NICA

Application of machine learning in the current BM@N model for particle tracking

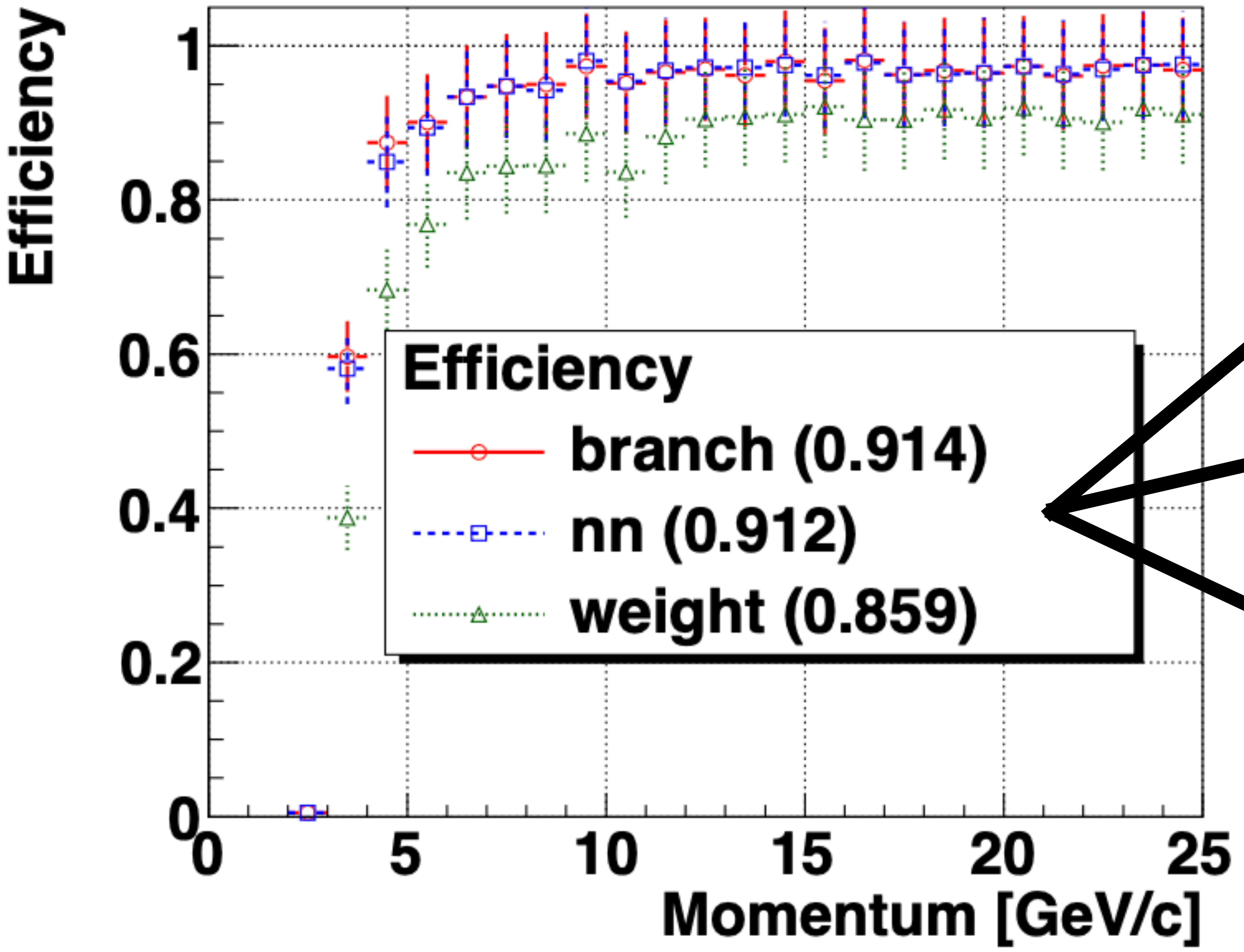
Anatoly Aleksandrov,
SPbU student

BM@N

