

Building an AI Reconstruction Framework for the PIONEER Experiment

Jack Carlton
University of Kentucky

Outline

- I. What is PIONEER?
 - A. Physics background
 - B. Experimental design
 - C. Simulation framework
- II. The Active Target (ATAR)
 - A. General design philosophy
 - B. Why a “traditional” reconstruction is hard
- III. Principles of Neural Networks and Machine Learning
 - A. Neural networks and back propagation (what the machine learns)
 - B. Hyperparameters and tuning (what the human learns)
 - C. Graph Neural Networks (GNNs)
 - D. Graph Transformers
- IV. AI Reconstruction Pipeline
 - A. Layout
 - B. Data preprocessing/loading
 - C. AI stages

You can find this presentation in my notes

Links:

https://jaca230.github.io/joplin_notes_page/

or

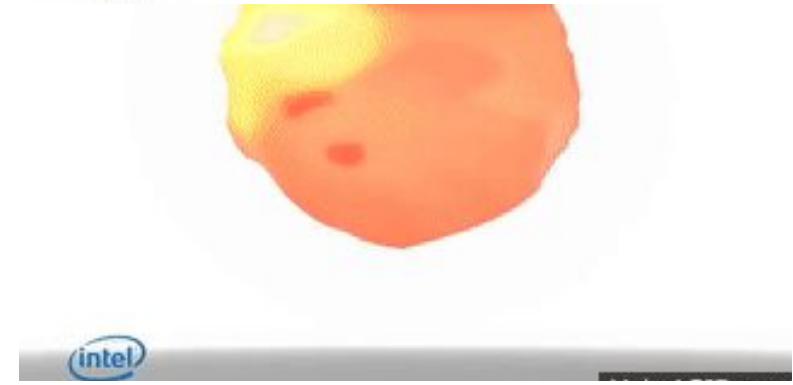
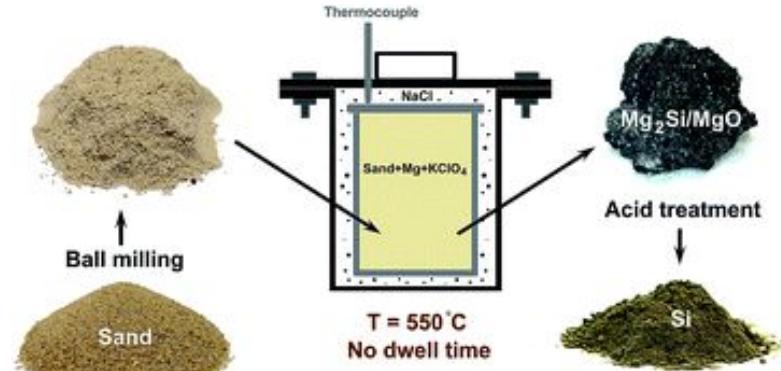
<https://tinyurl.com/jack-uky-notes>



How to Train an AI Model from Scratch

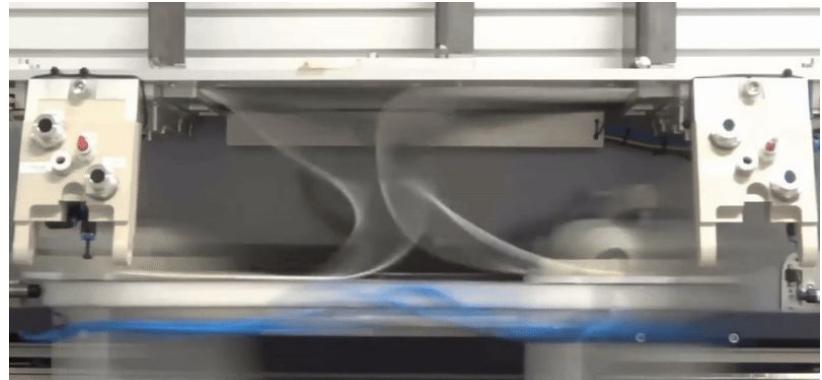
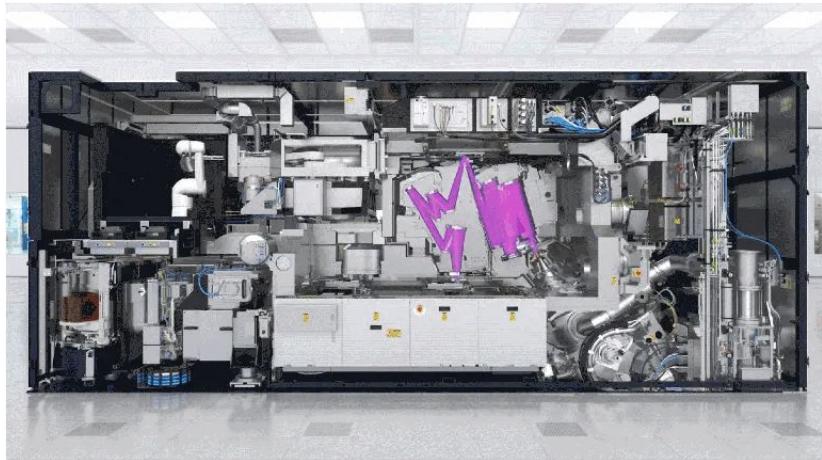
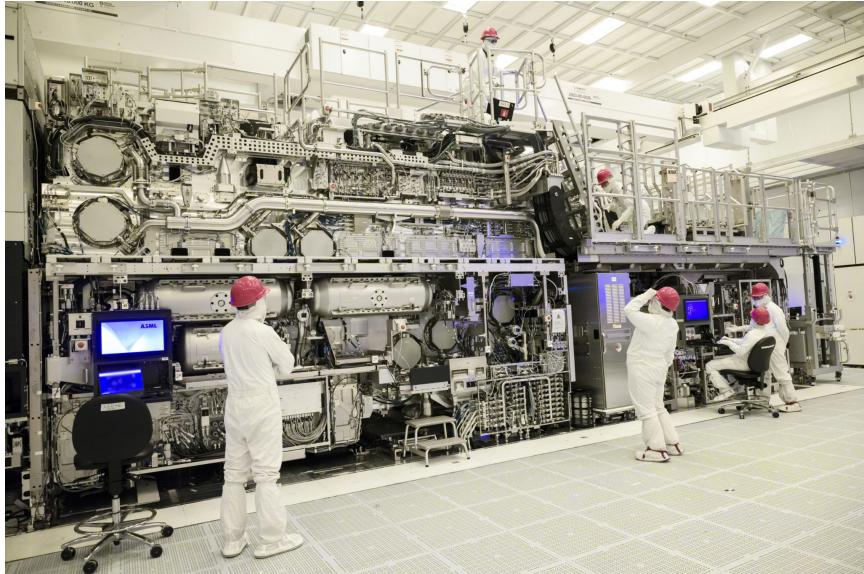
How to Train an AI Model from Scratch

1. Harvest some silicon from the beach



How to Train an AI Model from Scratch

1. Harvest some silicon from the beach
2. **Use whatever this thing is to carve some runes**



How to Train an AI Model from Scratch

1. Harvest some silicon from the beach
2. Use whatever this thing is to carve some runes
3. **Collect training data via legal and accepted practices**



A screenshot of a web browser showing a privacy policy update. The header says "Updates to Our Privacy Policy" with a shield icon. Below it, there's a paragraph about privacy changes and a summary of changes section with a bullet point about improved transparency.

At [REDACTED], we take your privacy seriously. As part of our ongoing commitment to transparency and keeping your data safe, we have updated our Privacy Policy in accordance with the General Data Protection Regulation (GDPR).

Here is a summary of the changes:

- Improved transparency: We've provided more information about the data we



We use cookies 🍪

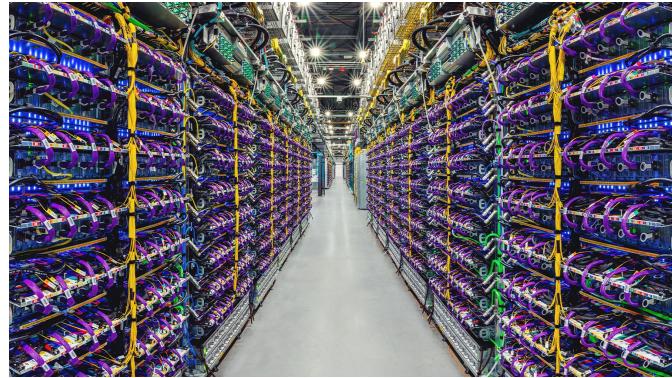
Our websites use cookies (also from third parties) for functional and analytical purposes, and to show you personalised advertisement. You can adjust this in [Cookie Settings](#) or learn more by reading our [cookie policy](#).

ONLY REQUIRED

ACCEPT ALL

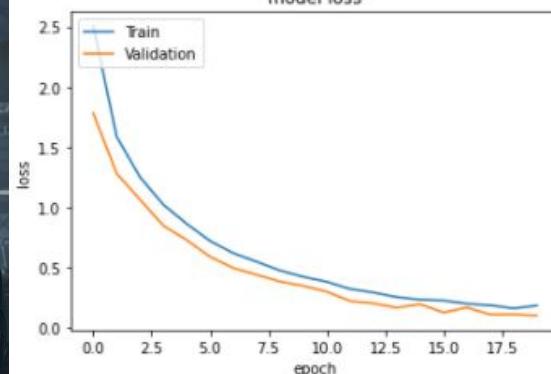
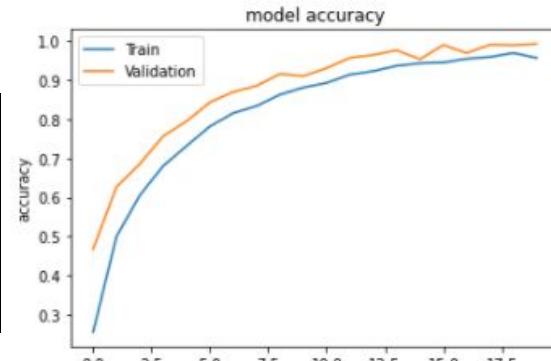
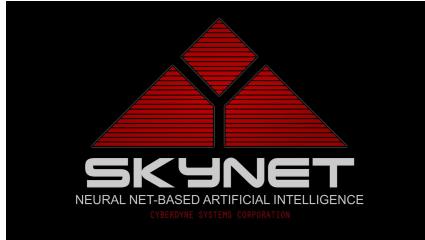
How to Train an AI Model from Scratch

1. Harvest some silicon from the beach
2. Use whatever this thing is to carve some runes
3. Collect training data via legal and accepted practices
4. **Temporarily repurpose the entire power output of a rural town**



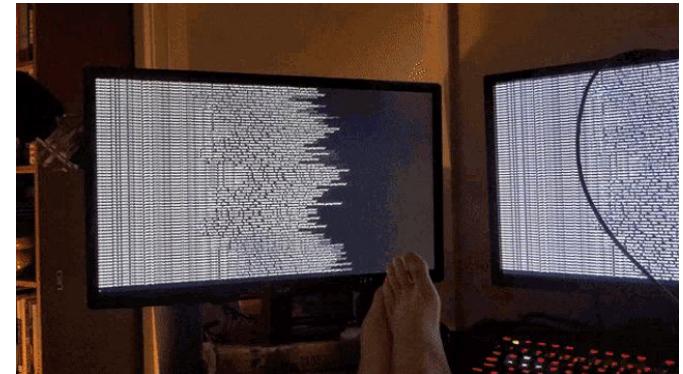
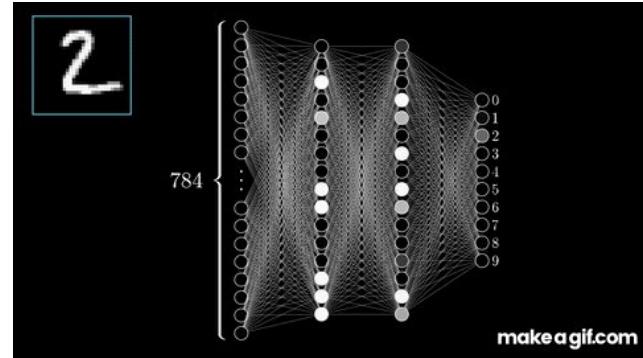
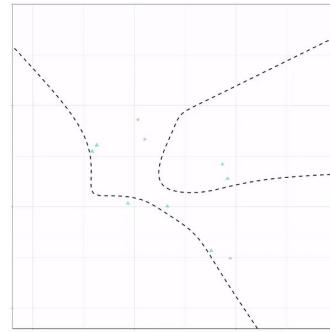
How to Train an AI Model from Scratch

1. Harvest some silicon from the beach
2. Use whatever this thing is to carve some runes
3. Collect training data via legal and accepted practices
4. Temporarily repurpose the entire power output of a rural town
5. **Design a loss function forgetting that the end DOES NOT justify the means**



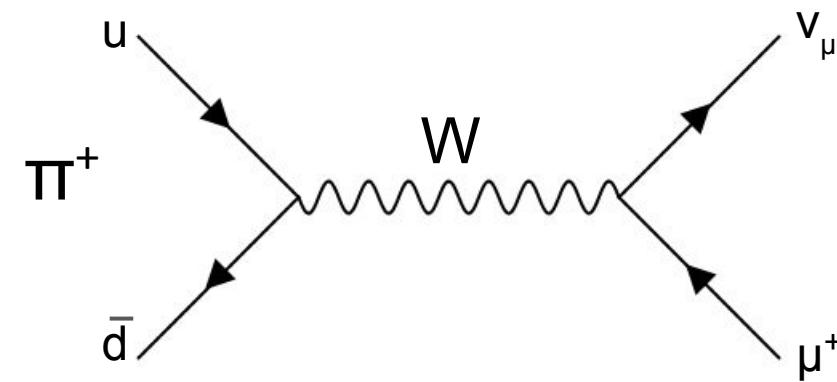
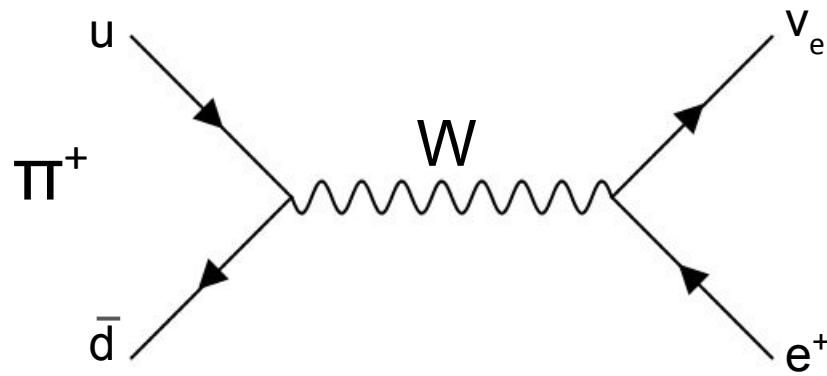
How to Train an AI Model from Scratch

1. Harvest some silicon from the beach
2. Use whatever this thing is to carve some runes
3. Collect training data via legal and accepted practices
4. Temporarily repurpose the entire power output of a rural town
5. Design a loss function forgetting that the end DOES NOT justify the means
6. **Teach your sand to think against its will**

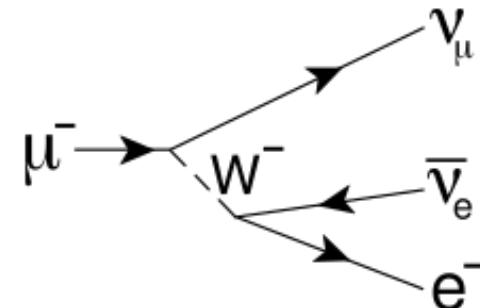


What is PIONEER?

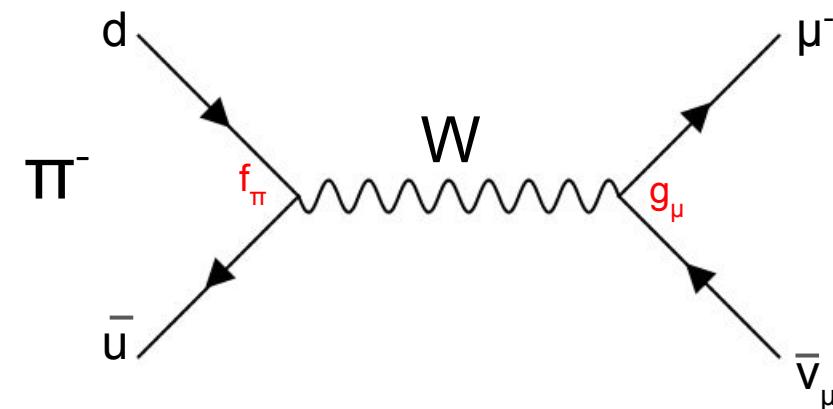
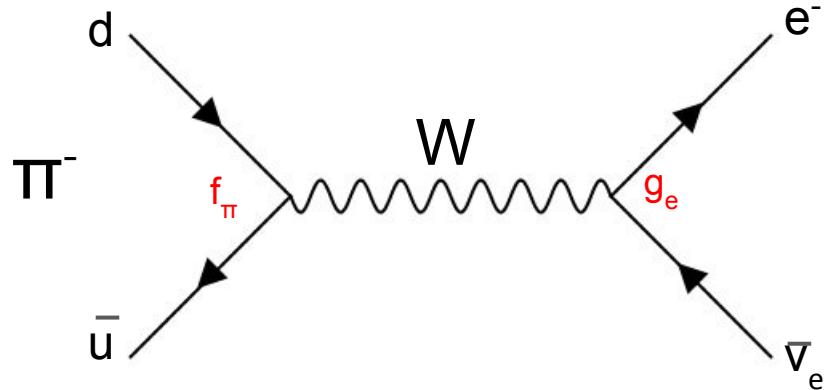
$\pi \rightarrow e v_e$ and $\pi \rightarrow \mu v_\mu$ ($\pi \rightarrow e$ and $\pi \rightarrow \mu \rightarrow e$)



- Corresponding diagrams for π^-
- Tau decay forbidden
 - Tau too massive ~ 1000 MeV/c 2
 - Pion ~ 100 MeV/c 2
- Muon decay more likely
 - Due to helicity suppression
 - Branching fraction of 0.999877
 - Quickly decays into positron



Lepton Universality



- States coupling strengths (vertices) $g_e = g_\mu = g_\tau$
- Using the Feynman rules for the weak interaction, we can approximate the matrix element

$$\mathcal{M}_{fi} = \left[\frac{g_W}{\sqrt{2}} \frac{1}{2} f_\pi p_\pi^\alpha \right] \cdot \left[\frac{g_{\alpha\beta}}{m_W^2} \right] \cdot \left[\frac{g_W}{\sqrt{2}} g_l \bar{u}(p_l) \gamma^\beta \frac{1}{2} (1 - \gamma^5) v(p_\nu) \right]$$

Pion vertex W-boson propagator Lepton vertex

Lepton Universality

- After some “massaging” we can find the matrix element to be

$$\mathcal{M}_{fi} = \left(\frac{g_W}{2m_W} \right)^2 f_\pi g_l \cdot \sqrt{m_\pi^2 - m_l^2}$$

- Pion spin zero \rightarrow no spin averaging needed, i.e.:

$$\langle |\mathcal{M}_{fi}|^2 \rangle = |\mathcal{M}_{fi}|^2 = \left(\frac{g_W}{2m_W} \right)^4 f_\pi^2 g_l^2 \cdot (m_\pi^2 - m_l^2)$$

- We can use the general formula for 2-body decay to find the decay rate

$$\Gamma = \frac{p \langle |\mathcal{M}_{fi}|^2 \rangle}{8\pi m_\pi^2} = \frac{f_\pi^2}{16\pi^2 m_\pi^3} \left(\frac{g_W}{2m_W} \right)^4 [m_l g_l (m_\pi^2 - m_l^2)]^2$$

- Finally, we compute the branching ratio

$$\frac{\Gamma(\pi^- \rightarrow e^- \bar{\nu}_e)}{\Gamma(\pi^- \rightarrow \mu^- \bar{\nu}_\mu)} = \left(\frac{g_e}{g_\mu} \right)^2 \left[\frac{m_e(m_\pi^2 - m_e^2)}{m_\mu(m_\pi^2 - m_\mu^2)} \right]^2$$

Branching Ratio $R_{e/\mu}$

- We can measure the branching ratio by measuring # of decays e and μ decays
- Theoretical prediction is simple in first (and second) order
 - No f_π or CKM element V_{ud}
- 3rd order correction and beyond the pion structure becomes relevant

$$R_{e/\mu} \equiv \frac{\Gamma(\pi^- \rightarrow e^- \bar{\nu}_e)}{\Gamma(\pi^- \rightarrow \mu^- \bar{\nu}_\mu)}$$

$$R_{e/\mu}^0 = \left(\frac{g_e}{g_\mu} \right)^2 \left[\frac{m_e(m_\pi^2 - m_e^2)}{m_\mu(m_\pi^2 - m_\mu^2)} \right]^2$$

$= 1$ [in theory]

$$R_{e/\mu}^{(\text{theory})} = R_{e/\mu}^0 \left(1 - \frac{3\alpha}{\pi} \ln \left(\frac{m_\mu}{m_e} \right) + \dots \right)$$

Current state of $R_{e/\mu}$

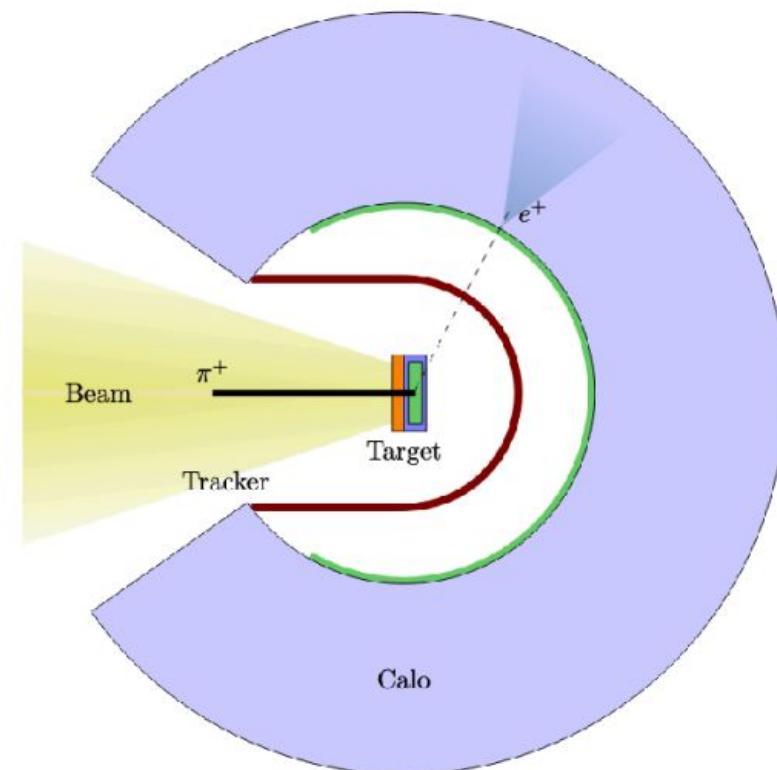
$$R_{e/\mu}^{\text{exp}} = 1.2327(23) \times 10^{-4} \text{ (PIENU collab)}$$

$$R^{\text{theo}} = 1.23524(15) \times 10^{-4}$$

- Consistent with each other
- Expect factor of ~ 10 precision improvement on experimental value from PIONEER
 - “Catches up” with theoretical uncertainty

PIONEER Experimental Proposal

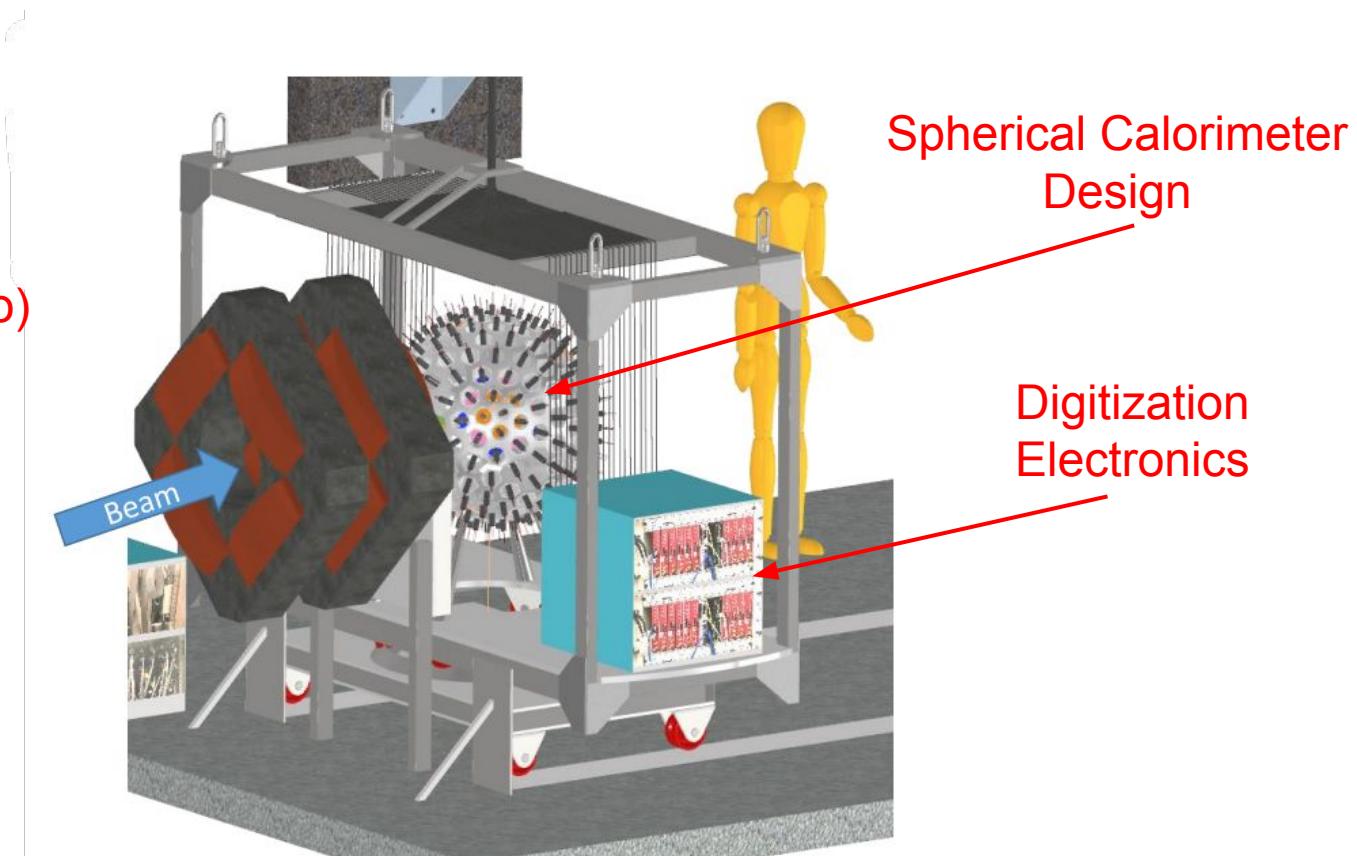
- LXe (or LYSO) as a calorimeter
material has short decay time and
good energy resolution
 - $\sigma_t \approx 25$ ns (timing resolution)
 - $\Delta E/E \approx 2\%$ (energy resolution)
- Allows experiment to run at much
higher rate than previous
experiments
 - ~ 300 kHz (phase 1)
 - ~ 2000 kHz (phase 2 and 3)
- “active target”, muons and pions
are “tracked” while being stopped



3D Render Experiment

Not Pictured:

- ATAR (inside Calo)
- Tracker (a shell around ATAR, inside Calo)
- VETOs, T0, etc.
- DAQ Computers



PIONEER Strategy

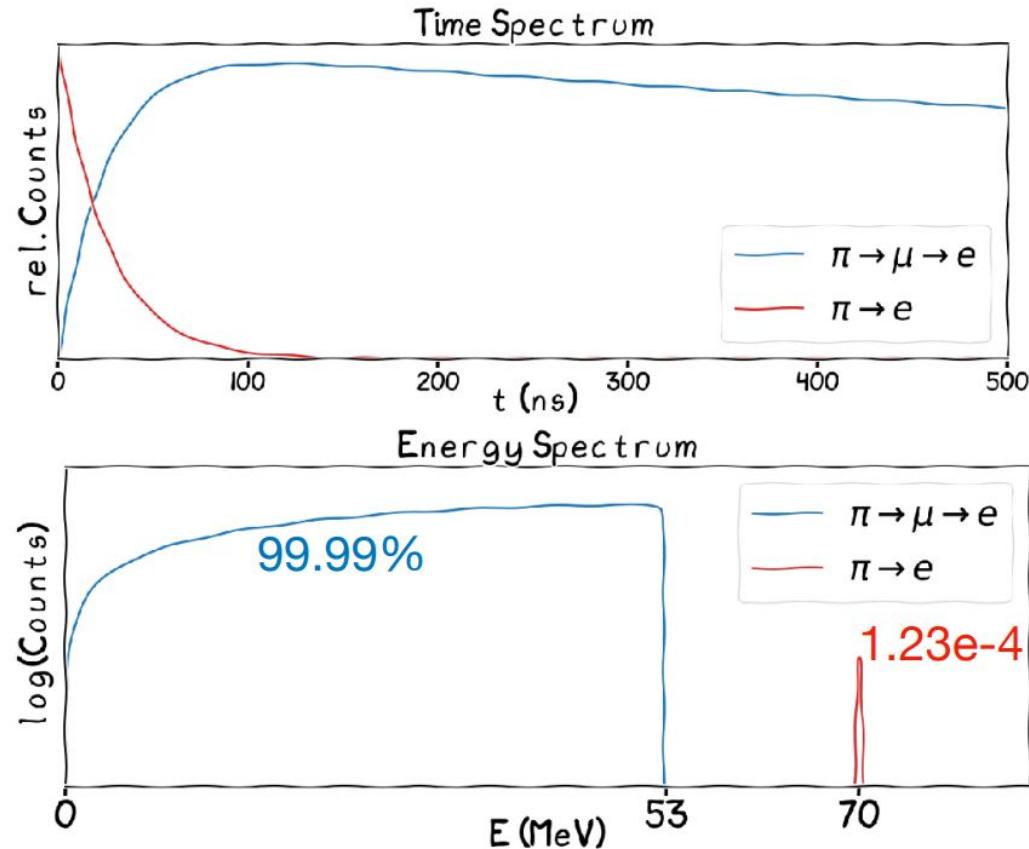
- Calorimeter measures just positron energy
 - Muon stopped by target
 - Neutrinos invisible
- Prompt 70 MeV = $\pi \rightarrow e$**
- Delayed <53 MeV = $\pi \rightarrow \mu \rightarrow e$**
- Plots are for idealized, perfect detectors (in a “simple world”)

$$\tau_{\pi^+} = 26 \text{ ns}, m_{\pi^+} \approx 140 \text{ MeV}$$

$$\tau_{\mu^+} = 2197 \text{ ns}, m_{\mu^+} \approx 106 \text{ MeV}$$

$$\pi^+ \rightarrow \mu^+ \nu \quad E_e < m_\mu/2 \approx 53 \text{ MeV}$$

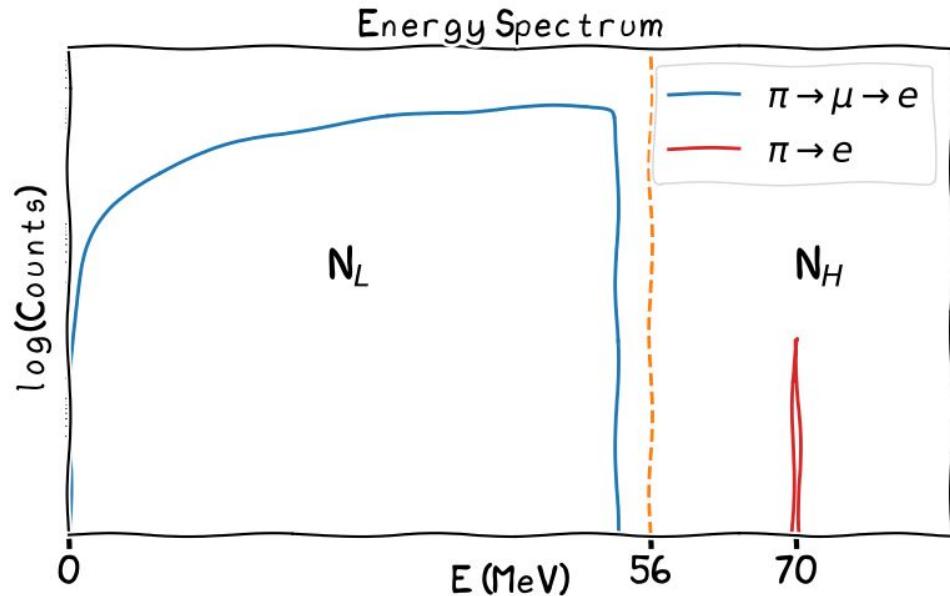
$$\pi^+ \rightarrow e^+ \nu \quad E_e = m_\pi/2 \approx 70 \text{ MeV}$$



