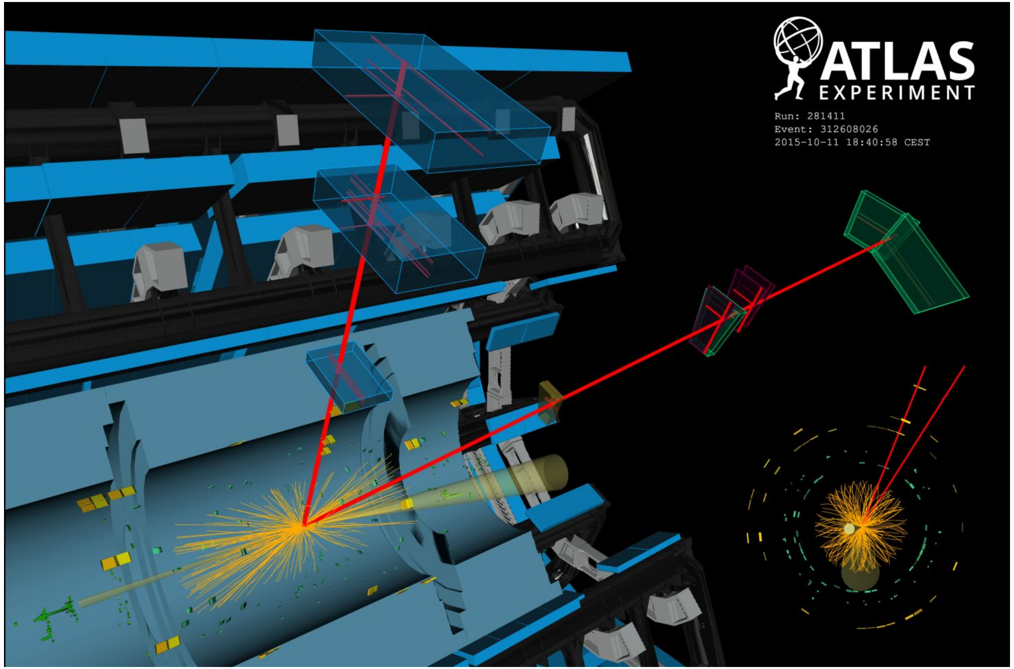
Ambiguity Resolution with Machine Learning and ACTS

Xiaocong Ai, Irina Ene, Heather Gray, Tomohiro Yamazaki
UC Berkeley

IRIS-HEP Topical Meeting
Sep 2, 2020

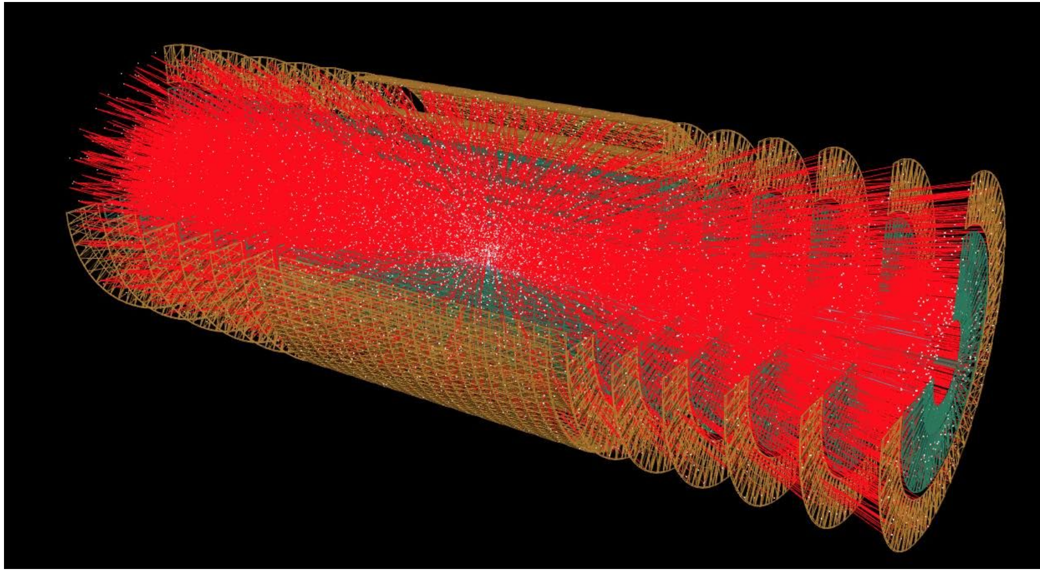
Outline

- Introduction / motivation
- Training the neural network
  - Training set and features
  - Network architecture
  - Validating network learning
- Implementation in ACTS
  - `MLTrackClassifier` class
  - Various options
- Discussion / future directions



