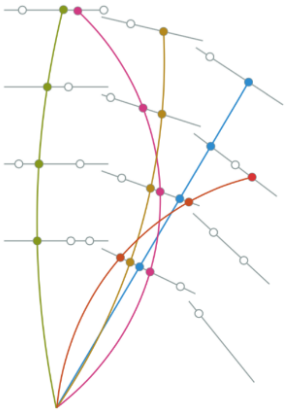
Machine Learning for Track Reconstruction at the LHC

Louis-Guillaume Gagnon

AI4EIC-Exp Workshop
2021/09/08

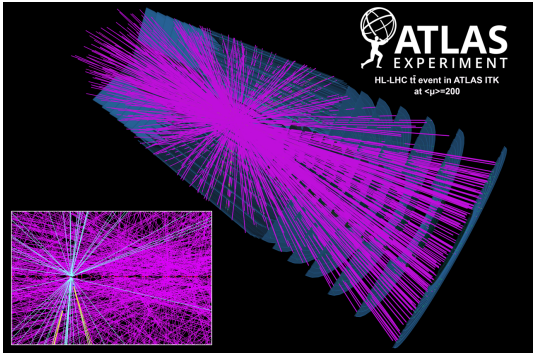
Introduction: Particle Trajectory Reconstruction

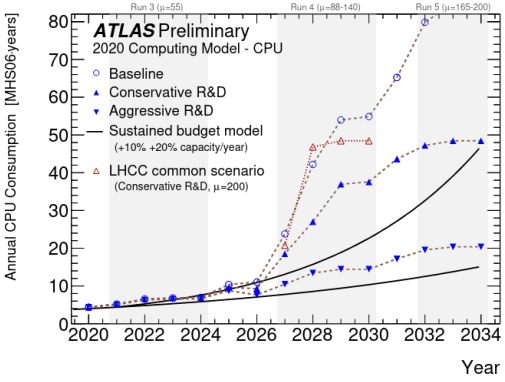
- Particle trajectory reconstruction (Tracking) is a clustering problem
- Given a set of points in 3D space (Hits), cluster in sets which originate from the same particles
  - Hits correspond to signals from subdetetectors
  - Today, focus on silicon tracking detectors at p–p colliders
- The small number of hits in a typical reconstructed track compared to the total number of hits in an event makes the problem particularly challenging
- Most experiments in LHC setting use variations of the Kalman Filter (KF) algorithm to find and fit tracks
- ☺Physics performance is excellent
- ☹Runtime scales badly with N_{hits}



[1904.06778]

Introduction: Towards the High-Luminosity LHC (HL-LHC)





- HL-LHC: Circa 2027, pileup increase to ≈ 200
  - ATLAS Run-2 had ≈ 30
- KF-based methods runtime is $\gtrsim \mathcal{O}(N_{\text{hits}}^2)$
- Combinatorial explosion $\rightarrow$ runtime explosion
- Also: trigger rate increase, more readout channels ...

- Current budget prediction do not cover CPU needs of current methods
- Need "Aggressive R&D": New and/or improved methods
- More details in ATLAS computing CDR

Introduction: Roadmap

- Where does Machine Learning (ML) fits into this problem?
- Kalman Filter is currently still by far the best method available

- Approach 1: Make the KF methods faster
- Approach 2: Keep the current KF-based methods and use ML to control the combinatorics
- Approach 3: Replace the KF altogether with more sophisticated ML
  (so-called "end-to-end" methods)

Approach 1: Accelerated Kalman Filter

Trigger-level track finding on GPU with ALICE

- Track finding in the ALICE TPC at software trigger level has been ported to GPUs
  - Tracklet finding with cellular automaton
  - Track finding & fitting with simplified Kalman Filter
- One of the earliest successful use of GPU for tracking in a realistic setting!