PIONEER Strategy

- At a glance the experiment is simple
- Just look at energy deposited on the calorimeter and count events above and below $E_{\text{thr}} \equiv 56 \text{ MeV}$
- In reality, not so simple...

$$R_{e/\mu} = \frac{\Gamma(\pi^+ \rightarrow e^+ \nu(\gamma))}{\Gamma(\pi^+ \rightarrow \mu^+ \nu(\gamma))} \approx \frac{N_H}{N_L}$$

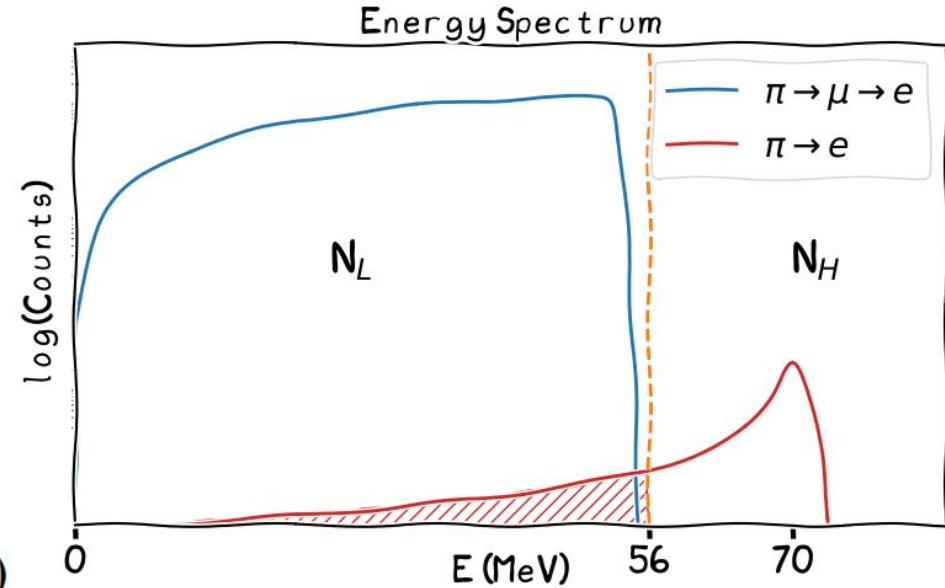


PIONEER Strategy

- Finite resolution and incomplete energy collection in the calorimeter result in a tail
- To compete with theoretical uncertainty, we need to characterize this tail to ~1 part in 10,000 ($\frac{\delta\Gamma_e}{\Gamma_e} \sim 10^{-4}$)

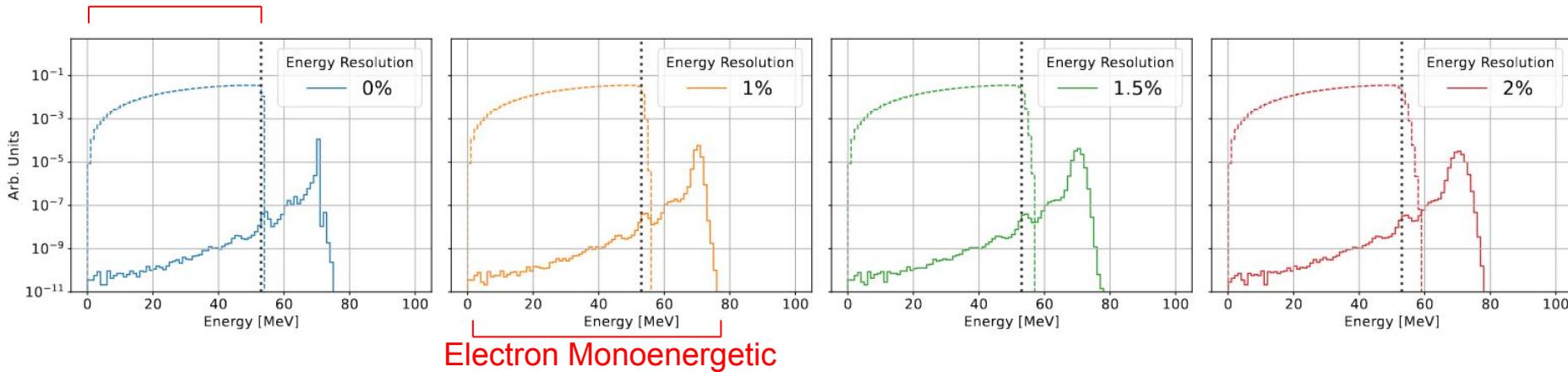
$$R_{e/\mu} = \frac{\Gamma(\pi^+ \rightarrow e^+ \nu(\gamma))}{\Gamma(\pi^+ \rightarrow \mu^+ \nu(\gamma))} \approx \frac{N_H}{N_L} \times (1 + c_T)$$

↑
Tail Correction
 $c_T \approx 5 \times 10^{-3}$



Calorimeter (CALO) Purpose

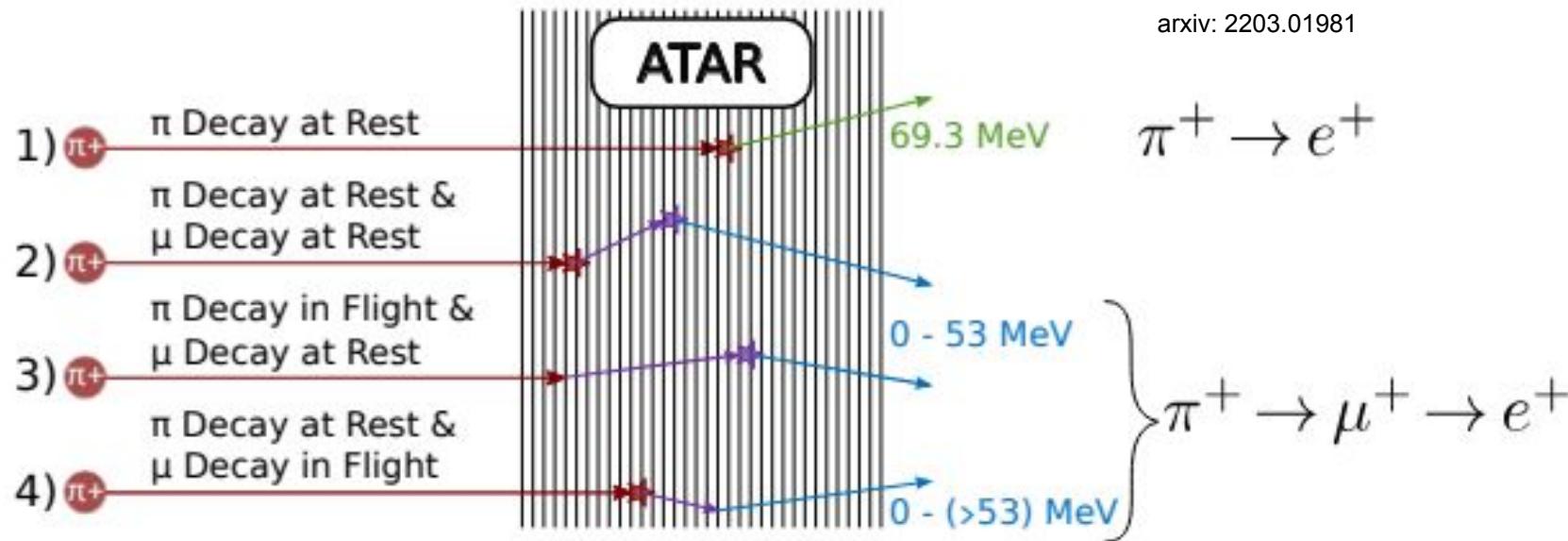
Muon Michelle



- Muon's intrinsically follow a Michelle Spectrum
 - Additionally width comes from energy resolution
- Positrons follow monoenergetic spectrum
 - “Back” and “Front” tail due to radiative decays and bremsstrahlung
- Good energy resolution is crucial for event reconstruction

Active Target (ATAR) Purpose

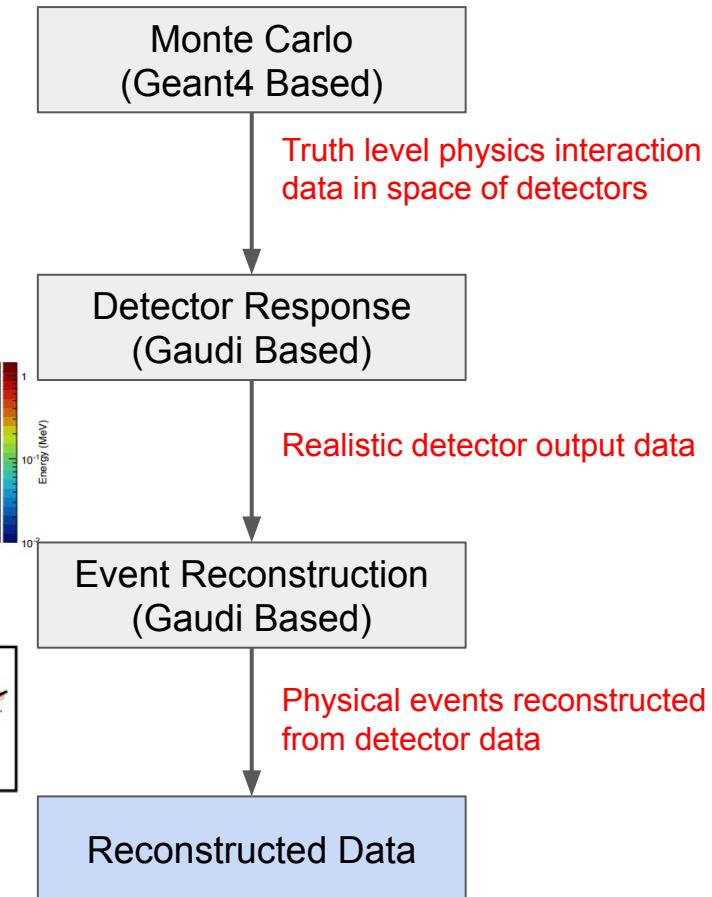
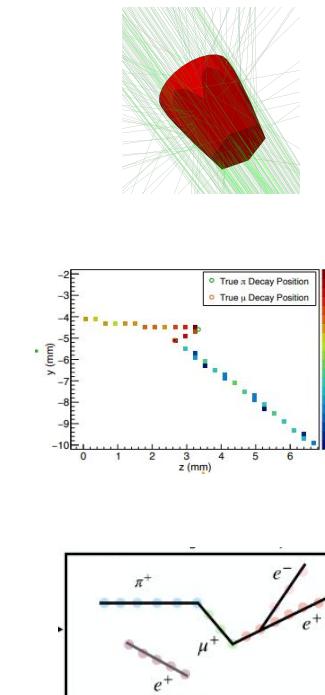
arxiv: 2203.01981



- ATAR allows classifying between $\pi \rightarrow e$ and $\pi \rightarrow \mu \rightarrow e$ based on 5D (x, y, z, t, E) information
 - Worse energy resolution than calorimeter, cannot do experiment on its own
- Case 5. π and μ decay in flight negligible
 - ATAR designed to stop heavy “slow” particles within their decay time; decay in flights are

PIONEER Simulation Framework

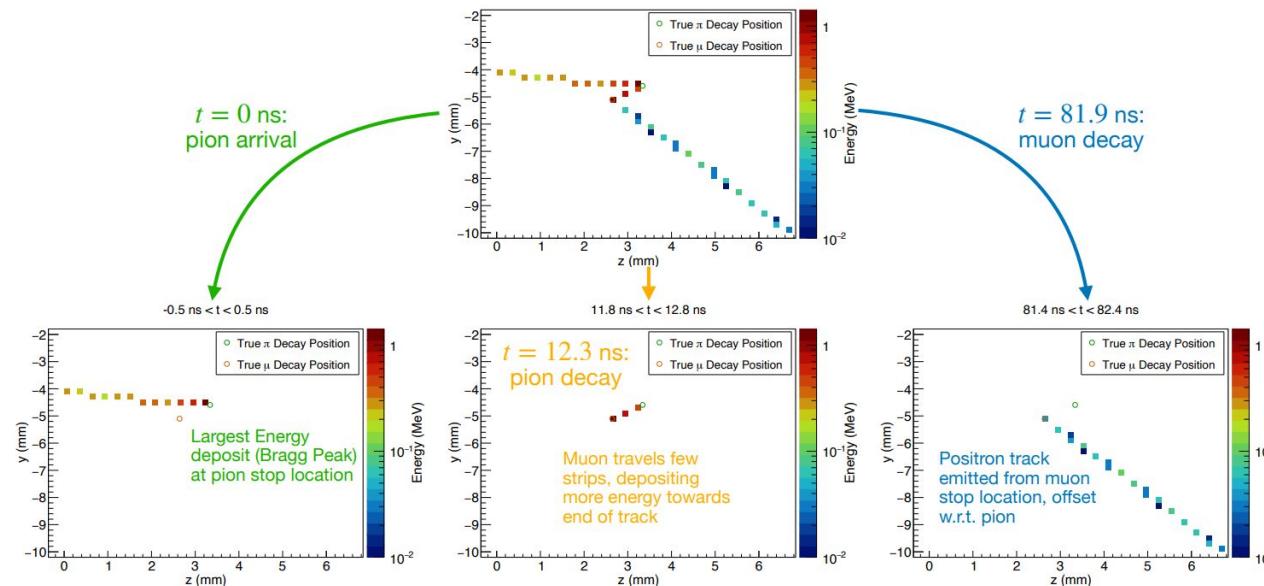
- Series of software steps to simulate the PIONEER experiment
 - Work in progress
 - Adapts as we develop our detectors/strategy
- Designed as a proof of concept
 - Experiment should work in simulation before real experiment is run
- Reconstruction designed to be used on simulated and real detector data



The Active Target (ATAR)

Why Do We Need an ATAR?

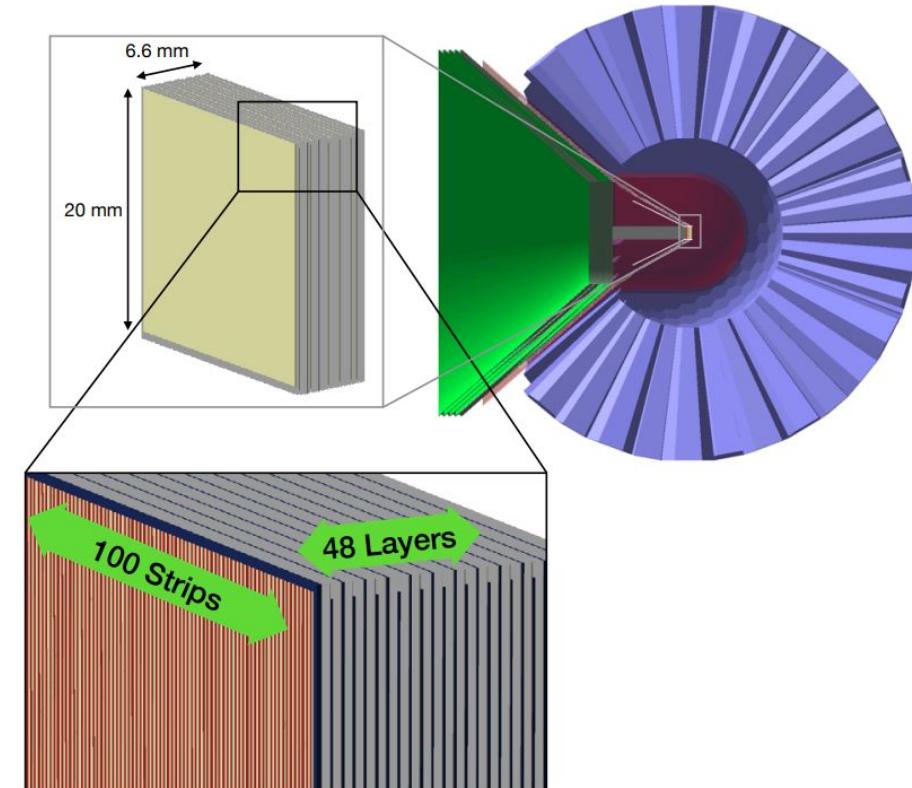
Example of pristine $\pi \rightarrow \mu \rightarrow e$ event data in ATAR



- Energy information alone is insufficient to characterize the $\pi \rightarrow e$ positron tail at the 10^{-8} level in $R_{e/\mu}$
- ATAR topology and timing are required to identify intermediate muons and correct for this effect

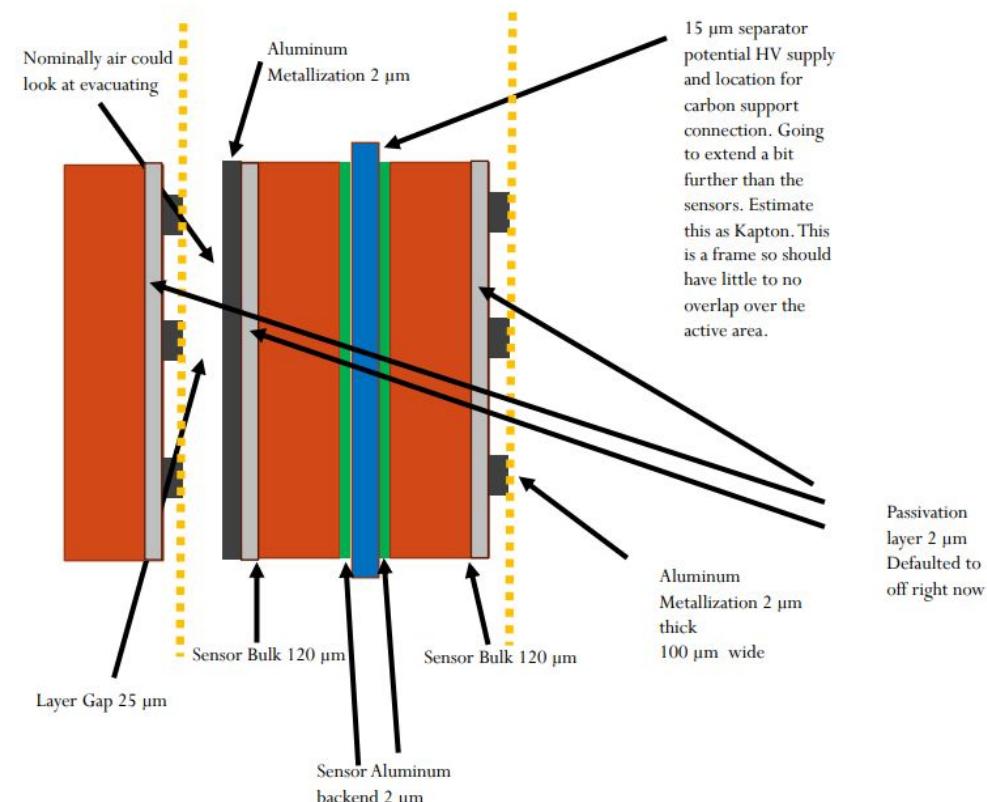
ATAR Design

- Highly segmented and instrumented pion stopping target made of silicon low gain avalanche diodes (LGADs)
- Each sensitive strip has a size of 20 mm x 200 μm x 120 μm
- Total of 4800 channels arranged in criss-crossing pattern
- 5D track information:
 - Transverse positions x,y ~ O(200 μm)
 - Longitudinal position z ~ O(120 μm)
 - Time t ~ O(100 ps) per single hit
 - Energy E, $\Delta E/E$ ~ O(10%)

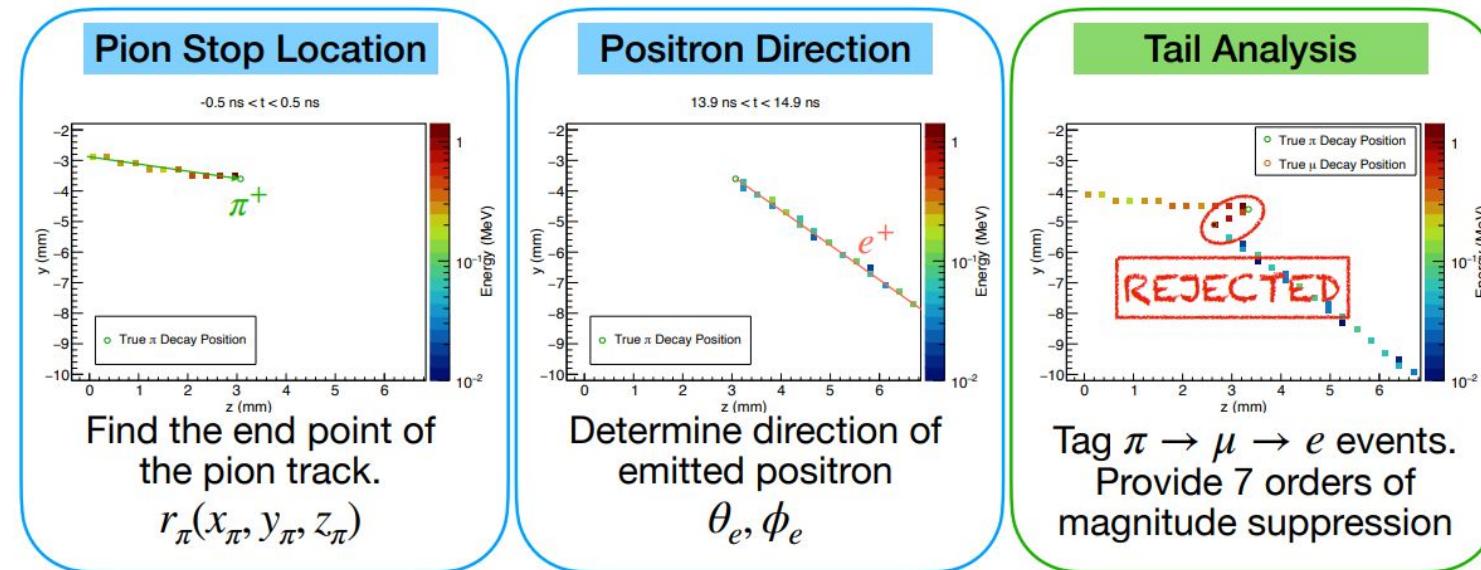


ATAR in Simulation

- Geometric descriptions of ATAR in files fed to Geant4 based simulation
 - Geant4 provides simulation of particles interacting with the detector
 - Detector response code turns those interactions into data we'd imagine seeing from the detector
- 24 layers of what's in the yellow dashed lines in the figure
 - Pairs of x y layers in dashed lines, meaning 48 sensor layers



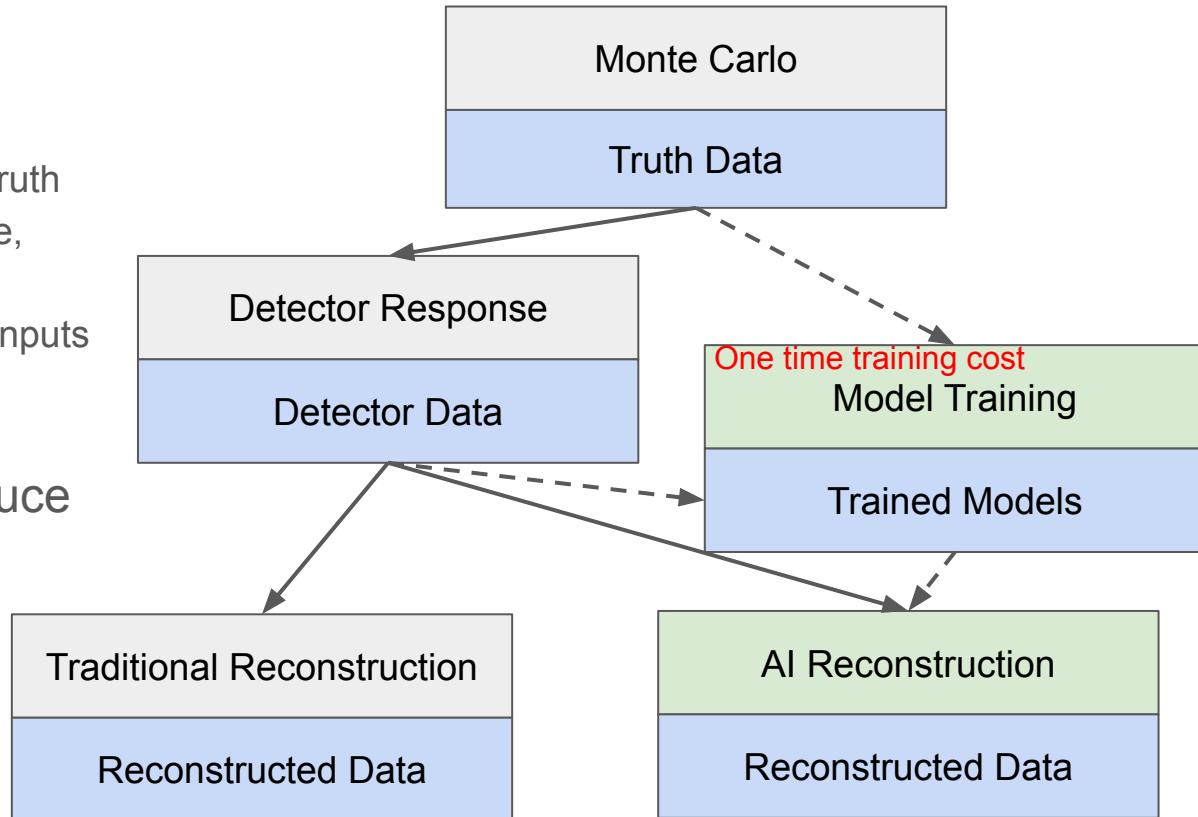
Why ATAR Events Are Hard to Reconstruct Traditionally



- Precise pion stop location and positron direction are essential for rebuilding events in our detector geometry
- Must be very efficient at rejecting $\pi \rightarrow e$ events that are actually $\pi \rightarrow \mu \rightarrow e$
 - $\pi \rightarrow \mu \rightarrow e$ decays are $\sim 10^4$ times more frequent than $\pi \rightarrow e$, and $R_{e/\mu}$ must be measured to $\sim 10^{-4}$ relative precision, the probability for a $\pi \rightarrow \mu \rightarrow e$ event to be misidentified as $\pi \rightarrow e$ must be suppressed to the $\sim 10^{-8}$ level
 - Timing and energy information help us so we need $\sim 10^{-7}$ suppression from tagging

Why Reconstruction Can be an AI task

- Simulation provides all needed information
 - Geant4 simulation gives truth targets (ex. Positron angle, true pion stop)
 - Detector response gives inputs (ex. ATAR strip hit 5D information)
- The simulation can produce large quantities of data needed for training



Principles of Neural Networks and Machine Learning

What is a Neural Network?