Introduction

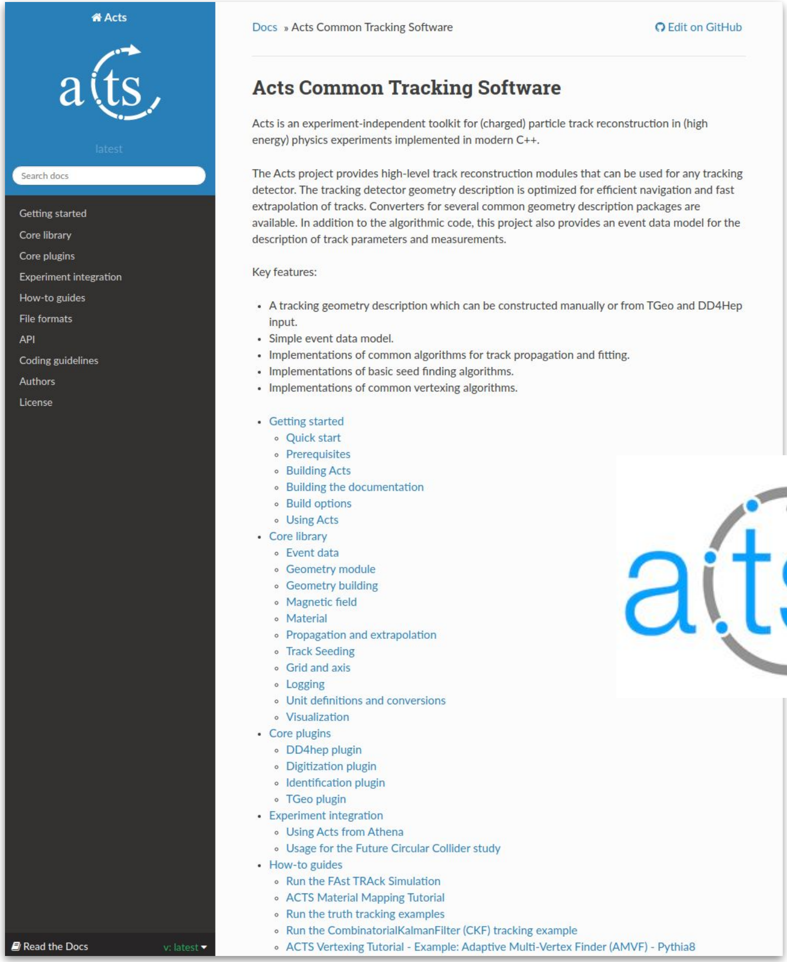
- LHC Run 1: ~280 tracks/event
- HL-LHC expect to see ~10k tracks/event
- This will stress computational resources
- Need fast, accurate, and efficient track reconstruction algorithms



TrackML Accuracy Phase Paper

ACTS

- Acts Common Tracking Software / A Common Tracking Software
- Experiment-independent toolkit for track reconstruction (for future detectors)
- Open-source platform for implementing new tracking techniques and hardware architectures



Acts

acts

latest

Search docs

Getting started
Core library
Core plugins
Experiment integration
How-to guides
File formats
API
Coding guidelines
Authors
License

Read the Docs v: latest

Docs » Acts Common Tracking Software Edit on GitHub

Acts Common Tracking Software

Acts is an experiment-independent toolkit for (charged) particle track reconstruction in (high energy) physics experiments implemented in modern C++.

The Acts project provides high-level track reconstruction modules that can be used for any tracking detector. The tracking detector geometry description is optimized for efficient navigation and fast extrapolation of tracks. Converters for several common geometry description packages are available. In addition to the algorithmic code, this project also provides an event data model for the description of track parameters and measurements.

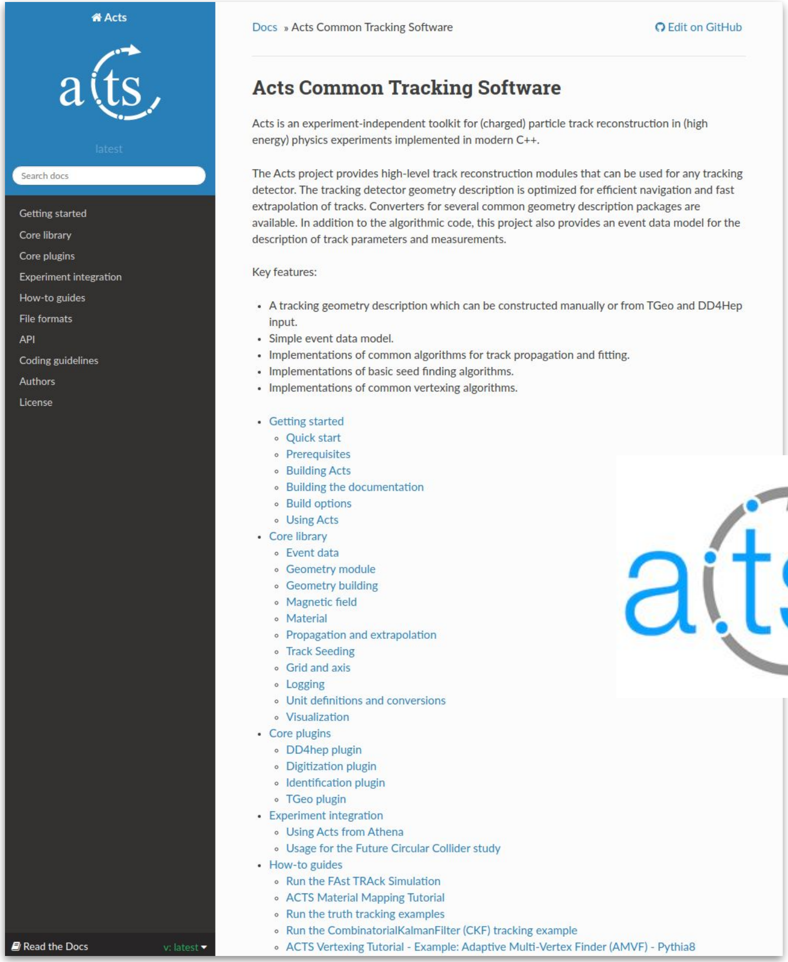
Key features:

- A tracking geometry description which can be constructed manually or from TGeo and DD4Hep input.
- Simple event data model.
- Implementations of common algorithms for track propagation and fitting.
- Implementations of basic seed finding algorithms.
- Implementations of common vertexing algorithms.

