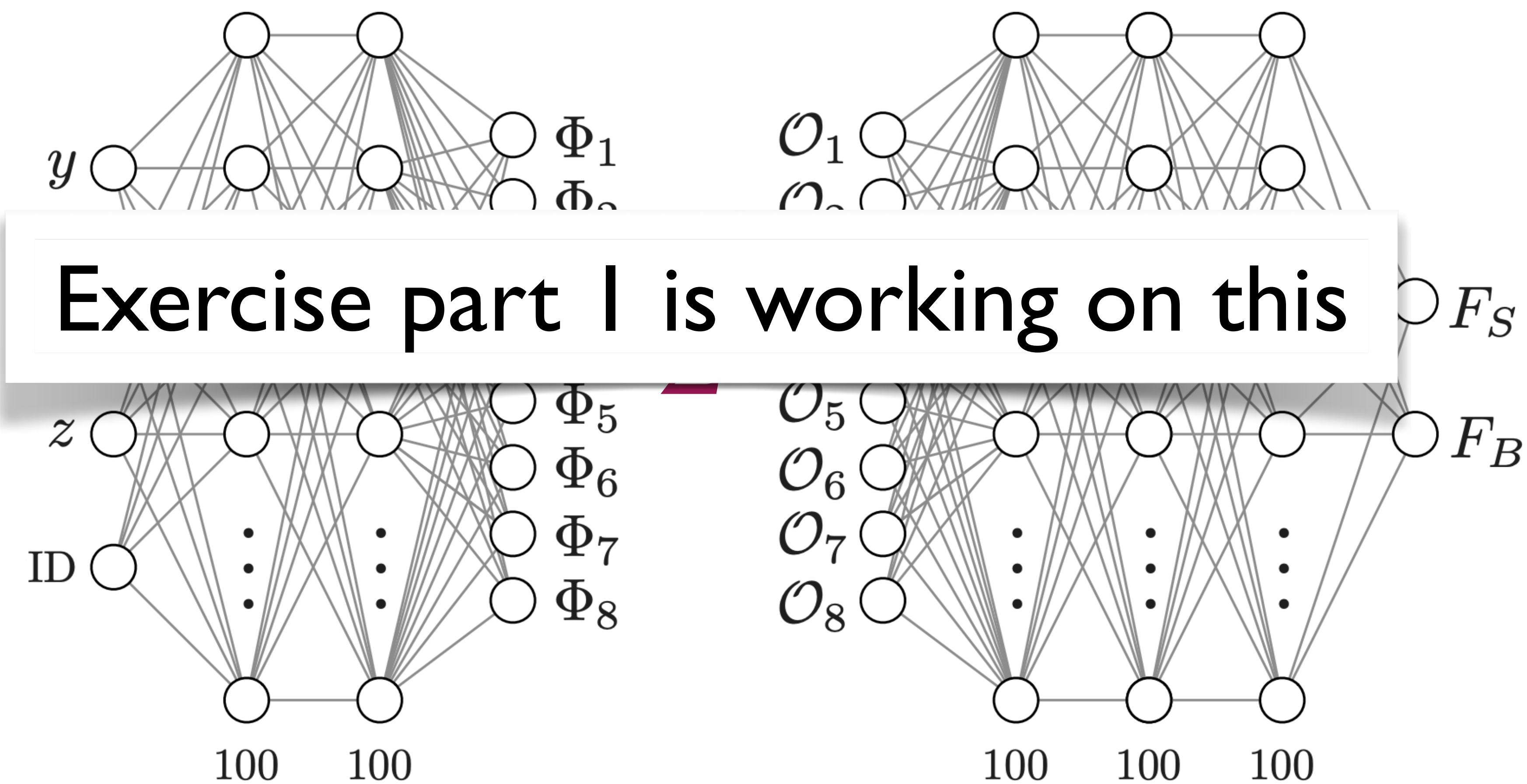
F_S

F_B

DeepSet Architecture

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i \phi(X_i)\right)$$

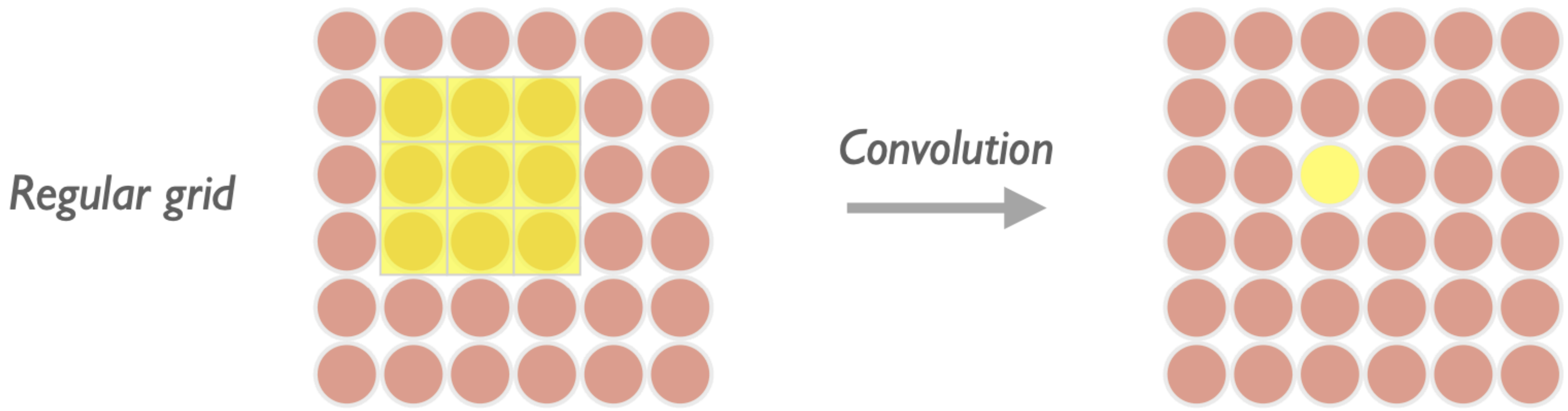
- How do we get the functions F and ϕ ?
  - Approximate them using the neural networks!



Exercise part 1 is working on this

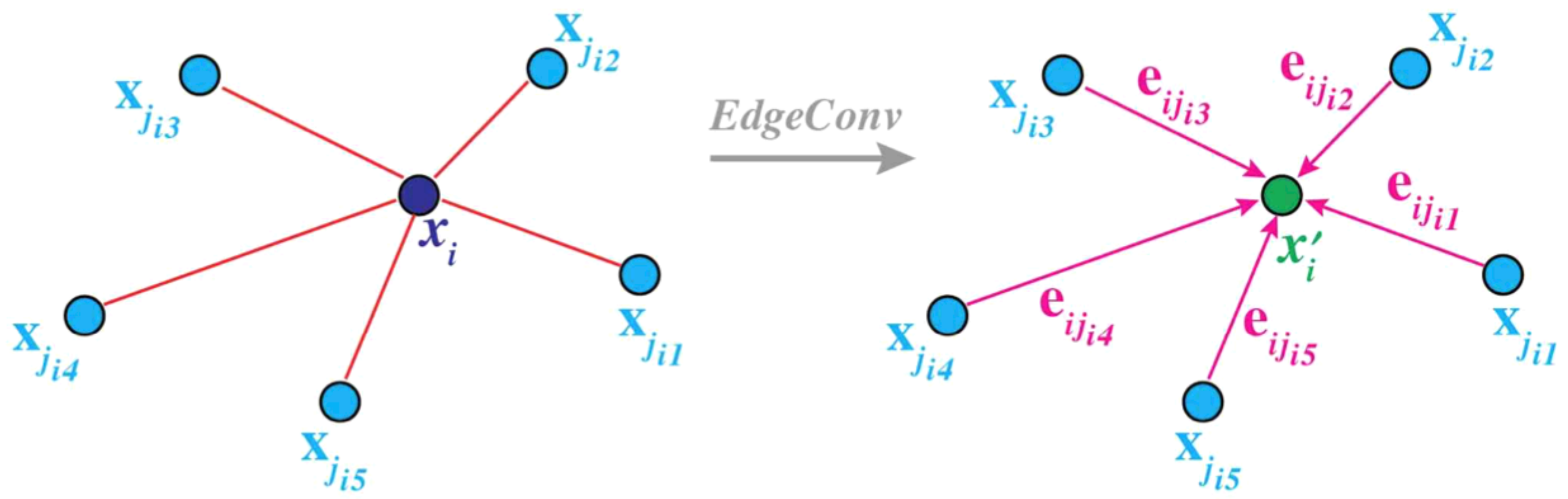
How to capture local properties ?

- May be we can implement something like Convolution in CNNs



How to capture local properties ?

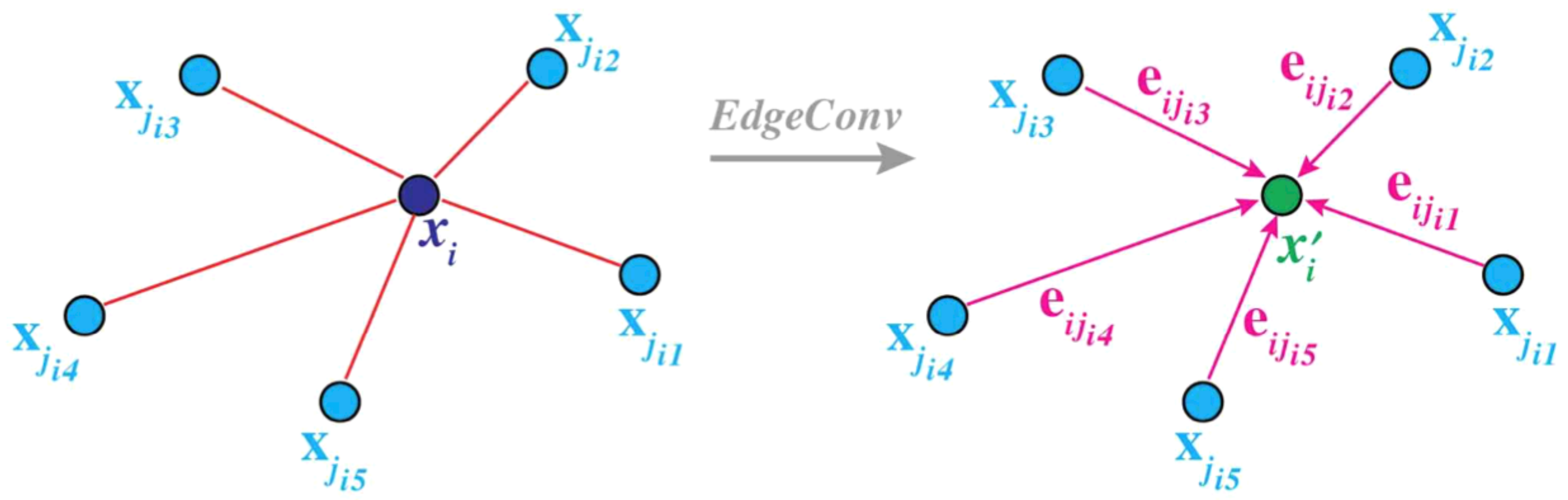
- May be we can implement something like Convolution in CNNs
  - Enter EdgeConv: Convolution on point clouds!