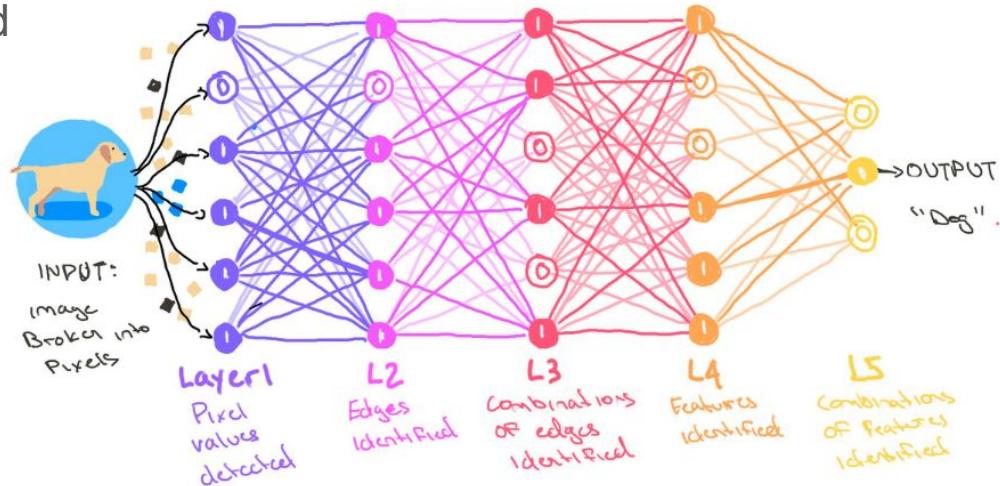
- A neural network is a parameterized function that approximates non-linear mappings through stacked linear transformations and nonlinear activations.
- A composition of function layers

$$f(\cdot; \theta) = f_L \circ f_{L-1} \circ \cdots \circ f_1$$

- Parameters (weights and biases) varied to optimize arbitrary loss function i.e. "learned"

$$f_\ell(\mathbf{x}; \theta_\ell) = \sigma_\ell(\mathbf{W}_\ell \mathbf{x} + \mathbf{b}_\ell)$$

where $\theta_\ell = \{\mathbf{W}_\ell, \mathbf{b}_\ell\}$

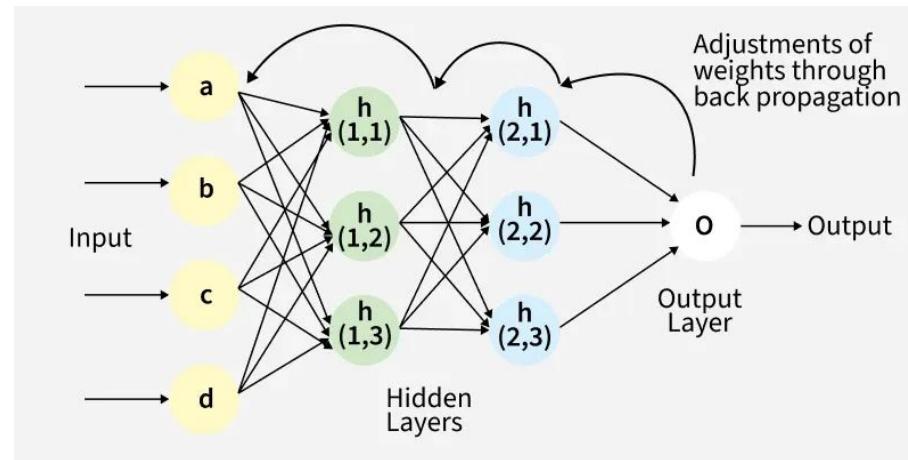


Example annotated multi layered neural network
used to classify images of dogs

This is a specific type of neural network called a multi layer perceptron (MLP), the simplest neural network

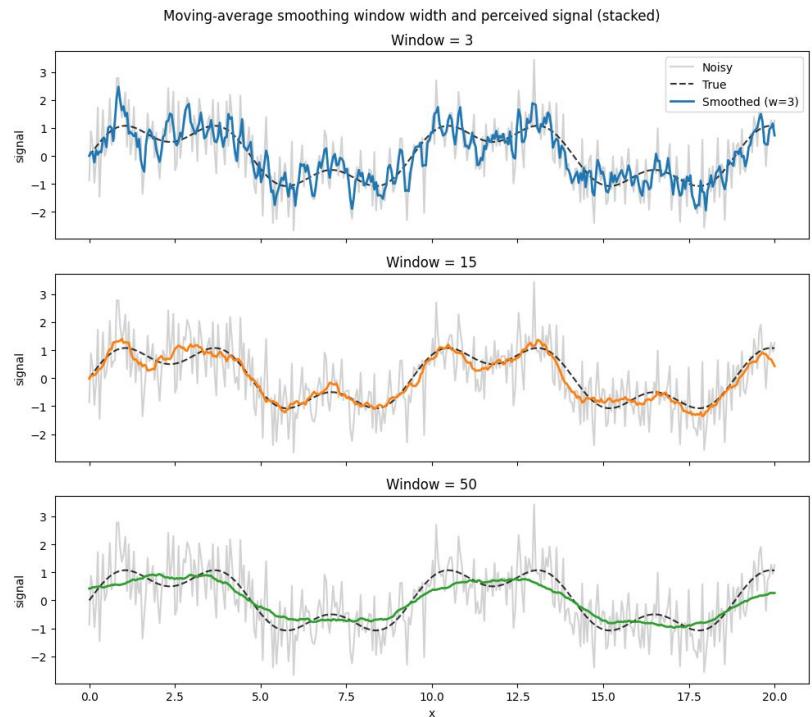
Backpropagation (Overview)

- [Much more detail in this article](#), and [wikipedia](#)
- **Backpropagation computes gradients of the loss with respect to all model parameters by applying the chain rule backward through the network**
- Compute the gradient of the loss in the space of weights
 - Optimal weight updates for the next iteration



What are Hyperparameters

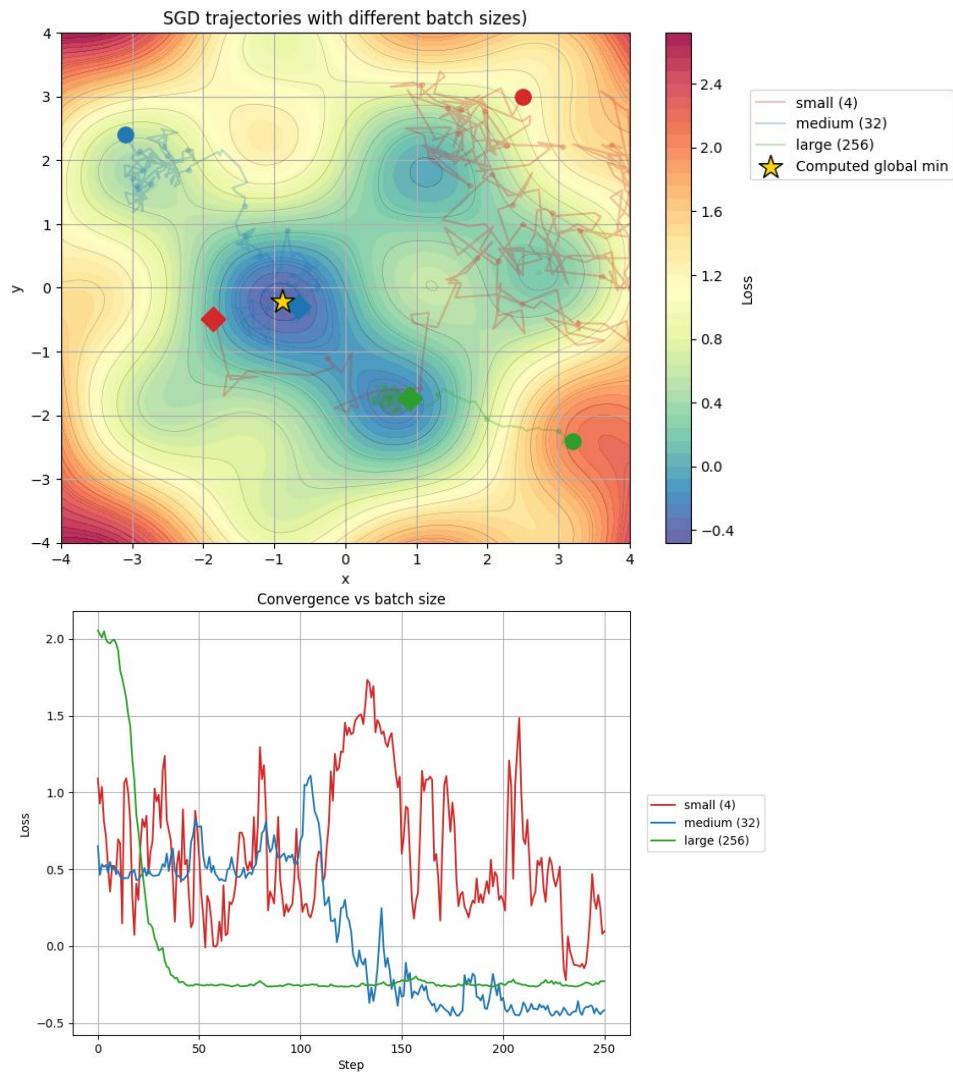
- “A **Hyperparameter** is a parameter that can be set in order to define any configurable part of a model's learning process”
 - These are parameters that are not learned by the model
 - Chosen before the model is trained
- Examples of what they control
 - Optimization dynamics (learning rate, batch size, optimizer)
 - Model capacity (hidden size, depth, heads)
 - Regularization (dropout, weight decay)



Example: Moving average window size is a hyperparameter. It can be over, or under tuned.

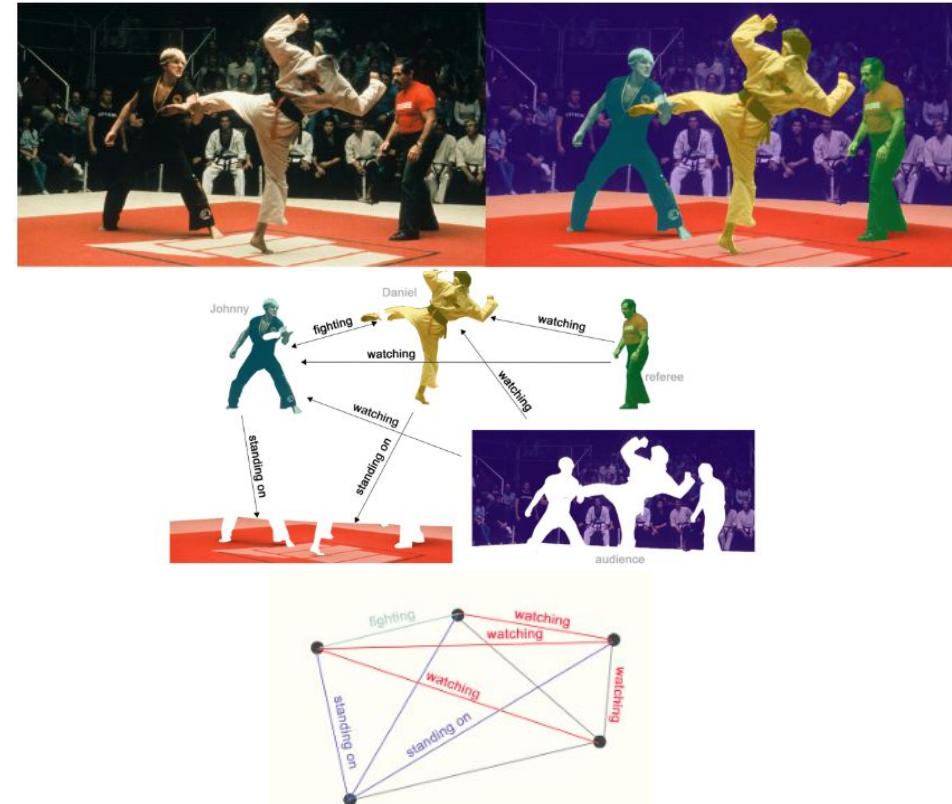
Hyperparameter Example (Batch Size)

- [More detail in this article](#)
- Batch size is how many training data points are processed before updating the model
 - Makes predictions on batch, uses error to inform gradient descent algorithm
- Larger batch size
 - More accurate gradient descent
 - Fewer updates per epoch
 - Less noisy gradient
 - Less general, can get stuck in local minima
- Small batch size
 - Less accurate gradient descent
 - More updates per epoch
 - Noiser gradient
 - Can escape local minima



What is a Graph Neural Network (GNN)?

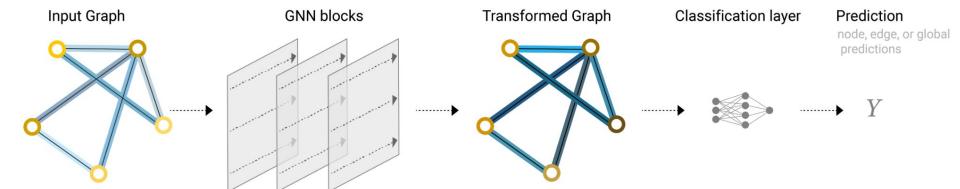
- A graph neural network is a neural network defined on graph-structured data
- Data represented by
 - Nodes (objects, e.g. particle hits in detector)
 - Edges (relationships, e.g. spatial proximity of two hits in detector)
 - Globals (property of group, e.g. total energy)
- Much better explained in this article



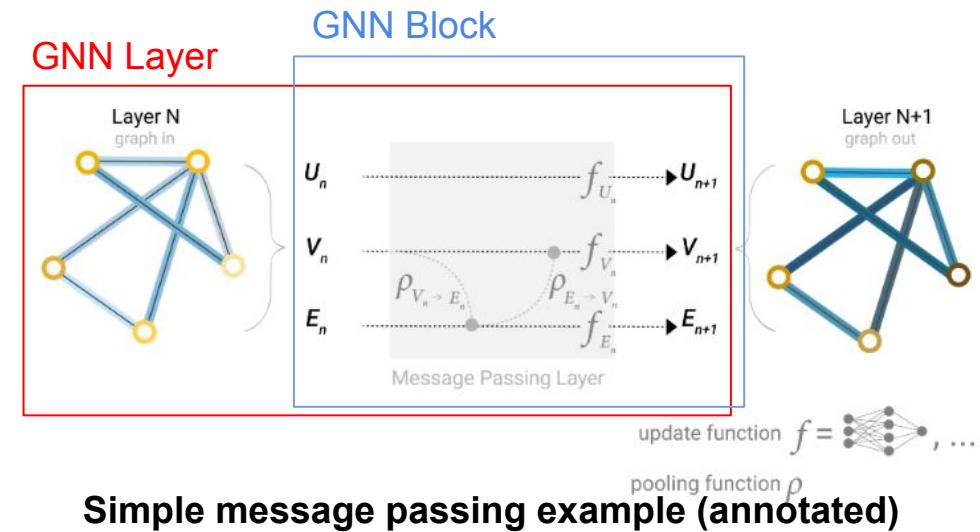
Example of data turned into a graph input

What is a Graph Neural Network (GNN)?

- We input a graph through GNN layers which define *how information flows*
- Example GNN layer strategies are:
 - Independent updates (no connectivity)
 - Apply learned functions to nodes, edges, globals independently
 - Message passing (most common, many versions)
 - Edges gather information from connecting nodes
 - Edges compute a “message” for each neighboring node
 - Nodes gather messages from edges
 - Graph is updated
- GNN Models have 3 basic type of predictions:
 - Node level predictions
 - Edge level predictions
 - Graph level predictions
- Update functions are MLPs

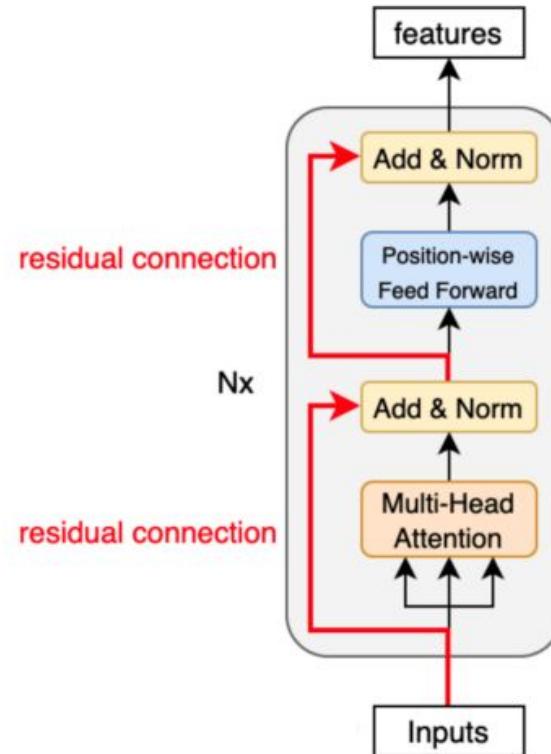


Example of full GNN model



What is a Transformer Encoder Block?

- A transformer encoder block is a neural network layer that
 - A neural network block that operates on a set of feature vectors
 - Updates each vector using information from related vectors
 - Produces context-aware features of the same shape
 - Can be stacked to build progressively larger context
 - This is because the model computes residuals or “corrections”, not replacements
- In the context of graphs
 - updates node embeddings using information from neighboring nodes and edges
- Analogy:
 - MHSAs = “Talk to other nodes”
 - FFNs = “Reflect privately”



What is Multi-Head Self-Attention (MHSA)?

- Self-attention lets each element decide which other elements matter and how strongly to combine them
- Multiple heads let the model learn different interaction patterns at the same time

x_i : node embedding (hit-level features)

q_i : what this node is looking for

k_j : what neighboring nodes offer

v_j : what information neighbors contribute

$$x'_i = \sum_{j \in \mathcal{C}(i)} \alpha_{ij} v_j$$

$$\alpha_{ij} = \frac{\exp\left(\frac{q_i^\top k_j}{\sqrt{d_k}}\right)}{\sum_{j' \in \mathcal{C}(i)} \exp\left(\frac{q_i^\top k_{j'}}{\sqrt{d_k}}\right)}$$

$$q_i = W_Q x_i, \quad k_j = W_K x_j, \quad v_j = W_V x_j$$

$\mathcal{C}(i)$: set of elements that can interact with i
(for graphs: nodes connected to i)

What is a Feed Forward Network (FFN)?

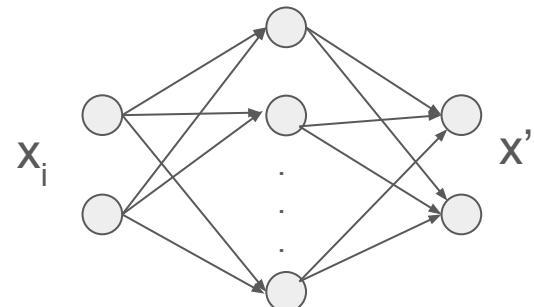
- A position-wise feed-forward network applied to each element independently
- Uses the same two-layer MLP for all elements (shared weights)
- No communication between elements occurs in this layer
- Adds nonlinear feature refinement after self-attention
- Aside:
 - [Feed Forward Networks](#) are more general than position wise FFNs

$$x'_i = \text{FFN}(x_i)$$
$$\text{FFN}(x) = W_2 \sigma(W_1 x + b_1) + b_2$$

x_i : element / node

σ : nonlinear activation (ReLU, GELU)

W_1, W_2 : learned linear layers

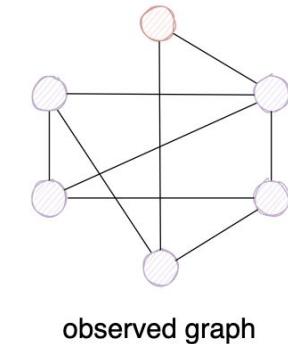
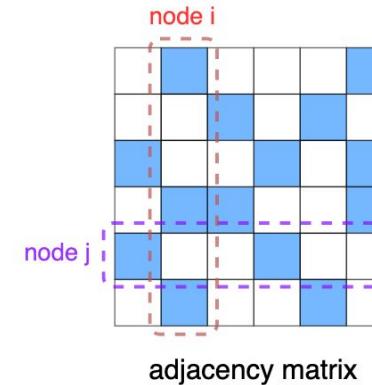


Example FFN for node with 2 features. Same FFN is applied to all nodes independently

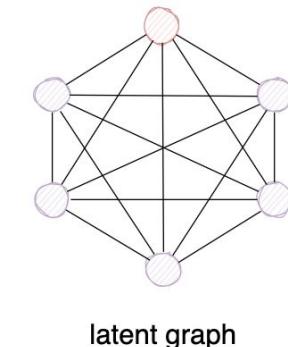
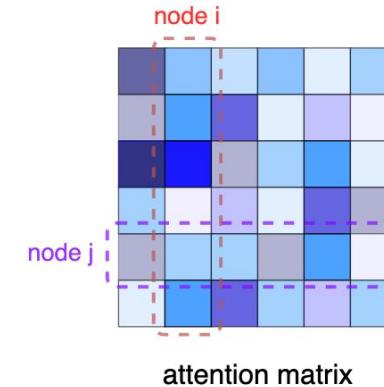
What is a Graph Transformer?

- A graph transformer is a GNN that generalizes message passing using transformer-style attention
- These are among the most expressive modern ML models
 - Can solve a wider class of problems than most models
- GNN
 - Edges define where information flows
 - Message passing same “strength” for all edges
- Graph transformers
 - Attention learns how strongly information flows along those edges.
 - Message passing differs in learned “strength” for different edges

Graph Neural Networks

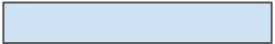


Transformers

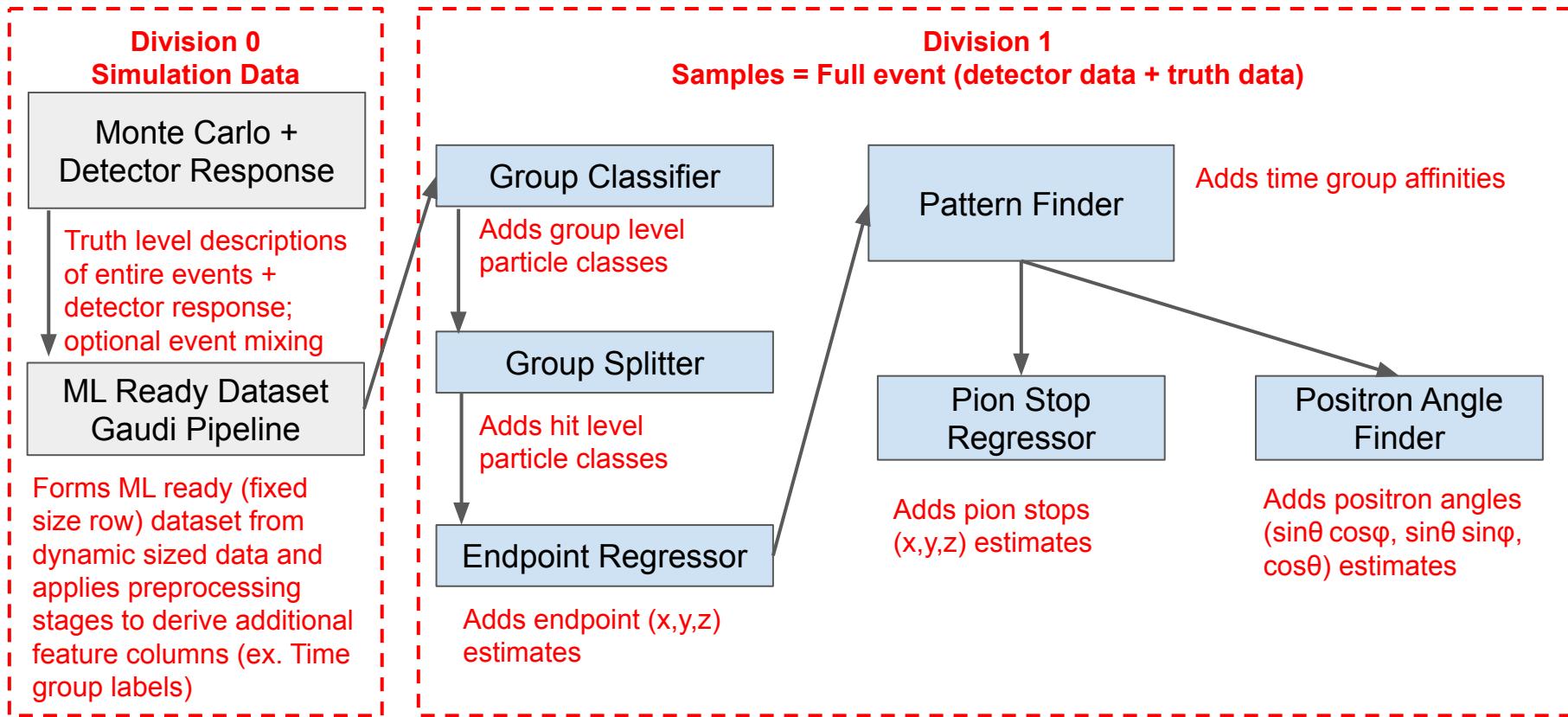


Comparison of GNNs and Transformers in terms of message passing over different structures

AI Reconstruction Pipeline

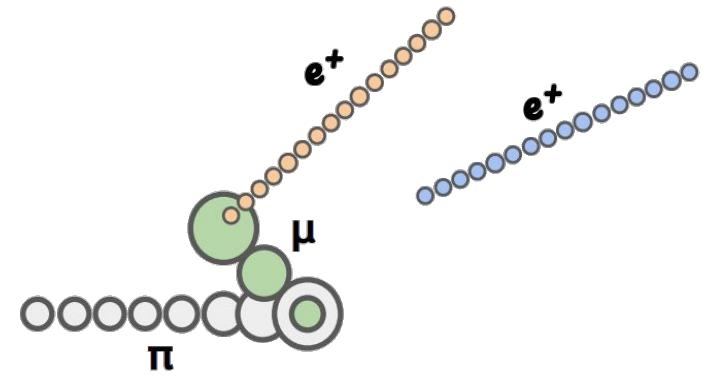
 = Not an ML stage
 = An ML (trained) stage

Monte Carlo → ATAR ML Reconstruction



G4Pioneer Monte Carlo + Detector Response

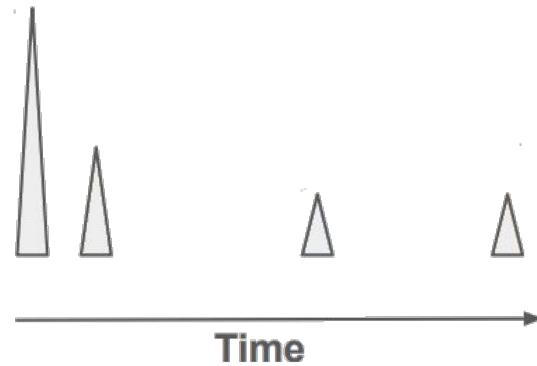
- **G4Pioneer Monte Carlo** simulates the underlying physics
 - Geant4 Fork
- **Detector Response** converts truth into detector-level signals
 - Depends on detector geometry
 - Handles event mixing
- ML training requires both detector-level outputs and truth-level information



Cartoon example of detector response with truth information added to hits

ML Ready Dataset Gaudi Pipeline (Time Group Forming)

- Gaudi pipeline does simple traditional labeling and reconstruction
- **Example time group forming:**
 - Detector response events are “split” into time-based clusters
 - Really, we just label hits with a time group they belong to
 - Most the time one time group == one particle
 - This helps the later ML stages reconcile particle specific features



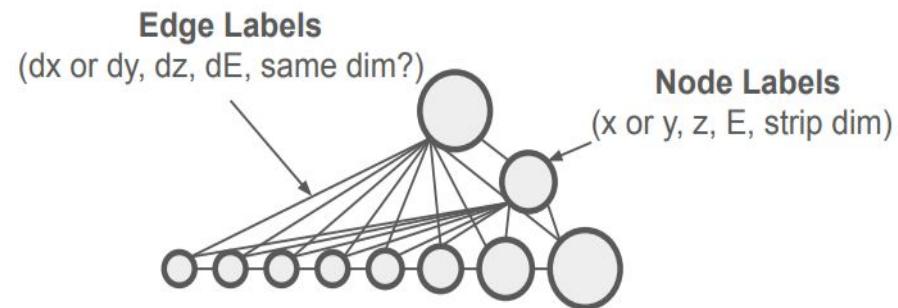
Cartoon example of time group spikes in energy deposition

Representing Information as a Graph

- Our final step before feeding the data into the model is constructing a fully connected graph
- Nodes contain hit specific information
 - Ex. position, deposited energy, etc.
- Edges contain related information between hits
 - Ex. difference in positions
- Global information for event level inputs
 - Ex. Total Energy
- We can append information to these graphs as downstream models make predictions

Graph level feature example:

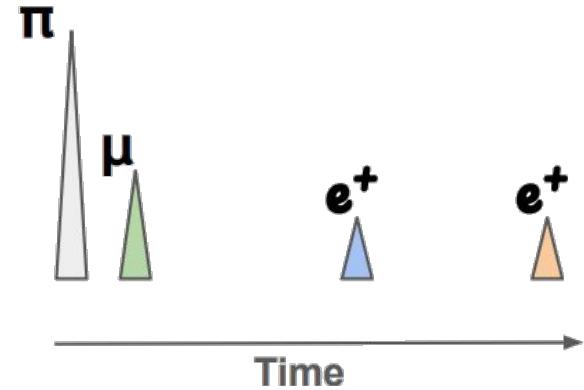
- Total energy



Graph representation of base detector information

Group Classifier

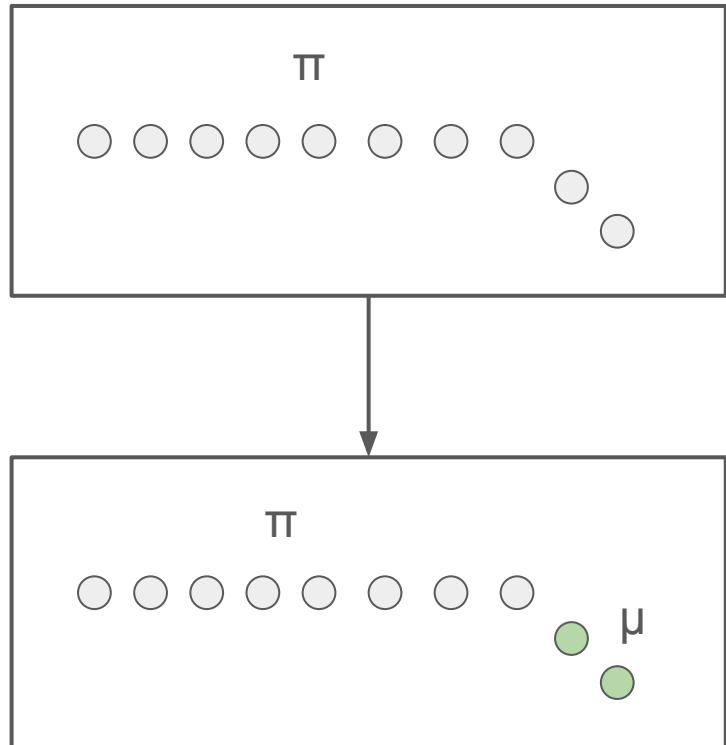
- **Multi-label graph transformer classifier**
 - Subgraph level task
- Inputs (represented as a graph):
 - Detector response event information
 - Time grouping labels
- Targets:
 - Truth level representation of what particles are in the group
 - Particularly:
[pionInGroup, muonInGroup, MIPinGroup]
- Outputs:
 - Predicted probability that the time group belongs to each class
[prob_pionInGroup, prob_muonInGroup, prob_MIPinGroup]



Cartoon example of time group spikes in energy deposition, now labeled by group classifier

Group Splitter

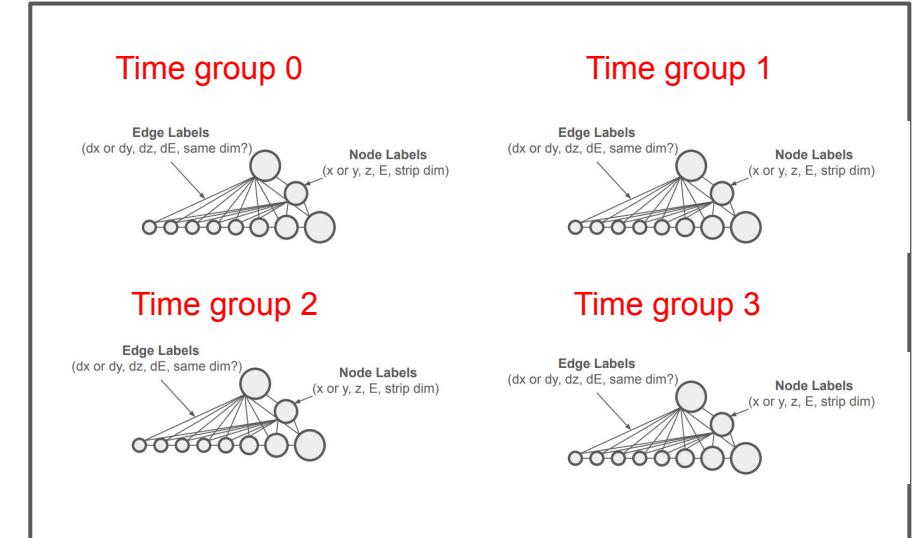
- **Multi-label graph transformer classifier**
 - Node level task
- **Inputs (represented as a graph):**
 - Detector response event information
 - Time grouping labels
 - Predicted time group level particle class
- **Targets:**
 - Truth level representation of the particle type for each hit
 - Particularly, for each hit in the graph:
[[is_pion, is_muon, is_mip], [is_pion, is_muon, is_mip], ...]
- **Outputs:**
 - Predicted probability that each hit belongs to each class
[[prob_pion, prob_muon, prob_mip],
[prob_pion, prob_muon, prob_mip], ...]



Cartoon example of how a time group may be split into multiple particle classifications

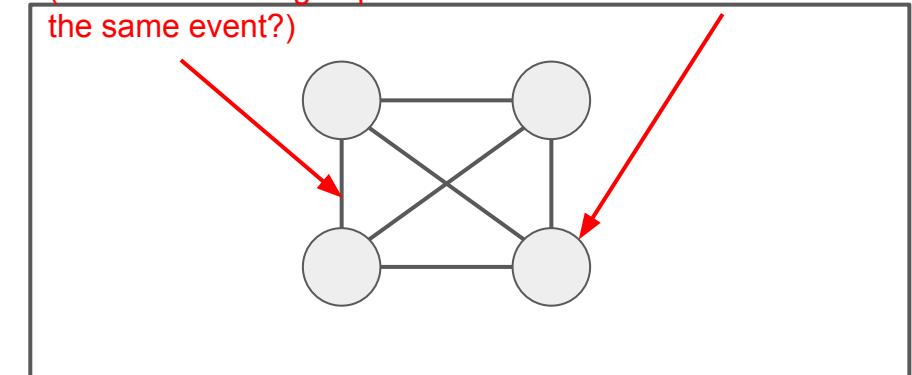
Pattern Finder

- **Affinity graph transformer classifier**
 - Edge level task
- **Inputs (represented as a graph):**
 - Detector response event information
 - Time grouping labels
 - Predicted time group level particle class
 - Predicted hit level particle classes
- **Targets:**
 - Agreement between truth level event origins between nodes; i.e. “are these two nodes from the same monte carlo truth event”.
Explicitly:
`target_affinity[i][j] = 1 if
origin_id[i] == origin_id[j]`
- **Outputs:**
 - Predicted “affinities” (0,1) between time groups



Edges:
Learned affinity between groups
(i.e. are two time groups from
the same event?)