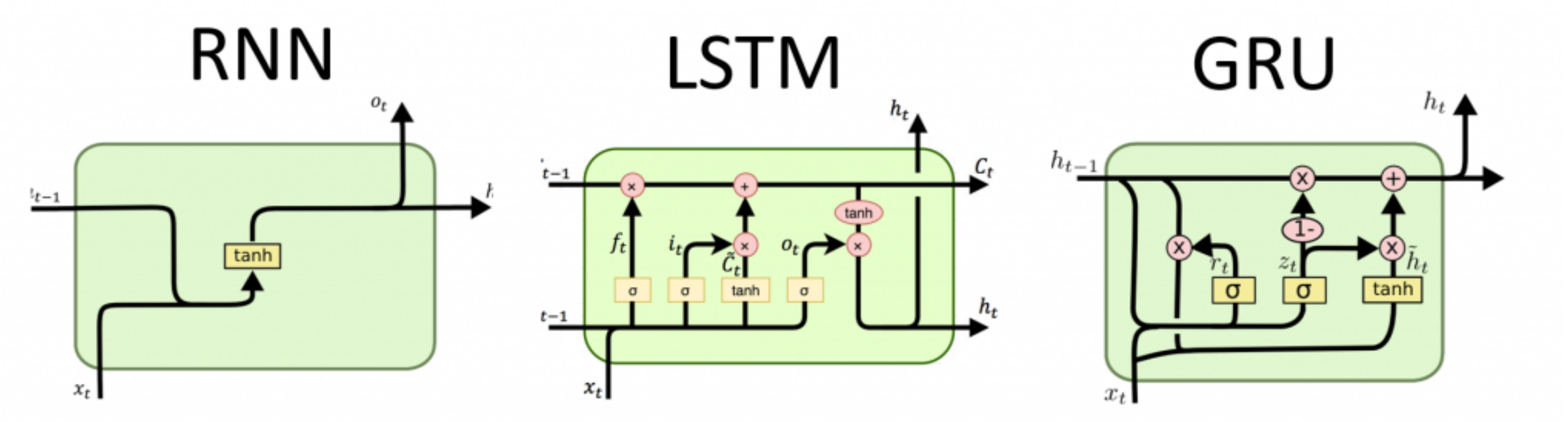
St Petersburg University

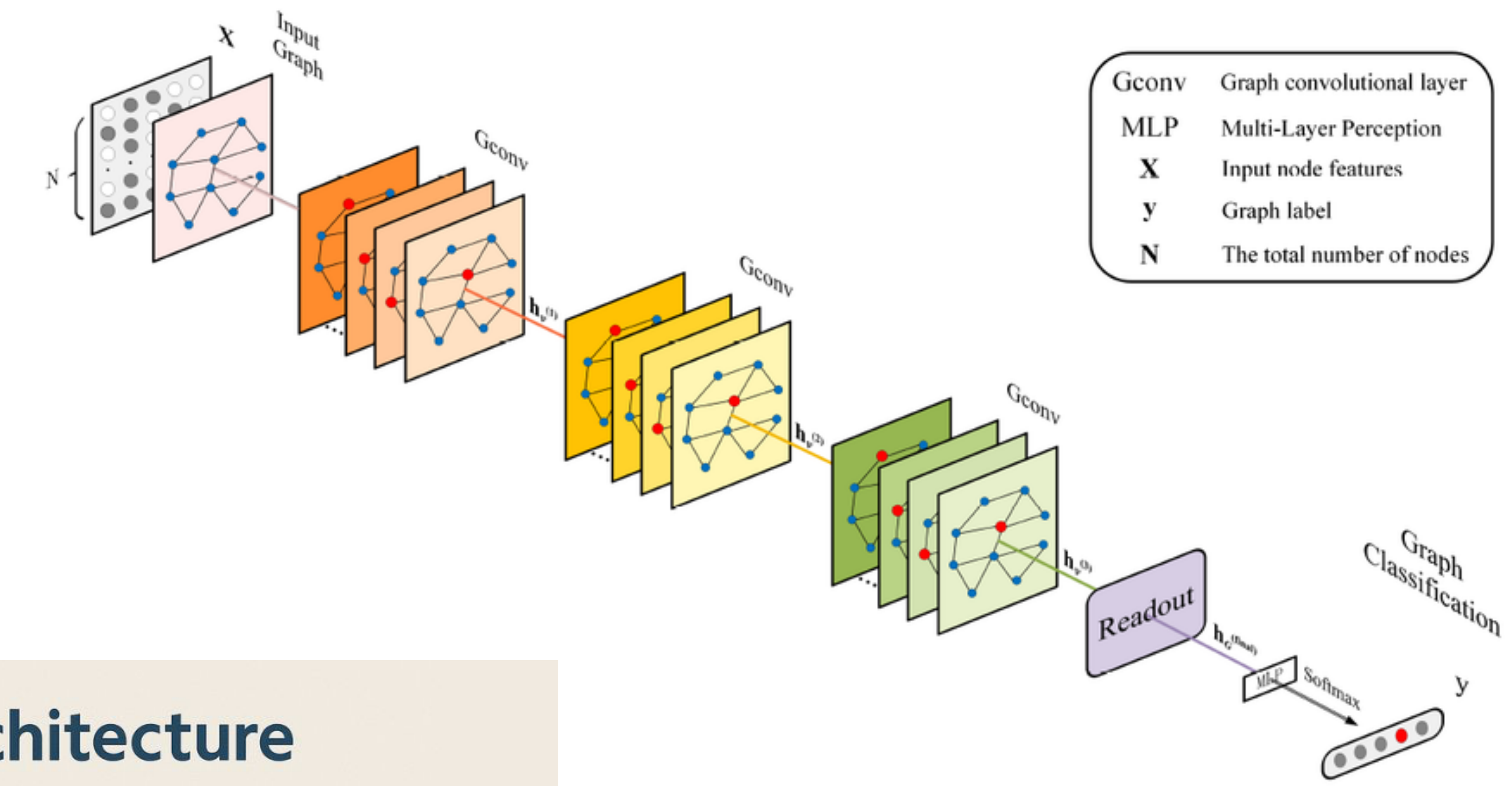
14th Collaboration Meeting of The BM@N Experiment at NICA