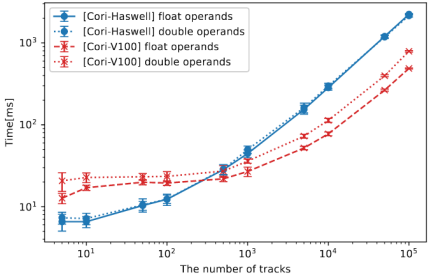
Processing step	AMD Radeon 7	RTX 2080 Ti	Intel CPU	2080 Ti / CPU
Zero Suppression Decoding	38 ms	19 ms	986 ms	52x
Cluster Finding	87 ms	79 ms	21343 ms	270x
Track Finding	109 ms	65 ms	8759 ms	135x
Track Fit	284 ms	243 ms	7204 ms	30x
Cluster Compression	137 ms	105 ms	1452 ms	14x
Synchronous Processing Total	657 ms	511 ms	39816 ms	78x
$\mathrm{d}E/\mathrm{d}x$ Calculation	61 ms	22 ms	906 ms	41x
Asynchronous Processing Total	304 ms	237 ms	13381 ns	56x

Table 1: Processing time of reconstruction steps on GPU and CPU. The CPU was measured on an Intel CPU clocked at 4.5 GHz with the Skylake architecture (clock fixed, turbo disabled) on a single core.

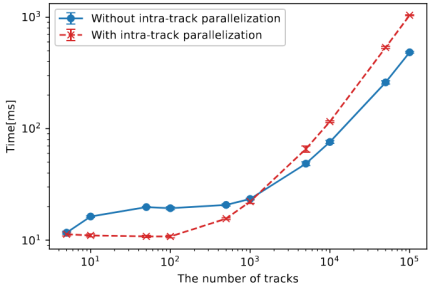
- Total speedup for GPU is 56x!
- Connecting the dots 2020 paper

Kalman Filtering on the GPU

- The KF algorithm is fully expressible with linear algebra
- However, peculiarities of the algorithm makes it a challenge to efficiently code it on a GPU (e.g. with CUDA)
  - E.g. Matrix sizes are typically very small
- Ai et al, [2105.01796], explored two strategies
  - Fitting many tracks in parallel (using CUDA threads or blocks)
  - Additionally parallelizing single-track fits (Using CUDA threads)



- Scaling remains the same, but GPU implementation is faster for large datasets



- Intra-track parallelization is mainly good for smaller datasets

Approach 2: Strategic Use of Machine Learning

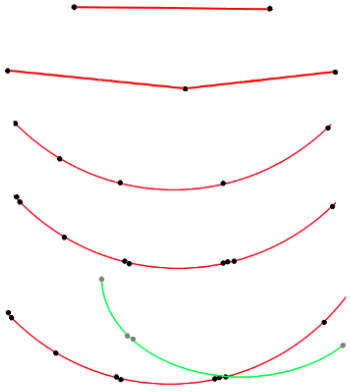
The TrackML challenge

- TrackML: Machine Learning competition aiming to encourage development of fast and high performance tracking methods by leveraging ML expertise
- Split into two different stages:
  - Accuracy phase: Hosted on Kaggle, aimed to attain maximum physics performance
  - Throughput phase: Hosted on Codalab, aimed to attain maximum inference speed
- Main insight: Most winning methods leverage "classical" track reconstruction techniques, using ML at strategic points to control combinatorics and increase performance

- Accuracy phase:
  - Kaggle webpage
  - Summary paper: [1904.06778]
- Throughput phase:
  - Codalab webpage
  - Summary paper: [2105.01160]

TrackML Accuracy Phase Winner: "Top-Quarks"

- Modular algorithm, similar to "typical" combinatorial KF pipelines (e.g. ATLAS):
  1. Seeding: create pairs of seed with a pairwise logistic regression model, using 50 different layer pairings as input
  2. Triplet formation: Extend the resulting doublets to triplets and filter them with another logistic regression model
  3. Track following: Build tracks by helical extrapolation
  4. Ambiguity resolution: Simple cut on number of wrongly associated hits (estimated from a fit to data)
- Logistic regression is leveraged to control the combinatorics
- Dedicated data structures were developed to allow an efficient implementation



- [1904.06778]
- Code Repository: github.com/top-quarks/top-quarks

TrackML Throughput Phase Winner: "Mikado"