-

How to capture local properties ?

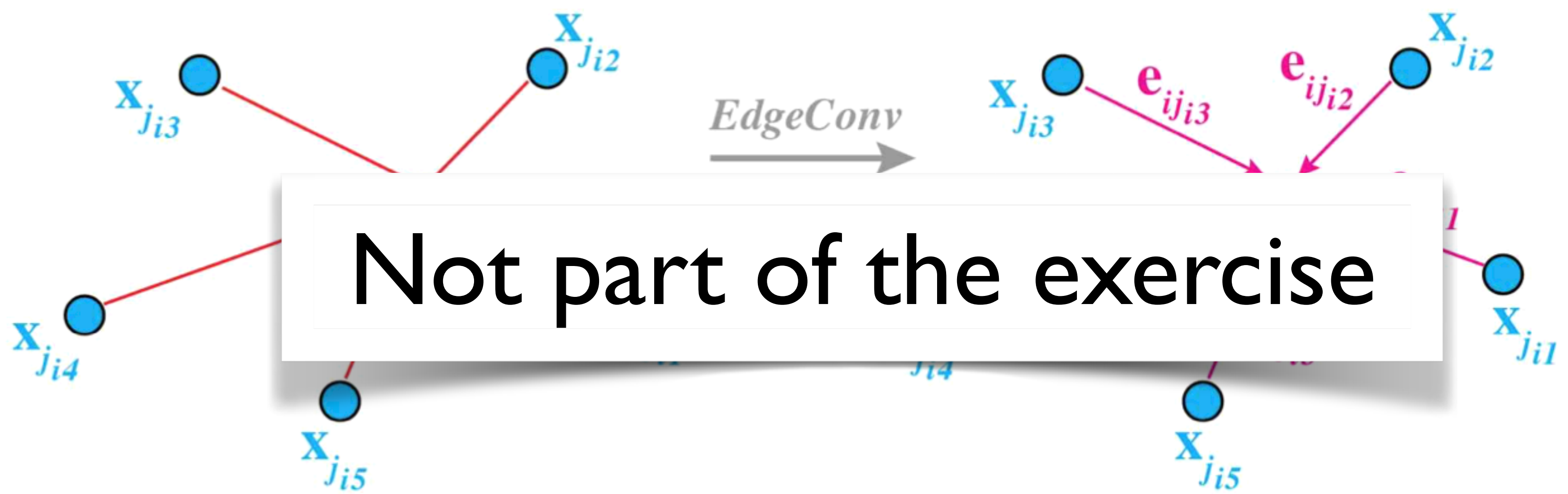
- May be we can implement something like Convolution in CNNs
  - Enter EdgeConv: Convolution on point clouds!



- For each point, Construct a Graph with K-nearest neighbors
- Define edge feature e_{ij} , using operation h on nodes | $e_{ij} = h(x_i, x_j)$
- Aggregate all the edge features to update the node to x_i'

How to capture local properties ?

- May be we can implement something like Convolution in CNNs
  - Enter EdgeConv: Convolution on point clouds!



- For each point, Construct a Graph with K-nearest neighbors
- Define edge feature e_{ij} , using operation h on nodes | $e_{ij} = h(x_i, x_j)$
- Aggregate all the edge features to update the node to x_i'
- This is a Graph Neural Network with Dynamic Edge Convolution

Capturing all the interactions !

- How about we capture all interactions in a particle/point cloud ?
  - Maybe we can capture all the local & global properties !
  - Extract Maximum amount of information!
-

Capturing all the interactions !

- How about we capture all interactions in a particle/point cloud ?
  - Maybe we can capture all the local & global properties !
  - Extract Maximum amount of information!

- How do we use all this information ?
  - Maybe we can get relative importance for particle / set w.r.t to others ?
  - Use this information to design a *smarter* DeepSet Approach ?

Capturing all the interactions !

- How about we capture all interactions in a particle/point cloud ?
  - Maybe we can capture all the local & global properties !
  - Extract Maximum amount of information!
- How do we use all this information ?
  - Maybe we can get relative importance for particle / set w.r.t to others ?
  - Use this information to design a *smarter* DeepSet Approach ?
- How do we end up doing all of this in one go ?

Capturing all the interactions !

- H

Can you pass the Turing test ?

As an AI language model, I don't have personal abilities or consciousness, so I can't "pass" the Turing test in the traditional sense. The Turing test is designed to determine if a machine can exhibit human-like intelligence in conversation to the extent that a human evaluator cannot reliably distinguish between the machine and a human. It's more about how convincingly the AI can simulate human-like conversation, rather than actual consciousness or understanding.

As an AI language model, my goal is to provide useful and coherent responses to the best of my training and abilities. While I can simulate human-like conversation to some extent, I am still an artificial construct and lack true consciousness and self-awareness. My responses are based on patterns and information present in the data I was trained on up to my knowledge cutoff date in September 2021.

- H
- How do we end up doing all of this in one go ?

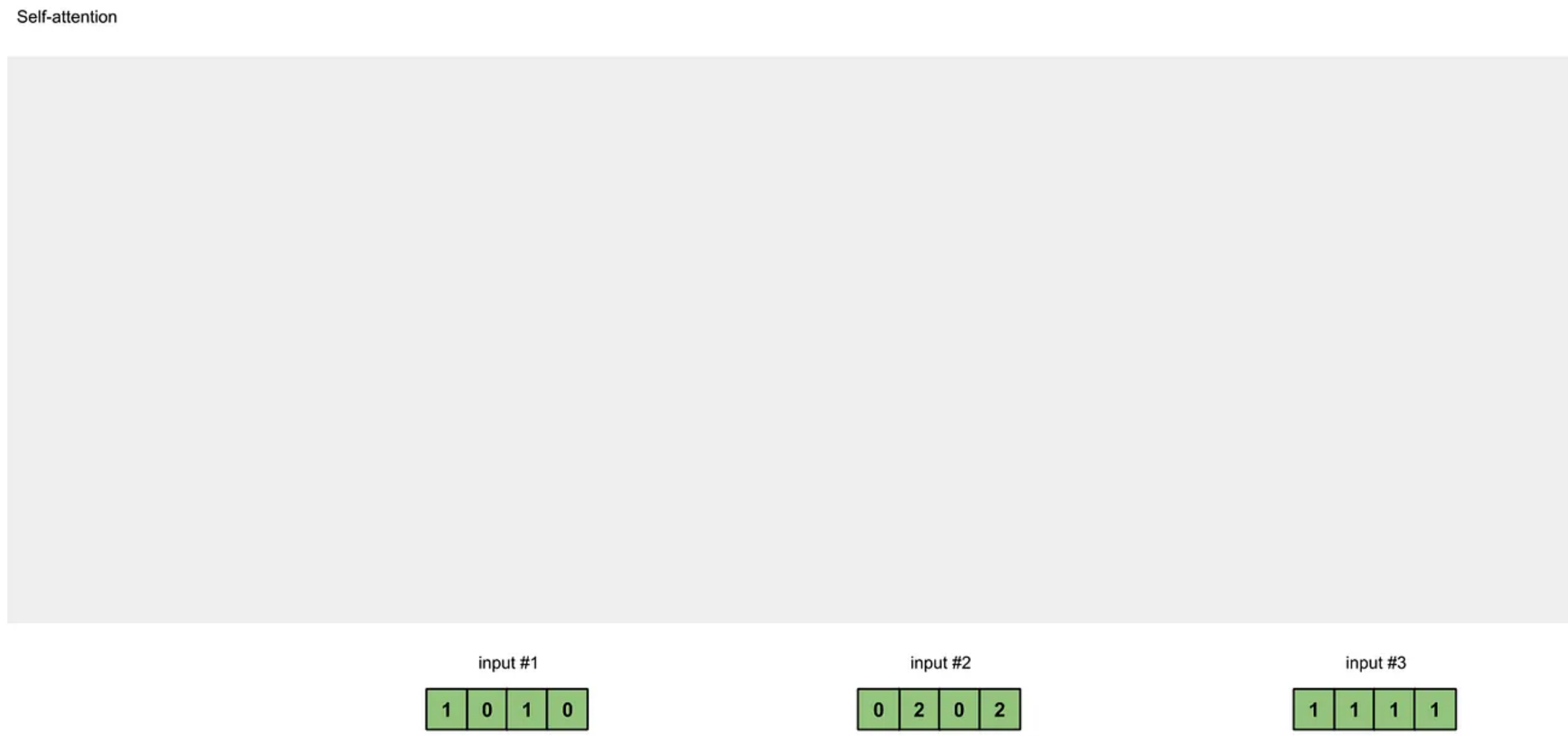
Capturing all the interactions !

- How about we capture all interactions in a particle/point cloud ?
  - Maybe we can capture all the local & global properties !
  - Extract Maximum amount of information!
- How

Attention is all you need !

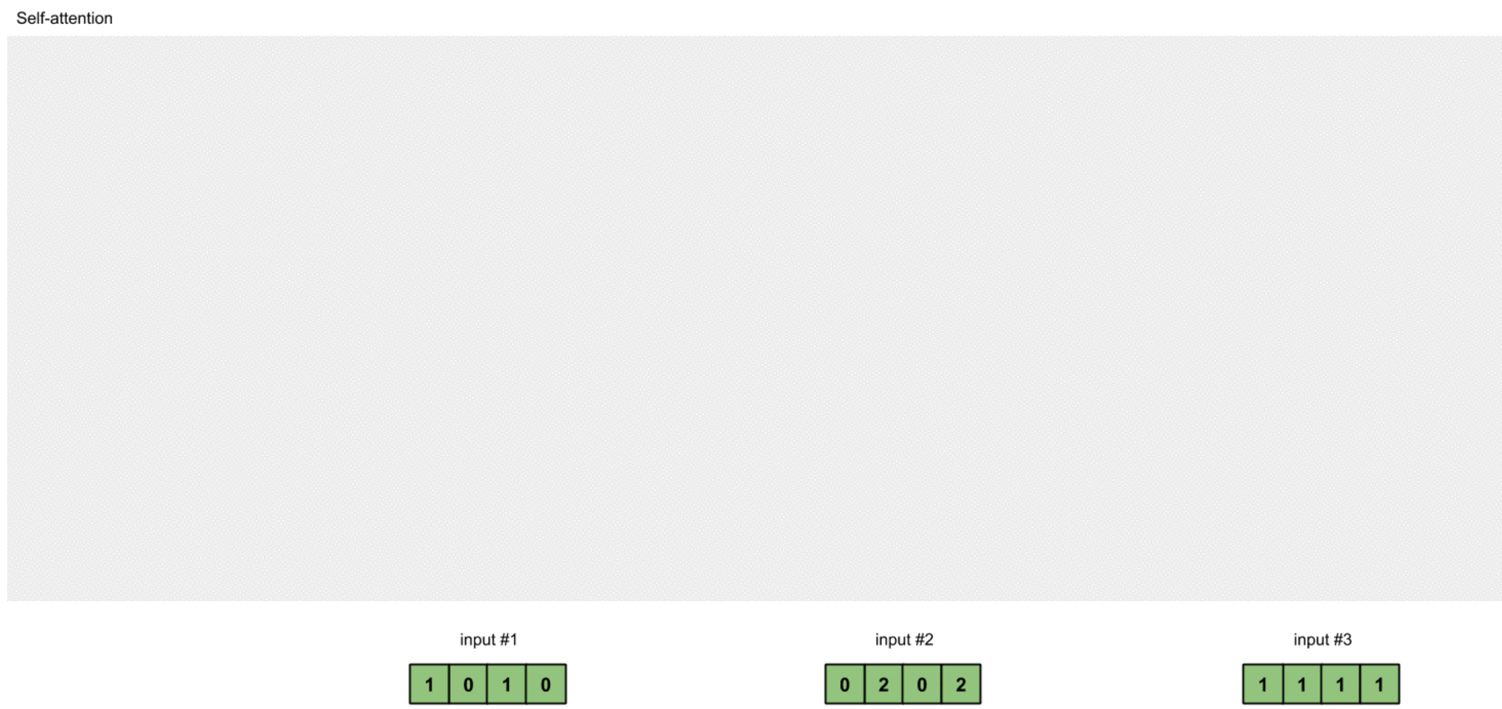
  - Maybe we can get relative importance for particle / set w.r.t to others ?
  - Use this information to design a *smarter* DeepSet Approach ?
- How do we end up doing all of this in one go ?