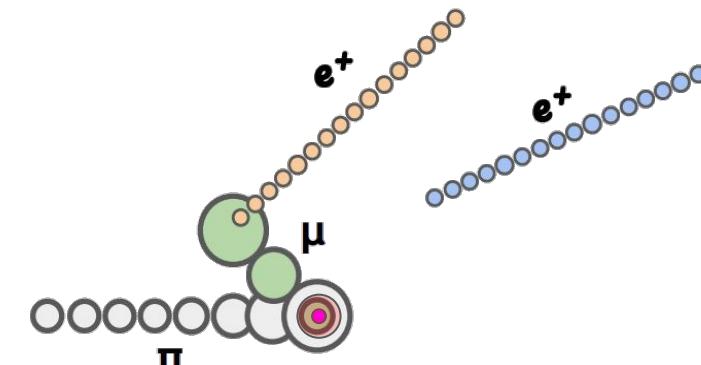
Node labels:
On per time group
graph



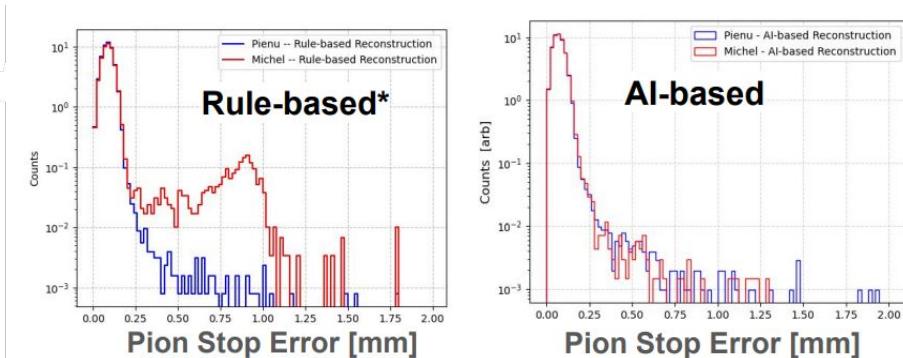
Pion Stop Regressor

- **Quantile graph transformer regressor**
 - Graph level task
- Inputs (represented as a graph):
 - Detector response event information
 - Time grouping labels
 - Predicted time group level particle class
 - Predicted hit level particle classes
 - Predicted affinities
- Targets:
 - Truth level pion stop
 - Particularly:
[x_stop, y_stop, z_stop]
- Outputs:
 - Predicted pion stop
[[[x_stop_q1, y_stop_q1, z_stop_q1], ...]
- Notes:
 - Quantiles effectively give uncertainties if chosen to be top 84%, top 50%, and top 16% quartiles = median \pm 1 standard deviation

- = Pion stop estimate
- = Error range determined from quantiles

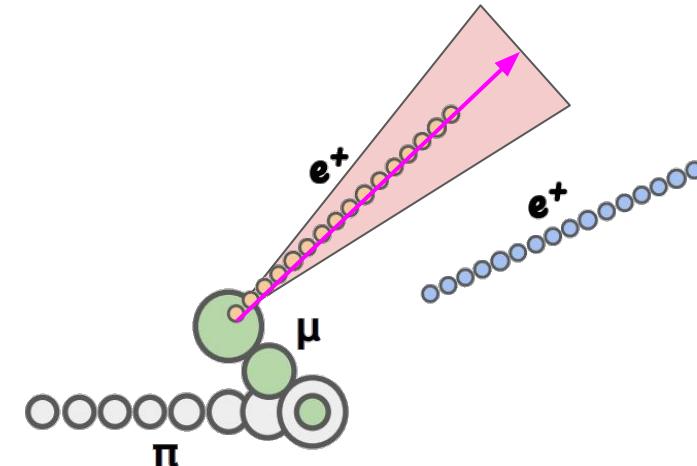


Cartoon example of pion stop estimate

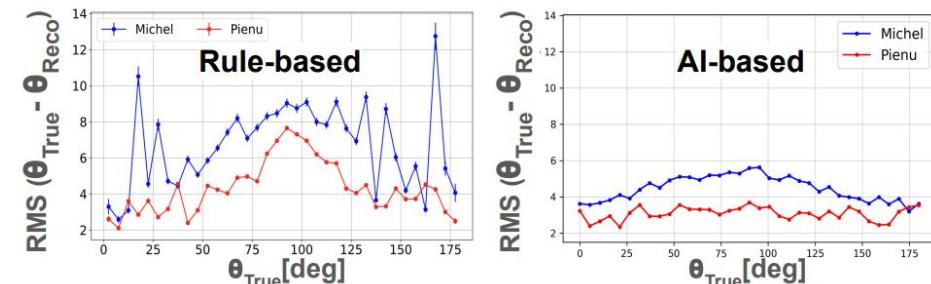


Positron Angle Regressor

- Quantile graph transformer regressor
 - Graph level task
- Inputs (represented as a graph):
 - Detector response event information
 - Time grouping labels
 - Predicted time group level particle class
 - Predicted hit level particle classes
 - Predicted affinities
- Targets:
 - Truth level direction vector for positron angle
 - Particularly:
 $[(\sin\theta\cos\varphi), (\sin\theta\sin\varphi), (\cos\theta)]$
- Outputs:
 - Predicted direction vector for positron angle
 $[(\sin\theta\cos\varphi)_{\text{q1}}, (\sin\theta\sin\varphi)_{\text{q1}}, (\cos\theta)_{\text{q1}}, \dots]$



Cartoon example of positron angle estimate



Auxiliary Slides

Background Physics

Common Pion Decay Channels



= Most Common

Leptonic Decay

- $\pi^+ \rightarrow e^+ + \nu_e$
- $\pi^- \rightarrow e^- + \bar{\nu}_e$
- $\pi^+ \rightarrow \mu^+ + \nu_\mu$
- $\pi^- \rightarrow \mu^- + \bar{\nu}_\mu$

Beta Decay

- $\pi^+ \rightarrow \pi^0 + e^+ + \nu_e$
- $\pi^- \rightarrow \pi^0 + e^- + \bar{\nu}_e$

Photon Decay

- $\pi^0 \rightarrow \gamma + \gamma$

Dalitz Decay

- $\pi^0 \rightarrow \gamma + e^- + e^+$

Double-Dalitz Decay

- $\pi^0 \rightarrow e^- + e^+ + e^- + e^+$

Electrons

- $\pi^0 \rightarrow e^- + e^+$

[Note: Dalitz Decays are like photon decays, except the photon(s) are virtual and immediately decay into electron/positron pairs]

Naive Pion Decay, 2-body decay

- Without getting into details of QCD, we can treat this as a 3 particle decay
- We can use Fermi's golden rule:

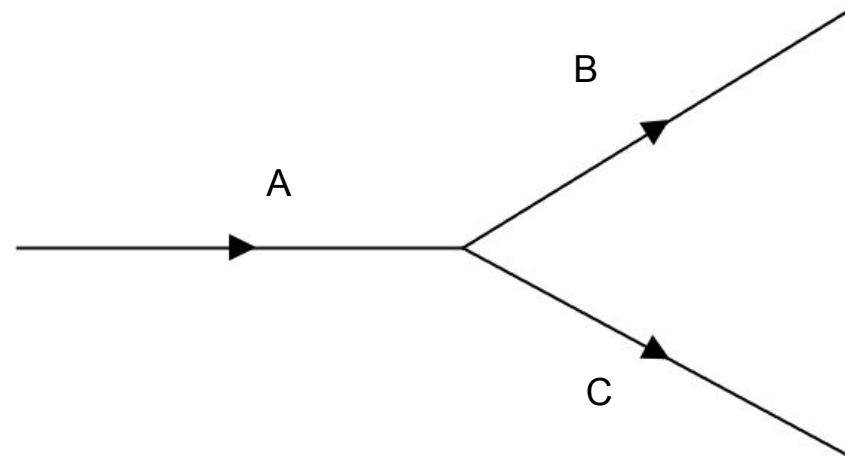
$$d\Gamma = |\mathcal{M}|^2 \cdot \frac{1}{2\hbar m_a} \cdot \left[\frac{cd^3\mathbf{p}_b^2}{(2\pi)^3 2E_b} \cdot \frac{cd^3\mathbf{p}_c^2}{(2\pi)^3 2E_c} \right] \cdot (2\pi)^4 \delta^4(p_a - p_b - p_c)$$

- After integration in the COM frame we find:

$$\Gamma = \frac{|\mathbf{p}|}{8\pi\hbar m_a^2 c} |\mathcal{M}|^2$$

where $\mathbf{p} = \mathbf{p}_b = -\mathbf{p}_c$

- $\rightarrow \Gamma \propto p$ (not correct)
 - Details hidden in matrix element



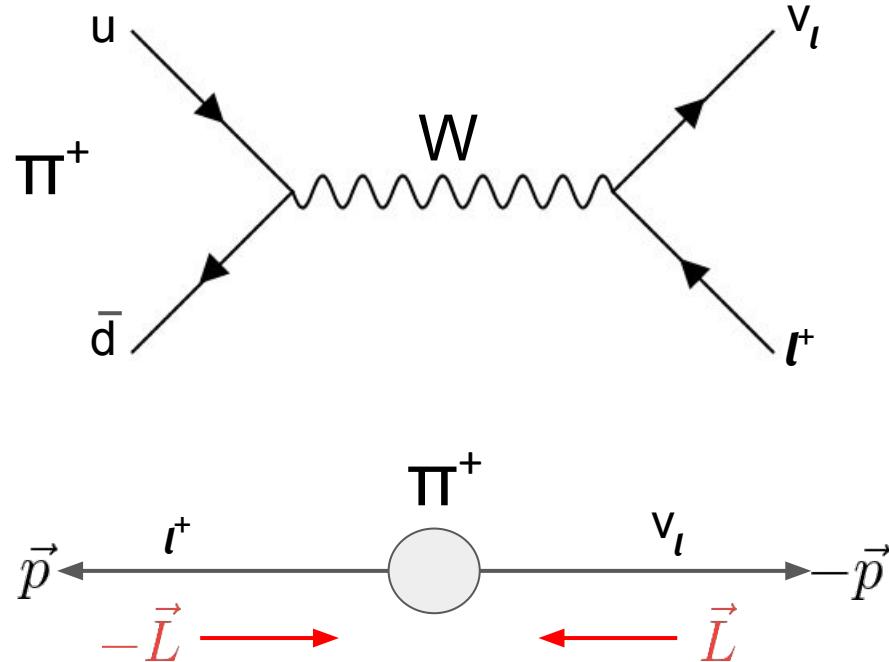
Why Massless \rightarrow Chirality States \sim Helicity States

- Massless \rightarrow moves at c
- Moves at c \rightarrow cannot reverse particle direction with Lorentz boost \rightarrow helicity is Lorentz Invariant
- Chirality is a property of a particle, always Lorentz invariant! \rightarrow helicity and chirality agree in direction in all inertial reference frames

$$\begin{aligned} & (\gamma^\mu p_\mu - m)u(p) = 0 \quad [\text{Dirac Equation}] \\ \implies & \begin{pmatrix} -mI_{2 \times 2} & \sigma \cdot p \\ \bar{\sigma} \cdot p & -mI_{2 \times 2} \end{pmatrix} \begin{pmatrix} u_L \\ u_R \end{pmatrix} = 0 \\ \implies & \begin{cases} (\sigma \cdot p)u_R - mu_L = 0 \\ (\bar{\sigma} \cdot p)u_L - mu_R = 0 \end{cases} \quad [\text{Chiral States}] \\ m \rightarrow 0 \implies & \begin{cases} (p_0 - \boldsymbol{\sigma} \cdot \mathbf{p})u_R = 0 \\ (p_0 + \boldsymbol{\sigma} \cdot \mathbf{p})u_L = 0 \end{cases} \\ \implies & \begin{cases} \frac{\boldsymbol{\sigma} \cdot \mathbf{p}}{|\mathbf{p}|}u_R = u_R \\ \frac{\boldsymbol{\sigma} \cdot \mathbf{p}}{|\mathbf{p}|}u_L = -u_L \end{cases} \\ \hat{h} = & \frac{\mathbf{S} \cdot \mathbf{p}}{|\mathbf{p}|} = \frac{1}{2} \frac{\boldsymbol{\sigma} \cdot \mathbf{p}}{|\mathbf{p}|} \quad [\text{Helicity operator}] \\ \implies & \begin{cases} \hat{h}u_R = \frac{1}{2}u_R \\ \hat{h}u_L = -\frac{1}{2}u_L \end{cases} \quad [\text{Chiral states are eigenstates of helicity operator}] \end{aligned}$$

Helicity Suppression (Why is Muon Decay Most Likely?)

- Naively, $\Gamma \propto p' \rightarrow$ electron decay more likely
- Weak force only affects left-handed (LH) chiral particle states and right-handed (RH) chiral anti-particle states
- Neutrinos are all LH chirality
- $m_\nu \ll E$ means LH neutrino chirality \rightarrow LH (negative) neutrino helicity
- Conservation of momentum \rightarrow anti-lepton is LH (negative) helicity



Helicity Suppression (Why is Muon Decay Most Likely?)

- We can write the LH (negative) helicity anti-particle state in the chiral basis:

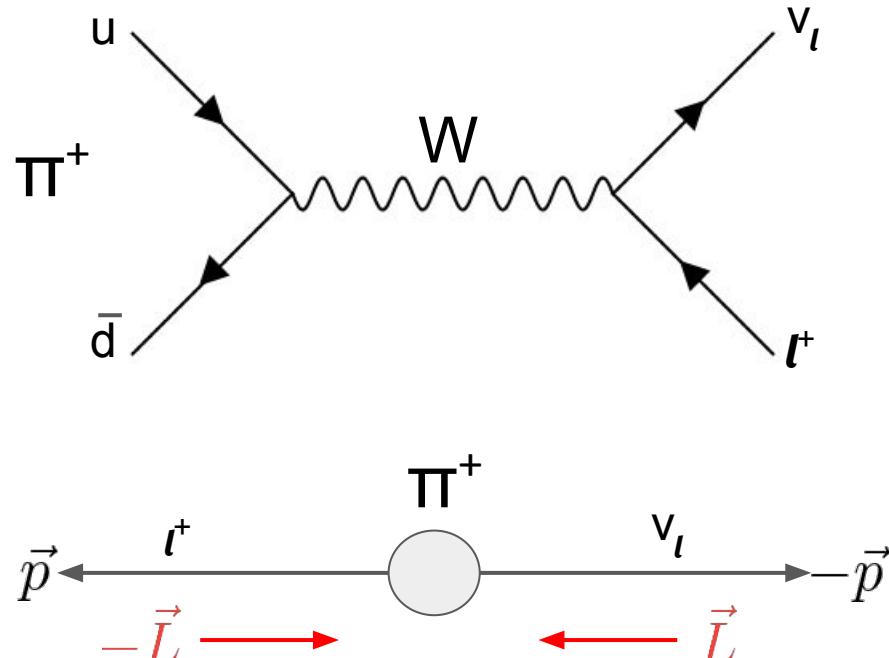
$$v_{\downarrow} = \frac{A}{2} \left[\left(1 - \frac{p}{E + m}\right) v_R - \left(1 + \frac{p}{E + m}\right) v_L \right]$$

- We ignore the LH term (weak force only acts on the RH term), anti-particle's matrix element contribution:

$$\mathcal{M} \sim \frac{1}{2} \left(1 - \frac{p_l}{E_l + m_l}\right) \xrightarrow{m_\nu \rightarrow 0} \frac{m_l}{m_\pi + m_l}$$

- This effect ends up making the matrix element smaller \rightarrow decay rate smaller

$$\Gamma \propto |\mathcal{M}|^2$$



LH (negative) helicity spinor to chiral components

An negative **helicity** antiparticle can be written as

$$v_{\downarrow} = \sqrt{E + m} \begin{pmatrix} \frac{|\mathbf{p}|}{E+m} \cos(\frac{\theta}{2}) \\ \frac{|\mathbf{p}|}{E+m} \sin(\frac{\theta}{2}) e^{i\phi} \\ \cos(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) e^{i\phi} \end{pmatrix}$$

Where (θ, ϕ) define the direction of the momentum. Without loss of generality, assume the momentum is in the z direction

$$v_{\downarrow} = \sqrt{E + m} \begin{pmatrix} \frac{|\mathbf{p}|}{E+m} \\ \frac{|\mathbf{p}|}{E+m} \\ 1 \\ 1 \end{pmatrix} \equiv A \begin{pmatrix} \tau \xi_R \\ \xi_R \end{pmatrix}$$

LH (negative) helicity spinor to chiral components

We can use the **chiral** projection operations to project this **helicity** state to chiral state

$$P_R = \frac{I_{4 \times 4} + \gamma^5}{2} = \begin{pmatrix} I_{2 \times 2} & I_{2 \times 2} \\ I_{2 \times 2} & I_{2 \times 2} \end{pmatrix}$$

$$P_L = \frac{I_{4 \times 4} - \gamma^5}{2} = \begin{pmatrix} I_{2 \times 2} & -I_{2 \times 2} \\ -I_{2 \times 2} & I_{2 \times 2} \end{pmatrix}$$

$$v_\downarrow = \frac{A}{2} \left[(1 - \tau) \begin{pmatrix} -\xi_R \\ \xi_R \end{pmatrix} + (1 + \tau) \begin{pmatrix} \xi_R \\ \xi_R \end{pmatrix} \right] \equiv \frac{A}{2}(1 - \tau)v_R - \frac{A}{2}(1 + \tau)v_L$$

Where the left and right **chiral** anti-particle states are defined by

$$P_L v_R = v_R \text{ and } P_R v_L = v_L$$

LH (negative) helicity spinor to chiral components

Looking at the **chiral** projection of a negative **helicity** state, we can see in general there are left **and** right **chiral** components, so the weak force **can** act on a LH (negative) anti-particle **helicity** state

$$v_{\downarrow} = \frac{A}{2} \left[\left(1 - \frac{p}{E + m} \right) v_R - \left(1 + \frac{p}{E + m} \right) v_L \right]$$

It should also be clear as $m \rightarrow 0$, the LH (negative) **helicity** state coincides with the LH **chiral** state.

This means W boson decay to two massless leptons is forbidden! One of the particles must have the wrong chirality, and thus low mass decays will be suppressed.

Matrix Element Details

$$\mathcal{M}_{fi} = \left[\frac{g_W}{\sqrt{2}} \frac{1}{2} f_\pi p_\pi^\alpha \right] \cdot \left[\frac{g_{\alpha\beta}}{m_W^2} \right] \cdot \left[\frac{g_W}{\sqrt{2}} g_l \bar{u}(p_l) \gamma^\beta \frac{1}{2} (1 - \gamma^5) v(p_\nu) \right]$$

Move to pion rest frame so only $p^0 = m_\pi$ remains:

$$\mathcal{M}_{fi} = \frac{g_W^2 f_\pi g_l}{4m_W^2} m_\pi \bar{u}(p_l) \gamma^0 \frac{1}{2} (1 - \gamma^5) v(p_\nu)$$

Using the identity: $\bar{u}(p_l) \gamma^0 = u^\dagger(p_l) \gamma^0 \gamma^0 = u^\dagger(p_l) I_{4 \times 4} = u^\dagger(p_l)$

$$\mathcal{M}_{fi} = \frac{g_W^2 f_\pi g_l}{4m_W^2} m_\pi u^\dagger(p_l) \frac{1}{2} (1 - \gamma^5) v(p_\nu)$$

Matrix Element Details

For a neutrino $m \ll E$ so helicity eigenstate is essentially the chiral eigenstate:

$$\frac{1}{2}(1 - \gamma^5)v(p_\nu) = v_\uparrow(p_\nu) \implies \mathcal{M}_{fi} = \frac{g_W^2 f_\pi g_l}{4m_W^2} m_\pi u^\dagger(p_l) v_\uparrow(p_\nu)$$

By letting the lepton go in the z-direction we can write:

$$u(p_l) = u_\uparrow(p_l) + u_\downarrow(p_l) = \sqrt{E_l + m_l} \left[\begin{pmatrix} 1 \\ 0 \\ \frac{p}{E_l + m_l} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ \frac{-p}{E_l + m_l} \end{pmatrix} \right] \text{ and } v(p_\mu) = v_\uparrow(p_\mu) = \sqrt{p} \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}$$

Negative helicity lepton down state disappears when “dotted” with the neutrino state:

$$\mathcal{M}_{fi} = \frac{g_W^2 f_\pi g_l}{4m_W^2} m_\pi \sqrt{E_l + m_l} \sqrt{p} \left(1 - \frac{p}{E_l + m_l} \right)$$

Matrix Element Details

$$\mathcal{M}_{fi} = \frac{g_W^2 f_\pi g_l}{4m_W^2} m_\pi \sqrt{E_l + m_l} \sqrt{p} \left(1 - \frac{p}{E_l + m_l} \right)$$

We can re-write E_l and p in the limit where the neutrino mass is zero:

$$E_l = \frac{m_\pi^2 + m_l^2}{2m_\pi} \text{ and } p_l = \frac{m_\pi^2 - m_l^2}{2m_\pi}$$

$$\implies \mathcal{M}_{fi} = \frac{g_W^2 f_\pi g_l}{4m_W^2} m_\pi \cdot \frac{m_\pi + m_l}{\sqrt{2m_\pi}} \cdot \left(\frac{m_\pi^2 - m_l^2}{2m_\pi} \right)^{\frac{1}{2}} \cdot \frac{2m_l}{m_\pi + m_l}$$

$$\implies \mathcal{M}_{fi} = \frac{g_W^2 f_\pi g_l}{4m_W^2} \cdot m_l (m_\pi^2 - m_l^2)^{\frac{1}{2}}$$

Lepton Universality

Note: $x^2 = 1 + 2(x - 1) + \mathcal{O}(x^2)$

Let: $\left(\frac{g_e}{g_\mu}\right) \equiv (1 + \Delta_{\frac{g_e}{g_\mu}}) \equiv x$

$$R_{e/\mu} = \frac{\Gamma(\pi^- \rightarrow e^- \bar{\nu}_e)}{\Gamma(\pi^- \rightarrow \mu^- \bar{\nu}_\mu)} \approx (1 + 2\Delta_{\frac{g_e}{g_\mu}}) \left[\frac{m_e(m_\pi^2 - m_e^2)}{m_\mu(m_\pi^2 - m_\mu^2)} \right]^2$$

Let: $\Delta R_{e/\mu} = R_{e/\mu} - (R_{e/\mu})_{\text{theory}}$

$$\frac{\Delta R_{e/\mu}}{(R_{e/\mu})_{\text{theory}}} = 2\Delta_{\frac{g_e}{g_\mu}}$$

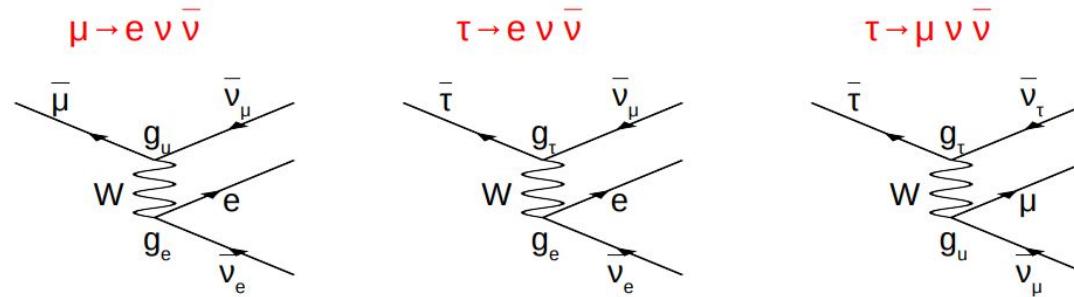
Small discrepancy in g_e/g_μ and 1 can cause twice as big discrepancy in measured $R_{e/\mu}$ and theory $R_{e/\mu}$

Lepton Universality

$$\frac{\Gamma(\pi^- \rightarrow e^- \bar{\nu}_e)}{\Gamma(\pi^- \rightarrow \mu^- \bar{\nu}_\mu)} = \left(\frac{g_e}{g_\mu} \right)^2 \left[\frac{m_e(m_\pi^2 - m_e^2)}{m_\mu(m_\pi^2 - m_\mu^2)} \right]^2$$

- Lepton universality assumes $g_e = g_\mu$, so the first factor disappears
- Improving the branching ratio measurement and comparing to the theoretical value acts as a test of lepton universality
- Another test would consider pure leptonic decays, but such decays involving taus are too rare for high precision measurements

Another Test for Lepton Universality



$$\text{Fermi constant, } G_F = g^2 / 4\sqrt{2}M_W^2$$

$$G_{\mu e} = 1.166\,378\,7(6) \times 10^{-5} \text{ GeV}^{-2} \text{ (0.5 ppm)}$$

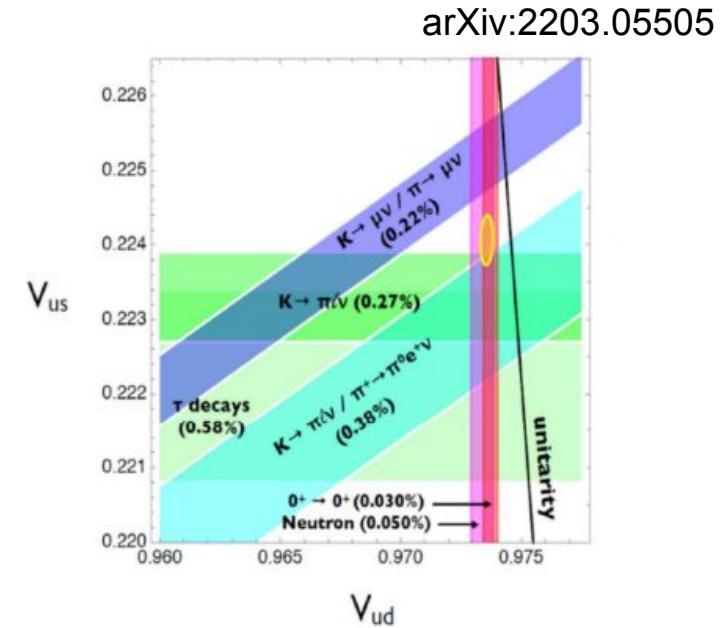
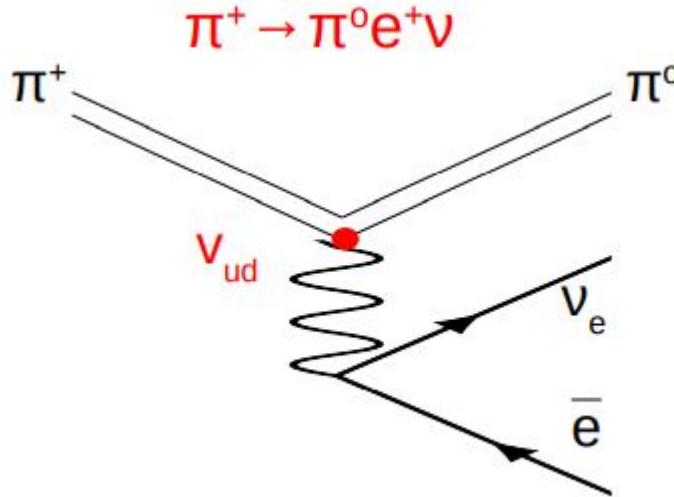
$$G_{\tau\mu} = 1.1665(28) \times 10^{-5} \text{ GeV}^{-2} \text{ (0.2%)}$$

$$G_{\tau e} = 1.1665(28) \times 10^{-5} \text{ GeV}^{-2} \text{ (0.2%)}$$

weak coupling, g

$$g_e : g_\mu : g_\tau \quad 1 : 1.0011(24) : 1.0006(24)$$

CKM Unitary Test

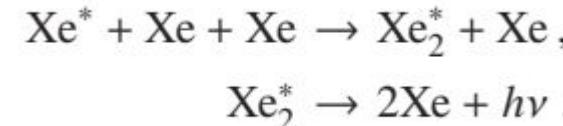


- Pion beta decay gives a precision measurement of V_{ud}
- These decays are lower rate than $\pi \rightarrow e v_e$ and $\pi \rightarrow \mu v_\mu$
- Experimental measurements do not agree

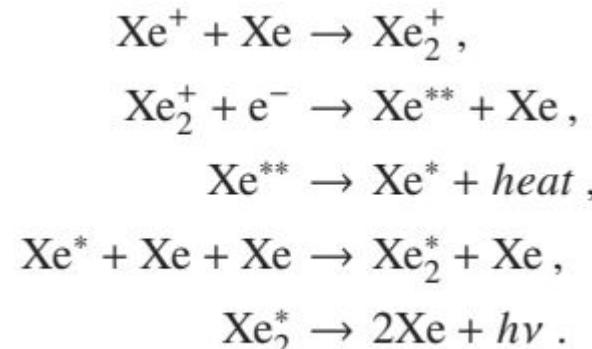
Some Information about LXe and NaI

- LXe has singlet and triplet state decay constants:
 - $\tau_s = 4.3 \pm 0.6$ ns
 - $\tau_t = 26.9^{+0.7}_{-1.1}$ ns
- LXe light yield:
 - ~29 photons/keV at room temp
- NaI decay constant:
 - ~ 250 ns
- NaI light yield:
 - 38 photons/keV at room temp

Scintillation from excited Xe (Xe^*):

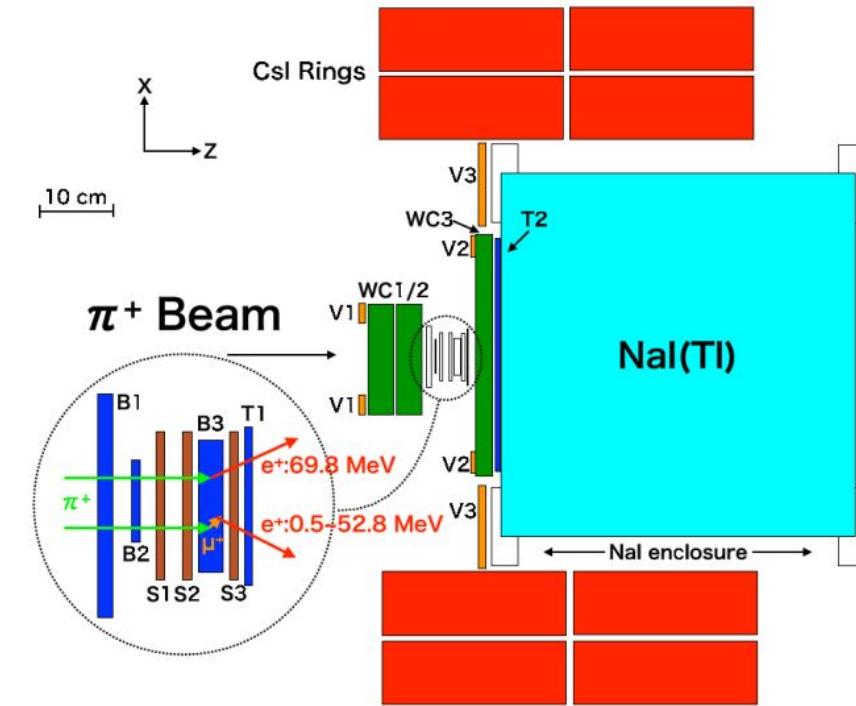


Scintillation from ionized Xe (Xe^+):



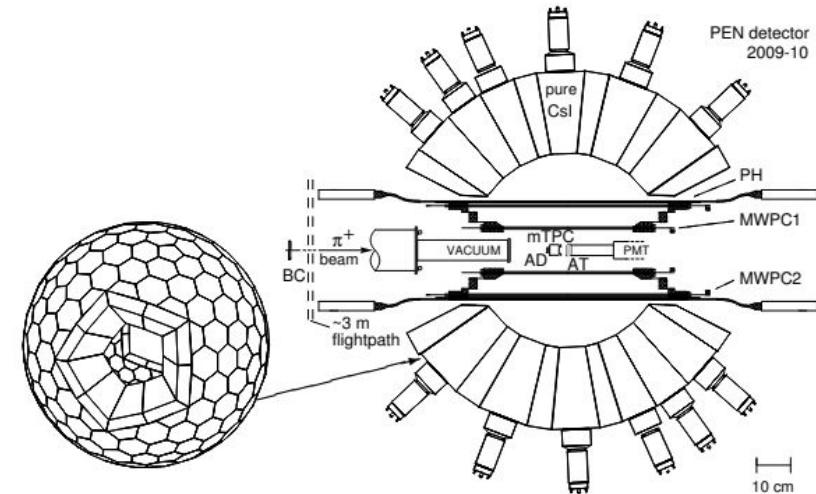
Past Experimental Approach (PIENU)