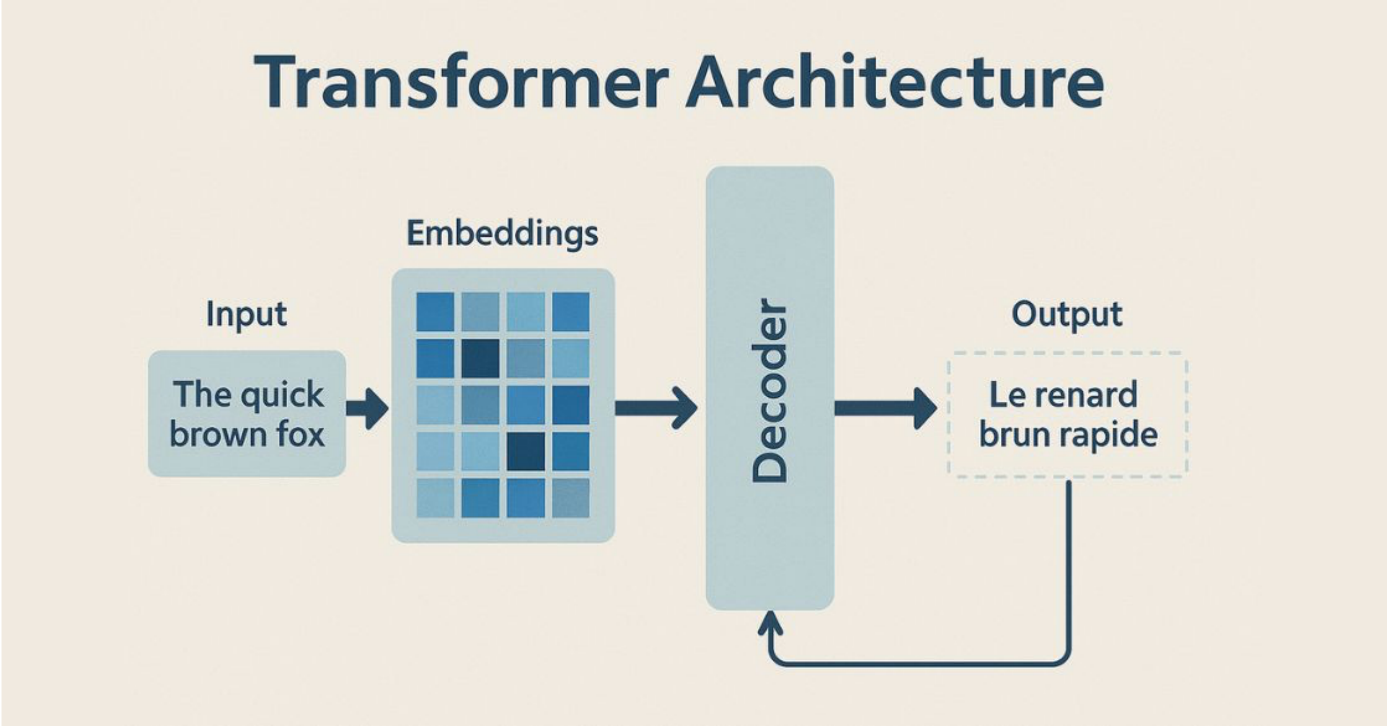
Key Challenge



Efficiency of track search algorithms used in bmnroot







Attempting to increase accuracy and speed up tracking performance using ML

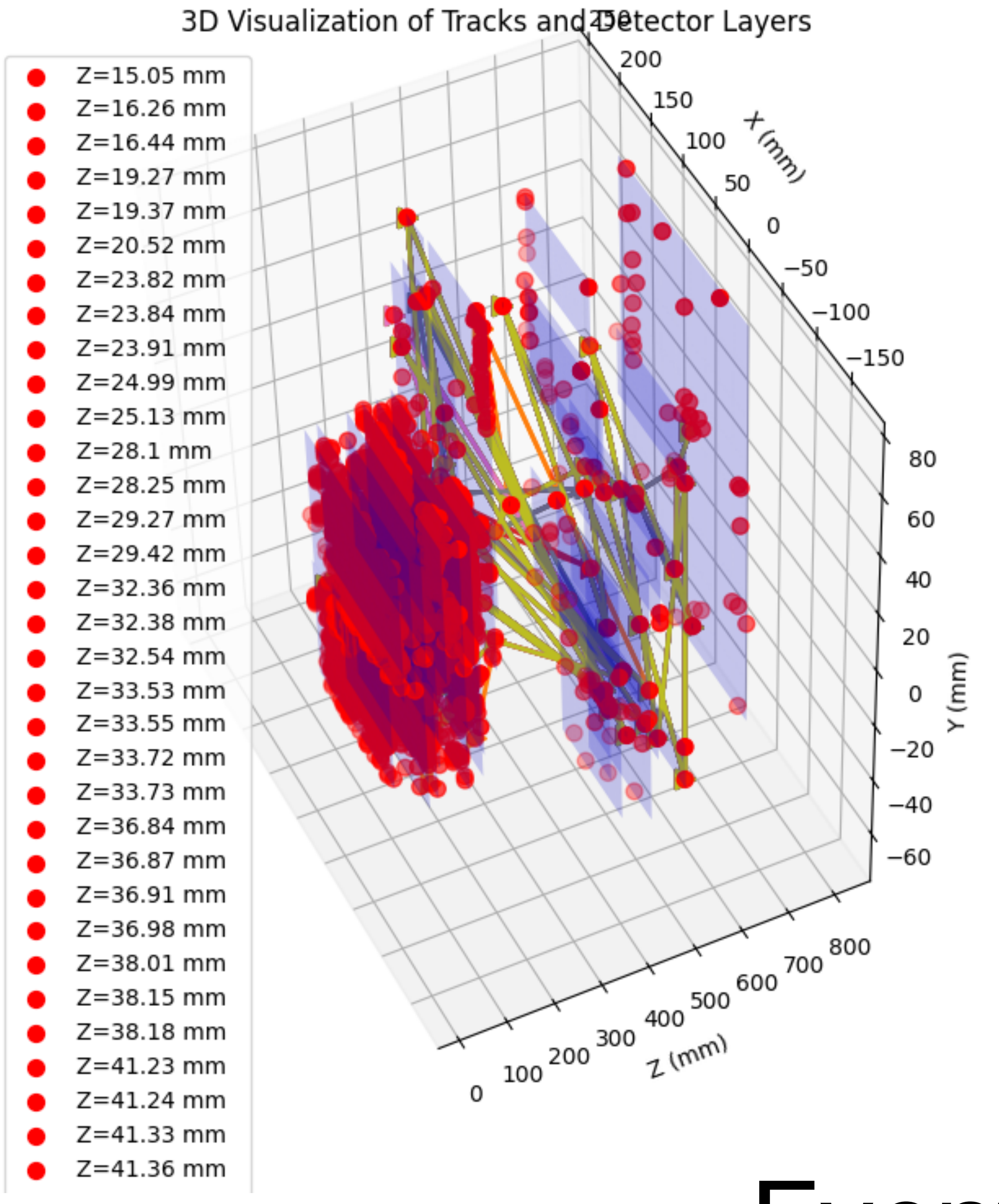
Software stack

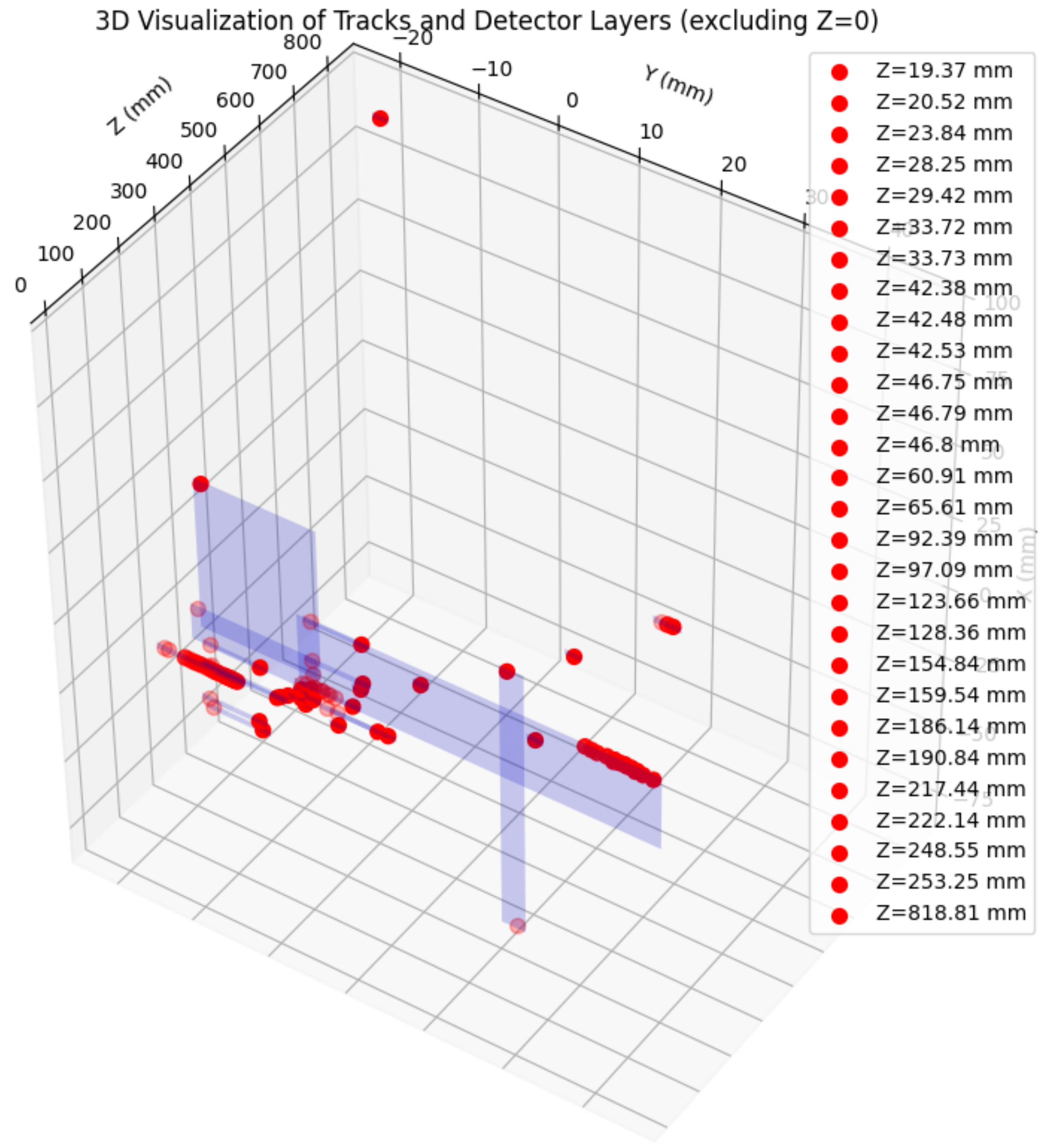
- Python
- Torch library for machine learning