https://acts.readthedocs.io/en/latest/

ACTS

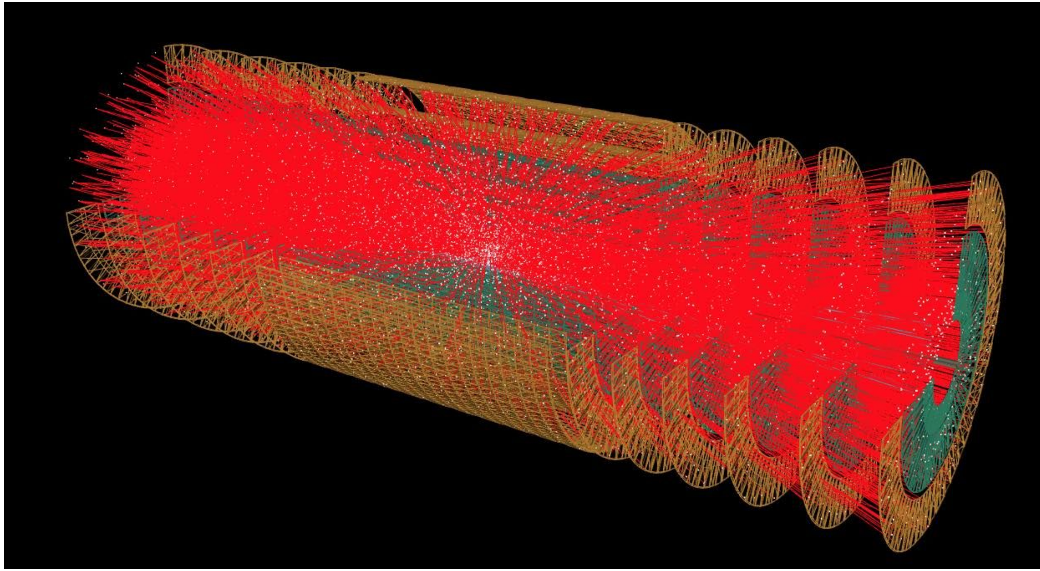
- Modern C++ 17
- Efficient memory allocation & access
- Strict thread-safety
- Rigorous unit tests
- Highly configurable
- Well-documented



https://acts.readthedocs.io/en/latest/

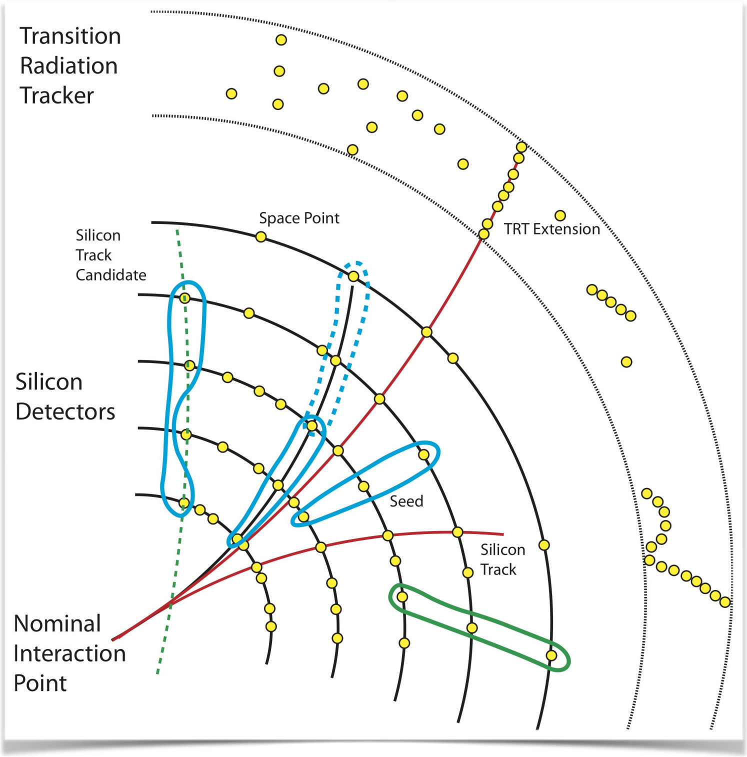
Ambiguity resolution

- LHC Run 1: ~280 tracks/event
- HL-LHC expect to see ~10k tracks/event
- Tracking performance is not 100% → not all reconstructed tracks are real tracks
- Need to ambiguity resolve to distinguish “good” tracks from “duplicate” and “fake” tracks



TrackML Accuracy Phase Paper

Track reconstruction in ATLAS



Space point formation

↓

Seed finding

↓

Track finding — → Combinatorial Kalman Filter
All realistic combinations of hits

↓

Ambiguity Solving — ← Track candidates with overlapping or incorrectly assigned hits

↓

TRT Extension

Motivation: Machine Learning

- For ATLAS, use simple scoring function based on track quality for ambiguity resolution
- In ACTS, currently have performance validation using truth-based information (i.e. truth-match probability for fakes, particle barcode for duplicates)

```cpp
// Check if the trajectory is matched with truth.
// If not, it will be classified as 'fake'
bool isFake = false;
if (nMajorityHits * 1. / trajState.nMeasurements >=
    m_cfg.truthMatchProbMin) {
  matched[majorityParticleId].push_back(
      {nMajorityHits, fittedParameters});
} else {
  isFake = true;
  unmatched[majorityParticleId]++;
}
// ...
// Sort the reco tracks matched to this particle by the number of majority
// hits
std::sort(matchedTracks.begin(), matchedTracks.end(),
          [](const RecoTrackInfo& lhs, const RecoTrackInfo& rhs) {
            return lhs.first > rhs.first;
          });
for (size_t itrack = 0; itrack < matchedTracks.size(); itrack++) {
  const auto& [nMajorityHits, fittedParameters] = matchedTracks.at(itrack);
  // The tracks with maximum number of majority hits is taken as the 'real'
  // track; others are as 'duplicated'
  bool isDuplicated = (itrack != 0);
}
```

Motivation: Machine Learning

- For ATLAS, use simple scoring function based on track quality for ambiguity resolution
- In ACTS, currently have performance validation using truth-based information (i.e. truth-match probability for fakes, particle barcode for duplicates)
- A possibility for ambiguity resolution is to use ML (e.g. neural network) to predict whether a given reconstructed track is good/duplicate/fake based on certain track features
    - Caveat: track information currently available is rather simple

