An updated hybrid deep learning algorithm for identifying and locating primary vertices

Simon Akar[1] Thomas Boettcher[2] Sarah Carl[1] Henry Schreiner[3] Mike Sokoloff[1]
Marian Stahl[1] Constantin Weisser[2] Mike Williams[2]
on behalf of the LHCb Real Time Analysis project

[1] University of Cincinnati [2] Massachusetts Institute of Technology [3] Princeton University

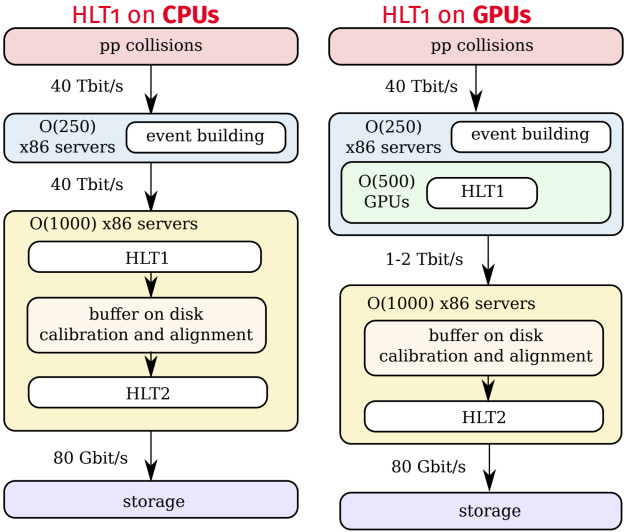
April, 20th 2020

Supported by:

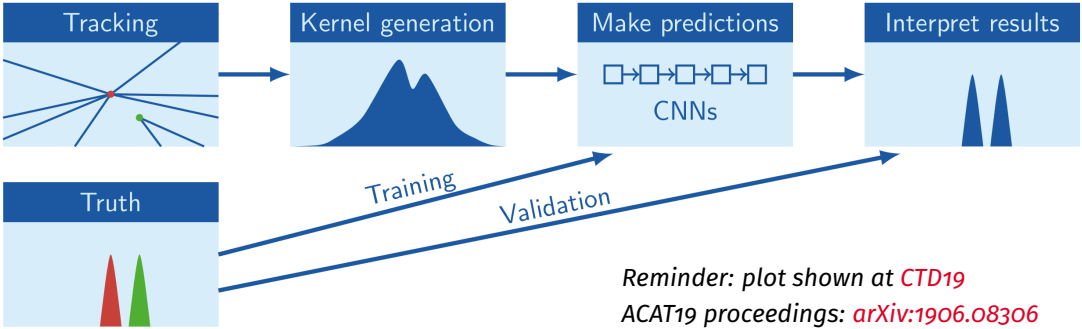
- All tracking detectors are being replaced, PID detectors upgraded; hardware triggers will be removed and luminosity increased ⇝ there will be a **new experiment** at Point 8 after LS2!

- Build on success of offline-quality reconstruction, alignment and calibration in Run 2, LHCb is moving to a **real-time analysis** approach
- Running a software trigger at 30 MHz with limited resources poses **major challenges** ⇝ **rethink** data structure, reconstruction and selection **from the ground up**

  dedicated LHCb HLT talks at CTD: [CPU] [GPU]
- **Machine learning** algorithms have potential to improve fidelity and run at high rate



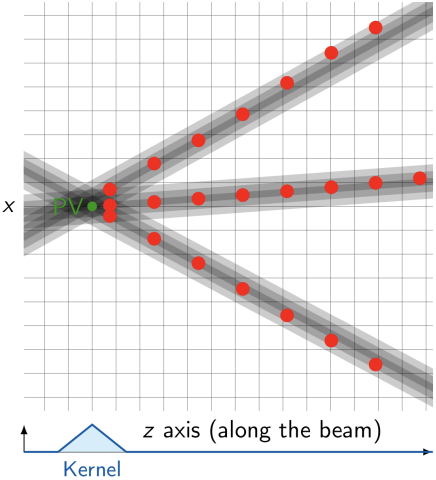
- Challenge: number of visible PVs will increase from $\sim$ 1.1 to 5.6 in Run 3 in LHCb
- Efficiency mainly driven by cluster search $\rightsquigarrow$ use machine learning
- The project is standalone, and uses toy data and it's own proto-tracking.
  This is the workflow in a nutshell:



Reminder: plot shown at CTD19
ACAT19 proceedings: arXiv:1906.08306

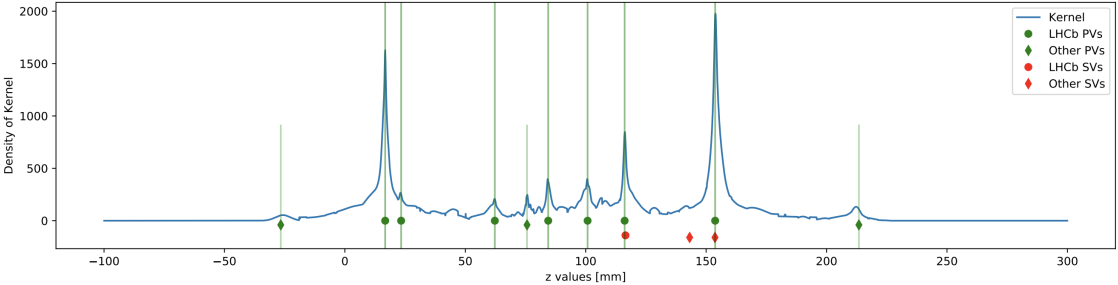
- Reduces sparse 3D data (41M pixels) to feature-rich 1D data – kernel densities in z
- Start from (LHCb Velo) tracks, *i.e.* closest to beam (x, y, z) position, slopes (t_x, t_y), (covariance matrix)
- Goal: find **kernel maximum in (x, y)** in each of the 4000, 100 μm wide z-bins. Assign it as z kernel value
- Currently, this is done by a coarse manual search followed by a `MINUIT` **minimization**
- Kernel $\mathcal{K}(z) := \frac{\sum_{\text{tracks}} \mathcal{G}(\text{IP}_{x,y}|z)^2}{\sum_{\text{tracks}} \mathcal{G}(\text{IP}_{x,y}|z)} - \sum_{\text{tracks}} \mathcal{G}(\text{IP}_{x,y}|z)$,

  where $\mathcal{G}$ is the product of Gaussian p.d.f.s in x and y with mean $(0, 0)$ and width given by **$\text{IP}_{x,y}$ uncertainties**. Those are **heuristic** with **proto-tracking**.



Reminder: plot shown at CTD19

KDE example



Reminder: plot shown at CTD19

- Typical KDE histogram (4k bins à 100 μm)
- Peaks generally correspond to PVs and SVs
- Challenges:
  - Vertex may be offset from peak
  - Vertices overlap
  - Some of the smaller peaks can be associated to vertices, others not

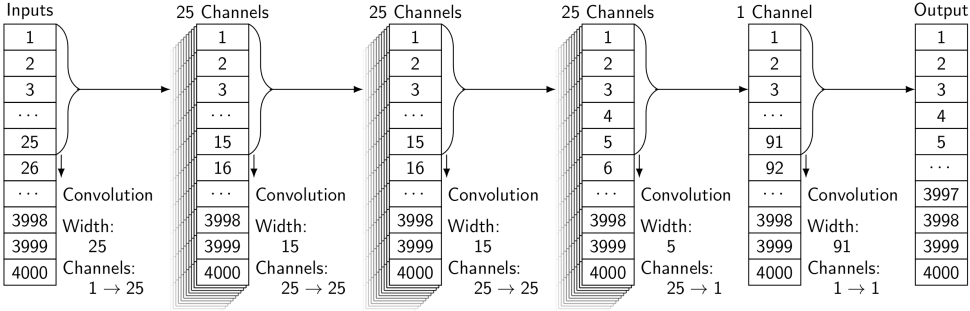
- Feed kernels to a convolutional neural network, implemented in `PyTorch`

  Further improvement possible with more layers and adding (x, y) perturbatively

- Cost function* modified with **asymmetry parameter****, which serves as powerful control to **balance efficiency to false-positive rate**

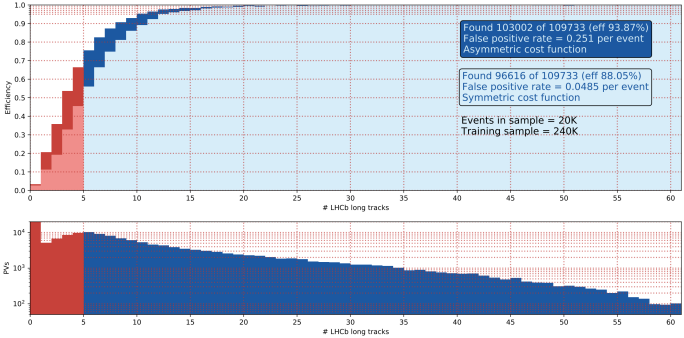
* inspired by cross entropy plus minimal offset ϵ to y and $\hat{y}$