- PostgreSQL for event data store, preparation and filttering

Architecture of neural particle tracker

Variable hits quantity

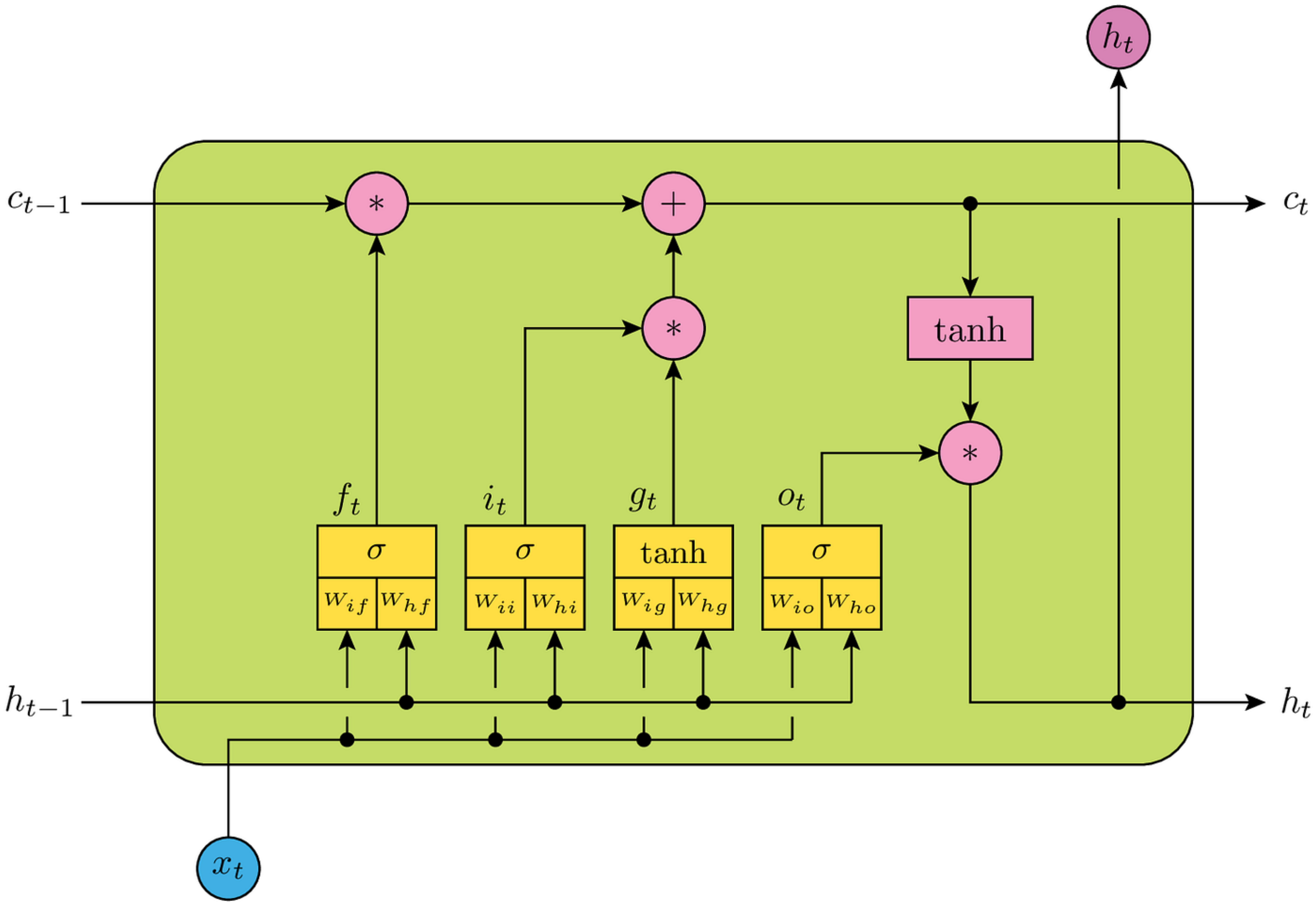
- Recurrent neural network (RNN)
- Convolution neural network (CNN)
- Density-based spatial clustering of applications with noise (classic clustering algorithm)