Intro to Self-Attention



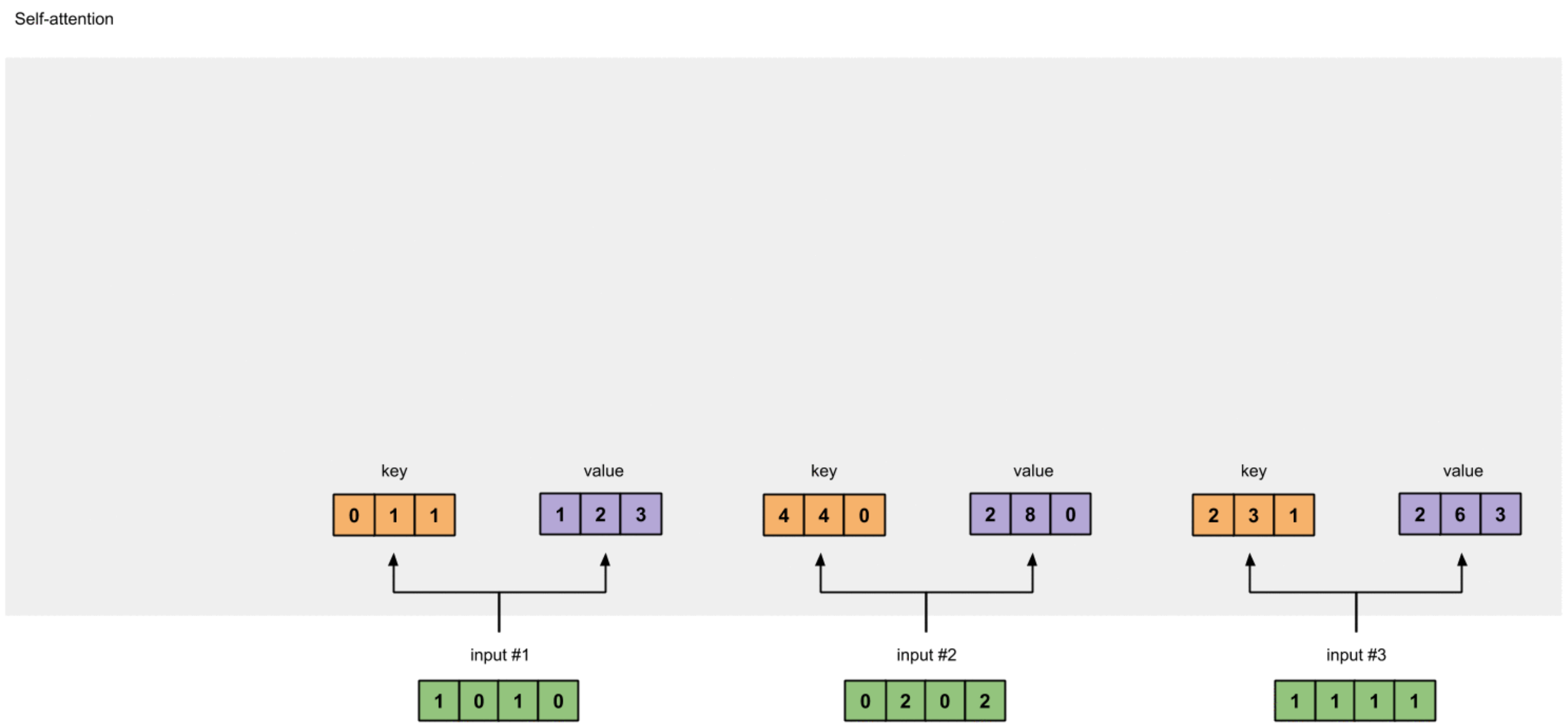
- Step 0: Lets take a case where our set / particle cloud has 3 elements in it
- Objective: Calculate relative importance of each element with others
- Output: Obtain a new embeddings with attention factored in

Intro to Self-Attention



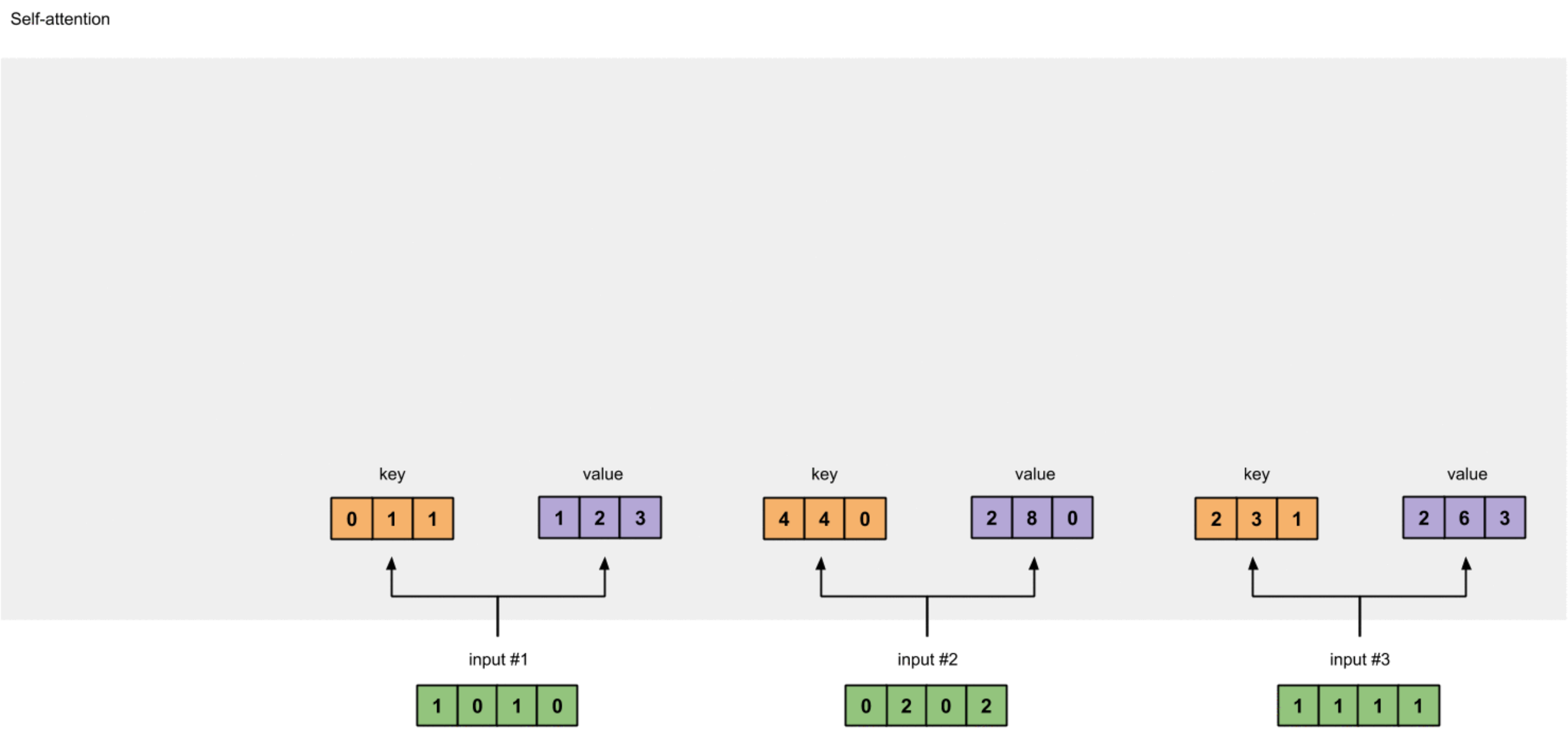
- Step 1:
  - Initialize three weight matrices W_K, W_Q, W_V
  - Multiply the Inputs with these weights
  - Obtain three different vectors Key, Query and Value for each set

Intro to Self-Attention



- Step 3:
  - Dot product the Query and Key vectors to get attention scores
  - Apply softmax on these scores to get the attention values
    - We get attention values w.r.t each of the set elements