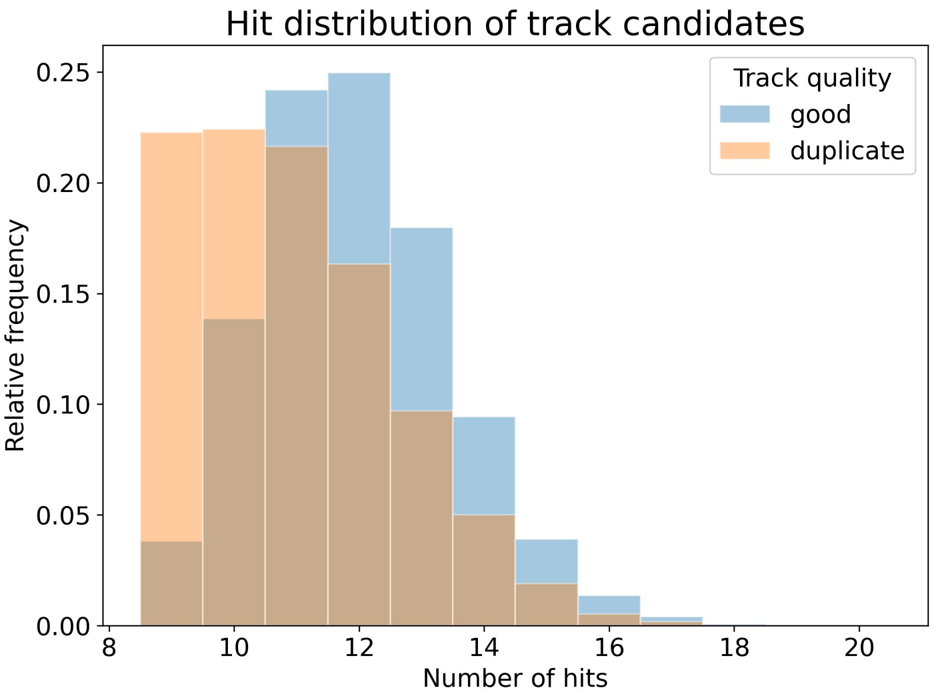
```
// Check if the trajectory is matched with truth.
// If not, it will be classified as 'fake'
bool isFake = false;
if (nMajorityHits * 1. / trajState.nMeasurements >=
    m_cfg.truthMatchProbMin) {
  matched[majorityParticleId].push_back(
      {nMajorityHits, fittedParameters});
} else {
  isFake = true;
  unmatched[majorityParticleId]++;
}
// ...
// Sort the reco tracks matched to this particle by the number of majority
// hits
std::sort(matchedTracks.begin(), matchedTracks.end(),
          [](const RecoTrackInfo& lhs, const RecoTrackInfo& rhs) {
            return lhs.first > rhs.first;
          });
for (size_t itrack = 0; itrack < matchedTracks.size(); itrack++) {
  const auto& [nMajorityHits, fittedParameters] = matchedTracks.at(itrack);
  // The tracks with maximum number of majority hits is taken as the 'real'
  // track; others are as 'duplicated'
  bool isDuplicated = (itrack != 0);
}
```

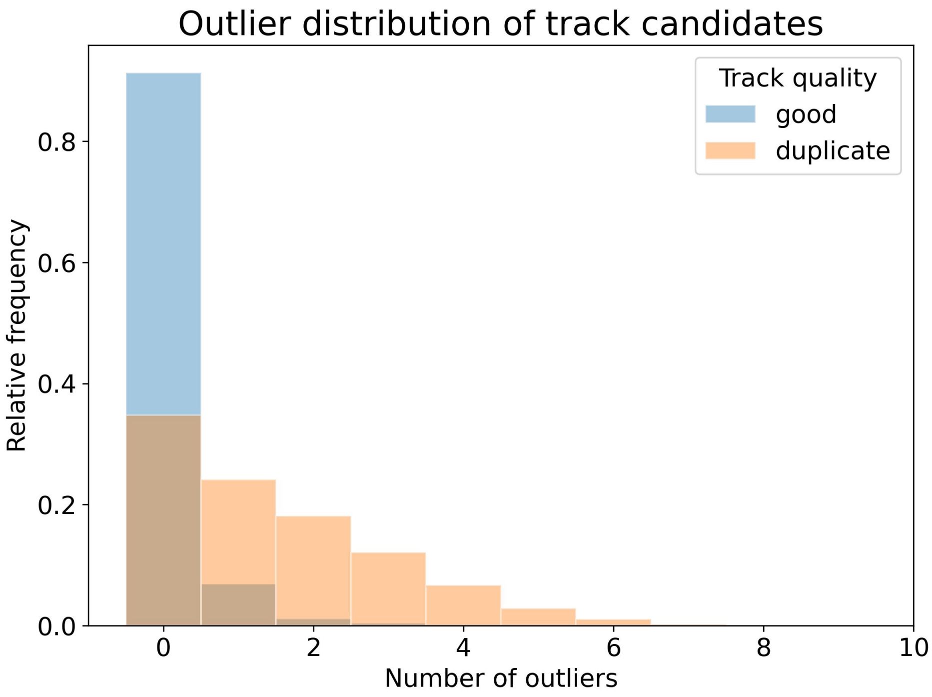
Generating the training data set

- Use ACTS FAst TRAck Simulation (FATRAS) to generate simulated particles/hits
  - 50 ttbar events at μ = 200
  - TrackML detector
  - 2T constant magnetic field (along z)
- Use Combinatorial Kalman Filter track finding and fitting: ~115k reconstructed tracks
  - ~85.96% **good** (truth-match prob >= 50%)
  - ~14.04% **duplicate** (same majority truth particle as good track, but less majority particle hits)
  - <0.001% **fake** (truth-match prob < 50%)
  - binary classifier (good/duplicate)
- Readily available features used for training
  - Number of hits
  - Number of outliers
  - Number of holes (didn't end up using, more details in a bit)
  - χ^2/dof
- With a more realistic detector description, can be extended to use additional information
  - e.g. ITk geometry was first integrated in ACTS just last week