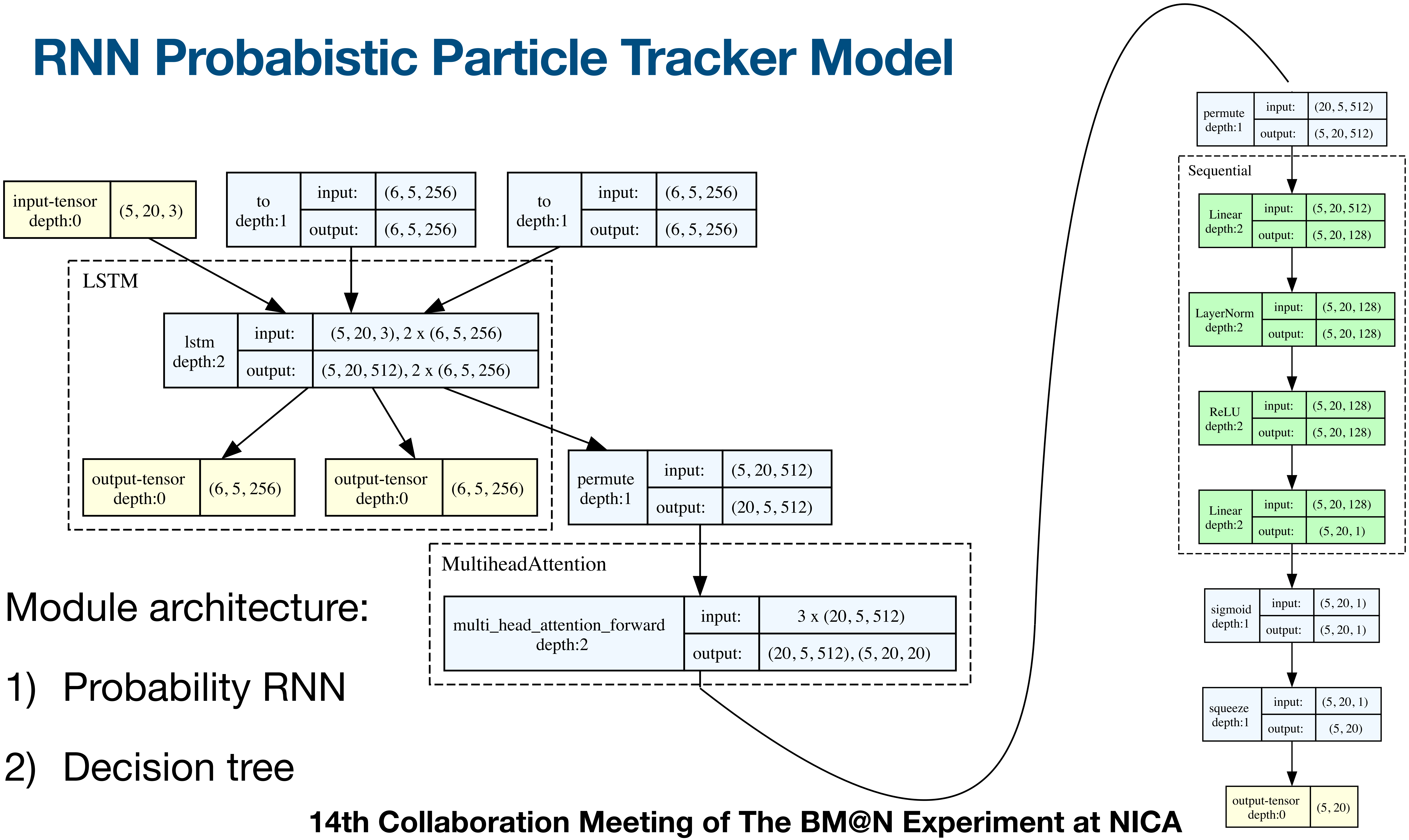
Event plots

RNN



$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