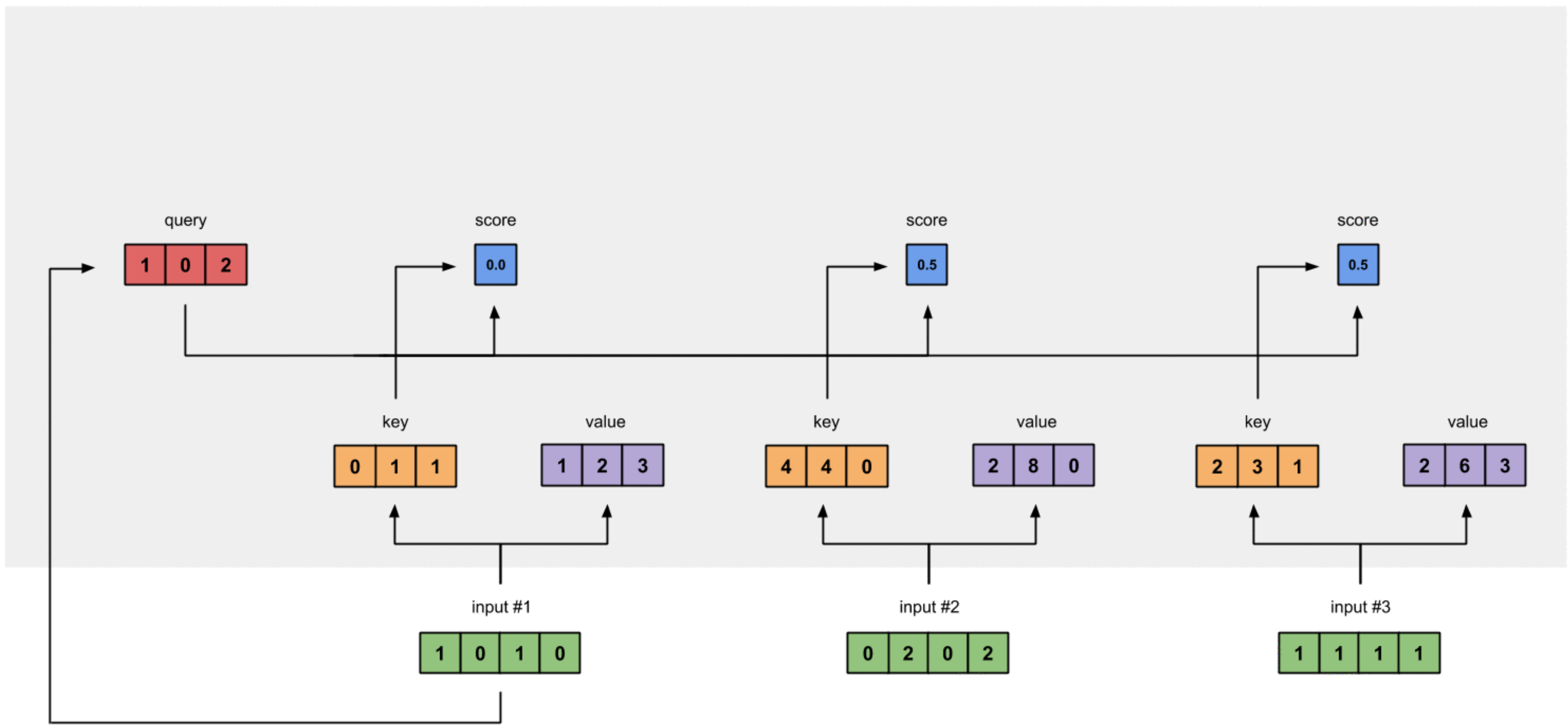
Intro to Self-Attention



- Step 3:
  - Dot product the Query and Key vectors to get attention scores
  - Apply softmax on these scores to get the attention values
    - We get attention values w.r.t each of the set elements

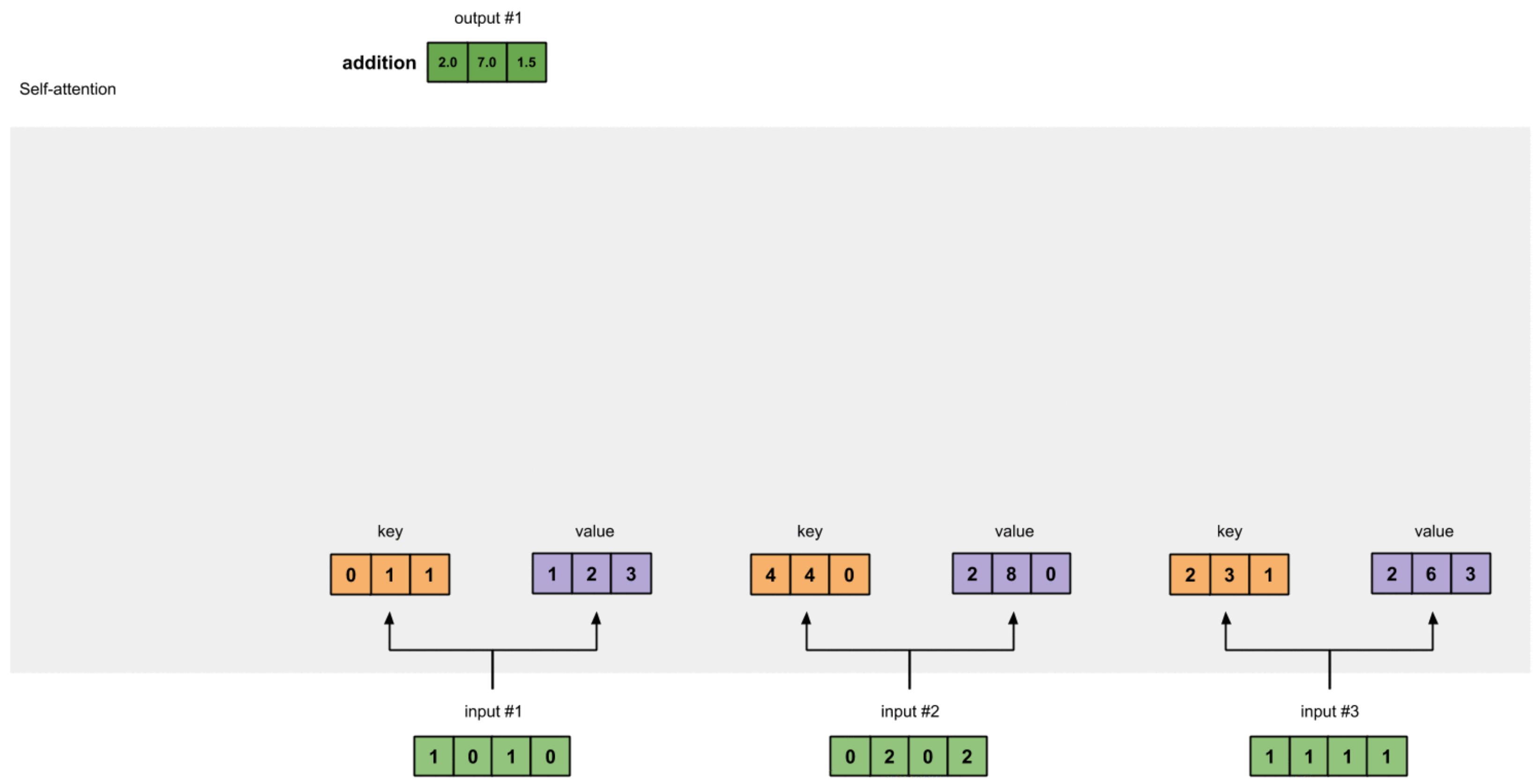
Intro to Self-Attention

Self-attention



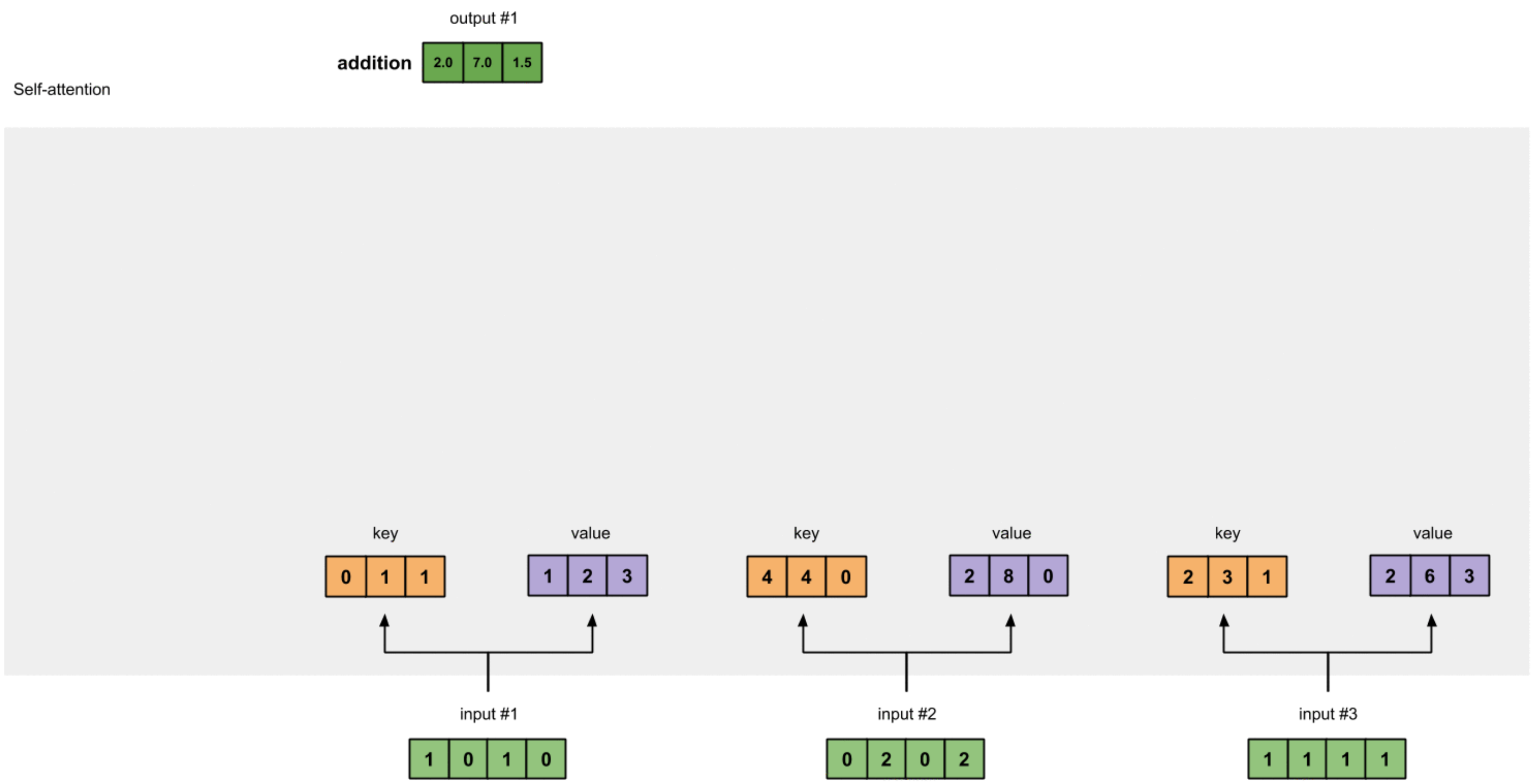
- Step 4:
  - Use these scores to obtain the weighted values
    - Multiply the attention score with value to obtain weighted values
    - Repeat this process to get all weighted values w.r.t to the first

Intro to Self-Attention



- Step 4:
    - Aggregate these values by sum across all elements
        - Obtain the weighted embeddings
    - Repeat this process for all the sets to get new embeddings