Training features

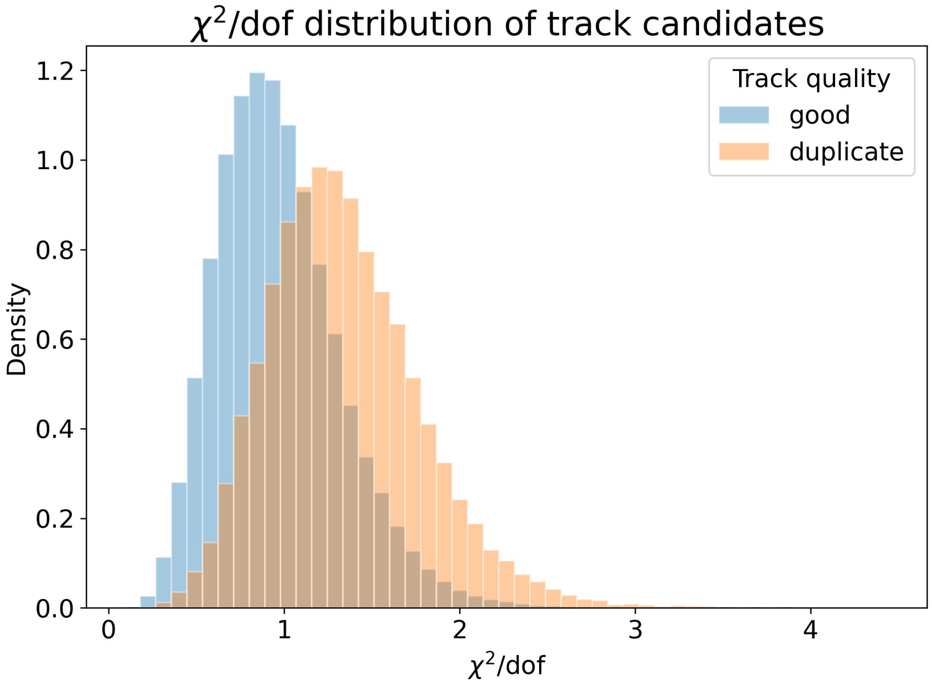
Separation, but also overlap!

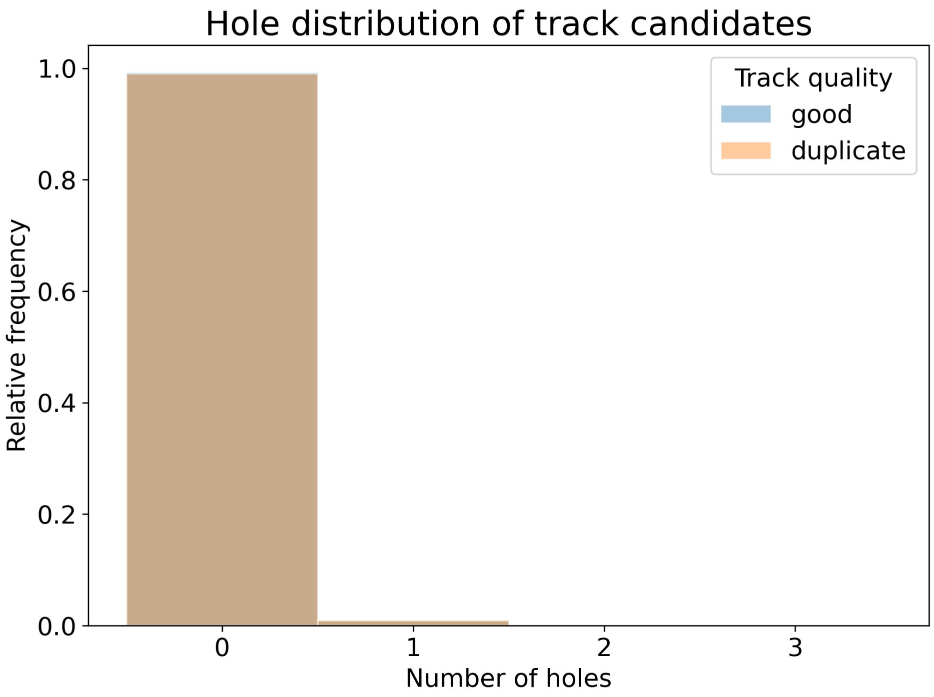




Training features

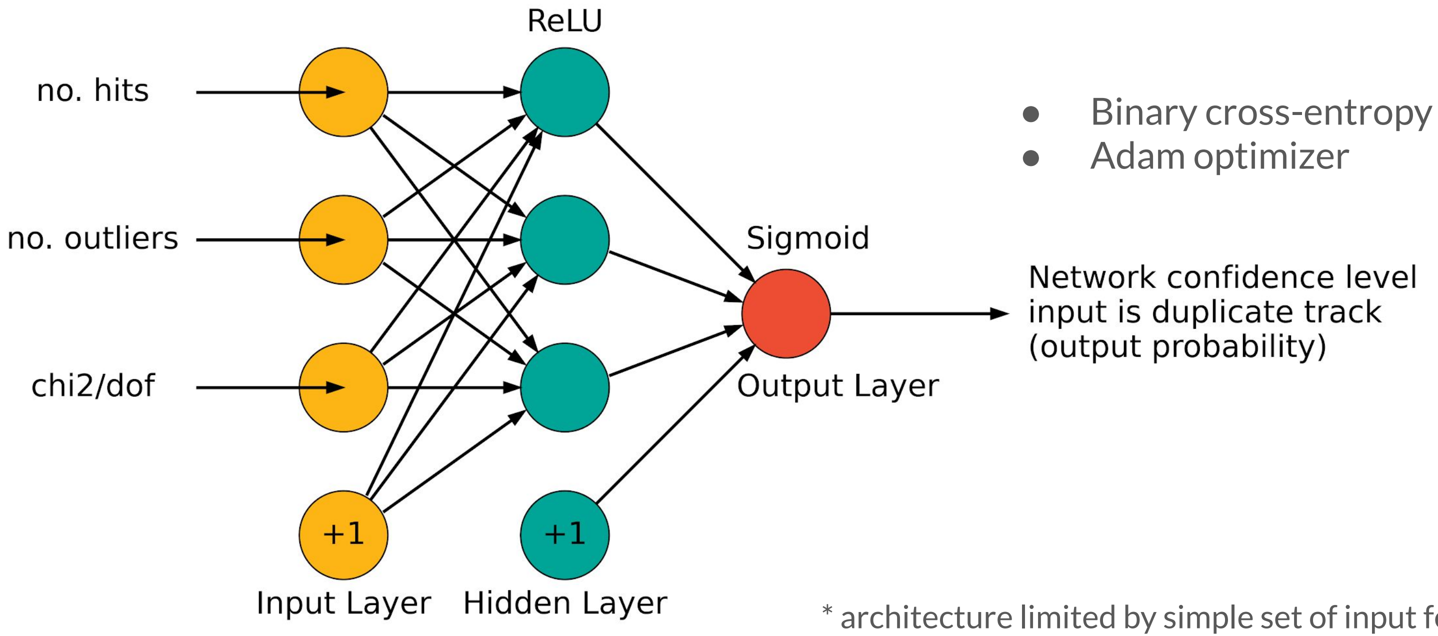
No inefficiencies in the modules → holes info not informative, don't use