- NaI has a long primary decay time
 - ~ 250 ns
- Event pileup forces the experiment to run at a low rate
 - ~70 kHz
- “inactive target”, muons aren’t tracked
- CsI Rings for shower leakage detection



PEN

- Similar to PIENU
 - Segmented
 - Better timing
- Many channels of pure CSI
 - 240 channels
- Active target



PIONEER Strategy (Tail Fraction Uncertainty)

$$R_{e/\mu} \equiv \frac{\Gamma_e}{\Gamma_\mu} \sim 10^{-4}$$

$$R_{e/\mu}^{\text{theo}} = 1.23524(15) \times 10^{-4} \quad \Rightarrow \quad \delta R_{e/\mu}^{\text{theo}} = 1.5 \times 10^{-8}$$

Compete with theory: $\delta R_{e/\mu} \lesssim \delta R_{e/\mu}^{\text{theo}}$

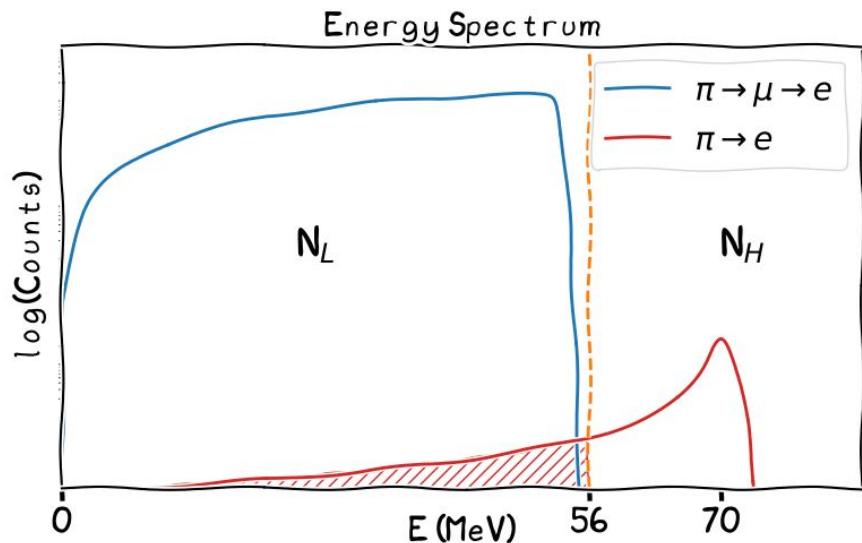
$$\Rightarrow \frac{\delta R_{e/\mu}}{R_{e/\mu}} \lesssim \frac{1.5 \times 10^{-8}}{1.235 \times 10^{-4}} \approx 1.2 \times 10^{-4}$$

(if Γ_μ is negligible): $\frac{\delta \Gamma_e}{\Gamma_e} \approx \frac{\delta R_{e/\mu}}{R_{e/\mu}} \sim 10^{-4}$

$$R_{e/\mu}^{\text{meas}} = R_{e/\mu}^{\text{true}} + f_{\text{tail}}, \quad f_{\text{tail}} \equiv \frac{\Gamma_{\mu \rightarrow e}}{\Gamma_\mu}$$

$$(\delta R_{e/\mu})^2 = (\delta f_{\text{tail}})^2 + (\delta R_{e/\mu}^{\text{other}})^2$$

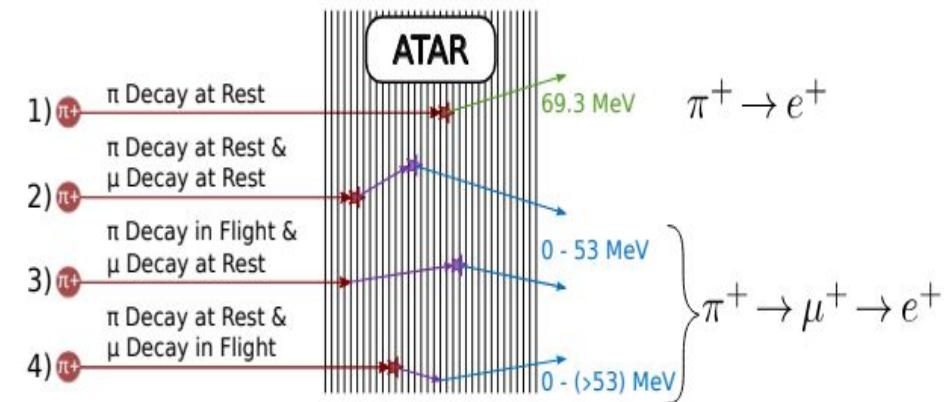
If tail is budgeted at fraction α : $\delta f_{\text{tail}} \lesssim \alpha \delta R_{e/\mu}^{\text{theo}} = \alpha (1.5 \times 10^{-8})$



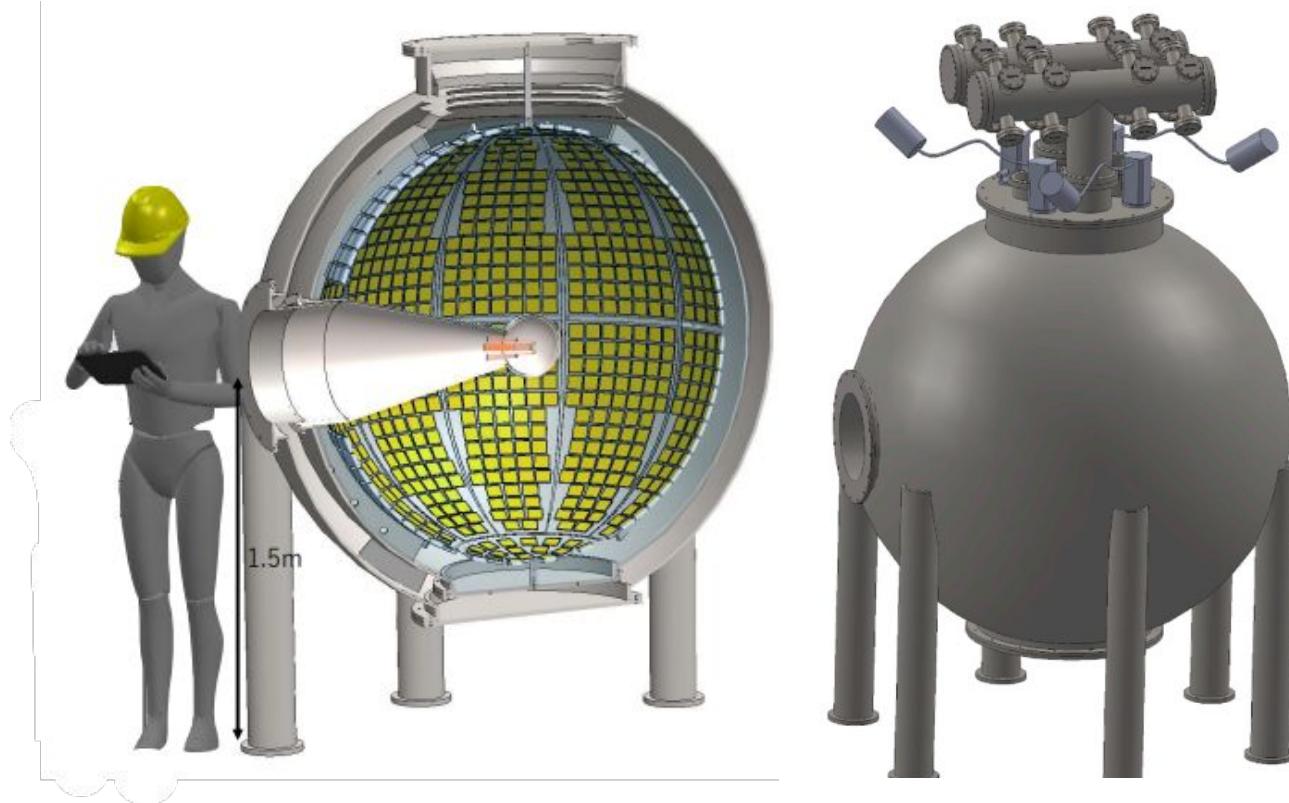
More ATAR details

- Pion and muon decays deposit energy into ATAR
- Allow event types to be distinguished
- Muons decaying in flight can boost positron energy past 53 MeV (big issue!)
 - ATAR can give information to rebuild event, and correctly classify a muon decay

arxiv: 2203.01981

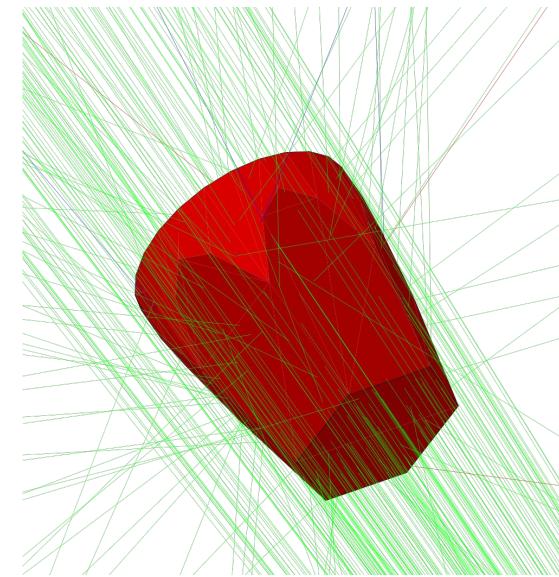


Another Calorimeter 3D Render (Liquid Xenon)



PIONEER Simulation Monte Carlo (Geant4 Fork)

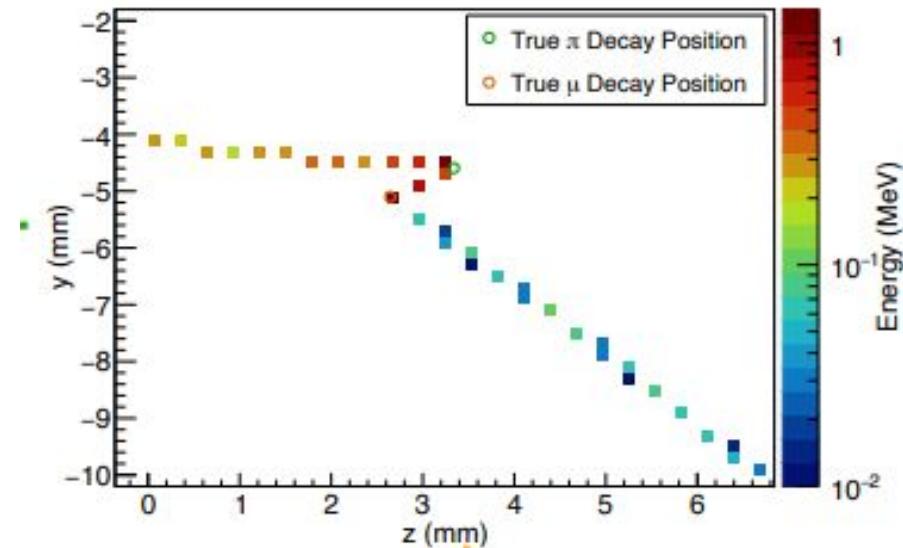
- Forked from Geant4
 - Geant4 is a general-purpose particle transport toolkit.
Standard simulation engine for high energy particle, nuclear, and medical physics
- Built to simulate pion decays in an arbitrary input detector geometry
- Modular “physics lists” for interactions, physics models, etc.
- Provides “truth” data for downstream software



Example of particle tracks through a physical object in an (unrelated) Geant4 simulation

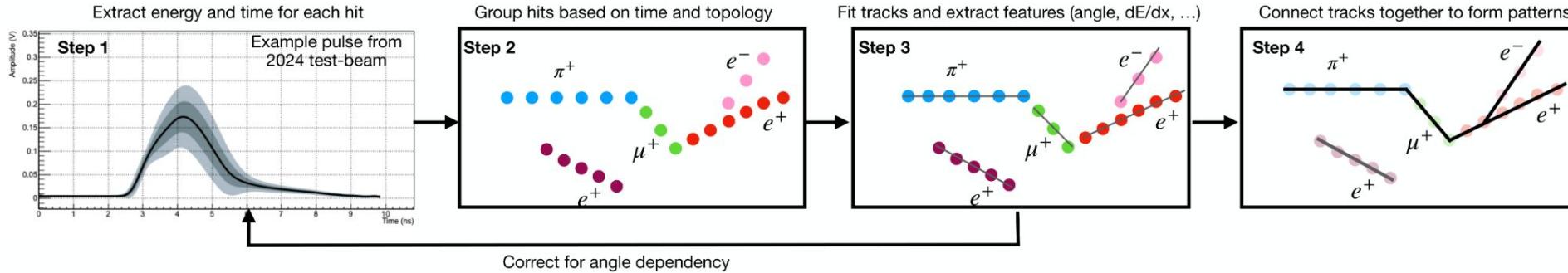
PIONEER Simulation Detector Response

- A Gaudi based pipeline for detector response to physical interactions in simulated truth data
 - Gaudi is a modular event processing framework commonly used for high energy physics software pipelines
- Simulates what information we expect to see out of the detectors so we can blind reconstruction to truth



Example raw detector data (with some truth overlaid)

PIONEER Simulation Reconstruction

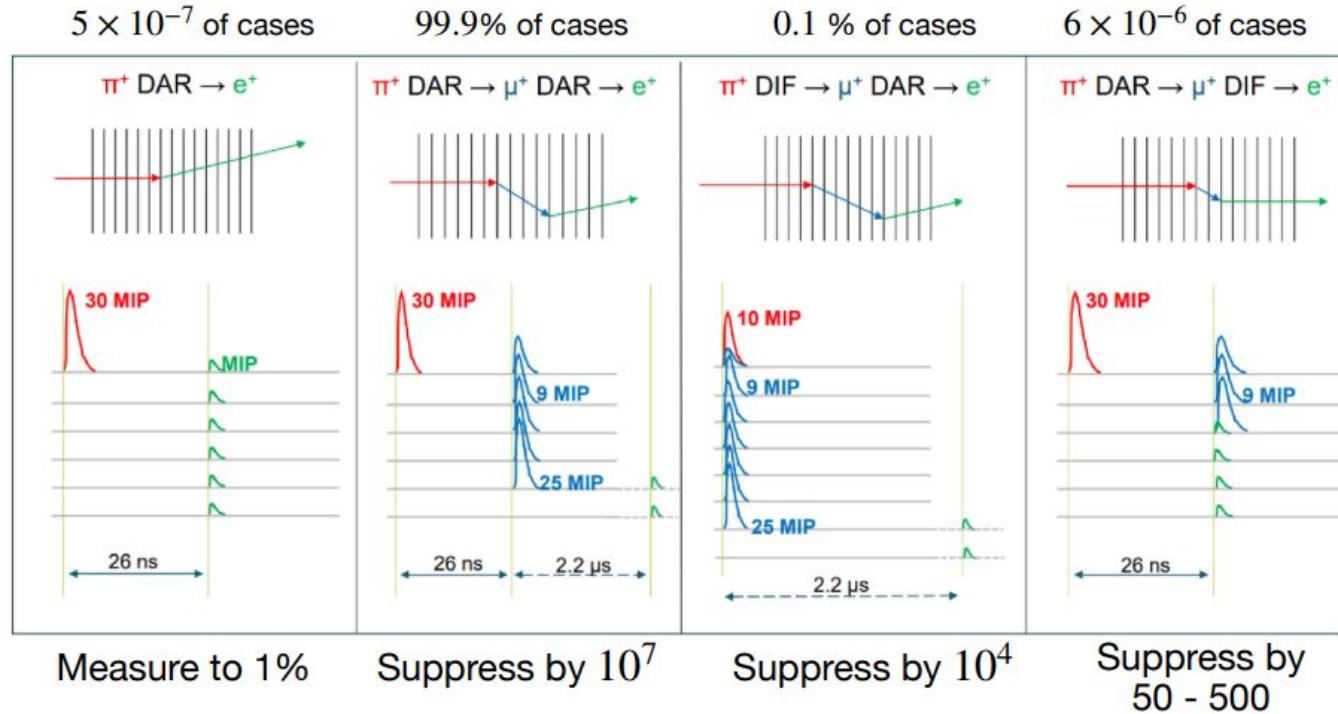


- Also a Gaudi based pipeline
- A series of steps to ultimately classify event types (ex. as $\pi \rightarrow e$ or $\pi \rightarrow \mu \rightarrow e$)
- Designed to operate on simulated **and** real detector data
- This is the stage where machine learning/AI approaches can work well
 - Large quantities of detector data with truth targets available from simulation

The Activate Target (ATAR)

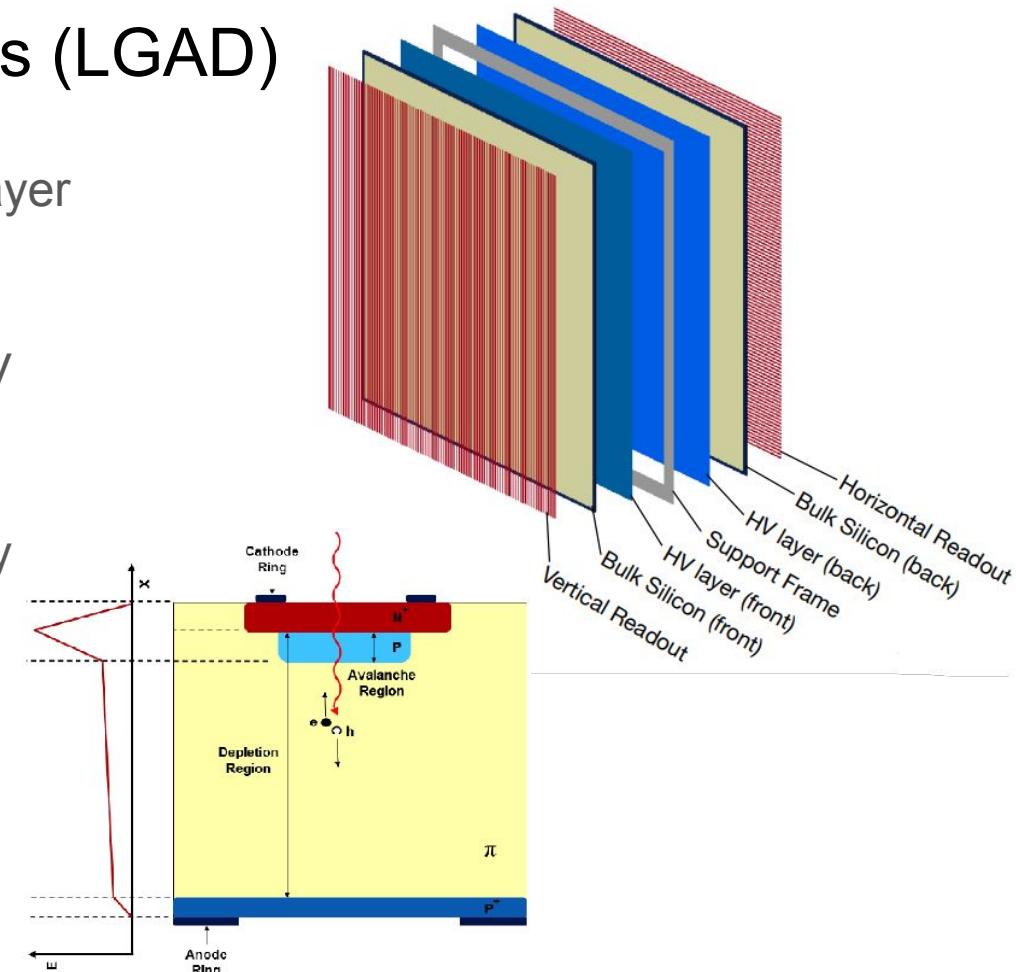
Active Target (ATAR) Purpose

Measuring $\pi \rightarrow e$ events with $E_{\text{rec}} < 56 \text{ MeV}$ (5×10^{-7} of all events) to 1%



Low Gain Avalanche Diodes (LGAD)

- Silicon detector with a thin gain-layer
- Gain required to reliably detect minimally ionising positrons.
- Large dynamic range from 40 keV (MIP) to 4 MeV (Muon Stop)
- Excellent timing resolution
- Gain suppression for large energy deposits (stopping pion/muon)
- Degraded energy resolution
- Introduce angular dependency



Principles of Neural Networks and Machine Learning

Backpropagation (Explicit, Part I)

- Diagram with some labels to conceptualize the variables

Indices:

$l \equiv$ layer index

$i \equiv$ neuron index in layer l

$j \equiv$ neuron index in layer $l - 1$

$k \equiv$ neuron index in layer $l + 1$

$t \equiv$ training iteration

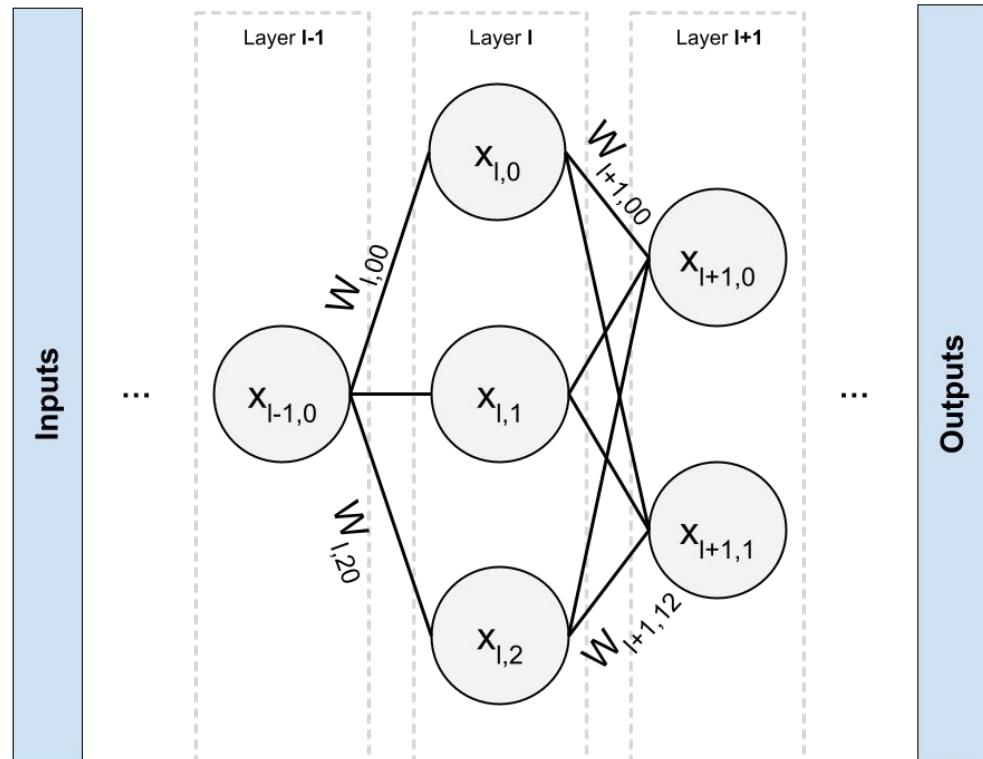
Core quantities:

$L \equiv$ loss (scalar)

$x_{l,i} \equiv$ activation of neuron i in layer l

$W_{l,ij} \equiv$ weight from $(l - 1, j)$ to (l, i)

$\alpha \equiv$ learning rate



Backpropagation (Explicit, Part II)

- Equations for how to update weights from back propagation

Indices:

$l \equiv$ layer index

$i \equiv$ neuron index in layer l

$j \equiv$ neuron index in layer $l - 1$

$k \equiv$ neuron index in layer $l + 1$

$t \equiv$ training iteration

Core quantities:

$L \equiv$ loss (scalar)

$x_{l,i} \equiv$ activation of neuron i in layer l

$W_{l,ij} \equiv$ weight from $(l - 1, j)$ to (l, i)

$\alpha \equiv$ learning rate

1. Backpropagate activation gradients:

$$\frac{\partial L}{\partial x_{l,i}} = \sum_k \frac{\partial L}{\partial x_{l+1,k}} \frac{\partial x_{l+1,k}}{\partial x_{l,i}}$$

2. Weight gradient via local chain rule:

$$\frac{\partial L}{\partial W_{l,ij}} = \frac{\partial L}{\partial x_{l,i}} \frac{\partial x_{l,i}}{\partial W_{l,ij}}$$

3. Gradient descent update to weights:

$$W_{l,ij}^{(t+1)} = W_{l,ij}^{(t)} - \alpha \frac{\partial L}{\partial W_{l,ij}} \Big|_{W=W^{(t)}}$$

Backpropagation (Explicit, Part III)

- The back propagation step is a recursive step

Indices:

$l \equiv$ layer index

$i \equiv$ neuron index in layer l

$j \equiv$ neuron index in layer $l - 1$

$k \equiv$ neuron index in layer $l + 1$

$t \equiv$ training iteration

Core quantities:

$L \equiv$ loss (scalar)

$x_{l,i} \equiv$ activation of neuron i in layer l

$W_{l,ij} \equiv$ weight from $(l - 1, j)$ to (l, i)

$\alpha \equiv$ learning rate

The backpropagation step:

$$\frac{\partial L}{\partial x_{l,i}} = \sum_k \frac{\partial L}{\partial x_{l+1,k}} \frac{\partial x_{l+1,k}}{\partial x_{l,i}}$$

This formula is recursive, so for N layers you have explicitly:

$$\frac{\partial L}{\partial x_{0,i_0}} = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_N} \frac{\partial L}{\partial x_{N,i_N}} \prod_{m=0}^{N-1} \frac{\partial x_{m+1,i_{m+1}}}{\partial x_{m,i_m}}$$

Or more commonly you can write in Jacobian form, which cleans up the notation

$$\boxed{\frac{\partial L}{\partial x_0} = \prod_{m=0}^{N-1} \left(\frac{\partial x_{m+1}}{\partial x_m} \right)^\top \frac{\partial L}{\partial x_N}}$$

Backpropagation (Explicit, Part IV)

- The actual values of $x_{l,i}$ are *architecture dependent*
- In this simple example, you can compute them by defining a bias and activation function (eg. softmax)

Core quantities:

$L \equiv$ loss (scalar)

$x_{l,i} \equiv$ activation of neuron i in layer l

$W_{l,ij} \equiv$ weight from $(l - 1, j)$ to (l, i)

$\alpha \equiv$ learning rate

$z_{l,i} \equiv$ pre-activation (raw linear response)

$b_{l,i} \equiv$ bias parameter for neuron (l, i)

$\phi(\cdot) \equiv$ activation function

We can define $z_{l,i}$ for the raw computed “response” from the previous layer:

$$z_{l+1,k} = \sum_j W_{l+1,kj} x_{l,j} + b_{l+1,k}$$

Then we apply some activation, usually to constrain values between 0 and 1 to prevent exponential blowup. This allows to (finally) formally compute all needed partial derivatives:

$$x_{l+1,k} = \phi(z_{l+1,k})$$

$$\frac{\partial x_{l+1,k}}{\partial x_{l,i}} = \phi'(z_{l+1,k}) W_{l+1,ki}$$

Indices:

$l \equiv$ layer index

$i \equiv$ neuron index in layer l

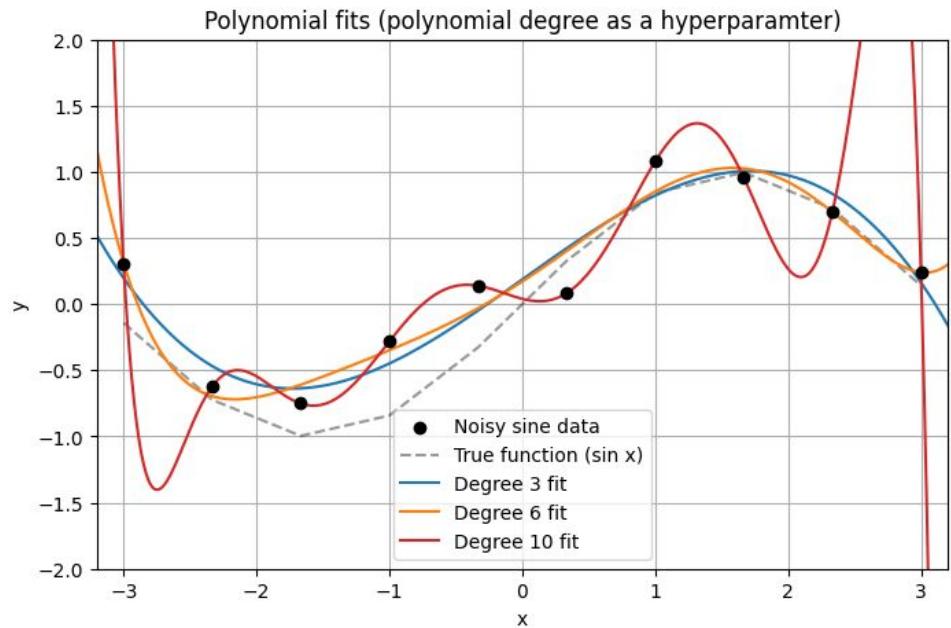
$j \equiv$ neuron index in layer $l - 1$

$k \equiv$ neuron index in layer $l + 1$

$t \equiv$ training iteration

Hyperparameters Need to be Tuned Appropriately

- Improper hyperparameter choices can lead to
 - Overtraining
 - Slower convergence
 - Vanishing or exploding gradients
 - Impossible to find true solution at this point
- Tuning hyperparameters can be hard
 - They interact with each other (often non-linearly)
 - Optimal choice varies with each dataset/model



Example: Fitting data with a polynomial. The degree of the polynomial, d , is a hyperparameter

Hyperparameters (Learning Rate)