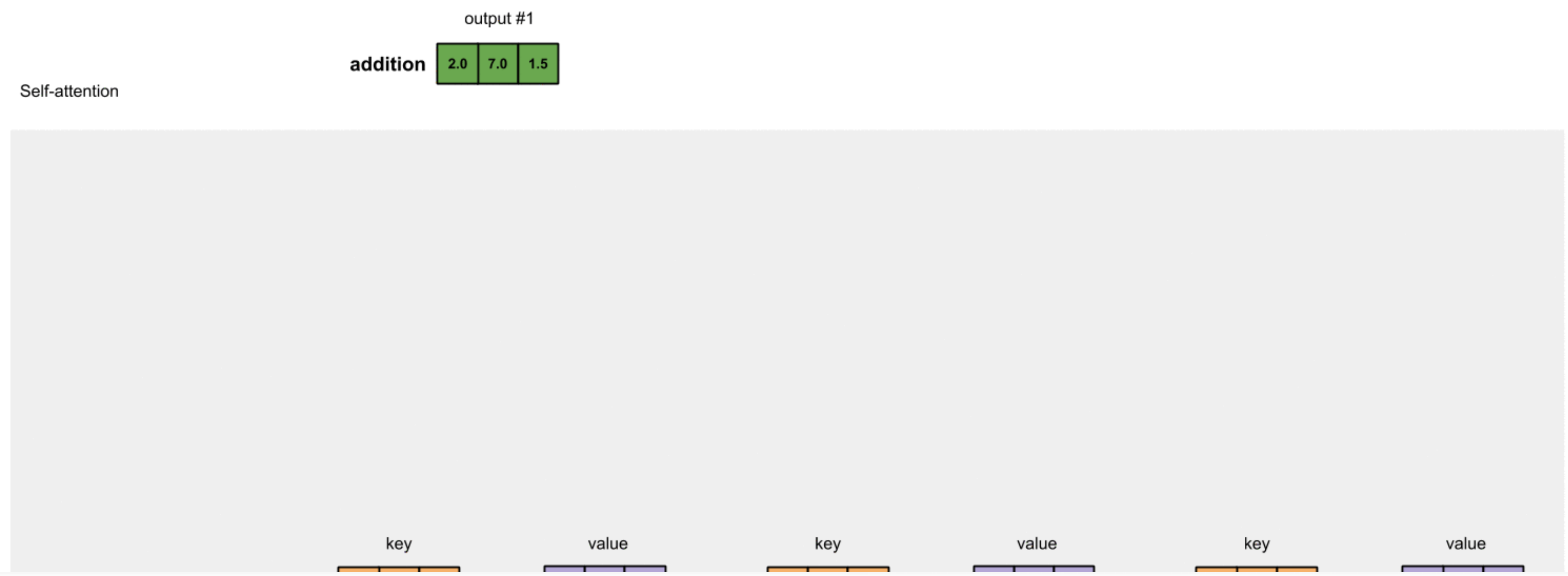
Intro to Self-Attention



- Step 4:
  - Aggregate these values by sum across all elements
    - Obtain the weighted embeddings
  - Repeat this process for all the sets to get new embeddings

Intro to Self-Attention

Fermilab

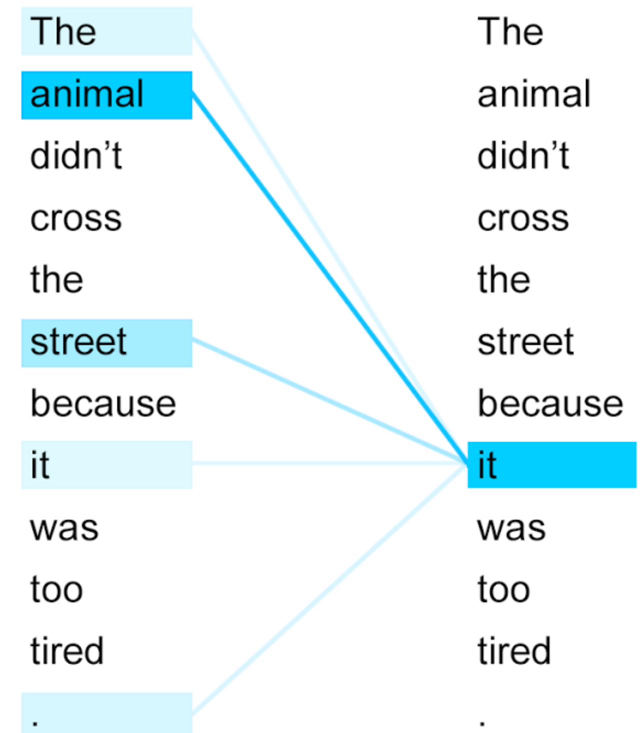


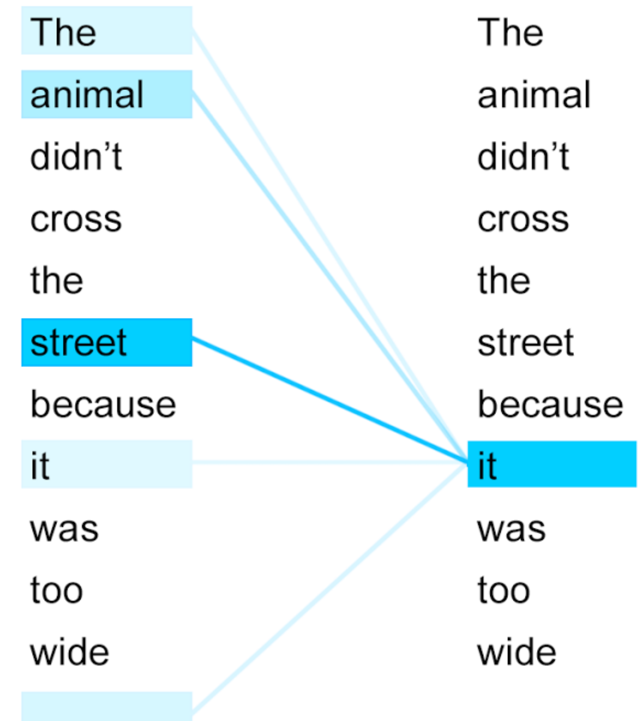
Now you got the attention !

- Step 4:
  - Aggregate these values by sum across all elements
    - Obtain the weighted embeddings
  - Repeat this process for all the sets to get new embeddings

Intro to Self-Attention

Now you got the n !

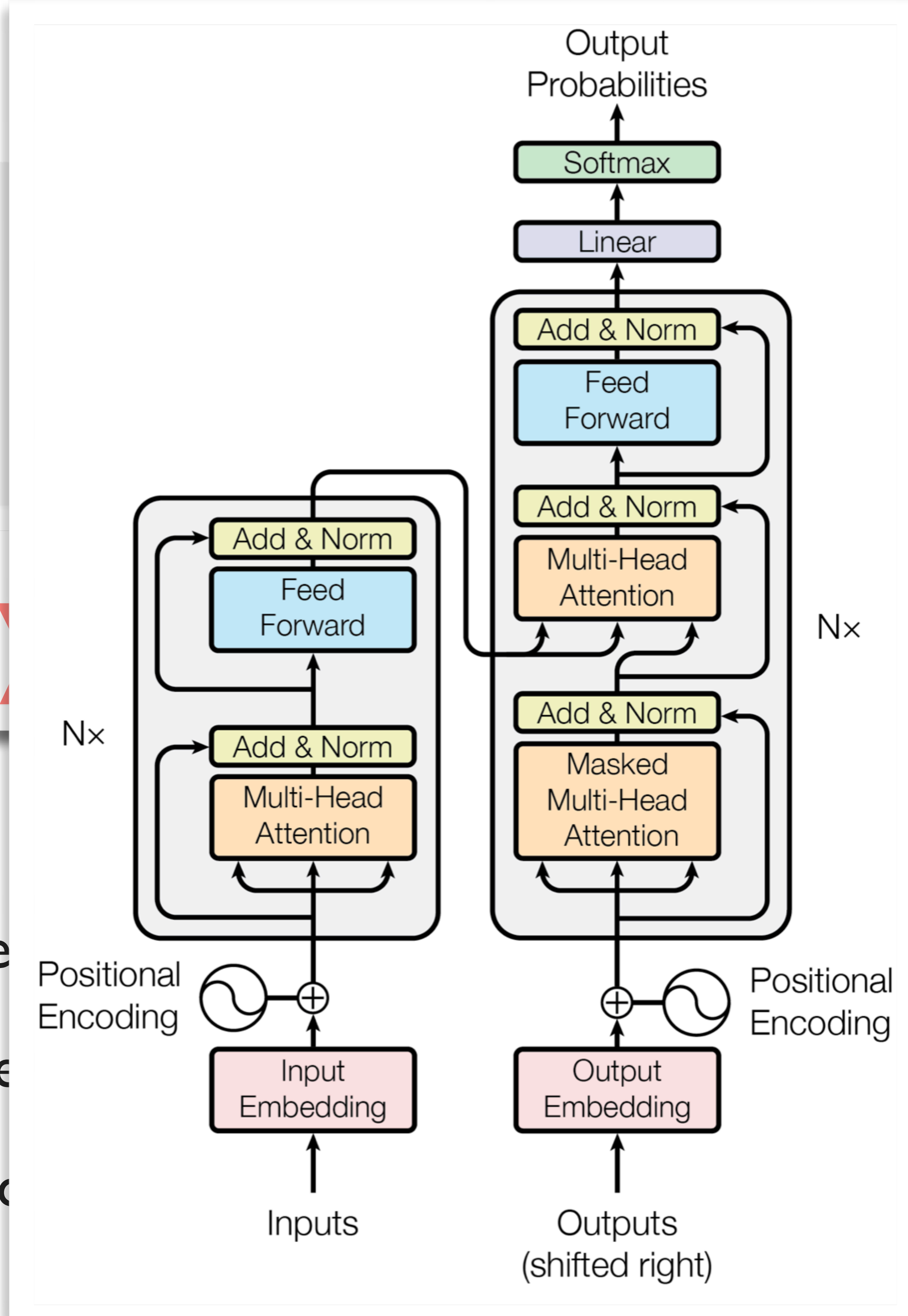




- Step 4:
  - Aggregate these values by sum across all ele
    - Obtain the weighted embeddings
  - Repeat this process for all the sets to get n

Intro to Self-Attention

Self-attention



value

Now y... ention !