Constructing the NN

TensorFlow → "Deep" Neural Network*



- Binary cross-entropy loss
- Adam optimizer

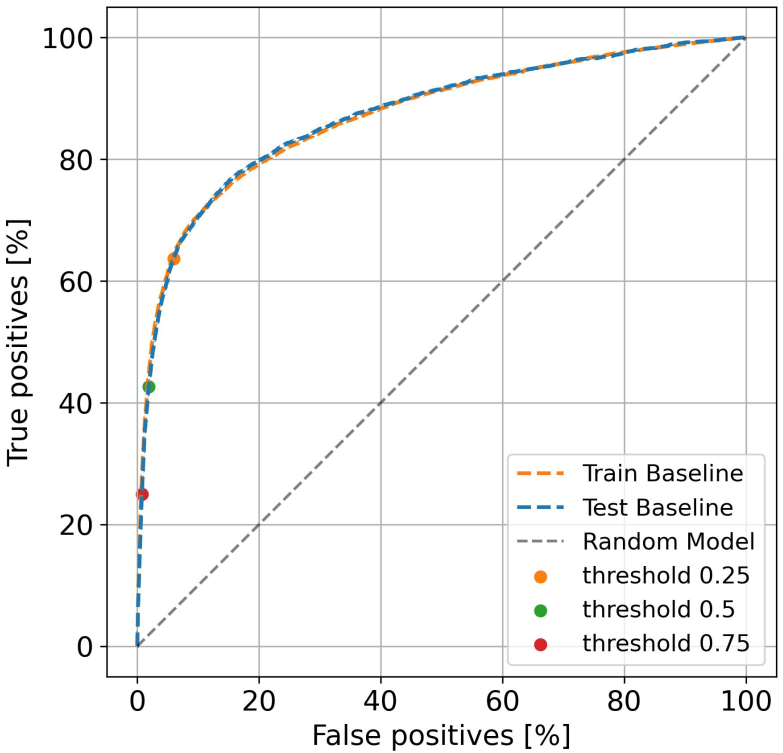
* architecture limited by simple set of input features

Training the NN

- 75 % train / 25% test split
- Train NN → maximize Area Under the Curve of the Receiver Operating Characteristic on test set
- Use trained model to make predictions
- Output probability > prediction threshold (=50%) → predicted duplicate track