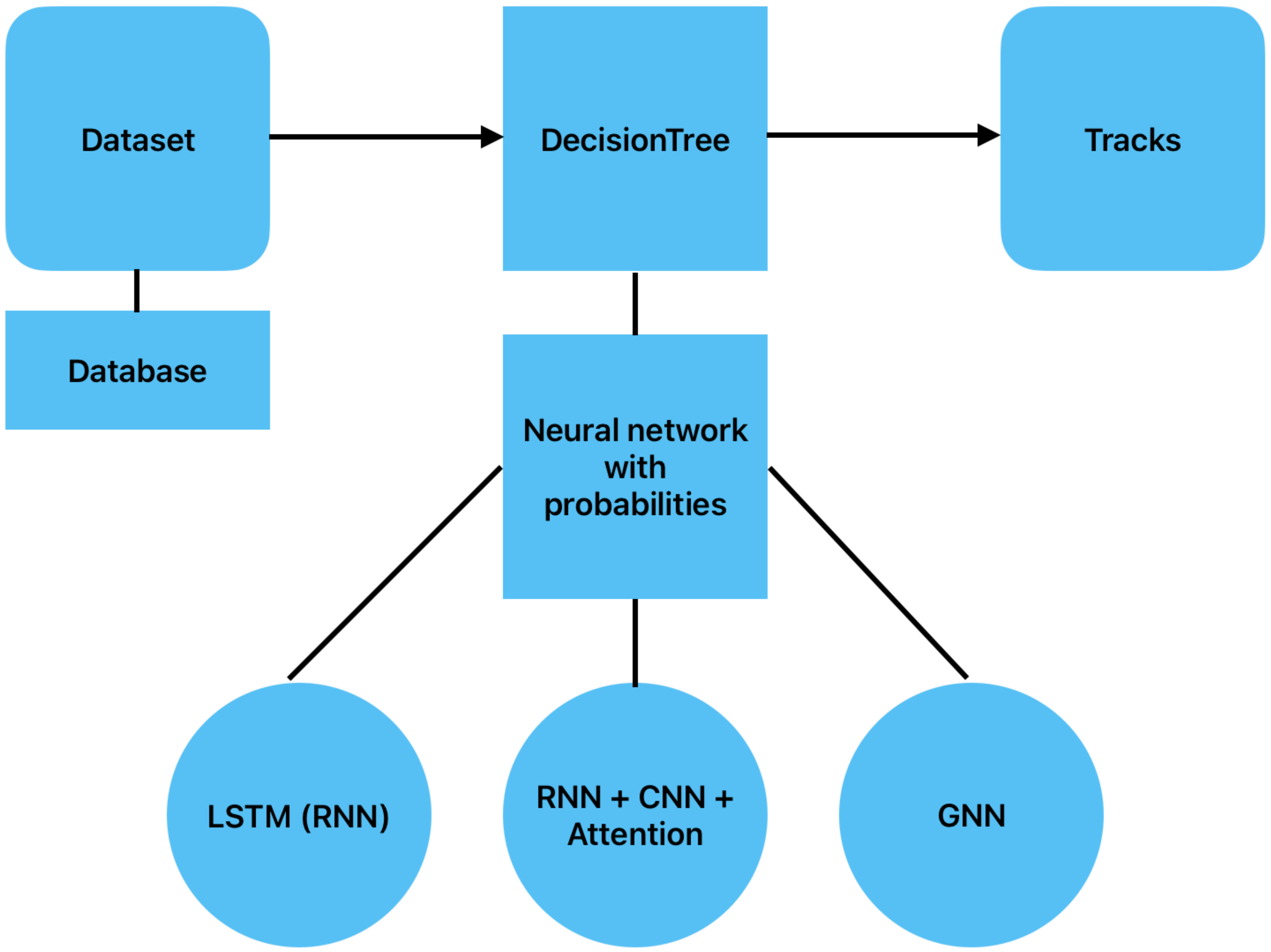
RNN Probabistic Particle Tracker Model



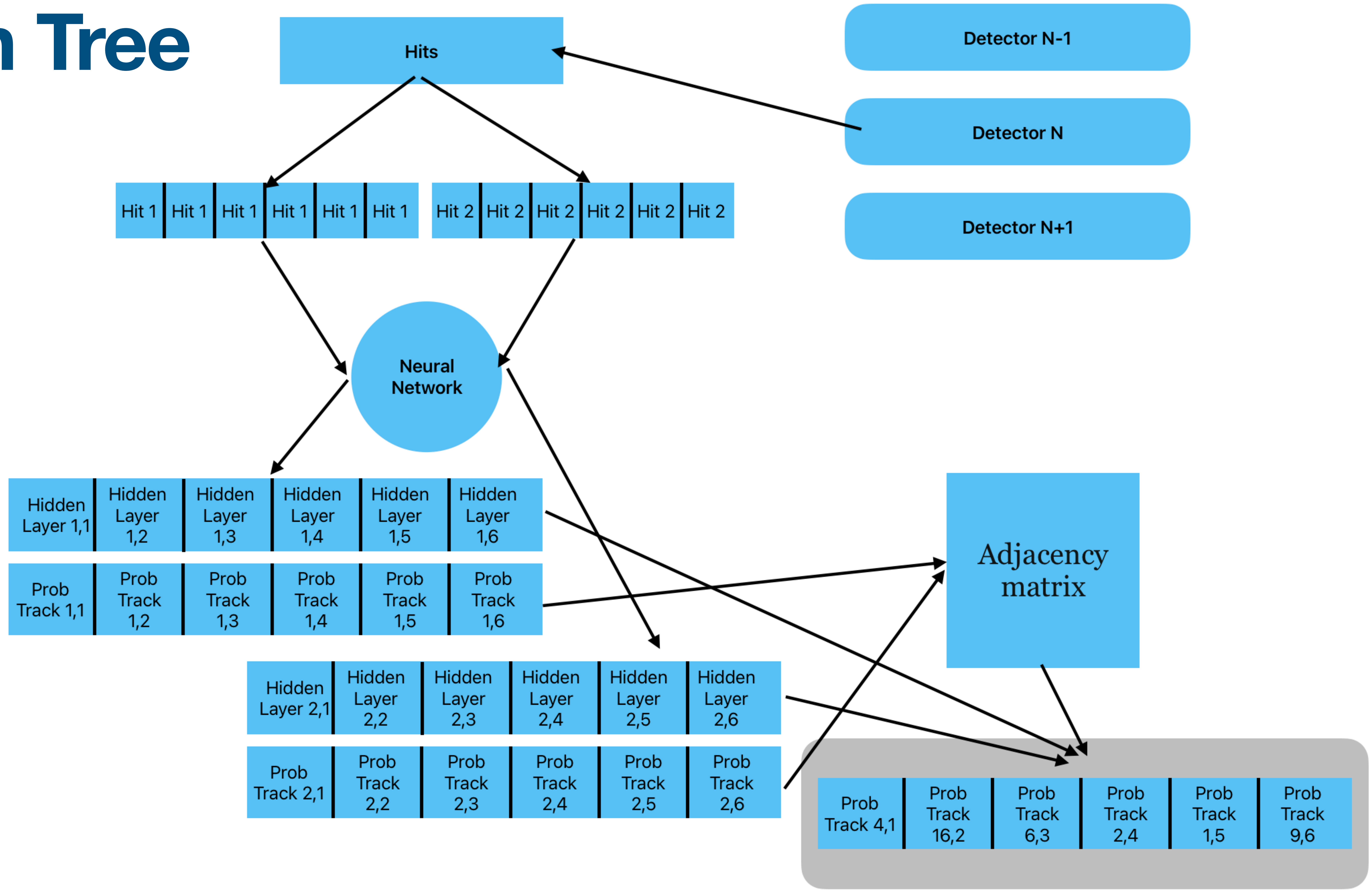
Module architecture:

1) Probability RNN
2) Decision tree

Modular architecture



Decision Tree



Supervised learning in particle tracking task

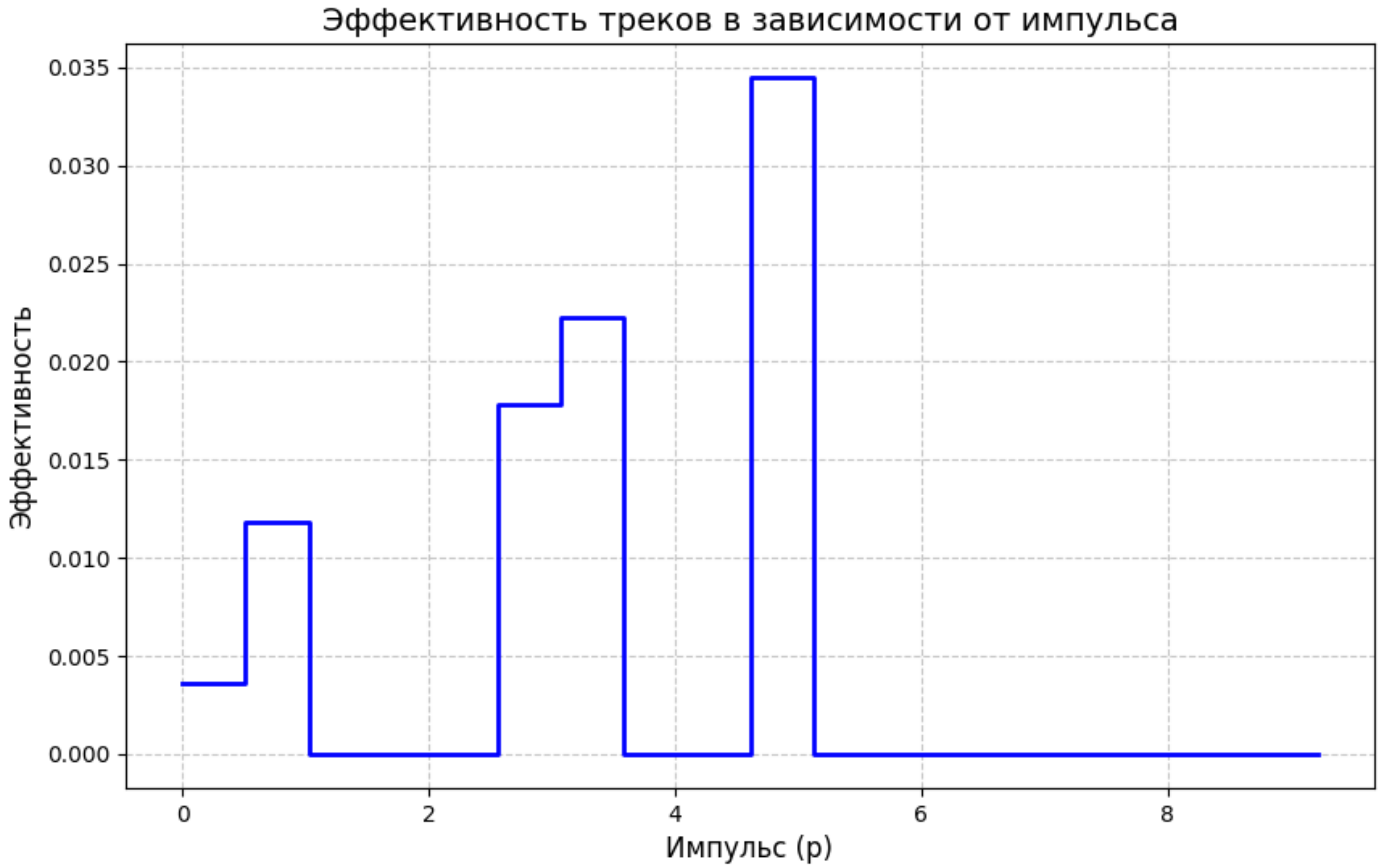
```
Используемое устройство: mps
INFO:root:Loaded 1005055 examples from 2064 batches in 0.10s
INFO:root:Loaded 557825 examples from 1276 batches in 0.07s
Потеря: 4.7148
Потеря: 10.1925
Потеря: 14.7846
Потеря: 20.8737
Потеря: 25.9207
Потеря: 30.0582
Потеря: 34.7613
Потеря: 40.1623
Accuracy %:  99.99282816273826
```

Train hits

Train labels

```
tensor([[[-2.8492e+00,  9.7046e+00,  4.5453e+01],
         [ 5.1564e+01, -5.0959e-01,  7.9268e+02],
         [-8.8787e+00,  4.0356e+00,  9.2392e+01],
         [-1.5752e+01,  5.4471e+00,  1.2836e+02],
         [-2.2155e+01,  6.4728e+00,  1.5484e+02],
         [-3.2799e+01,  7.9218e+00,  1.9084e+02],
         [-4.2054e+01,  9.0072e+00,  2.1744e+02],
         [-5.6337e+01,  1.0546e+01,  2.5325e+02],
         [-1.2602e+02,  1.7121e+01,  4.0065e+02],
         [-1.4565e+02,  1.9537e+01,  4.4131e+02],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
         [ 0.0000e+00,  0.0000e+00,  0.0000e+00]]], device='mps:0')
tensor([[1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], device='mps:0')
```