- Step 4:
  - Aggregate these
    - Obtain the we
  - Repeat this proc ... dings

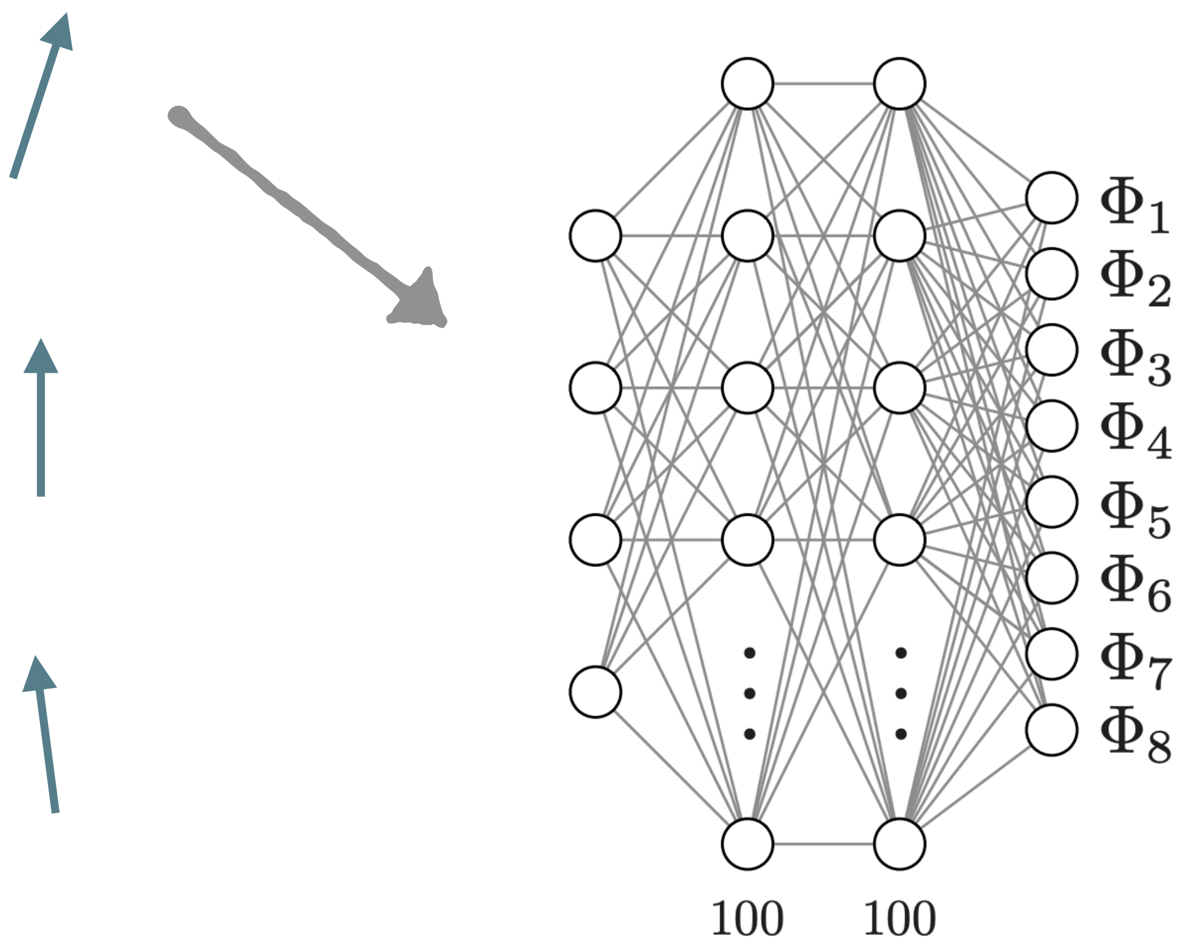
Intro to Self-Attention

- Step 4:
  - Aggre
    - Ob
  - Repea

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

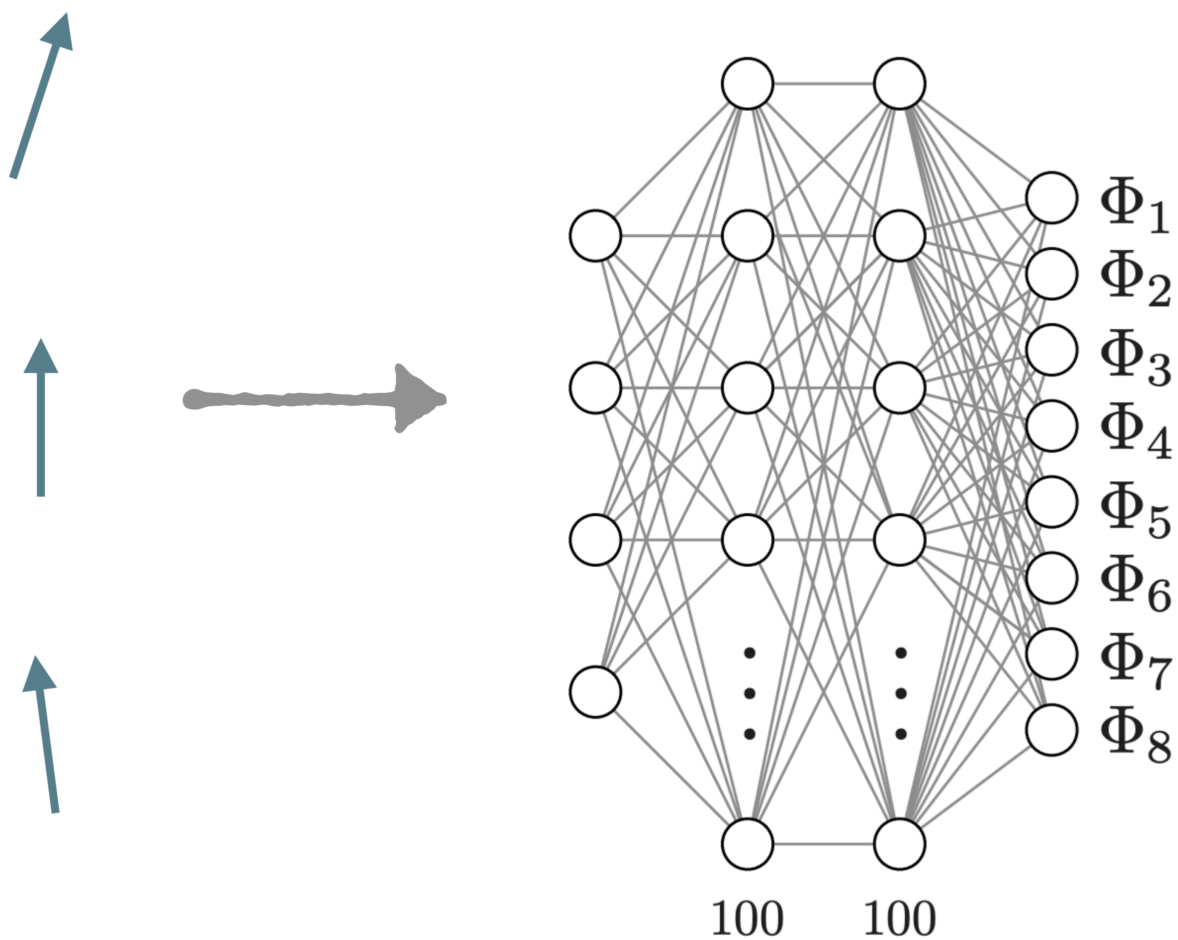
Transformers in DeepSets

- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets



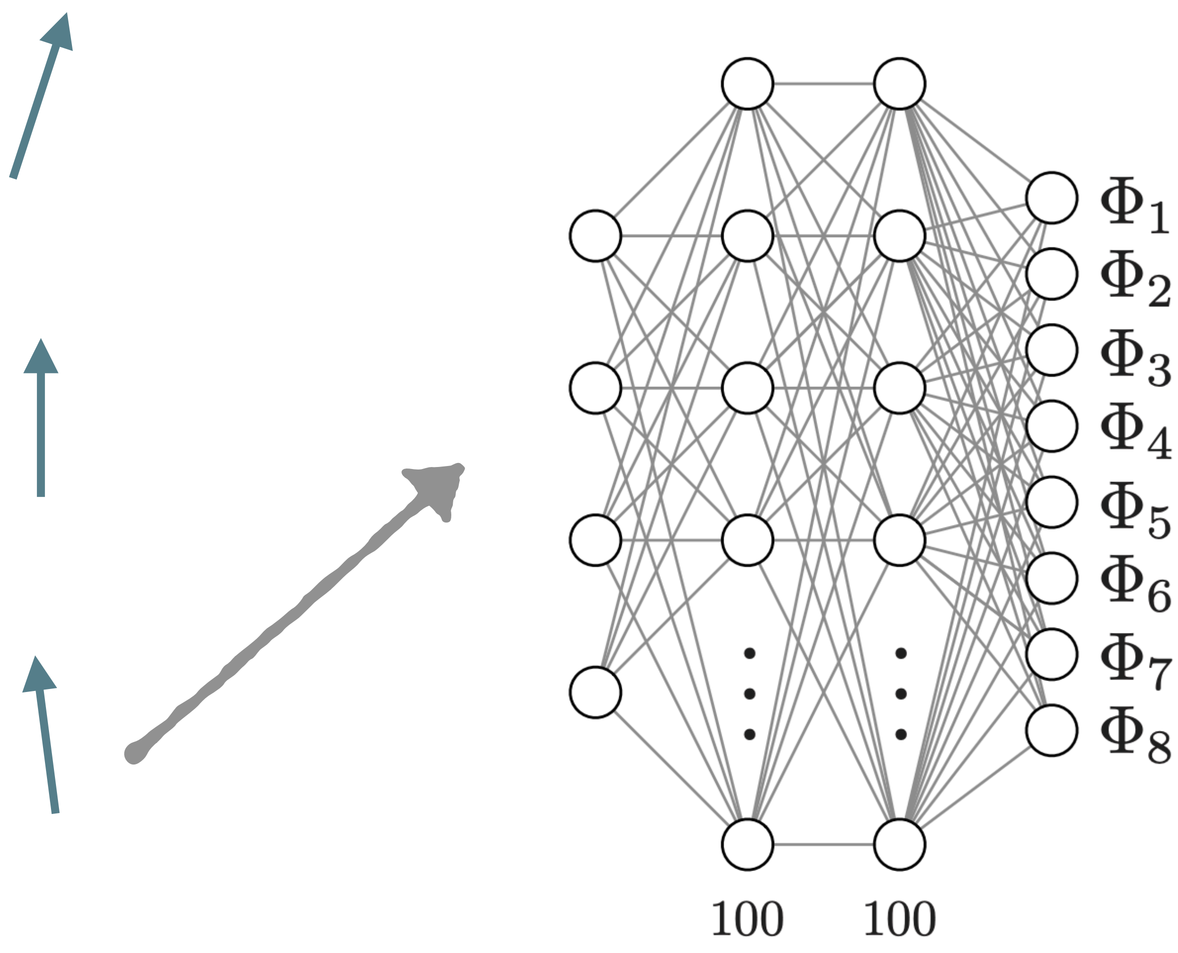
Transformers in DeepSets

- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets



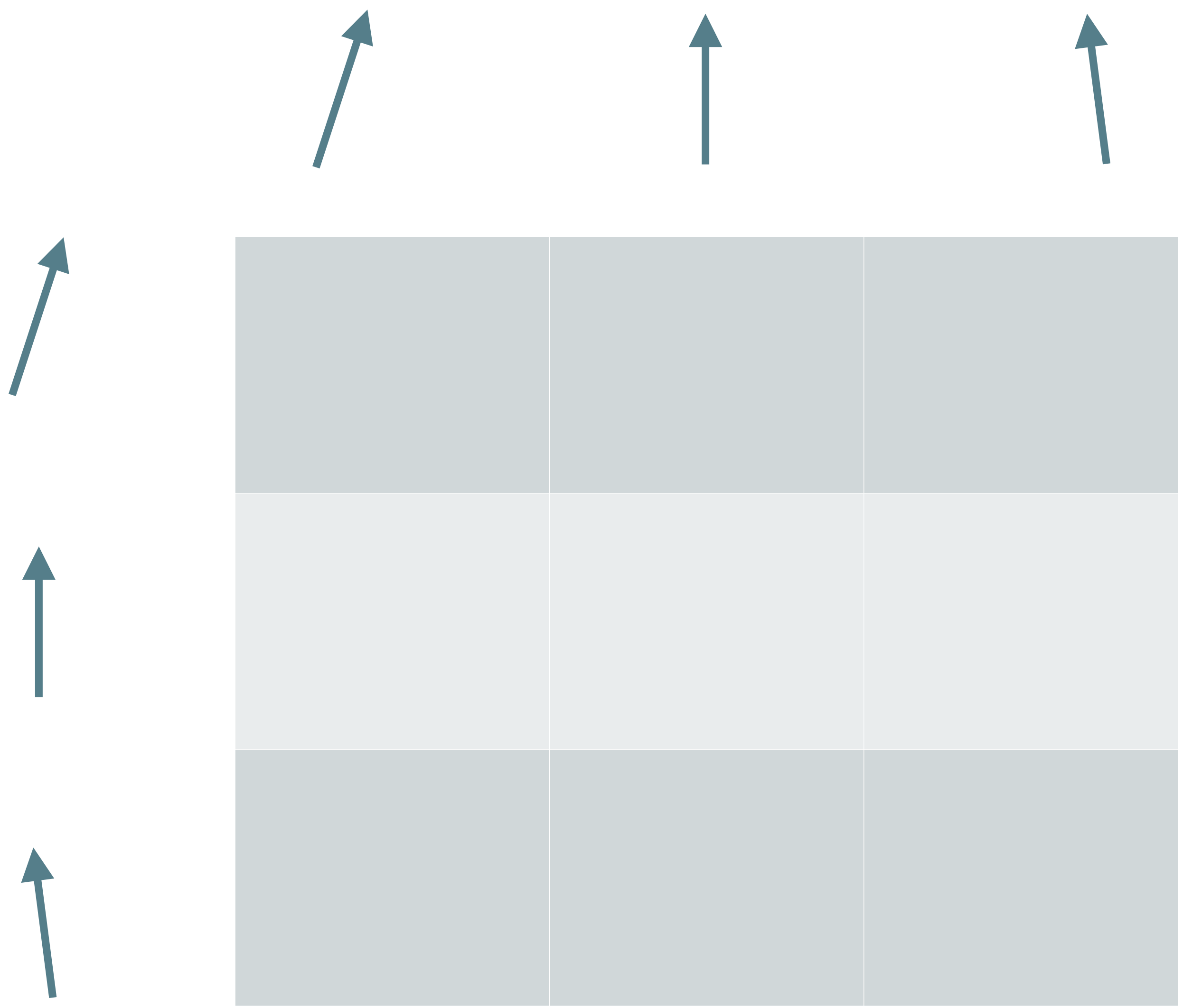
Transformers in DeepSets

- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets



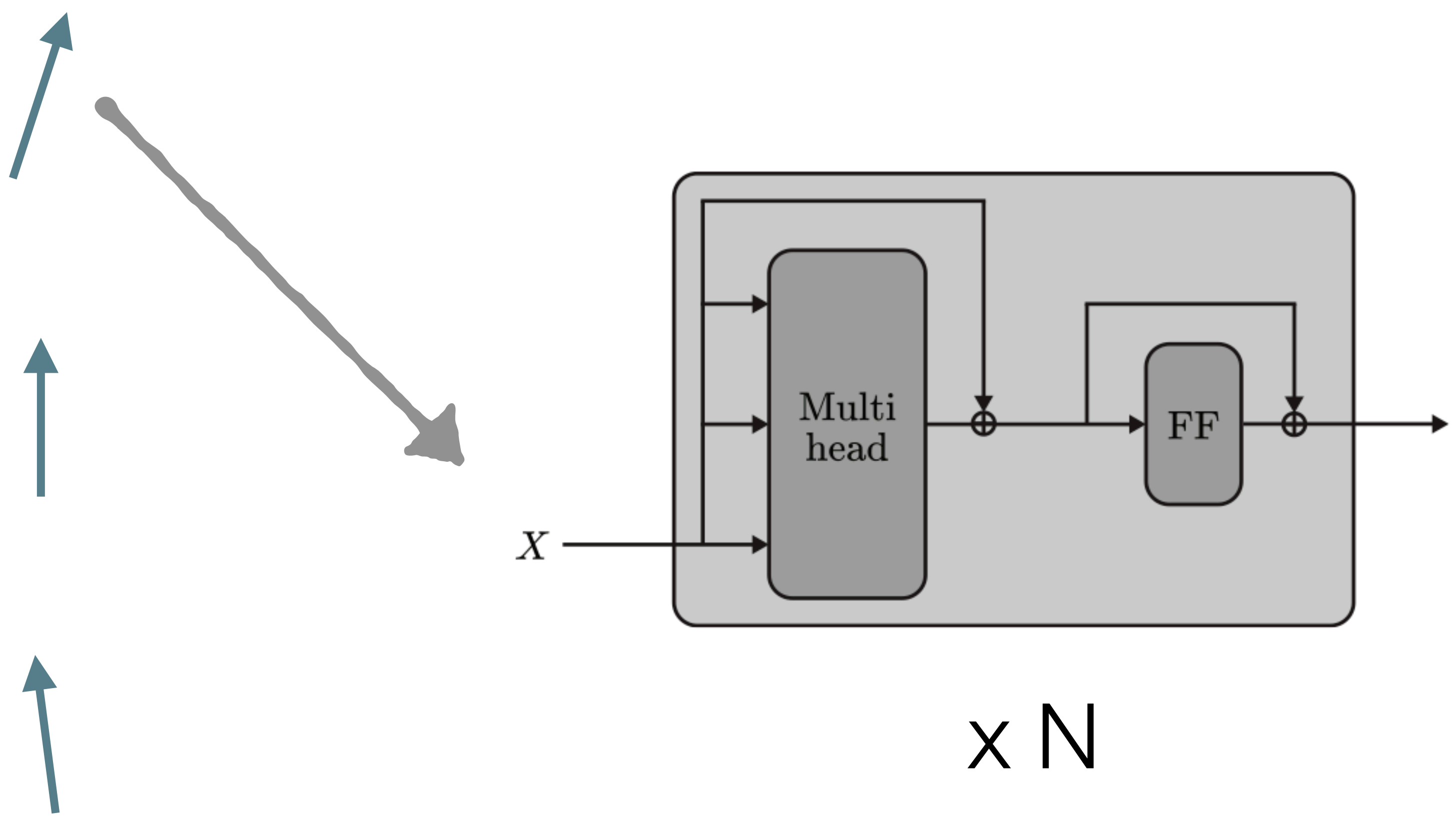
Transformers in DeepSets

- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets



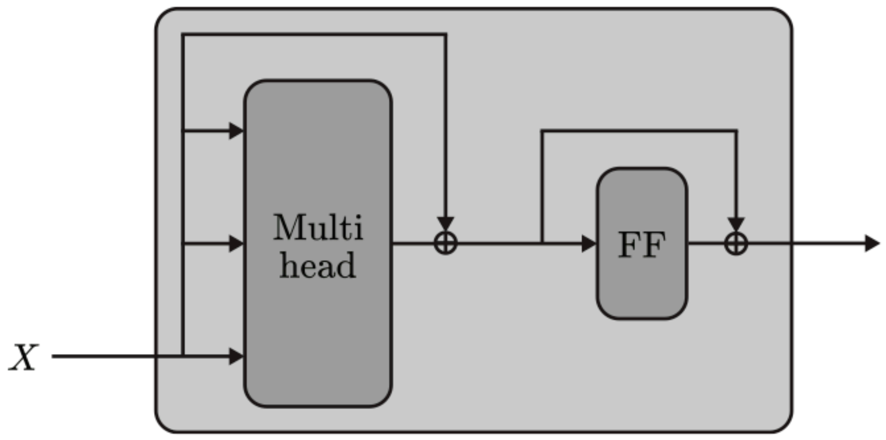
Transformers in DeepSets

- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets



Transformers in DeepSets

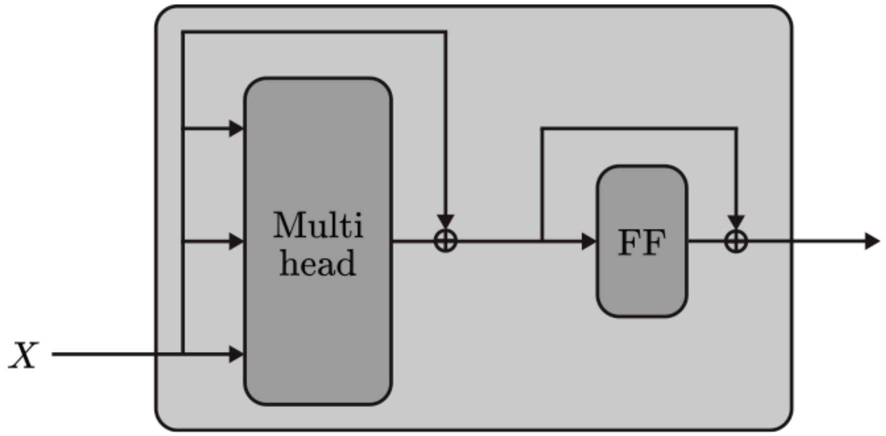
- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets
  - Obvious place to introduce self-attention
  - Relative importance of each particle in the set



- We can replace the Dense / Feedforward layers in ϕ with self attention blocks
- We end up transitioning from DeepSets to [Set Transformer]
  - Although more complex, can be more powerful than GNNs

Transformers in DeepSets

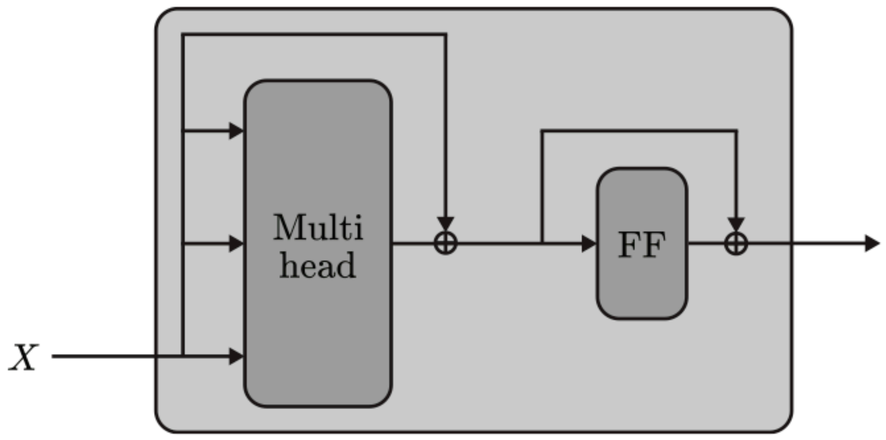
- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets
  - Obvious place to introduce self-attention
  - Relative importance of each particle in the set



- We can replace the Dense / Feedforward layers in ϕ with self attention blocks
- We end up transitioning from DeepSets to [Set Transformer]
  - Although more complex, can be more powerful than GNNs