Duplicate tracks predicted to be duplicate

AUC: ~0.87

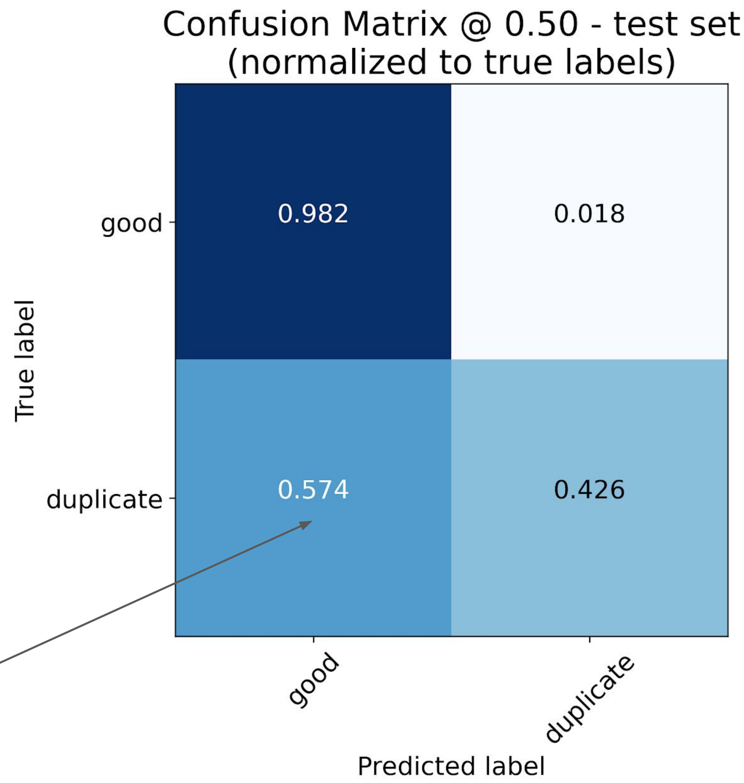


Good tracks predicted to be duplicate

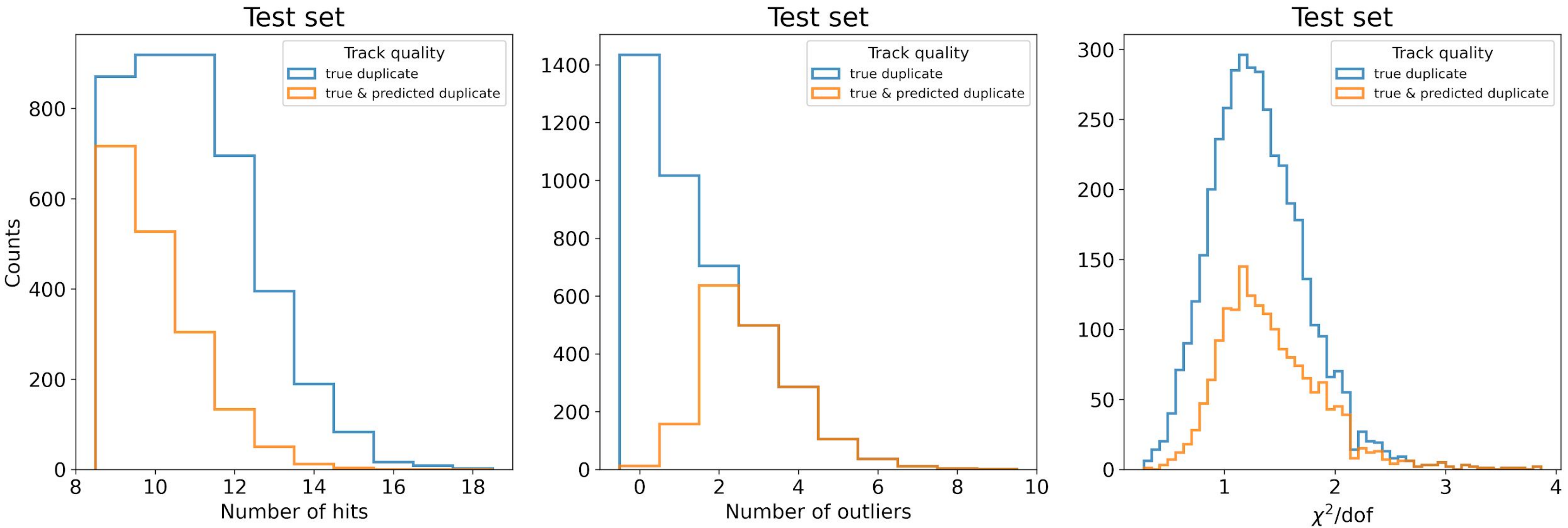
Training the NN

- 75 % train / 25% test split
- Train NN → maximize Area Under the Curve of the Receiver Operating Characteristic on test set
- Use trained model to make predictions
- Output probability > prediction threshold (=50%) → predicted duplicate track
- NN is doing great on good tracks, not so great on duplicate tracks
    - Good enough considering limitations of simple input features

Need extra info such as e.g. shared hits to resolve the duplicates!



Checking NN learning



- NN has learned that duplicate tracks have fewer hits, lots of outliers, and large χ^2/dof (easy case)
- Need extra info to resolve the duplicates that "look" like good tracks!
- Currently working on track-by-track basis, but eventually would like to find an architecture that can consider multiple tracks at a time!

C++

Implementing neural network into ACTS

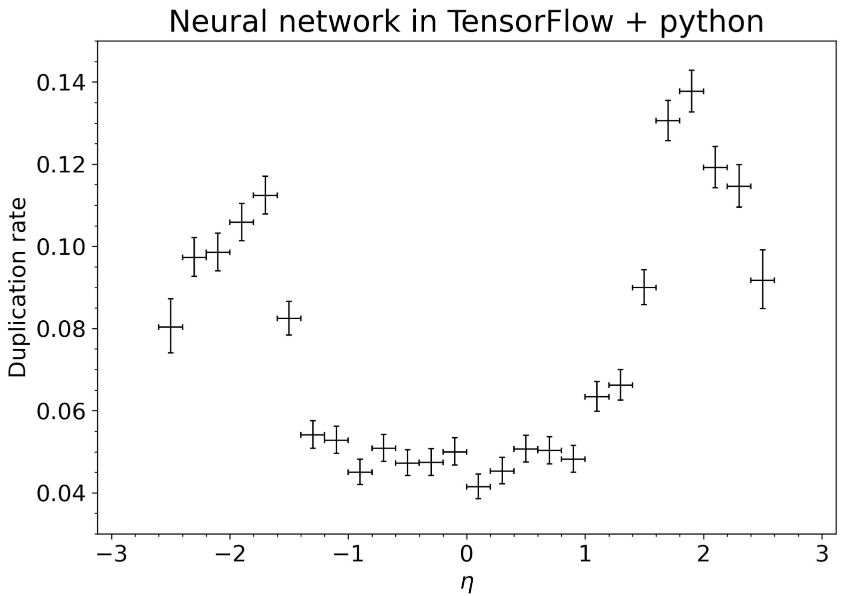
`MLTrackClassifier` class (v1)

- `m_weightsPerLayer`
  - Vector that stores trained weights matrices for each layer
  - weights → dynamic-size `Eigen::Matrix`
- `predictTrackLabel` function
  - For each layer:
    - Take input
    - Add bias term
    - Apply the weights
    - Apply activation function
  - Predict `good` or `duplicate` based on decision threshold probability
- Can use for duplication rate plots