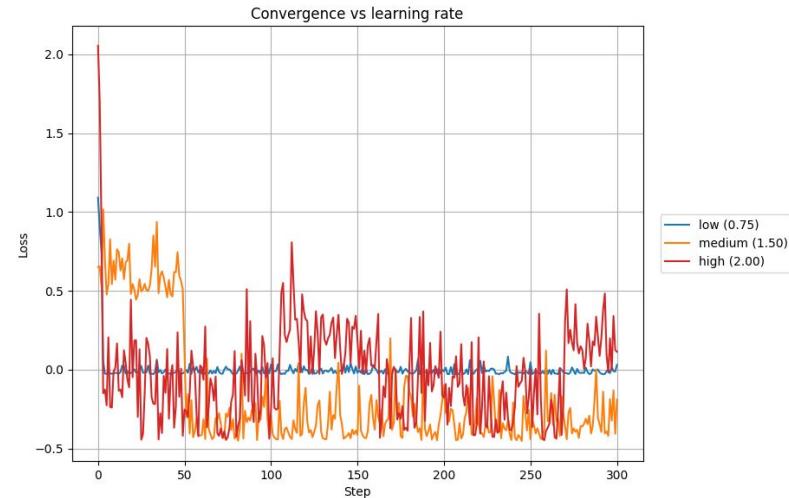
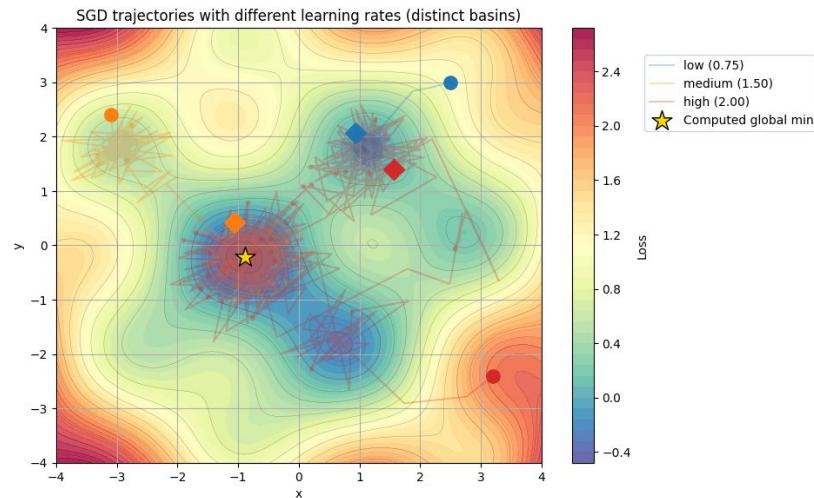
- [More detail in this article](#)
- Learning Rate controls how large each parameter update is in the direction of the gradient
- Larger learning rate:
 - Bigger, more exploratory steps
 - Less likely to get stuck in local minima
 - Harder to settle precisely at the optimum
- Smaller learning rate:
 - Smaller, more precise steps
 - More likely to get trapped in local minima
 - Better at fine-tuning near the optimum

$$w_{\text{new}} = w_{\text{old}} - \alpha \nabla_w L(w)$$

w = model weights

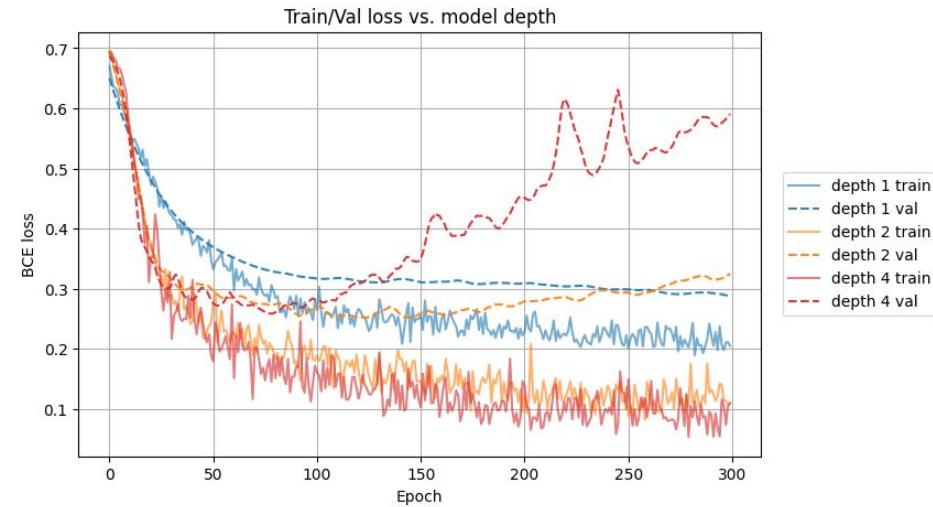
α = learning rate

$\nabla_w L(w)$ = gradient of the loss with respect to the weights

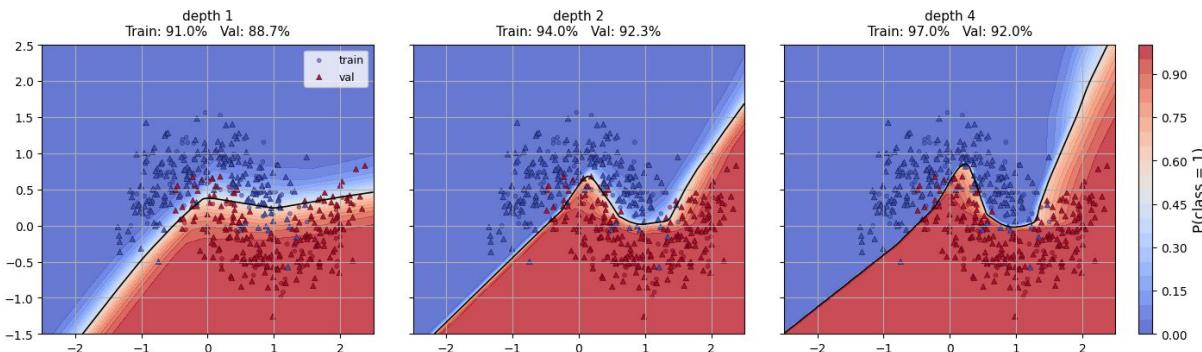


Hyperparameters Example (# of layers)

- [More detail in this article](#)
- Hidden dimension sets the size of the model's internal feature representations at each layer.
- More layers:
 - Learns more "hierarchical"/"abstract" features
 - Higher risk of overfitting or training instability (higher loss fluctuations)
 - Higher memory and compute cost
- Less layers:
 - Learns "simpler"/"shallow" features
 - Lower risk of overfitting
 - Lower memory/compute cost

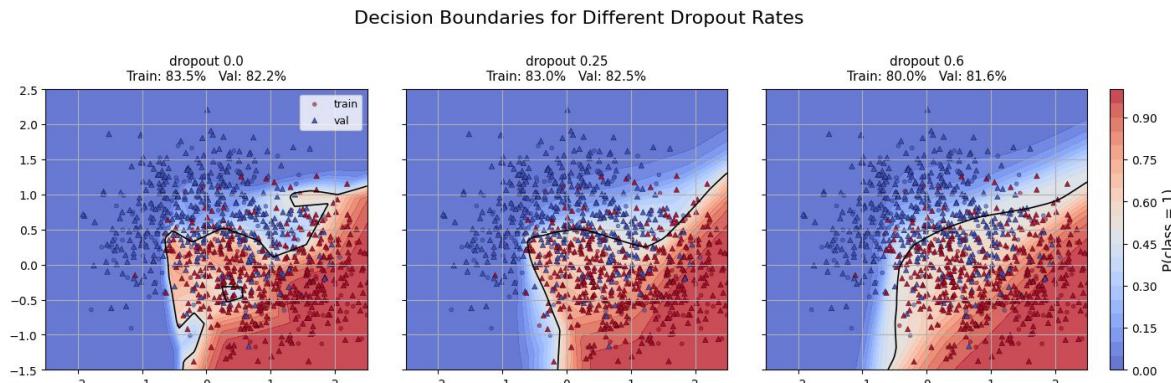
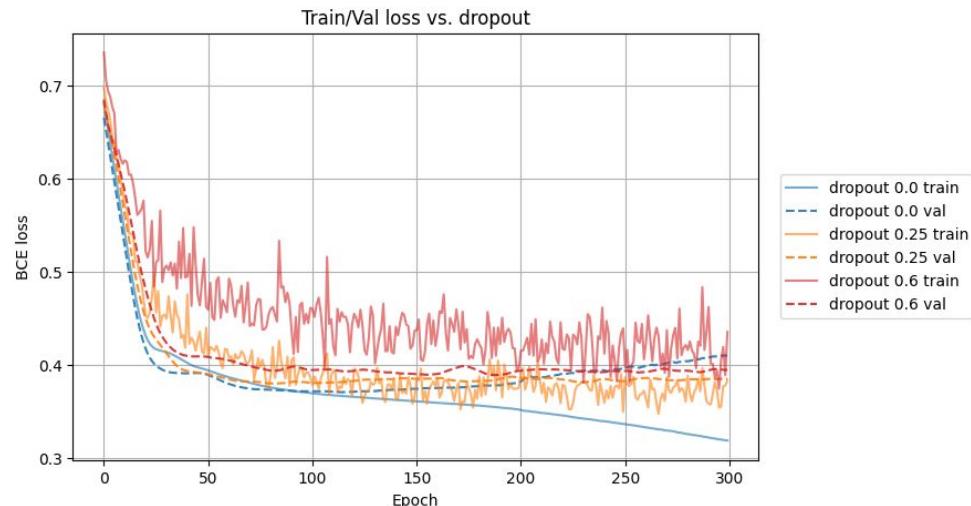


Decision Boundaries for Different Model Depths



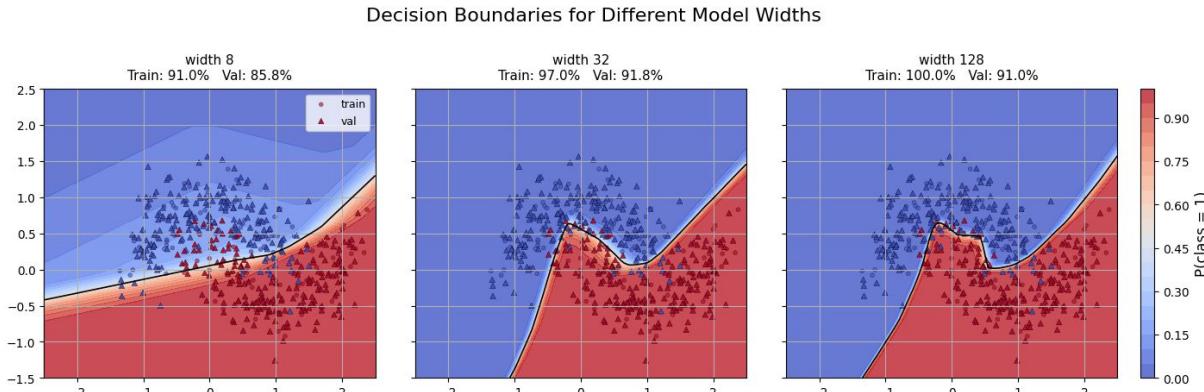
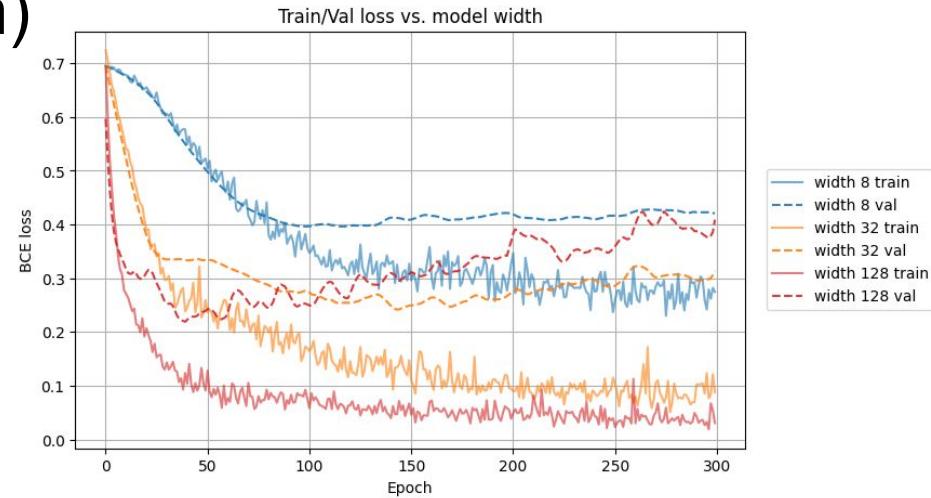
Hyperparameters (Dropout)

- [More detail in this article](#)
- **Dropout randomly disables a fraction of activations during training to reduce overfitting**
- Higher dropout:
 - Better for noisy datasets
 - Less risk of overfitting
 - Higher risk of underfitting
- Lower dropout:
 - Better for less noisy datasets
 - Less risk of underfitting
 - Higher risk of overfitting



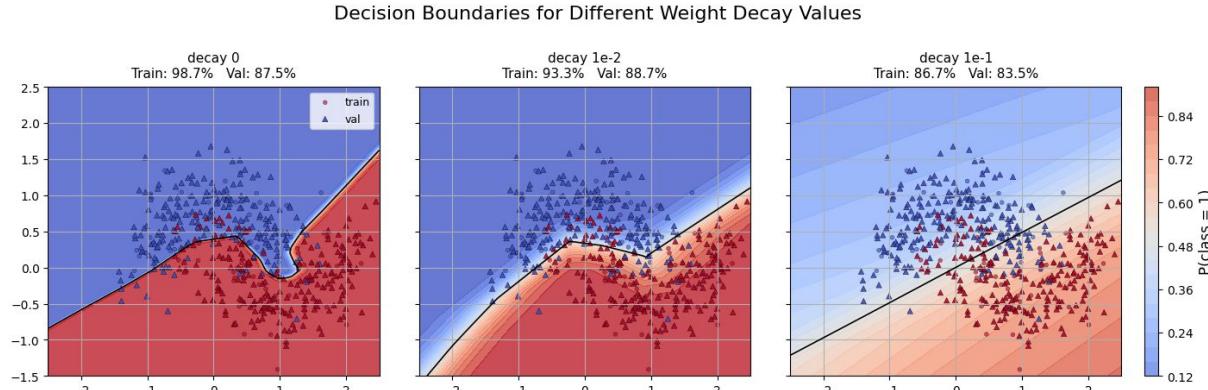
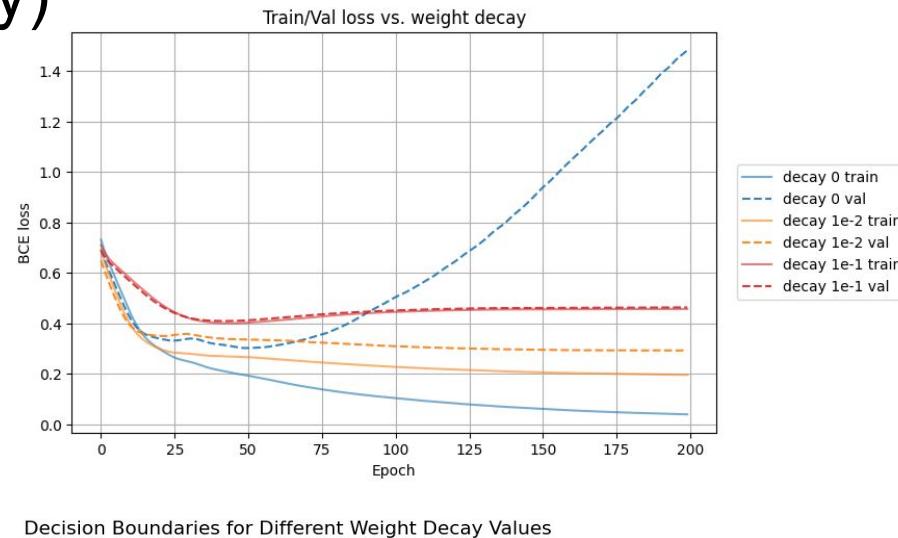
Hyperparameters (Hidden dim)

- [More detail in this article](#)
- **Hidden dimension sets the size of the model's internal feature representations at each layer.**
- **Larger hidden dim:**
 - Can represent more complex patterns
 - Higher risk of overfitting
 - Higher memory and compute cost
- **Smaller hidden dim:**
 - Limited ability to model complex patterns
 - Lower risk of overfitting
 - Lower memory/compute cost



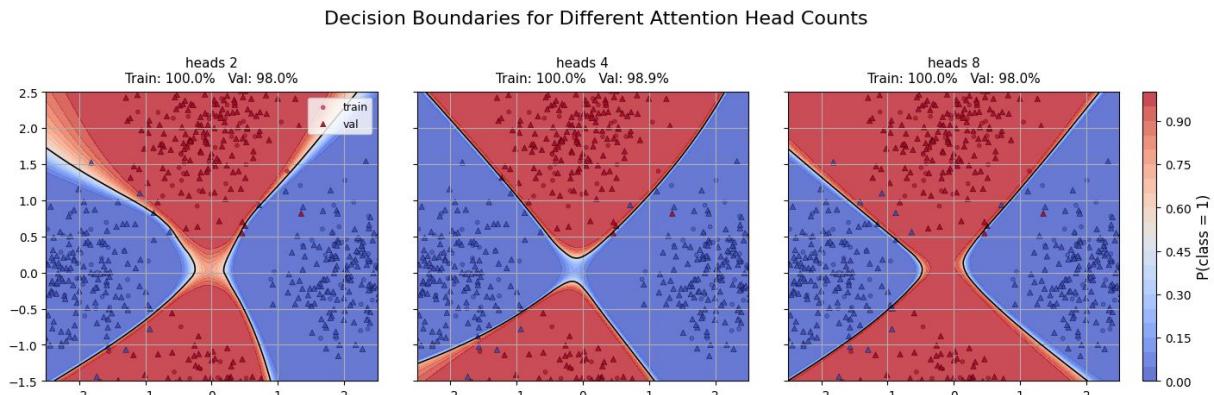
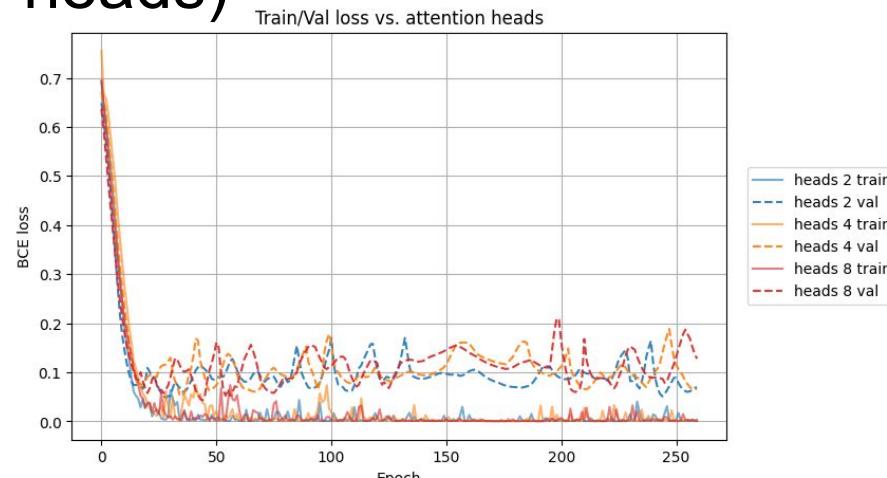
Hyperparameters (weight decay)

- [More detail in this article](#)
- Weight decay adds a penalty on large weights to the loss, encouraging simpler models. Similar to dropout.
- Higher weight Decay:
 - Better for noisy datasets, stronger regularization
 - Lower risk of overfitting
 - Higher risk of underfitting
- Lower weight decay:
 - Better for less noisy datasets, less regularization
 - Lower risk of underfitting
 - Higher risk of overfitting



Hyperparameters (# of attention heads)

- [More detail in this article](#)
- **The number of attention heads controls how many independent subspaces the model attends to in parallel within each attention layer**
- More attention heads:
 - Learns multiple attention patterns in parallel
 - Higher memory/compute cost
 - Diminishing returns if per-head dimension is too small
 - Can overfit and destabilize training
- Fewer attention heads:
 - Learns fewer attention patterns
 - Lower memory/compute cost
 - May miss distinct relationships
 - Can underfit if data is very heterogeneous

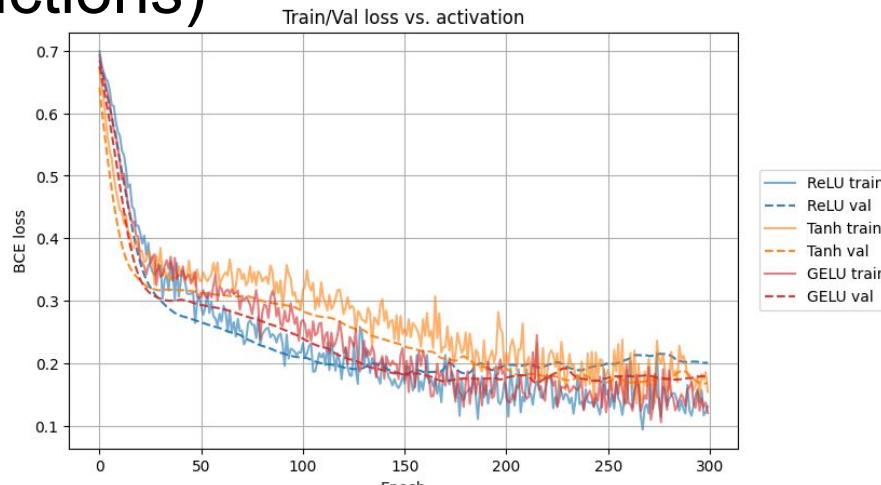


Hyperparameters (activation functions)

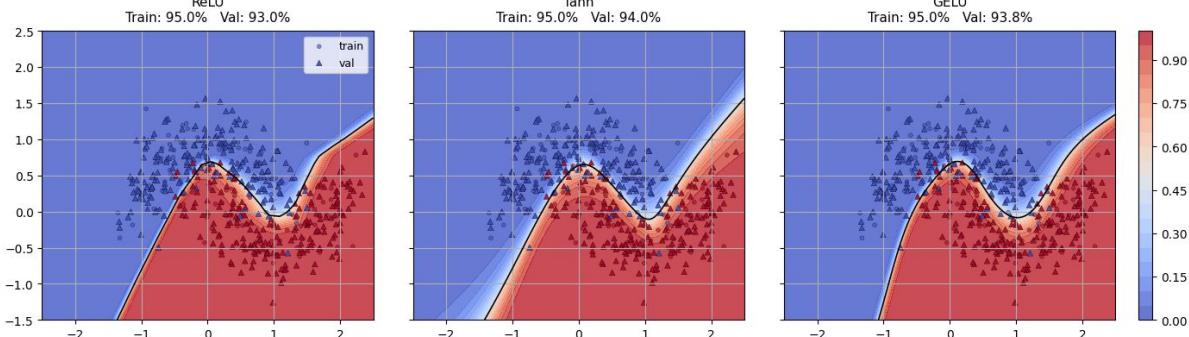
- [More detail in this article](#)
- The activation function controls the nonlinearity applied at each layer

Examples:

- ReLU
 - Simple, piecewise-linear activation
 - Fast training; can form sharp, noisy boundaries
- Tanh
 - Smooth, bounded activation
 - Produces smoother boundaries; may saturate
- GELU
 - Smooth, probabilistic gating
 - Balances smoothness and expressiveness

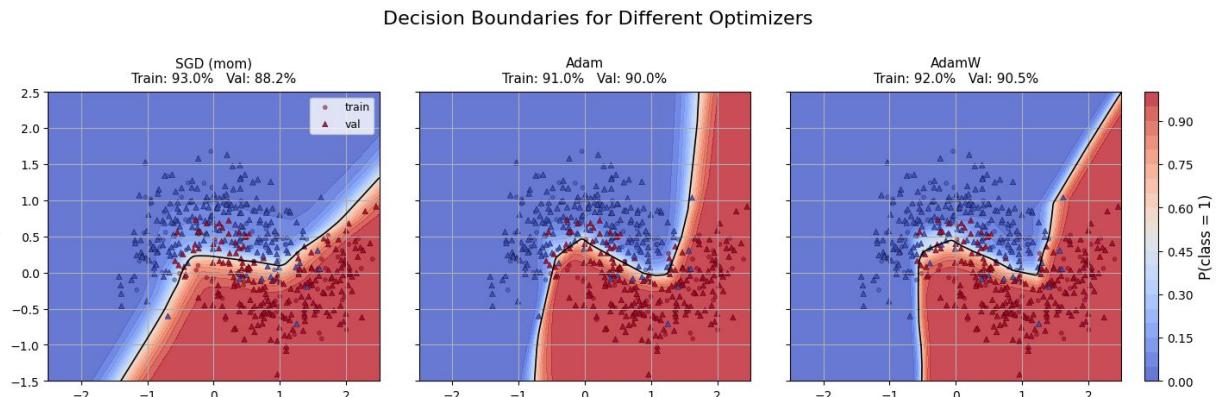
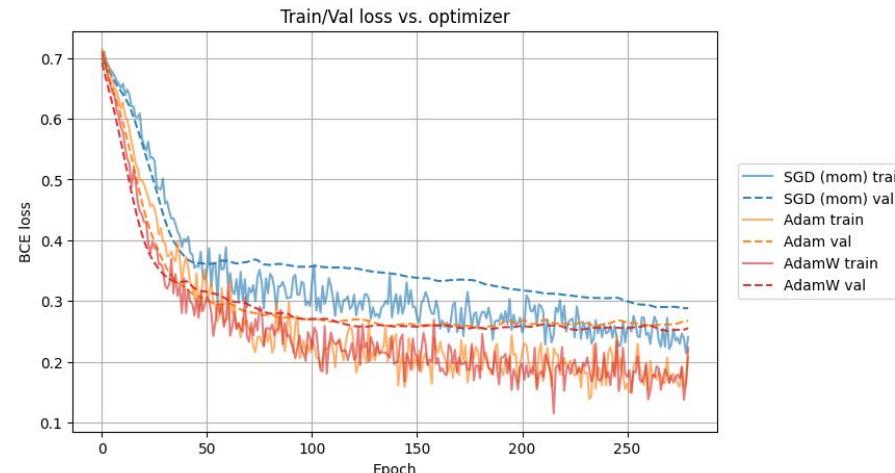


Decision Boundaries for Different Activation Functions



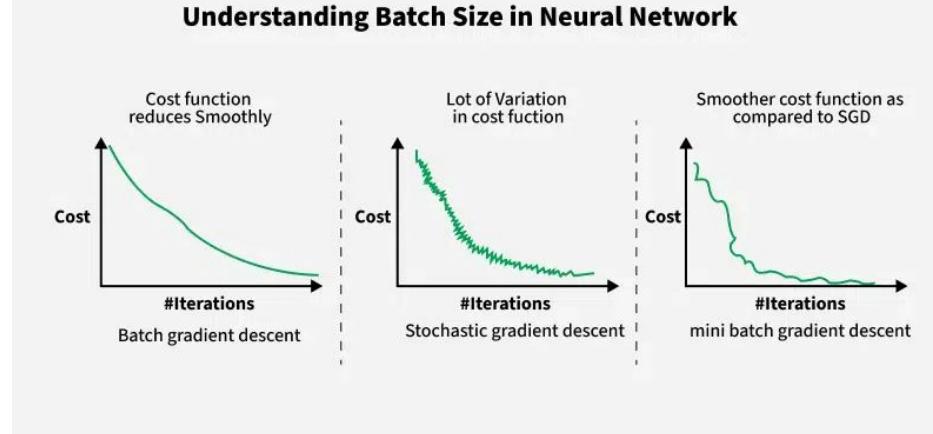
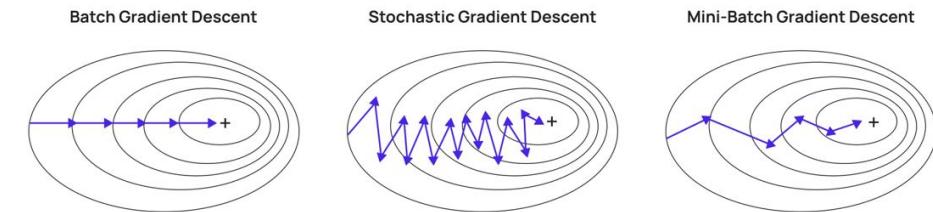
Hyperparameters (optimizers)

- [More detail in this article](#)
 - Optimizers control how gradients are transformed into parameter updates during training
- Examples:
- Stochastic Gradient Descent (SGD) with momentum
 - Simple, stable updates
 - Often performs well on unseen data, not just the training set
 - Sensitive to learning rate and scaling
 - Adaptive Moment Estimation (Adam)
 - Adaptive learning rates per parameter
 - Fast convergence, less sensitive to hyperparameter choice
 - Can overfit or generalize worse than SGD
 - AdamW
 - Adam with decoupled weight decay
 - More reliable regularization behavior
 - Standard choice for modern deep models



Hyperparameters (Gradient Descents)

- [Much more detail in this article](#)
- Batch Gradient Descent (BGD):
 - Uses the entire dataset in one iteration. The batch size is equal to the total number of training samples.
- Mini-Batch Gradient Descent (MBGD):
 - Uses a predefined number of samples from the dataset for each update. This method lies between batch and stochastic gradient descent (SGD).
- Stochastic Gradient Descent (SGD):
 - Processes only one sample at a time for each update, making the batch size equal to 1.



Hyperparameters (why are more layers harder to train)

- More layers → more loss instability
 - Due to back propagation
 - The size of your step becomes more “untrainable” the more layers you add
- Cases:
 - $\lambda \approx 1$
 - Gradients stable as model layers increase
 - $\lambda < 1$
 - Gradients vanish as model layers increase
 - No more learning occurs!
 - $\lambda > 1$
 - Gradients blow up as model layers increase
 - Cannot converge on solution
- One goal of model architecture choice is to get $\lambda \approx 1$

Backprop signal: $\frac{\partial L}{\partial x_0} = \prod_{m=0}^{N-1} \left(\frac{\partial x_{m+1}}{\partial x_m} \right)^\top \frac{\partial L}{\partial x_N}$

Layer Jacobian: $x_{m+1} = \phi(W_{m+1}x_m + b_{m+1})$
 $\frac{\partial x_{m+1}}{\partial x_m} = D_{m+1}W_{m+1}$
where $D_{m+1} = \text{diag}(\phi'(W_{m+1}x_m + b_{m+1}))$

Substitute: $\frac{\partial L}{\partial x_0} = \left(\prod_{m=0}^{N-1} W_{m+1}^\top D_{m+1} \right) \frac{\partial L}{\partial x_N}$

Take norms: $\left\| \frac{\partial L}{\partial x_0} \right\| \leq \left\| \frac{\partial L}{\partial x_N} \right\| \prod_{m=0}^{N-1} \|W_{m+1}\| \|D_{m+1}\|$

$$\Rightarrow E \left[\left\| \frac{\partial L}{\partial x_0} \right\| \right] \approx \left\| \frac{\partial L}{\partial x_N} \right\| (E[\|W\|] E[\|D\|])^N$$

Let $\lambda \equiv E[\|W\|] E[\|D\|]$

What is a Tree-Structured Parzen Estimator (TPE) (Part I)

- Given a (possibly stochastic) black box function you want to minimize (ex. model loss)

$$y = f(\theta)$$

- Hyperparams $\equiv \theta$
- Define the following:

$\gamma \equiv$ "good"/"bad" fraction (say, 0.2)

y^* is chosen such that $P(y < y^*) = \gamma$
so y^* is in, say, the best 20% of losses

$\max(y^* - y, 0) \equiv$ improvement

- Then, for any given theta we can define *expected improvement*

$$\text{EI}(\theta) \equiv \mathbb{E}[\max(y^* - y, 0) \mid \theta] = \int_{-\infty}^{y^*} (y^* - y) p(y \mid \theta) dy$$

- Goal: Maximize expected improvement

What is a Tree-Structured Parzen Estimator (TPE) (Part II)

- For a 1 dimensional y , it turns out to be easier to work with $p(\theta | y)$, we can invert using Bayes' rule:

$$p(y | \theta) = \frac{p(\theta | y) p(y)}{p(\theta)}$$

- And substitute

$$\text{EI}(\theta) \propto \int_{-\infty}^{y^*} (y^* - y) \frac{p(\theta | y)}{p(\theta)} p(y) dy$$

- Since we don't know every value of y this becomes an impossible task. We must make an approximation by dividing into "good" and "bad" distributions

$$p(\theta | y) \approx \begin{cases} p(\theta | y < y^*) \equiv \ell(\theta), & y < y^* \text{ (good)} \\ p(\theta | y \geq y^*) \equiv g(\theta), & y \geq y^* \text{ (bad)} \end{cases}$$

$$p(\theta) \approx \ell(\theta) \int_{y < y^*} p(y) dy + g(\theta) \int_{y \geq y^*} p(y) dy = \gamma l(\theta) + (1 - \gamma) g(\theta)$$

What is a Tree-Structured Parzen Estimator (TPE) (Part III)

- The integral, by construction, only cares about the “good” region, so the integral simplifies to

$$\text{EI}(\theta) \propto \frac{\ell(\theta)}{p(\theta)} \int_{-\infty}^{y^*} (y^* - y) p(y) dy$$

- But the integral is now constant in theta! So we have:

$$\text{EI}(\theta) \propto \frac{\ell(\theta)}{p(\theta)}$$

- Where $p(\theta) = \gamma \ell(\theta) + (1 - \gamma) g(\theta)$ so we can write:

$$\text{EI}(\theta) \propto \frac{\ell(\theta)}{\gamma \ell(\theta) + (1 - \gamma) g(\theta)}$$

What is a Tree-Structured Parzen Estimator (TPE) (Part IV)

- But γ is fixed, so

$$\begin{aligned}\arg \max_{\theta} EI(\theta) &= \arg \max_{\theta} \frac{\ell(\theta)}{\gamma \ell(\theta) + (1 - \gamma) g(\theta)} \\ &= \arg \max_{\theta} \frac{\ell(\theta)/g(\theta)}{\gamma \ell(\theta)/g(\theta) + (1 - \gamma)} \\ &= \arg \max_{\theta} \frac{\ell(\theta)}{g(\theta)}\end{aligned}$$

- Where the final step is because $x/(bx+c)$ is monotonic in x for $x > 0$, let $x = l/g$
- In other words, we just need to find $\arg \max_{\theta} \frac{\ell(\theta)}{g(\theta)}$ which is doable via algorithm!