Transformers in DeepSets

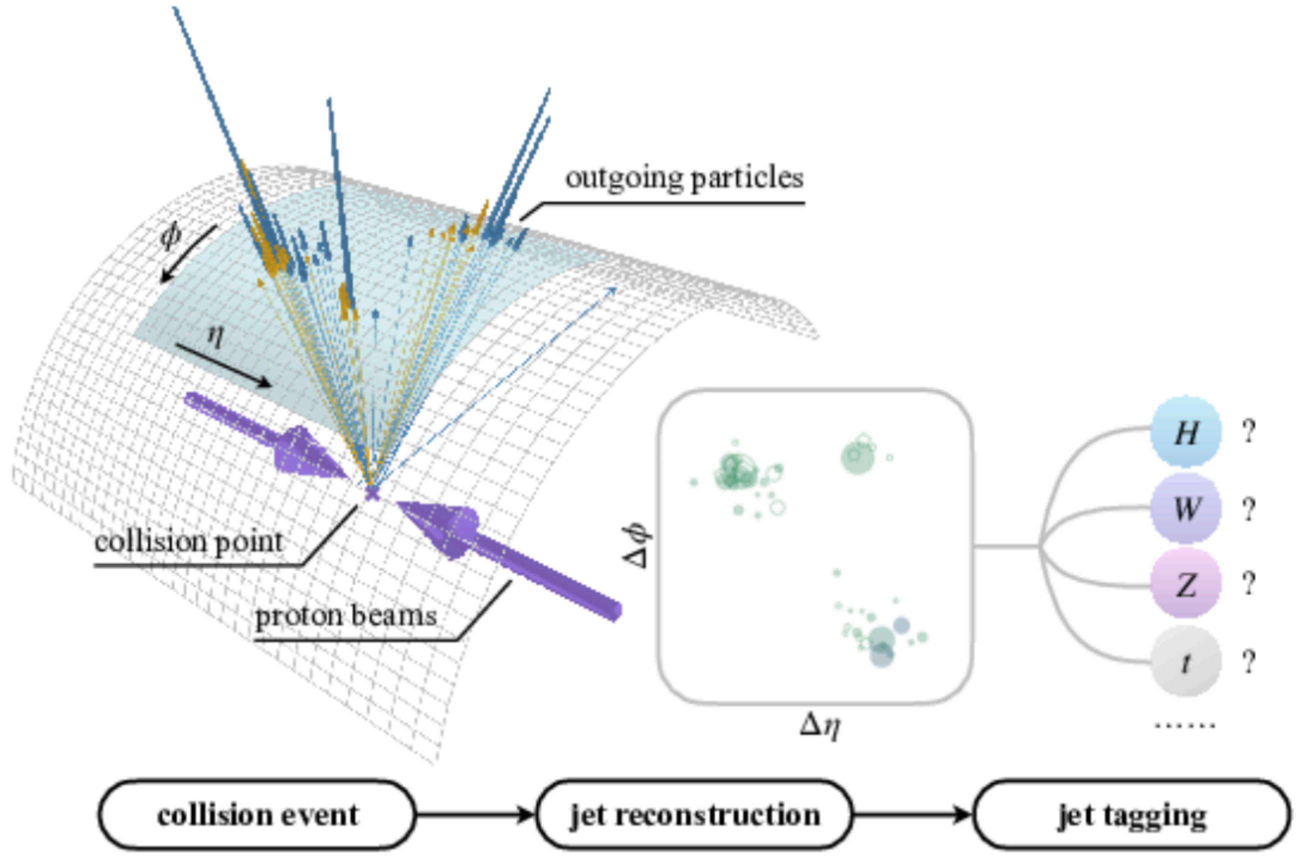
- Recap: DeepSets have NNs F and ϕ
  - But only ϕ NN sees the elements of the sets
  - Obvious place to introduce self-attention
  - Relative importance of each particle in the set

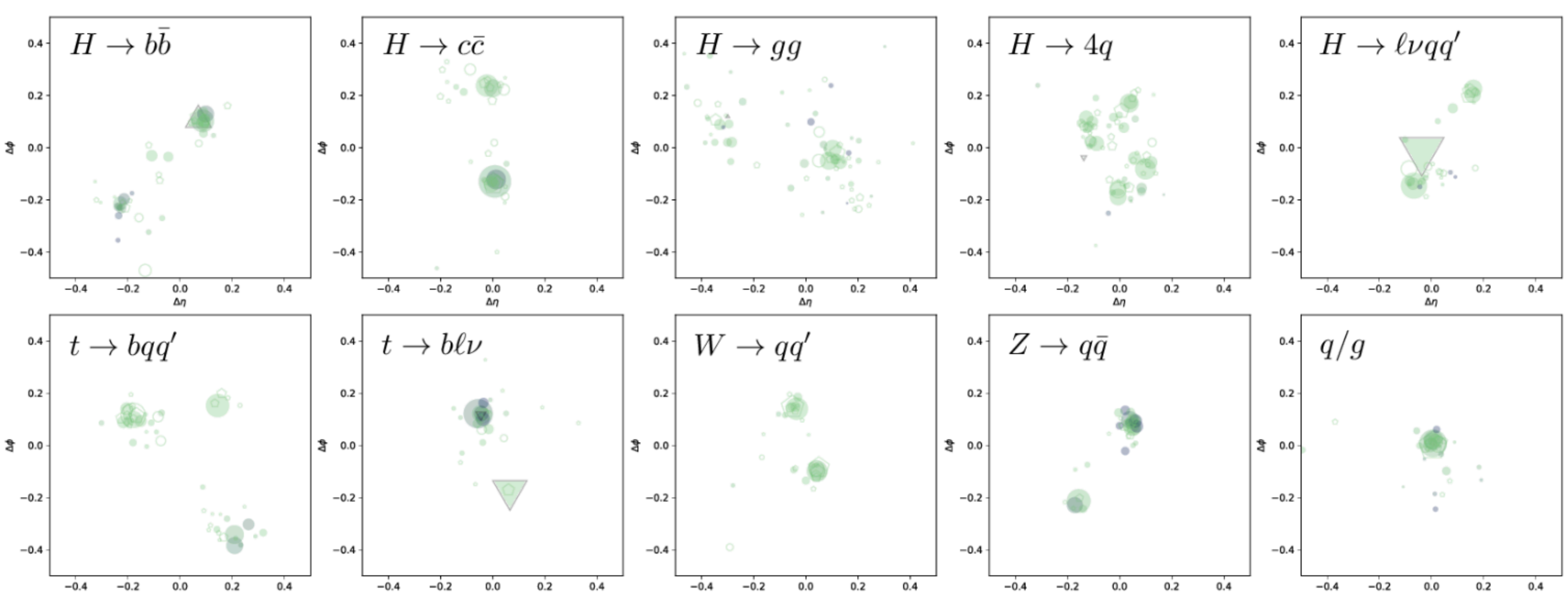


- We can replace the Dense / Feedforward layers in ϕ with self attention blocks
- We end up transitioning from DeepSets to [Set Transformer]
  - Although more complex, can be more powerful than GNNs

Transformers in Particle Physics

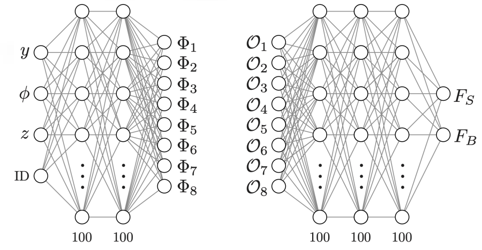
- [“Particle Transformer for Jet Tagging” by Huilin Qu et al. (ICML ’22)]()
- one model for many tasks
- key innovation: incorporating pairwise particle interactions in the attention mechanism
- also demonstrate fine-tuning for other jet tagging datasets





44

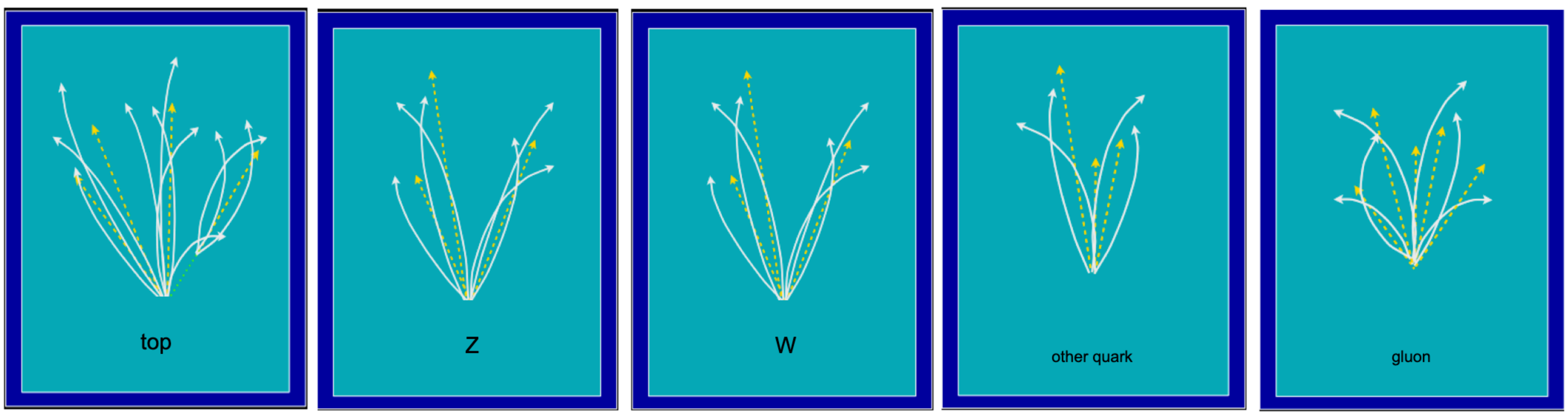

C'mon,
do something...


y
φ
z
ID
Φ1 Φ2 Φ3 Φ4 Φ5 Φ6 Φ7 Φ8
100 100
O1 O2 O3 O4 O5 O6 O7 O8
F_S
F_B
100 100 100

Exercise problem

Fermilab

- Identification of jets arising from hadronization of boosted W/Z/H/top
  - A key and important task in high energy physics
  - Load the reconstructed particles from the decay
    - Use DeepSets to get $f(\mathbb{X}) \rightarrow$ Jet Flavor



$t \rightarrow bW \rightarrow bqq$	$Z \rightarrow qq$	$W \rightarrow qq$	**q/g background**
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

Reconstructed as one massive jet with substructure

Training dataset

Fermilab

- Input:
  - Set of particle inputs from the decay $\mathbb{X}$
- Objective:
  - Tagging the origin of the jet
- Explore the dataset and get the best performance possible !

What to do ?

- Identify the best features architecture for this task
- Optimize the hyper parameters: embedding size, Aggregation method
- Change the architecture, make the network deeper and wider
- Can you Implement IRC safety in model

$$f(\{X_1, X_2, \ldots, X_n\}) = F\left(\sum_i Z_i\, \phi(X_i)\right)$$

  - Z_i is the relative momentum of the particle w.r.t the Jet

Got more time ?

- Play with the Set Transformer

- Rather than simple pooling methods, implement pooling by maximum attention

- Do you get the best performance possible ?