- Implemented by Sergey Gorbunov (Runner-up in accuracy phase)
- Iterative track finding algorithm with 60 passes
  - Earlier passes are very strict → high purity, low efficiency
  - Later passes are progressively looser → combinatorics kept under control by earlier passes
- Hits on every layer are quantized to a 2D grid
- Tracklets are built by looking through layer- and pass-specific search windows
- $\mathcal{O}(10^4)$ parameters used to define fixed-size search windows
- Parameters tuned with a fit to data

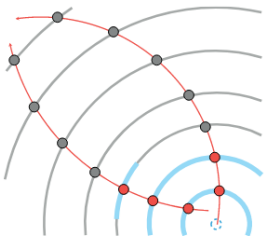


Fig. 7 Combinatorial Layers

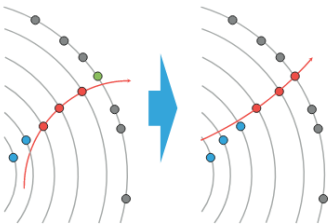
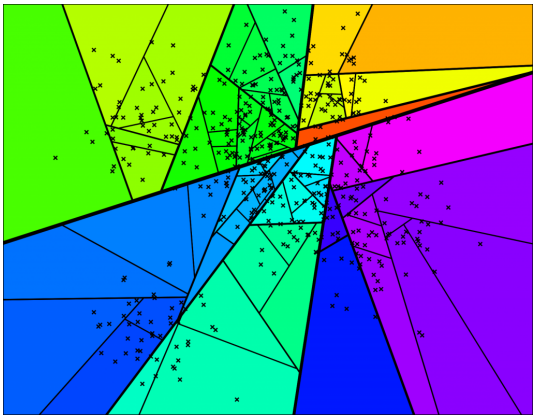


Fig. 8 Tracklet prolongation

[2105.01160]

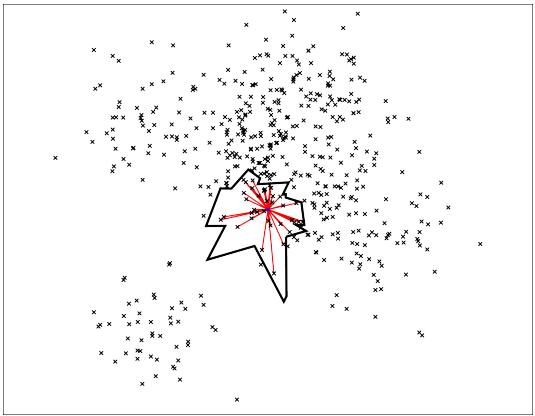
Approximate Nearest Neighbor Search

- Define a metric space (e.g. angular distance)
- Segment it in different regions, in $\mathcal{O}(\log N_{\text{hits}})$



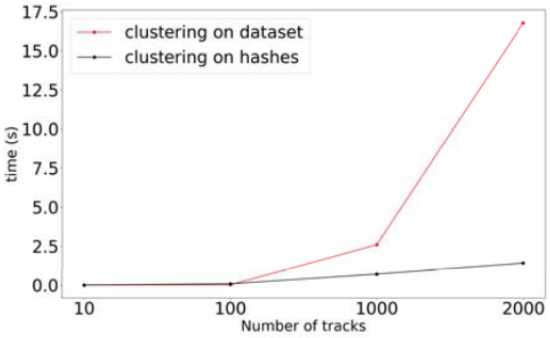
source

- Quickly lookup union of regions being approximately closest to a query point



Approximate Nearest Neighbor Search

- Different approach to track reconstruction, by Amrouche et al
- Divide-and-conquer approach used to control combinatorics
- Use "Approximate nearest neighbor" algorithms separate all hits in different regions
- Tracking can then be performed separately in each region
- $\mathcal{O}(N_{\text{hits}}^2)$ nature of tracking means that this scales better than the global approach
- Need to define a meaningful distance between hits
  - Can be as simple as the angular distance between hits...
  - ...or as complex as a learned embedding via neural network