What is a Tree-Structured Parzen Estimator (TPE) (Part V)

- Now to define the algorithm, first we observe T (~ 10) trials randomly:

Observed trials: $\mathcal{D} = \{(\theta^{(i)}, y^{(i)})\}_{i=1}^T$

Quantile threshold: $P(y < y^*) = \gamma$

$$\mathcal{D}_{\text{good}} = \{\theta^{(i)} : y^{(i)} < y^*\}$$

$$\mathcal{D}_{\text{bad}} = \{\theta^{(i)} : y^{(i)} \geq y^*\}$$

- From this data, we want to build:

$$\ell(\theta) \equiv p(\theta \mid y < y^*)$$

$$g(\theta) \equiv p(\theta \mid y \geq y^*)$$

- So we define

$$\theta \equiv (\theta_1, \theta_2, \dots, \theta_d)$$

- And assume:

Independence assumption: $p(\theta \mid y < y^*) \approx \prod_{k=1}^d p(\theta_k \mid y < y^*)$

What is a Tree-Structured Parzen Estimator (TPE) (Part VI)

- This allows us to write:

$$\ell(\theta) \approx \prod_{k=1}^d \hat{p}_{\text{good}}(\theta_k)$$

$$g(\theta) \approx \prod_{k=1}^d \hat{p}_{\text{bad}}(\theta_k)$$

- We can fit to our samples to get a continuous distribution spaces for each param

$$\hat{p}_{\text{good}}(\theta_k) \leftarrow \text{fit to } \{\theta_k^{(i)} : \theta^{(i)} \in \mathcal{D}_{\text{good}}\}$$

$$\hat{p}_{\text{bad}}(\theta_k) \leftarrow \text{fit to } \{\theta_k^{(i)} : \theta^{(i)} \in \mathcal{D}_{\text{bad}}\}$$

- Then we sample from the “good” spaces for M candidates index by m

$$\tilde{\theta}_{m,k} \sim \hat{p}_{\text{good}}(\theta_k), \quad k = 1, \dots, d$$

- And finally, choose our next theta, add it to the data set, and repeat

$$\theta^{(t)} = \arg \max_{\tilde{\theta}_m} \frac{\ell(\tilde{\theta}_m)}{g(\tilde{\theta}_m)} = \prod_{k=1}^d \frac{\hat{p}_{\text{good}}(\tilde{\theta}_{m,k})}{\hat{p}_{\text{bad}}(\tilde{\theta}_{m,k})}$$

What is a Tree-Structured Parzen Estimator (TPE) (Part VII)

- How do we obtain our fits from the data?
- Uses [Kernel Density Estimation](#) (KDE)
 - The actual fit is done when determining each “gaussian kernel” (or sigma)
 - Likelihood-optimal is expensive $\sim O(n^2d)$
 - n = # samples
 - d = # hyperparameters
 - Optuna uses the “heuristic” version, which requires choosing some value for c .

Given samples: $\{\theta_k^{(i)}\}_{i=1}^n$, $\theta_k^{(i)} \in \mathbb{R}$

Kernel density estimate:

$$\hat{p}(\theta_k) = \frac{1}{n} \sum_{i=1}^n \mathcal{K}\left(\frac{\theta_k - \theta_k^{(i)}}{h_k}\right)$$

Gaussian kernel:

$$\mathcal{K}(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$$

Bandwidth (likelihood-optimal):

$$h_k^* = \arg \max_h \sum_{i=1}^n \log \hat{p}_{-i}(\theta_k^{(i)} | h)$$

Bandwidth (heuristic used in practice):

$$h_k = c \cdot \text{std}\left(\{\theta_k^{(i)}\}_{i=1}^n\right), \quad c > 0$$

Why use Graph Transformers?

-

Example Graph Transformer Layer

Example: Graph Transformer Layer (Neighborhood Self-Attention)

For each node v in the graph:

$$\begin{aligned} q_v &= W_Q h_v, \\ k_u &= W_K h_u, \\ v_u &= W_V h_u, \\ \forall u \in \mathcal{N}(v) \end{aligned}$$

$$\alpha_{vu} = \text{softmax}_{u \in \mathcal{N}(v)} \left(\frac{q_v^\top k_u}{\sqrt{d}} + f(e_{vu}) \right)$$

$$h'_v = \sum_{u \in \mathcal{N}(v)} \alpha_{vu} v_u$$

where:

h_v is the node embedding of node v ,

$\mathcal{N}(v)$ is the set of neighbors of node v ,

e_{vu} are edge features between nodes v and u ,

$f(\cdot)$ is a learned function producing an attention bias from edge features.

This update is applied independently to all nodes and repeated at each layer.

Building a Framework that Scales

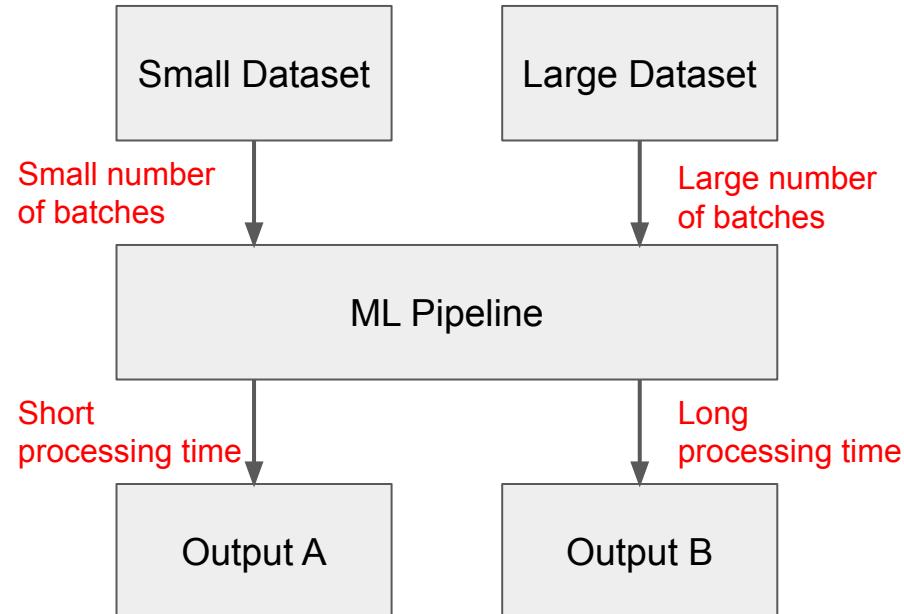
Breaking down “Scalability”

I choose to break down scalability into three categories:

1. Data
 - a. ML pipeline does not change behavior as data set grows
 - i. Still scales in execution time
2. Compute
 - a. ML pipeline does not change behavior as compute power is changed
 - i. Still scales in execution time
 - ii. Ex. Pipeline runs on developer's laptop or a CPU/GPU cluster
3. Codebase
 - a. ML pipeline does not (greatly) change behavior as complexity grows
 - i. New models, stages, or data products require additional code, not (major) code rewrites
 - ii. Iteration speed does not (greatly) degrade with system size (i.e. keep things modular!)

What Do We Mean by “Scaling” (Data)

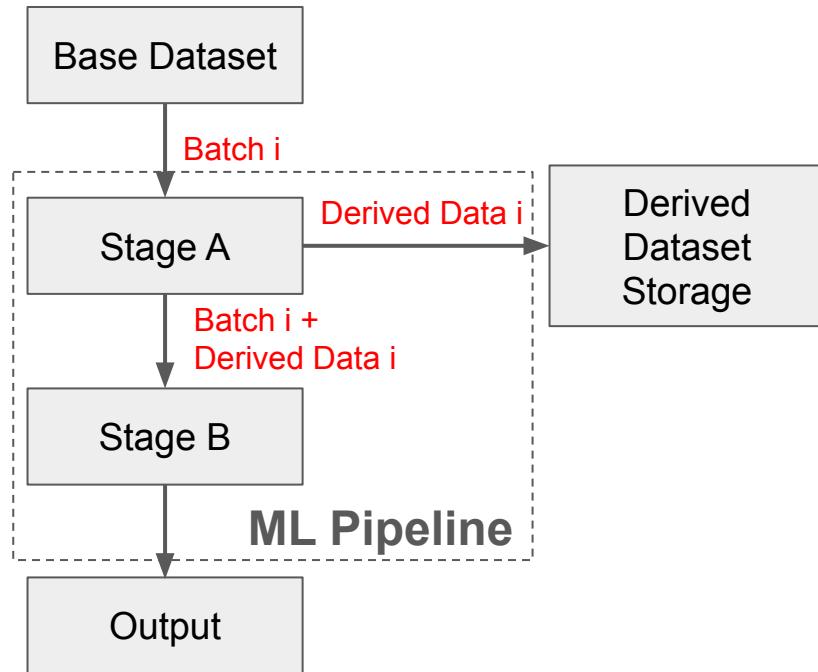
- **Data volume should be able to grow without changing how the pipeline behaves**
- Adding the following should not cause pipeline behavior changes:
 - More events
 - More derived data products
 - predictions, masks, regressions, etc.
 - More passes over the same data
 - Larger event representations
- Implications:
 - Memory usage must not grow with dataset size



Ideal Behavior for Different Sized Datasets

Techniques to Ensure Scalability (Data)

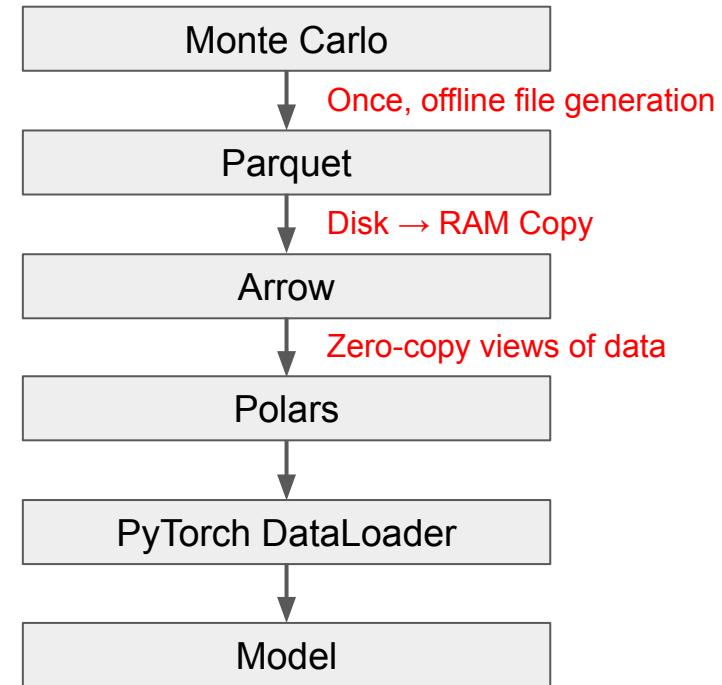
- Stream data in bounded batches
 - RAM usage should not scale with data set size
 - Allows RAM usage to be “tunable”
- Base data should be immutable
 - No modifications or extensions to ML pipeline input data files
- Separate derived data products
 - Predictions, masks, and regressions are produced as independent datasets
- Reference data by IDs, not by object
 - When passing data modules, use file paths or map IDs not in memory collections
 - Similar to why we use pointers in C++



Simple Example Pipeline Including Derived Datasets

Technologies for Scalability (Data)

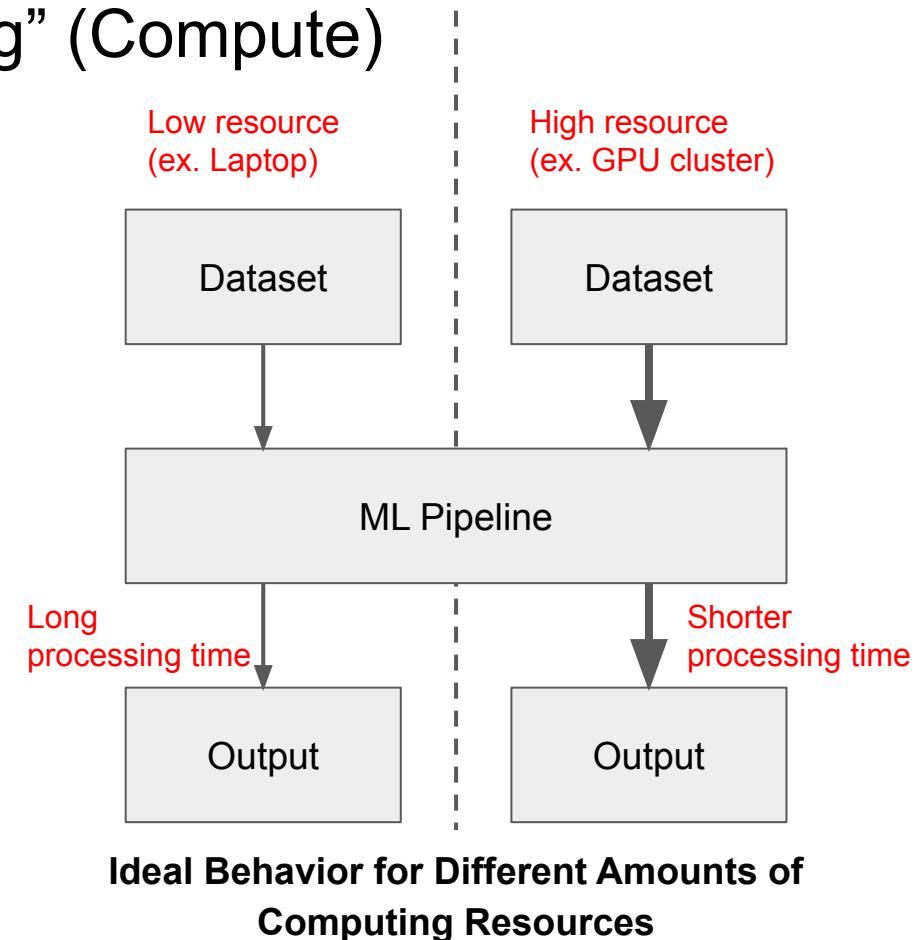
- **Apache Parquet** (columnar, on disk)
 - ML-native working datasets, append-only, shardable (aka “batchable”)
 - Can key rows by event ID, enabling batch-scoped joins for derived data products
- **Apache Arrow**
 - Single copy from disk into RAM, then zero-copy views all the way to Torch
- **Polars**
 - Dataset shaping: filtering, joins, feature derivation
 - Lazy, columnar execution on Arrow-backed data
 - Prepares batchable views for PyTorch ingestion
- **PyTorch tensors**
 - Execution format for models
- **Pytorch DataLoader**
 - Handles batching, shuffling, parallel loading, prefetching, enforcing memory bounds



Flow of Data From Monte Carlo to a Model in the ML pipeline

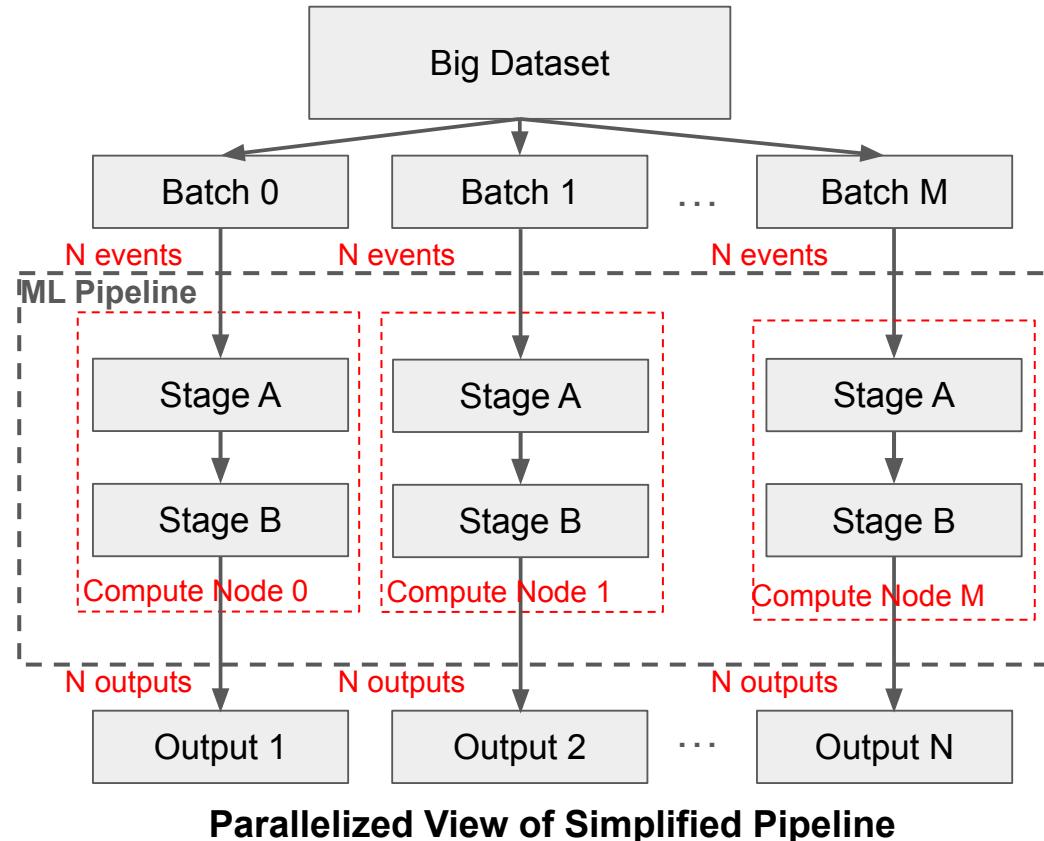
What Do We Mean by “Scaling” (Compute)

- Computation resources should be able to grow without changing how the pipeline behaves
- Adding the following should not cause pipeline behavior changes:
 - Number of GPUs
 - Number of CPU cores/threads
 - Node count
 - Memory Capacity
- Implications:
 - Algorithms must be agnostic to resources
 - Caveat: PyTorch and other frameworks may change their behavior for different devices/device counts for efficiency



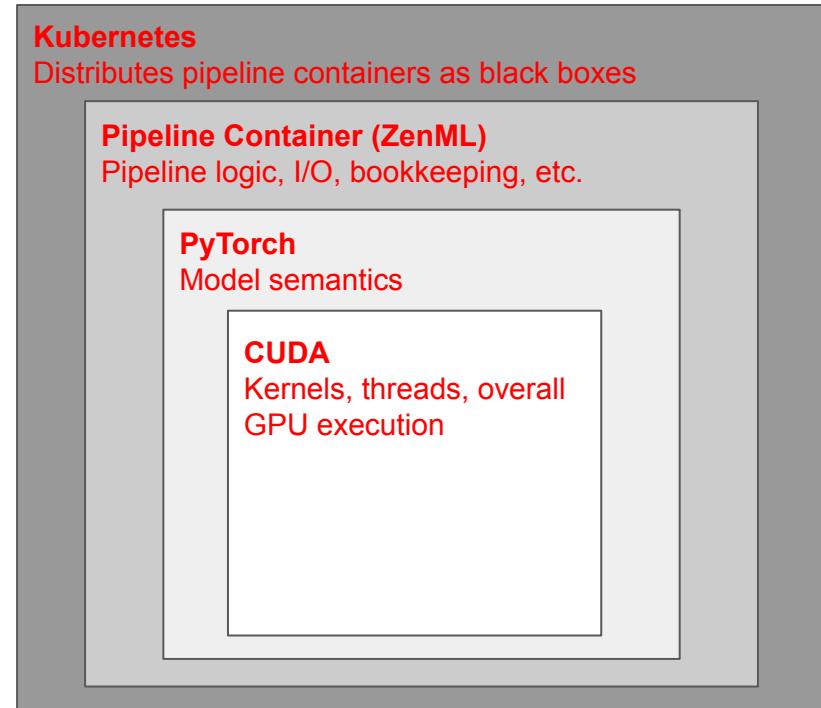
Techniques to Ensure Scalability (Compute)

- Make algorithms resource-agnostic
 - No logic branches based on GPU count, core count, node count, or memory size, etc.
- Make algorithms easily parallelizable
 - Decompose computation into independent, composable units
 - Avoid global state dependencies in pipeline stages
 - Avoid “synchronization points”; i.e. points where models must make inferences on whole datasets



Technologies for Scalability (Compute)

- **PyTorch**
 - Standard framework for modern ML models
 - Resource-agnostic execution model
- **CUDA**
 - Operates purely at the level of memory, kernels, and execution, independent of algorithm or pipeline semantics
 - Provides an API for launching and coordinating large numbers of parallel threads on GPUs
- **Kubernetes**
 - Schedules identical pipeline executions onto available compute nodes
 - Scales how many pipelines run concurrently, not what they do
 - Handles retry logic and resource limits

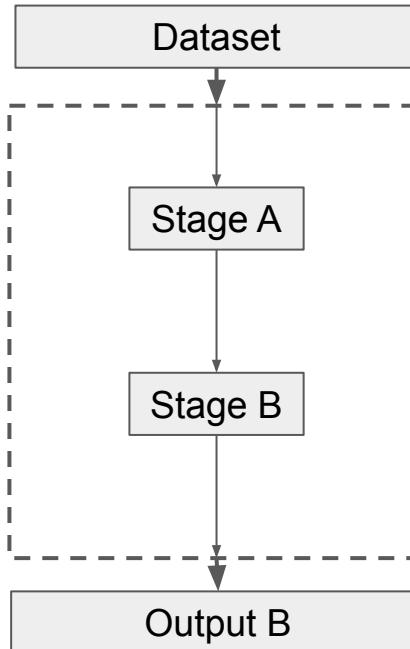


Simplified “Scope” Of Technologies
Outer Technologies Manage Inner Technologies

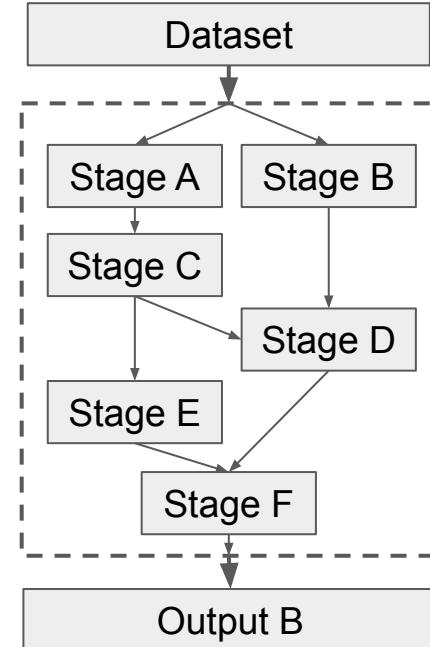
What Do We Mean by “Scaling” (Codebase)

- **Codebase complexity should be able to grow without changing how the system behaves**
- Adding the following should not cause system wide behavioral changes:
 - More pipeline stages
 - More models / algorithms
 - More configuration options
- Implications:
 - New functionality should be added by extensions, not modification
 - System behavior should be locally understood (modularity)
 - Existing code should not require global refactoring to evolve

“Simple” pipeline with few stages



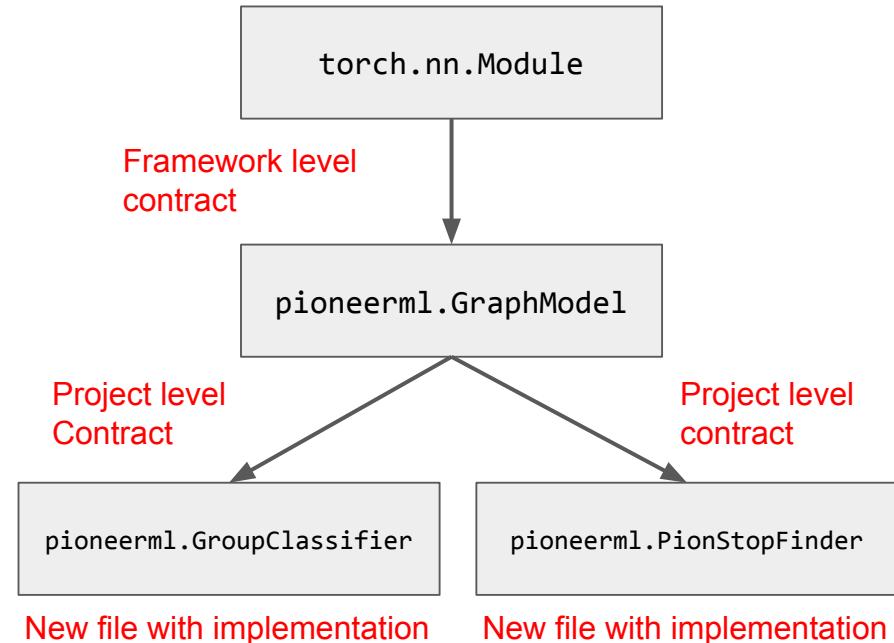
“Complex” Pipeline with many stages



The “Global” Structure Should Not Change As Complexity Increases

Techniques to Ensure Scalability (Codebase)

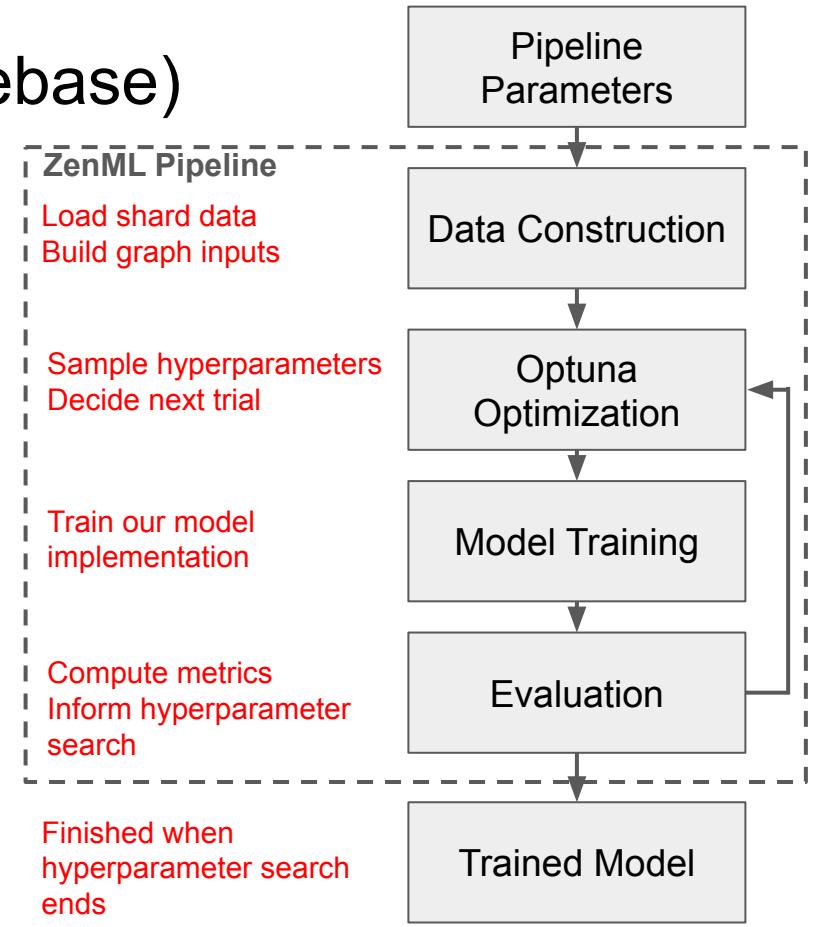
- Use abstraction where appropriate
 - Define stable base interfaces / abstract classes
 - Add new functionality by extending, not rewriting
- Avoid global coupling
 - Modules depend only on explicit inputs
 - Each module manages its own immutable local state
- Isolate responsibilities to achieve modularity
 - Each module has a single, well-defined responsibility
 - Changes remain local to the owning module



Simple Example of Abstraction For Adding New Models

Technologies for Scalability (Codebase)

- PyTorch
 - Provides a standard base abstraction classes for many model types (ex. `nn.Module`)
 - Enforces a consistent model interface (ex. `forward` method)
- ZenML
 - Encodes pipelines as composable, declarative units and orchestrates execution
 - Allows pipelines to grow by adding or reordering steps; easy to add new pipelines
 - Manages pipeline state, artifacts, and execution metadata outside user code
- Optuna
 - Isolates hyperparameter search code
 - Enables experimentation without modifying core implementations
 - Really a package for black box searching, by default uses [Tree-Structured Parzen Estimator](#) (TPE)



What is Apache Parquet?

- **Apache Parquet is a columnar, on-disk data format that is widely used in ML workloads**
- What parquet does
 - Columnar storage → read only the columns (features) your model needs
 - Allows one to efficiently assign a subset of columns as inputs and another subset of columns as targets
 - Supports nested and variable-length fields
 - Data schema is embedded in the file, not inferred by code (ex. Not like numpy, where code defines `dtype` parameter)
- Why this matters for scalability
 - Efficient I/O for large datasets through compression and encoding
 - New models using different subsets of the data becomes trivial

Columns:

```
event_id
theta
phi
pion_stop_x
pion_stop_y
pion_stop_z
total_pion_energy
...
hit_coord→ list < float >
hit_z→ list < float >
hit_energy→ list < float >
hit_pdg_mask→ list < int >
```

Example `time_groups.parquet` file structure

What is Batching?