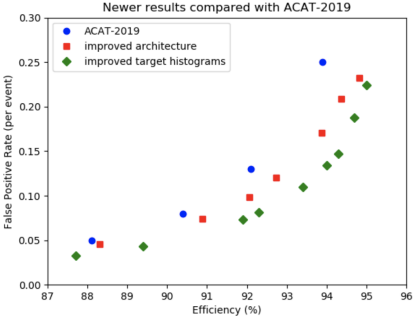
** (a-)symmetric w.r.t. $r = \frac{\hat{y}}{y}$



Reminder: plot shown at CTD19

Interpret results





Reminder: plot shown at CTD19/ACAT19

- Proof of principle established. Improved performance further by
  - modifying target histograms (learning proxies);
  - adding layers to CNN and adding x, y position information perturbatively.
- For a fixed efficiency of 94 %, the **false positive rate** is about **2× smaller**

- Algorithm is standalone, written mostly in `python` and has own data structure
- Need C++/CUDA versions adapted to LHCb (with ability to write data for standalone package)
- Kernel generation and **inference-engine** now deployed in **CPU** version of LHCb's HLT
- It runs on output of a new Velo tracking algorithm [arXiv:1912.09901]
- Decided for `TorchScript` as C++ inference engine

frameworks such as Caffe2, Microsoft Cognitive Toolkit, and MXNet. ONNX is still supported and actively worked in PyTorch v1.*x*, but it appears that TorchScript is the preferred way for model exporting. See the "Further Reading" section for more details on ONNX if you're interested.

Programming PyTorch for Deep Learning, I. Pointer, ISBN: 9781492045342

WARNING

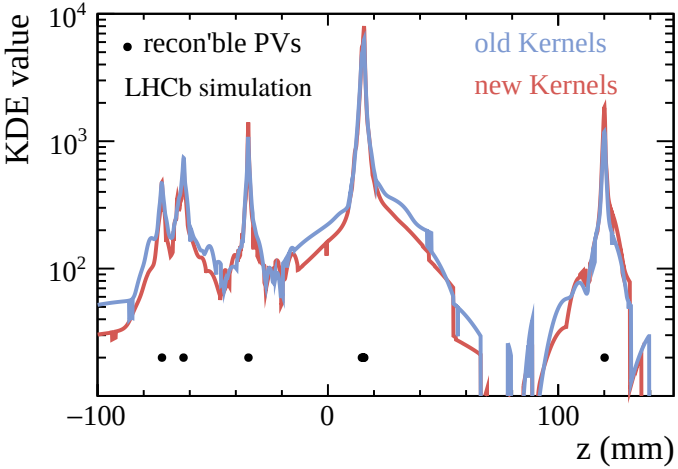
At the moment, the C++ API should be considered "beta" stability; we may make major breaking changes to the backend in order to improve the API, or in service of providing the Python interface to PyTorch, which is our most stable and best supported interface.

https://pytorch.org/cppdocs/

- We are currently collecting and evaluating results

  Running a model trained with toy data on official LHCb MC looks promising!

- With output of production Velo tracking, we can **use measured covariance matrix**.
  $\rightsquigarrow$ Uncertainty estimate for the impact parameters and their correlation in kernel-Gaussians $\mathcal{G}$, instead of approximated $\mathrm{IP}_{x,y}$ uncertainties.
- New kernels look **more pronounced in PV regions**



- A hybrid deep learning vertexing algorithm is introduced; it's proof of principle established
- It is (privately) deployed in the LHCb CPU software stack and it's performance improved
- Future milestones are well defined:
  - **Benchmark** performance with standard LHCb simulation and software, and compare to the current baseline PV finding algorithm [see dedicated talk by Florian Reiß]
  - **Re-train** the algorithm using full LHCb simulation in place of toy simulation
  - Develop an algorithm to **assign tracks to PVs probabilistically**
  - KDE generation too slow $\rightsquigarrow$ develop fast **ML algorithm for KDE generation**
  - **Prune** ML algorithms **if necessary** to fit into the HLT time budget

(Open) Source code of standalone package, including toy generation:

- `https://gitlab.cern.ch/LHCb-Reco-Dev/pv-finder`
- Runnable with Conda on macOS and Linux