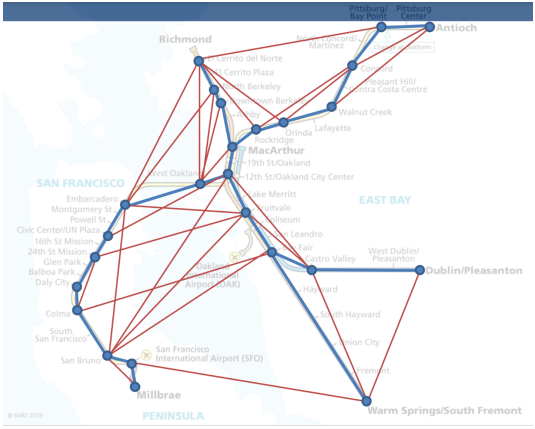


- [2101.06428]
- IEEE article
- Blog post by Sabrina Amrouche

Approach 3: End-to-end Machine Learning Algorithms

Graph Neural Networks

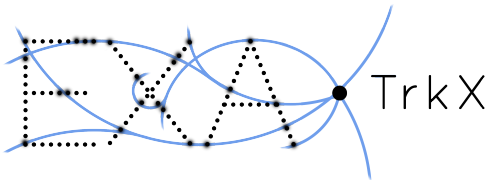
- Graph Neural Networks (GNNs):
  - Cast dataset as a set of nodes connect by edges
  - Both nodes and edges can have associated values and labels
  - GNN will do node and/or edge classification
- In context of track reconstruction:
  - Nodes are naturally hits in a detector layer
  - Edges connect hits together. "True" edge → hits from a same particle
- The edge classification task is leveraged to cluster hits into sets that are truly connected



- Example: Edge classification of San Francisco Bay Area transit network stations
- source

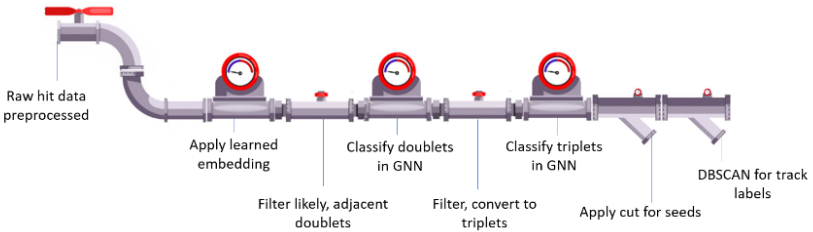
Graph Neural Networks: the Exa.TrkX Pipeline

- Exa.TrkX is a DOE project aims to leverage GNNs to form a complete track reconstruction pipeline at the exascale
- Multi-stage algorithm that can be used for seed formation or complete track reconstruction
- Core GNN model is "Interaction Network" [1612.00222]
- Project also aims to support parallelized deployment on accelerated hardware such as GPUs and FPGAs



- Project webpage: exatrkx.github.io
- Performance paper: [2103.06995]
- Code repository: github.com/exatrkx

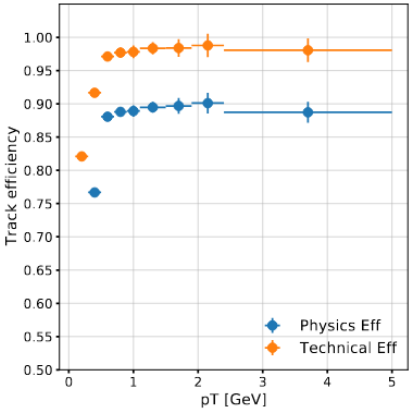
Graph Neural Networks: the Exa.TrkX Pipeline



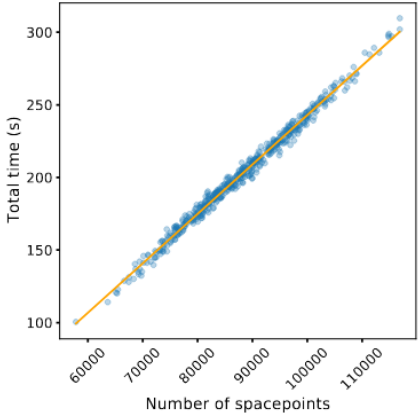
[2103.06995]

1. Construct a metric space in which distance between hits correlates with track membership $\rightarrow$ defines edges
2. Train an neural network to filter these edges to increase purity and make graph sparser
3. Core step: Use a GNN (interaction network) to classify edges
4. Post-processing, depending on the goal:
   - Seeding: Using the resulting doublet graph, build triplets using likelihood methodx
   - Track reconstruction: Partition the graph into connected components