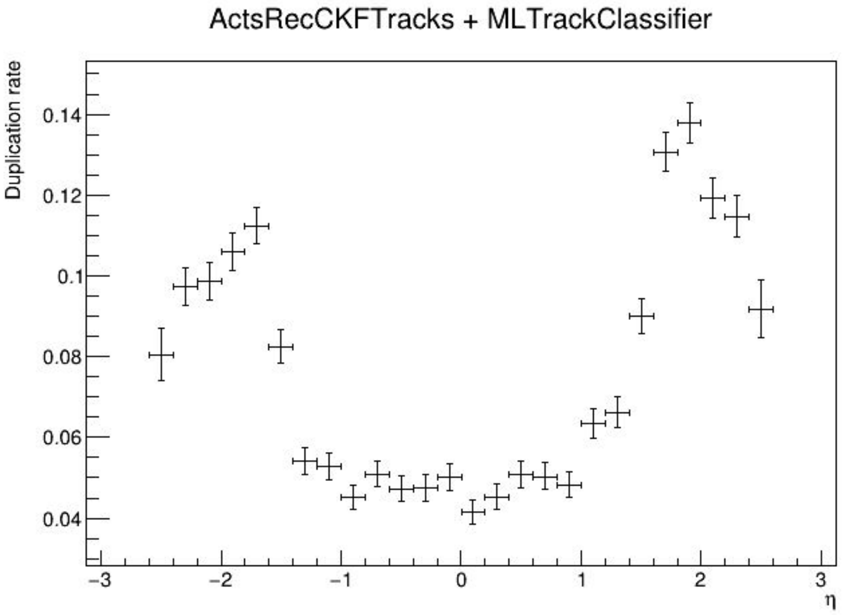
```cpp
FW::MLTrackClassifier::TrackLabels FW::MLTrackClassifier::predictTrackLabel(
    const Acts::MultiTrajectory<SimSourceLink>& multiTraj,
    const size_t& entryIndex, const double& decisionThreshProb) const {
  // ...
  // get the trajectory summary info
  auto trajState =
      Acts::MultiTrajectoryHelpers::trajectoryState(multiTraj, entryIndex);
  // the vector of input features
  Acts::ActsVectorXd inputFeatures(3);
  inputFeatures[0] = trajState.nMeasurements;
  inputFeatures[1] = trajState.nOutliers;
  inputFeatures[2] = trajState.chi2Sum * 1.0 / trajState.NDF;
  // linear algebra for layer 1 (hidden layer)
  Acts::ActsVectorXd wInputLayer1 =
      weightedInput(m_weightsPerLayer[0], inputFeatures);
  Acts::ActsVectorXd outputLayer1 = reluActivation(wInputLayer1);
  // linear algebra for layer 2 (output layer)
  Acts::ActsVectorXd wInputLayer2 =
      weightedInput(m_weightsPerLayer[1], outputLayer1);
  Acts::ActsVectorXd outputLayer2 = sigmoidActivation(wInputLayer2);
  // output layer prediction
  if (outputLayer2[0] > decisionThreshProb) {
    return TrackLabels::duplicate;
  }
  return TrackLabels::good;
}
```

Validating implementation

Duplication rate vs eta using predictions of NN directly in TensorFlow



Duplication rate vs eta using predictions of MLTrackClassifier instance



Thinking ahead

- Currently, `MLTrackClassifier` is customized to:
  - One architecture for neural network
  - Trained on data from one detector + magnetic field configuration
  - Some other aspects are hard-coded
- Would like the implementation to be more general
  - Multiple detector + magnetic field configurations
  - Use ML inference framework for other applications (e.g. seed finding)

Thinking ahead

- Currently, `MLTrackClassifier` is customized to:
  - One architecture for neural network
  - Trained on data from one detector + magnetic field configuration
  - Some other aspects are hard-coded
- Would like the implementation to be more general
  - Multiple detector + magnetic field configurations
  - Use ML inference framework for other applications (e.g. seed finding)
- Open Neural Network Exchange (ONNX) format and runtime
  - This has already been integrated into Athena for more general ML tasks in ATLAS
  - Exploring implementation as plugin for ACTS

https://onnx.ai/index.html

https://github.com/microsoft/onnxruntime

ONNX integration

- ONNX format
- Open-source format for DNN/ML models
- Supports many ML frameworks
    - Keras/TensorFlow
    - PyTorch
    - Scikit-learn
    - Matlab
    - LibSVM
    - MyCaffe
    - XGBoost
    - Etc.
- Save model architecture, trained weights, compiler info

- ONNX runtime
- Cross-platform inferencing accelerator (training feature in preview)
- Has APIs for
    - Python
    - C#
    - C/C++
    - Java
    - Node.js
- Abstracts away the complex linear algebra behind the ML model prediction function

C++

ONNX integration

`MLTrackClassifier` class (v2 - WIP)

- "Wrapper" around ONNX runtime
- `predictTrackLabel` function
  - Predict `good` or `duplicate` based on decision threshold probability
- NN prediction function
  - `Ort::Session`
    - Loads model
  - Inference in the usual way
    `output = model.predict(input)`

```cpp
FW::MLTrackClassifier::TrackLabel FW::MLTrackClassifier::predictTrackLabel(
    std::vector<float>& input_tensor_values,
    const double& decisionThreshProb) const {
  // run onnx inference
  float outputProbability = runONNXInference(input_tensor_values);
  if (outputProbability > decisionThreshProb) {
    return TrackLabel::duplicate;
  }
  return TrackLabel::good;
}

// onnxruntime inference function
float FW::MLTrackClassifier::runONNXInference(
    std::vector<float>& input_tensor_values) const {
  // ...
  // score model & input tensor, get output tensor
  std::vector<Ort::Value> output_tensors = m_session->Run(
      Ort::RunOptions{nullptr},
      m_inputNodeNames.data(), &input_tensor, m_inputNodeNames.size(),
      m_outputNodeNames.data(), m_outputNodeNames.size());

  // get pointer to output tensor float values
  float* floatarr = output_tensors.front().GetTensorMutableData<float>();
  // binary decision classification, only need first value
  return floatarr[0];
}
```

Stores representation of model

Future work

- ONNX format+runtime provide a general ML inference framework that can be integrated within ACTS
- But we have future plans to:
    - Improve network performance on duplicate tracks
        - Class imbalance problem
        - Tweaking network architecture
        - Access to additional features such as shared hits information is needed
    - Train network on more realistic detector description (i.e. decent fraction of fake tracks)

Thank you for your attention!