Supervised learning efficiency



Extra low efficiency:

20000 tracks recovered from 10^6

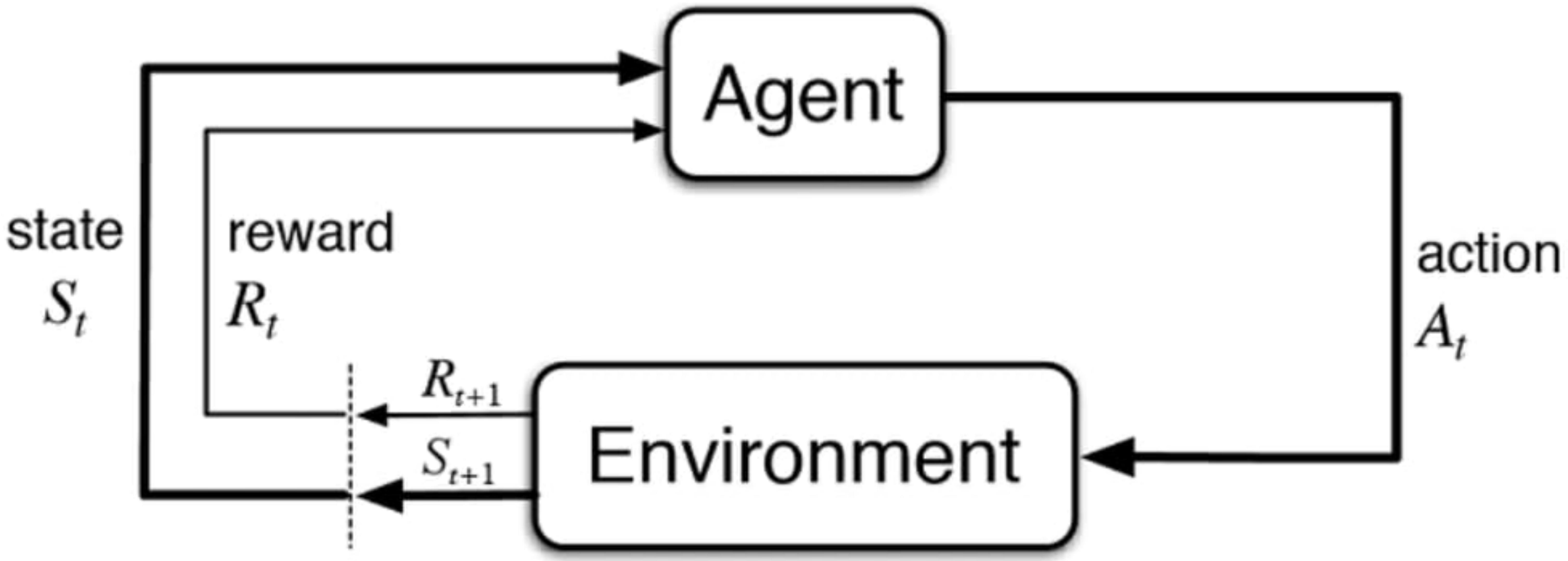
The model is missing information about the decision tree operation.

Solutions:

1) Upgrade dataset

2) Reinforcement learning

Reinforcement learning



RL in tracking task

```python
        self.active_tracks[track_id] = {
            'history': [hit],
            'hidden': None
        }

    return self._get_state()

def _get_state(self):
    return {
        'current_hits': self.levels[self.current_step],
        'next_hits': self.levels[self.current_step + 1]
                    if self.current_step + 1 < len(self.levels)
                    else torch.empty(0, 3, device=self.device),
        'track_states': self.active_tracks
    }
```

```python

if random.random() < epsilon:
    action_tensor = torch.randint(0, len(current_hits), (1,), device=device)
    action_scalar = action_tensor.item()

    probs = torch.ones(len(current_hits), device=device) / len(current_hits)
    dist = torch.distributions.Categorical(probs)
else:
    probs = torch.softmax(logits, dim=0)
    if torch.any(torch.isnan(probs)):
        probs = torch.ones_like(probs) / len(probs)
    dist = torch.distributions.Categorical(probs)
    action_tensor = dist.sample()
    action_scalar = action_tensor.item()
```

Rewards

```python
def calculate_reward(self, tracks):
        if not self.true_tracks:
            return torch.tensor(0.0, device=self.device)
        track_hits = torch.stack([torch.stack(t['history']) for t in tracks.values()])
        true_hits = torch.cat(self.true_tracks)

        dists = torch.cdist(track_hits, true_hits[:, :3])
        matches = (dists < 1e-4).any(dim=2).sum(dim=1)

        precision = matches / track_hits.size(1)
        recall = matches / true_hits.size(0)
        f1 = 2 * (precision * recall) / (precision + recall + 1e-8)
        return f1.mean()
```

Rewards

```python
def calculate_track_reward(self, tracks):
        if not self.true_tracks:
            return torch.tensor(0.0, device=self.device)

        tp, fp, fn = 0, 0, 0
        for true_track in self.true_tracks:
            best_overlap = 0
            for pred_track in tracks.values():
                pred_hits = torch.stack(pred_track['history'])
                true_hits = true_track[:, :3]
                dists = torch.cdist(pred_hits, true_hits)
                overlap = (dists < 1e-4).any(dim=1).sum().item()
                if overlap > best_overlap:
                    best_overlap = overlap

            if best_overlap / len(true_track) > 0.7:
                tp += 1
            else:
                fn += 1

        fp = len(tracks) - tp

        precision = tp / (tp + fp + 1e-8)
        recall = tp / (tp + fn + 1e-8)
        f1 = 2 * (precision * recall) / (precision + recall + 1e-8)
        return torch.tensor(f1, device=self.device)
```

In progress

- Optimization the implementation of reinforcement learning
- Preprocessing data for neural network. Finding pairs using algorithms
- Neural Kalman Filter
- Non-recurrent models