Graph Neural Networks: the Exa.TrkX Pipeline



- Performance adequate for proof-of-concept



- Good runtime scaling, albeit on a simplified dataset (TrackML)

[2103.06995]

ACTS: A Common Tracking Software project

- Experiment-independent toolkit for tracking
- Free software (Mozilla Public License v2.0)
- Considered for use by Belle II, CEPC, sPHENIX, PANDA, FASER, ATLAS, EIC, ...
- Three overarching goals:
  1. Preserve current tracking approaches while enabling development for HL-LHC
  2. Serve as an algorithmic test bed incl. ML-based algorithms and accelerated hardware
  3. Enable rapid and realistic development of new tracking detectors
- Includes an ONNX plugin, to enable import of various ML models anywhere in the tracking workflow
- Ongoing R&D for GPU tracking (traccc)

- Overview paper: [2106.13593]
- Project webpage: acts.readthedocs.io
- Code repository: github.com/acts-project/acts

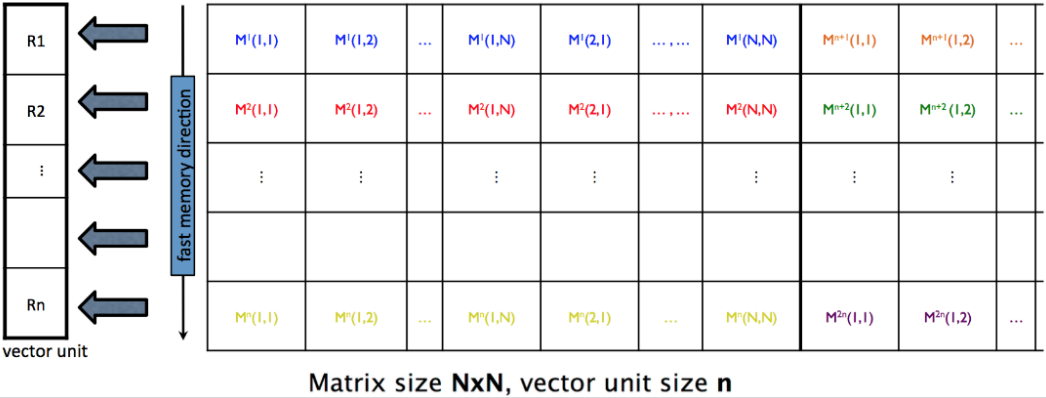
Conclusion

- Today: Only presented a small fraction of the total landscape of ML use in track reconstruction
  - Using accelerated hardware to speed-up current KF-based algorithms
  - Using ML at strategic points in track reconstruction algorithms
  - Using ML at all stages to build an end-to-end pipeline
- My personal takeaways:
  - The Kalman Filter remains too powerful to completely throw away
  - ML techiques are extremely useful to avoid the dreaded combinatorial explosion
  - Many techniques, from logistic regression to graph neural networks, are promising
  - An optimal tracking pipeline should be very modular in design
  - Accelerated implementations could help bridge the remaining performance gap
- Eager to learn about more cutting edge methods?
  - The accuracy and throughput trackML papers are highly recommended
  - The Living Review of Machine Learning for Particle Physics contains a "Tracking" section

BACKUP SLIDES

Parallelized track finding on CPU with CMS: Matriplex

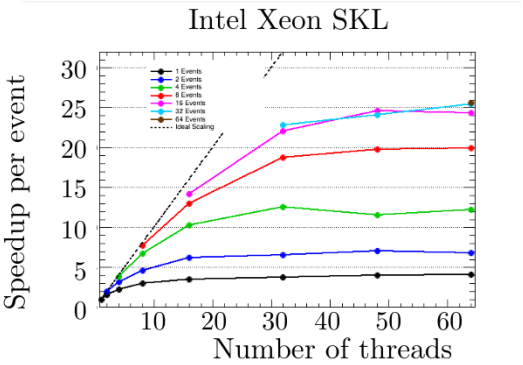
- Most recent CPUs have broad support for vectorized math operations
- Matriplex: Memory layout for efficient use of vector units to perform small matrix operations