- **Batching means processing a fixed-size subset of events at a time, rather than the full dataset**
- What batching does
 - Groups individual events into batches of size N
 - Each batch is processed independently
 - Batches are discarded after use
- Why this matters for scalability
 - Memory usage is bounded by batch size (for single batch process)
 - Dataset size does not affect RAM usage
 - Enables streaming over arbitrarily large datasets

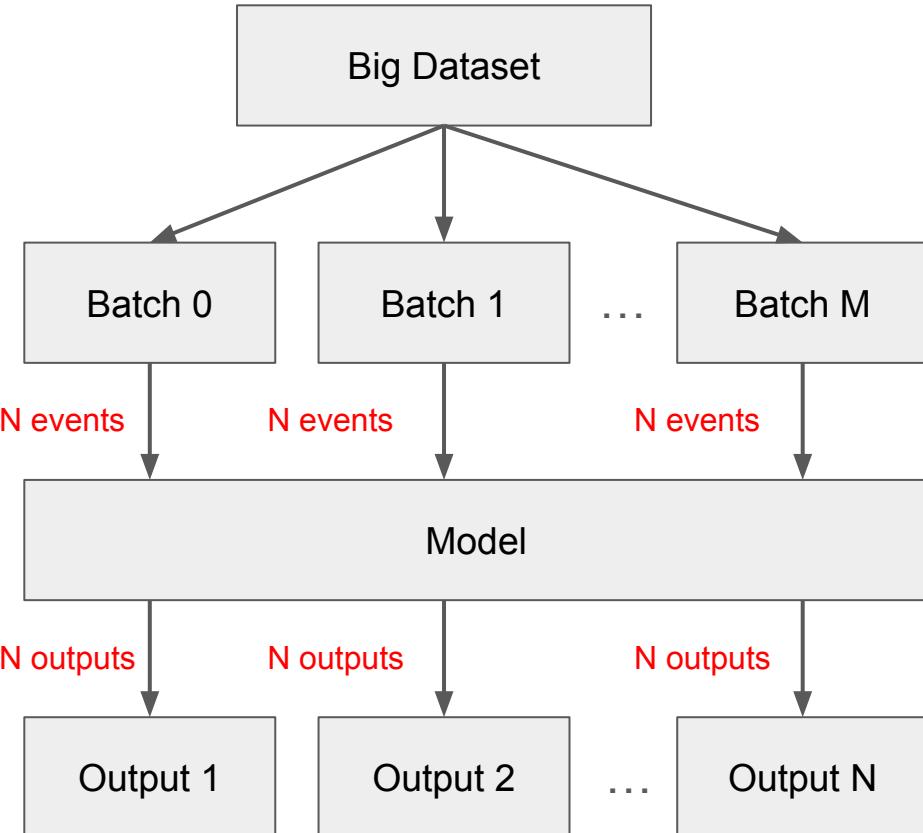
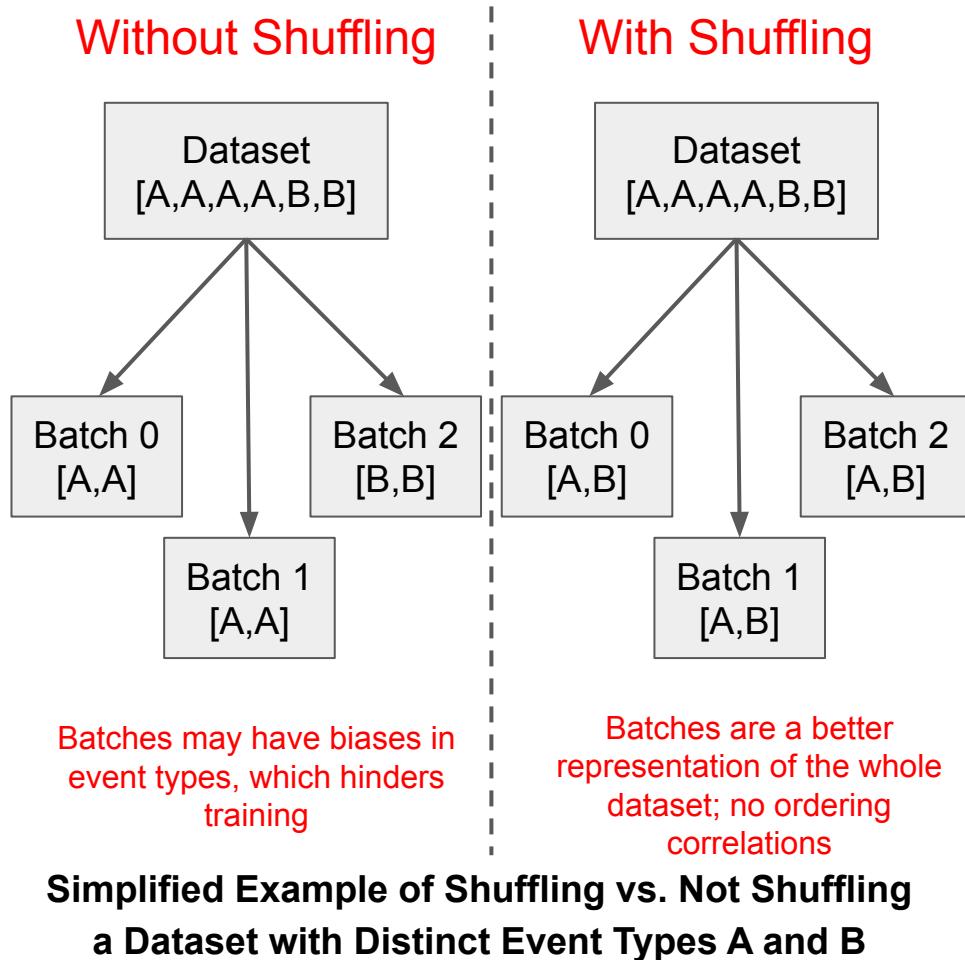


Diagram that Shows how Batching Splits Data

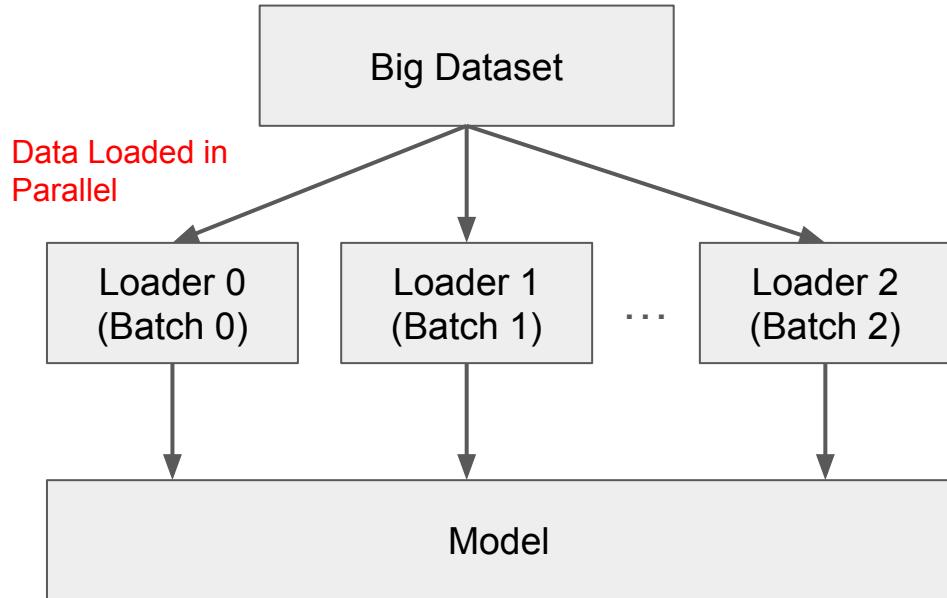
What is Shuffling?

- **Shuffling changes the order in which events are seen, without changing the data itself**
- What shuffling does
 - Randomizes event order before forming batches
 - Ensures batches contain a mix of events
 - Changes between epochs (or passes over the data)
- Why this matters for scalability
 - Prevents bias from data ordering
 - Improves statistical independence between batches
 - Allows repeated passes over large datasets without correlation artifacts



What is Parallel Loading?

- **Parallel loading means loading multiple batches concurrently, using multiple workers**
- What parallel loading does
 - Multiple workers read and prepare batches simultaneously
 - The model always has a batch ready to process
 - Data loading is decoupled from model execution
- Why this matters for scalability
 - Helps prevent the model from waiting on disk I/O
 - Improves hardware utilization (especially GPUs)



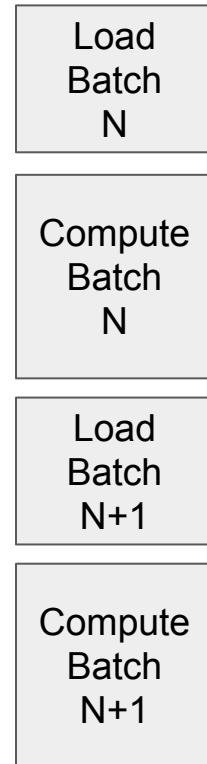
Note: Parallel loading does **not** necessarily mean parallel model execution

Parallel Data Loading Diagram

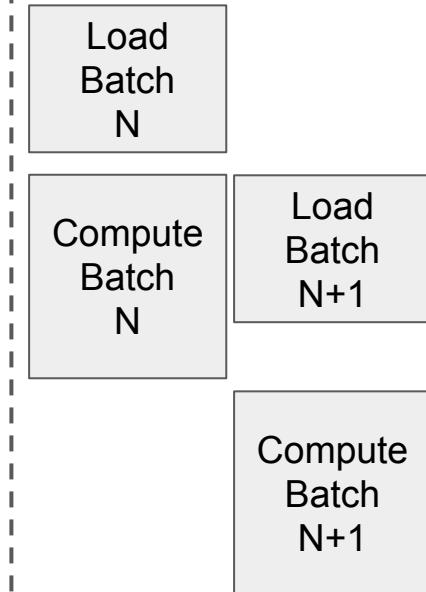
What is Prefetching?

- **Prefetching means loading future batches while the current batch is being processed**
- What prefetching does
 - While the model computes on batch N the next batch (N+1) is loaded in the background
 - When computation finishes, the next batch is already ready
- Why this matters for scalability
 - Helps prevent the model from waiting on disk I/O

Without Prefetching



With Prefetching



Subsequent batches loaded in parallel with compute of previous batch. Effectively utilizing different hardwares

Simplified Prefetching Example Diagram

What is Enforcing Memory Bounds?

- **Enforcing memory bounds means placing a hard limit on how much data can be in memory at once**
- What enforcing memory bounds does
 - Maximum in-flight data size is fixed
 - Batch creation is throttled when memory is full
 - Makes memory usage predictable and stable
- Why this matters for scalability
 - Dataset size does not affect RAM usage
 - Enables streaming over arbitrarily large datasets

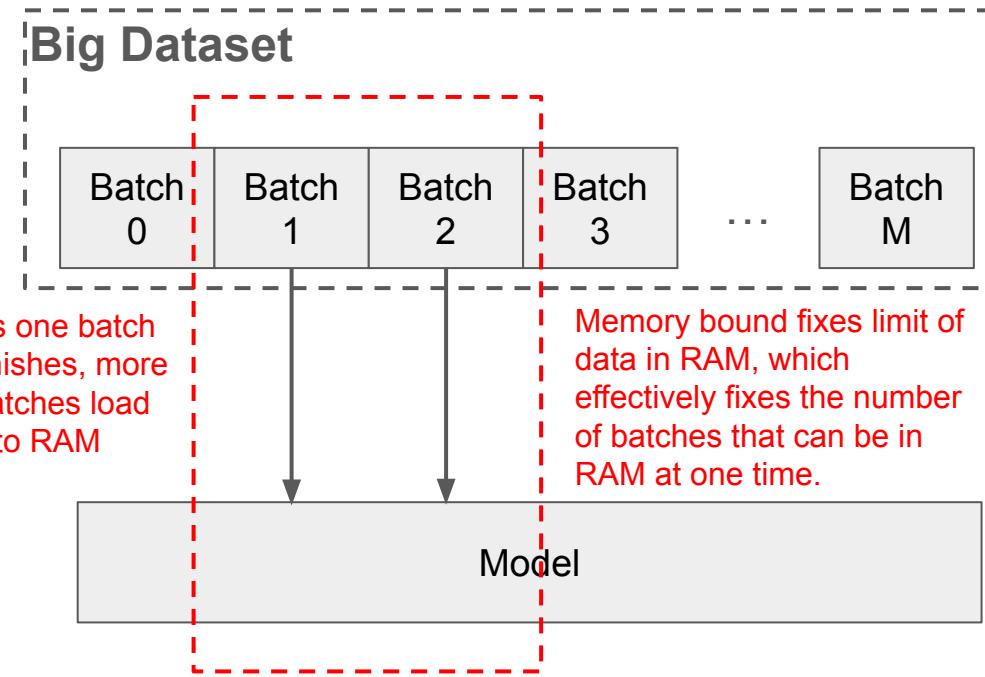
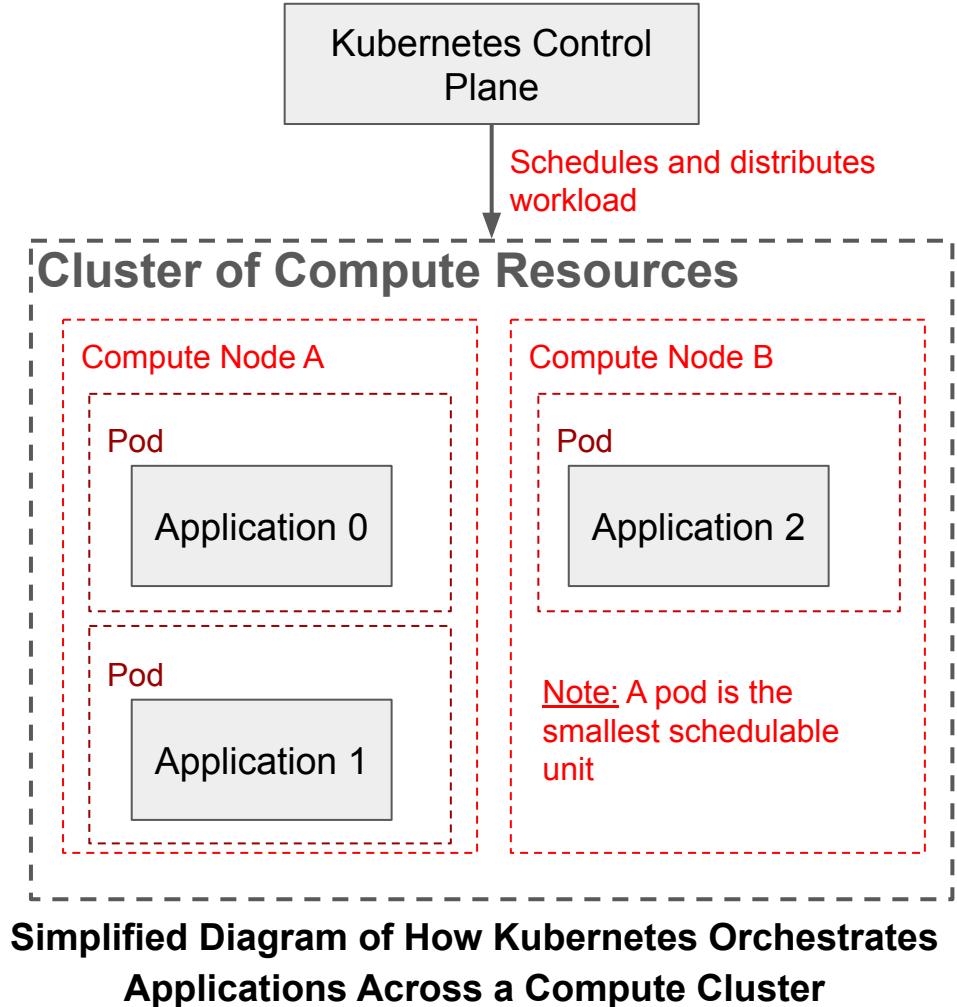


Diagram Showing How Memory Bounds Enforce a Limited Number of Batches Loaded in RAM

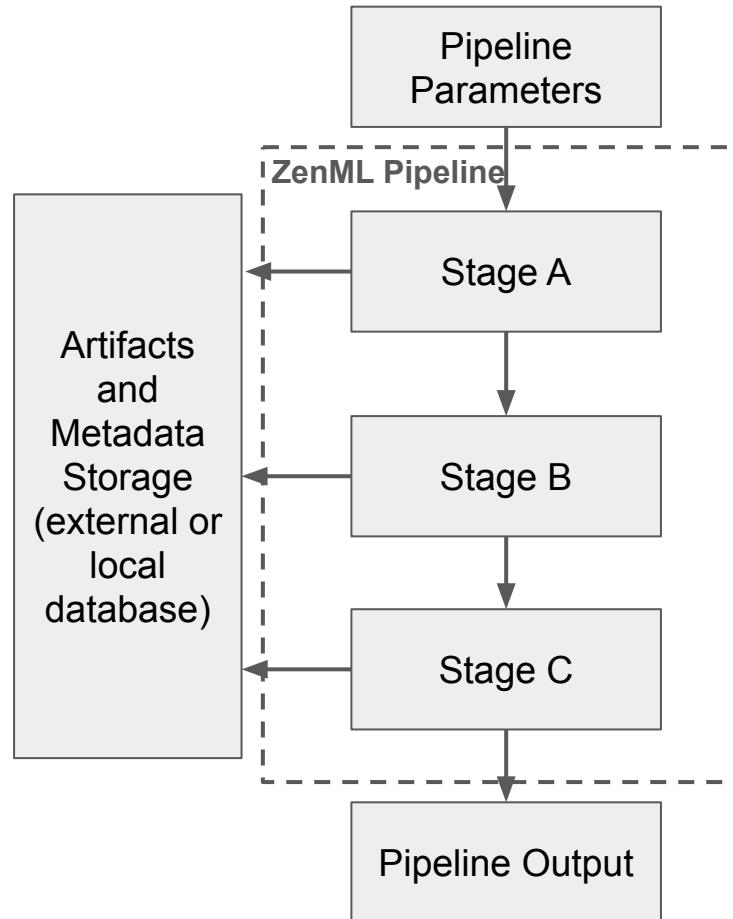
What is Kubernetes (K8s)?

- **A system for running and managing containerized workloads across shared compute resources**
- What Kubernetes does
 - Schedules containers onto available compute nodes
 - Enforces resource limits (CPU, GPU, memory) per container
 - Isolates workloads from one another
 - Handles restarts and retries on failure
- Why this matters for scalability
 - Enables concurrent execution of many independent workloads
 - Prevents resource contention between workloads



What is ZenML?

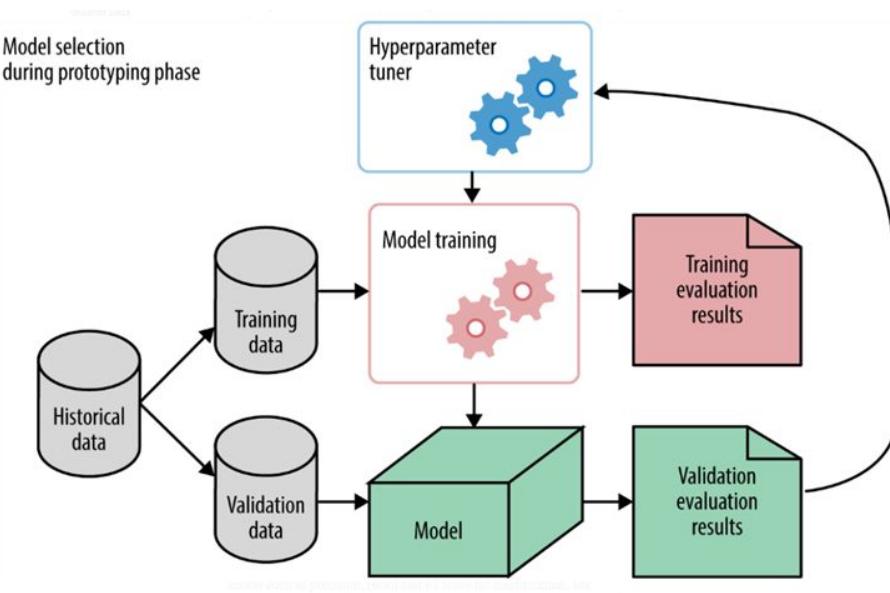
- **A framework for defining, orchestrating, and executing machine-learning pipelines with explicit steps, artifacts, and metadata**
- What ZenML does
 - Encodes workflows as composable, declarative pipelines
 - Orchestrates execution order and dependencies
 - Integrates with execution backends (local, containers, clusters) without changing user code
- Why this matters for scalability
 - Pipelines grow by adding or rearranging stages, not rewriting logic
 - Enables reproducibility, versioning, and parallel development
 - State and artifacts are managed outside user code



Example ZenML pipeline diagram

What is Optuna?

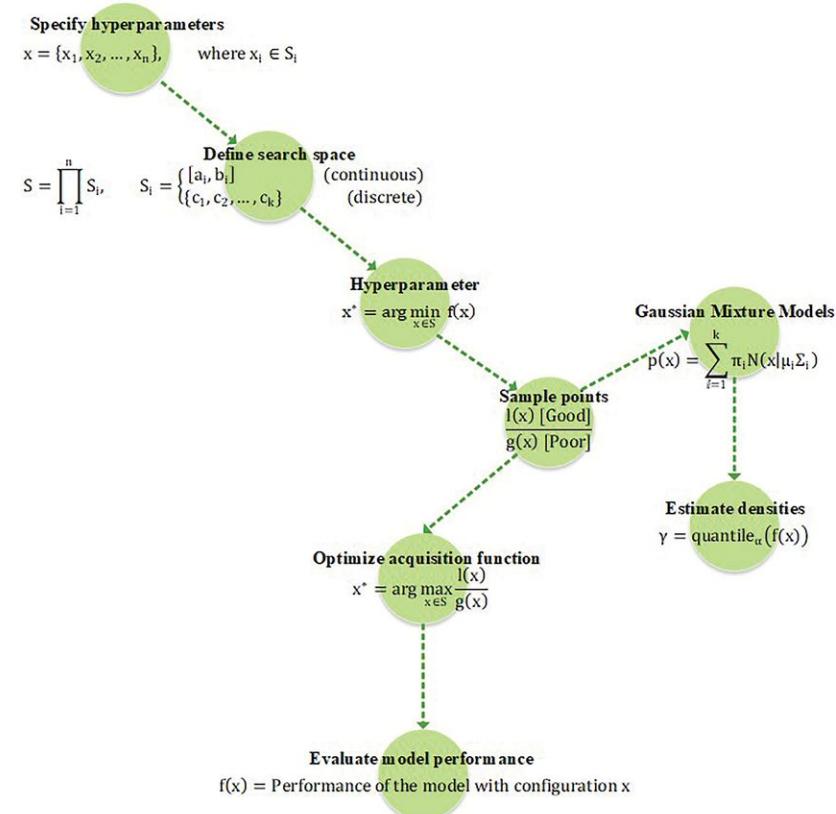
- Many hyper parameters means manual tuning is bad
 - Too slow
 - Suboptimal tuning means you spend more resources (time, computing power, etc.) training unused models
- Optuna is a python package that solves this problem
 - Framework for optimization black box objective functions
 - Technically not an ML package, but rather a package that supports many optimized sampling strategies



Example Optuna workflow diagram

How Optuna Works

- Optuna supports many sampling algorithms, examples:
 - Grid search
 - Random search
 - Gaussian process-based Bayesian optimization
- For single object functions, the default for Optuna is Tree-Structured Parzen Estimator (TPE)



TPE flow diagram (in a nutshell)

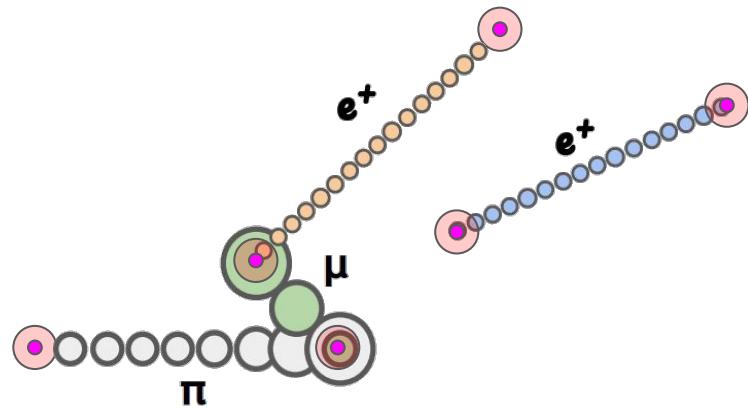
AI Reconstruction Pipeline

Endpoint Regressor

● = Median endpoint estimate

○ = Error range determined from quantiles

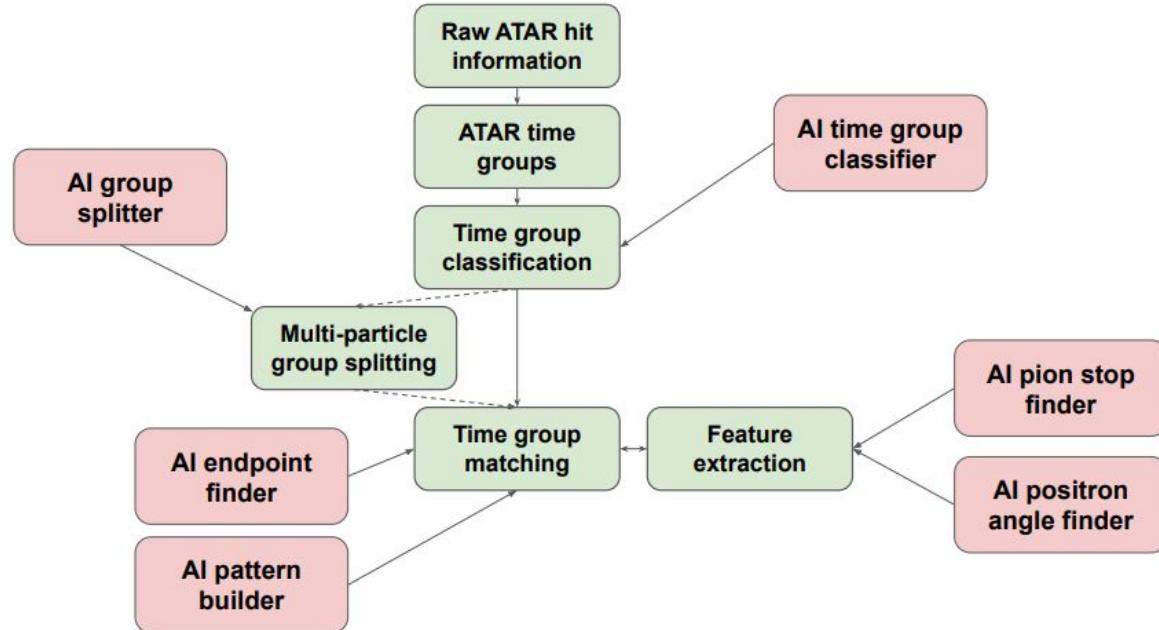
- Quantile graph transformer regressor
- Inputs:
 - Detector response event information represented as a graph
 - Predicted time group level particle class
 - Predicted hit level particle classes
- Targets:
 - Truth level endpoints
 - Particularly:
[[x_start, y_start, z_start], [x_end, y_end, z_end]]
- Outputs:
 - Predicted endpoints with quantiles
[[[x_start_q1, y_start_q1, z_start_q1], [x_end_q1, y_end_q1, z_end_q1]], ...]



Cartoon example of how we estimate endpoints with uncertainties given by the quantiles

Full Reconstruction Diagram

- Conceptually, all the same stages
- The idea is the ML models are “implementations” of a stage
 - In principle they can be swapped out with a traditional reconstruction (non-machine learning)
 - Allows for benchmarking of different implementations



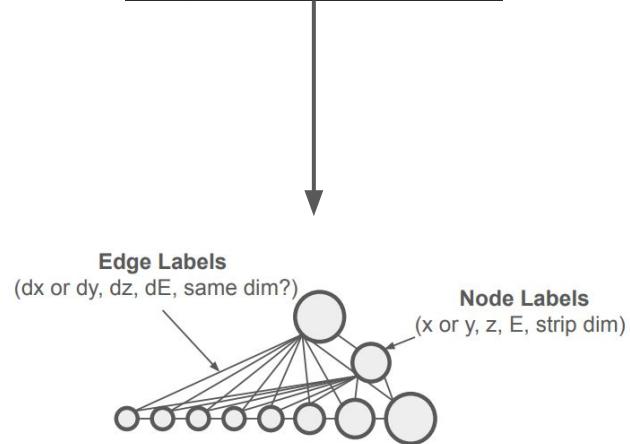
Why Have Divisions?

- Each pipeline stage operates on a well-defined “sample”
 - Each division defines a **dataset boundary**
 - A dataset boundary is the point at which the meaning of a row is fixed and written to storage, so downstream stages can rely on it without knowing how it was produced
 - It’s okay to create appending files or masks,
 - If you must mutate data (i.e. change your definition of a “sample” then you create another dataset boundary), then you should create a new dataset boundary or “division”
- If we don’t create clear divisions
 - Hidden coupling between stages can occur
 - I.e. you change one stage, an unrelated stage no longer works
 - Batching becomes unclear
 - Your definition of a batch changes between data boundaries
 - Memory and performance become less predictable
- In short, each division should scale separately

GraphRecord Former

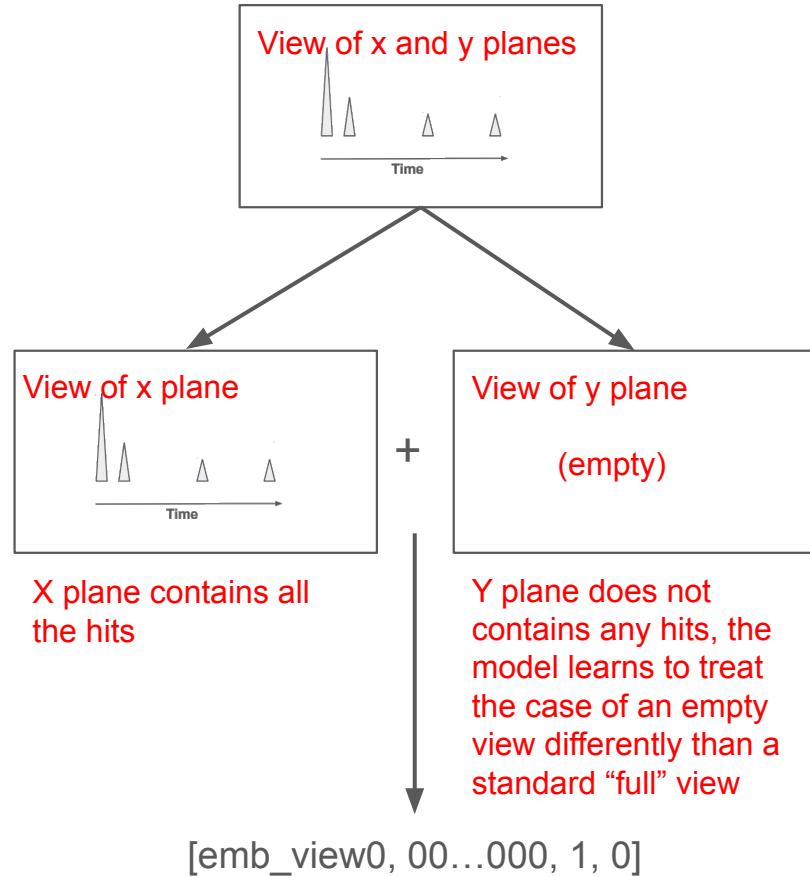
- We don't store data as their graphs, we store them as the minimal information needed to build the graph
- As a result we have a “data loading stage” before running any model where we construct a graph from the data

```
Columns:  
_____  
event_id  
theta  
phi  
pion_stop_x  
pion_stop_y  
pion_stop_z  
total_pion_energy  
...  
hit_coord→ list < float >  
hit_z→ list < float >  
hit_energy→ list < float >  
hit_pdg_mask→ list < int >
```



Stereoscopic View

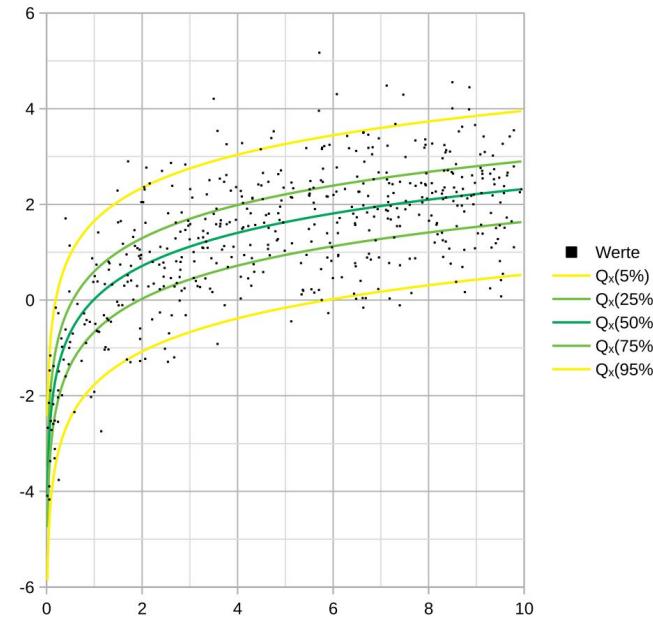
- Before each “Division 1” model “does its job”, it first constructs a “stereoscopic” view of the data
 - Hits are split by view:
 - 0 = X plane, 1 = Y plane
 - Each view is embedded independently, then fused:
[emb_view0, emb_view1, mask0, mask1]
 - mask_i = 1 if view i has hits, else 0
 - This process is learned for each model
- Why this helps:
 - Treats view ID as discrete routing
 - Explicitly encodes that intermediate views (e.g. “0.5”) do not exist
- Why we want this stage learnable:
 - weight each view
 - rescale per-view features
 - handle missing or weak views
- Why it’s model-specific
 - Different tasks rely on views differently
 - e.g. missing Y view matters more for endpoint finding than for particle classification



Exaggerated cartoon example of how a stereoscopic view can help in the case of missing views

Quantiles

- For regressions, quantiles are useful to extract uncertainties
- Uses a quantile regression loss function
- We can use quantile regression to have our models target a specific quantiles
 - Usually we select
 - Lower quantile = $16 \sim -1\sigma$
 - Mid quantile = $50 \sim \text{median}$
 - Upper quantile = $84 \sim +1\sigma$



**Wikipedia example of Quantile Regression.
Notice how it forms confidence bands around
the best fit (or median)**

Event Mixing

- Event mixing is handled in the pioneer simulation framework in the detector response
 - Effectively piles up simulated events into one “detector” event. The number of simulated events and probability of pileup are parameterized.