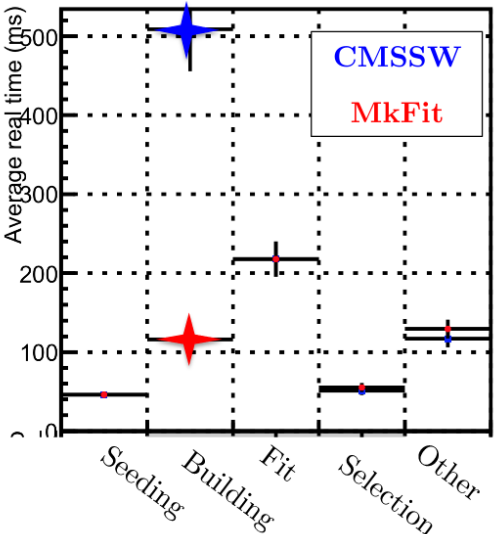


- From Connecting the dots 2019 talk

Parallelized track finding on CPU with CMS: mkfit

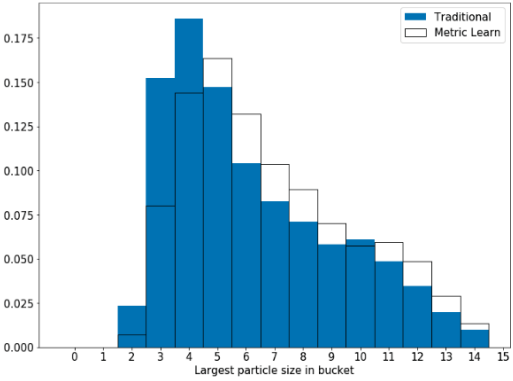
- Matriplex + threading leveraged in the CMS mkfit framework for parallel track reconstruction
- github.com/trackreco/mkFit





- Good speedup seen for single events
- Up to 25x with event batching

Approximate Nearest Neighbor Search



- Using dedicated learned metric space increases the bucket quality

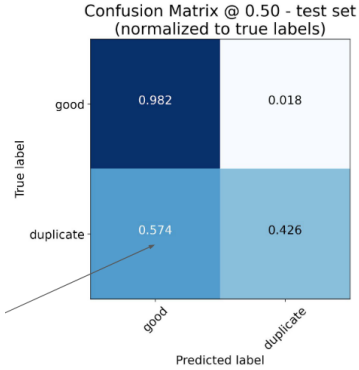
plots from IEEE article

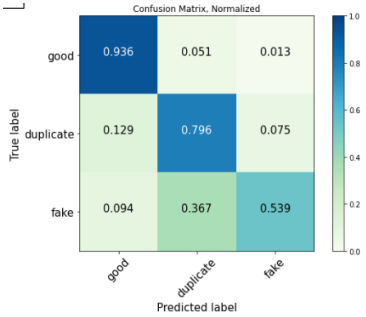
Algorithm 1 Track finding with hashing

Input : List of hits H
Output: List of tracks T
Require: $T \leftarrow findTracks(H)$
Require: $P \leftarrow metricLearn(H)$
Require: $ANN \leftarrow buildIndex(H, metric)$
  $ANN \leftarrow buildIndex(P, metric)$
  $foundTracks \leftarrow \emptyset$
  while $i \leq Nqueries$ **do**
    $n \leftarrow random()$
    $bucket \leftarrow ANN.query(n)$
    $t \leftarrow findTracks(bucket)$
    $foundTracks \xleftarrow{+} t$
    $i \leftarrow i + 1$
  end while
  return $foundTracks$

Example of ML R&D in ACTS: Ambiguity resolution

- Ambiguity resolution: Given a set of tracks, reject duplicate and fake tracks and keep only tracks corresponding to a real particle
- Using the ACTS framework, a dataset comprising real and fake tracks is easily obtained
- Regular NN and/or Recurrent NN can be integrated in ACTS with ONNX





- Earlier work by Irina Ene, grad student at UCB
- slides

- Extension to LSTM, using information about hits shared by many tracks
- Work by Nupur Oza, IRIS-HEP fellow