PVFinder for Real-Time Primary Vertexing at LHCb

Hybrid DNN PV reconstruction for 30 MHz triggers at LHCb

Mohamed Elashri[1*], Simon Akar[2], Conor Henderson[1], Michael David Sokoloff[1]

[1]University of Cincinnati (US)
[2]LPCA - Université Clermont Auvergne (FR)
*Corresponding author

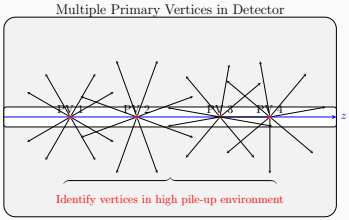
7th Inter-Experimental LHC Machine Learning Workshop

May 22, 2025

This work is supported by IRIS-HEP

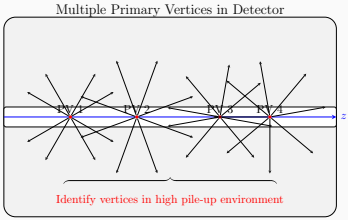
Why Fast PVs in LHCb Run 3?

- LHCb detector upgraded for Run 3 with **5-fold increase** in luminosity
- Pure software trigger requires fast, accurate PV reconstruction
- Flavour physics selections critically depend on prompt z_{PV}
- Strict latency budget: $\approx$ **200** μ**s** total processing time



Why Fast PVs in LHCb Run 3?

- LHCb detector upgraded for Run 3 with **5-fold increase** in luminosity
- Pure software trigger requires fast, accurate PV reconstruction
- Flavour physics selections critically depend on prompt z_{PV}
- Strict latency budget: $\approx$ **200** μ**s** total processing time

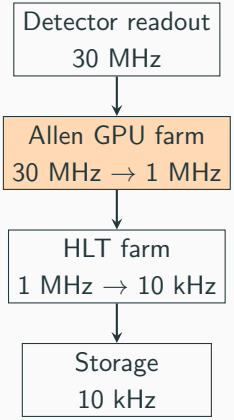


Challenge: Reconstruct all primary vertices at 30 MHz with high efficiency

LHCb Trigger Chain

Run 3:

- **Allen (GPU)** first-level trigger (30 MHz)
- Processing ∼40 Tbit/s raw data
- Triggerless readout to event building servers
- Full selection in software - no hardware trigger



GPU farm: ∼**500 GPUs** hosted in event building servers

First complete high-throughput GPU trigger in a HEP experiment

Inside Allen

Key Design Principles:

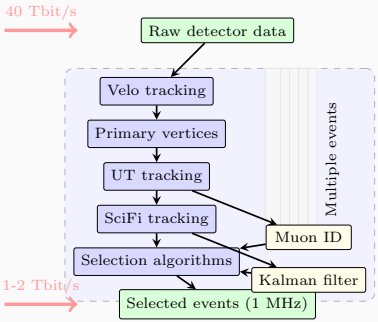
- Structure of arrays data model
- CUDA streams for event slices
- Zero dynamic allocation; pre-allocated pools
- One event ≈ one block
- Intra-event parallelism mapped to threads

Allen Pipeline:

- Track reconstruction (Velo, UT, SciFi)
- Vertex finding, muon ID, parameterized Kalman filter
- 1 MHz output with tracks & vertices

Integration Challenge: Adapt external libraries to Allen's thread/memory model

Allen GPU trigger paper
(arXiv:1912.09161)

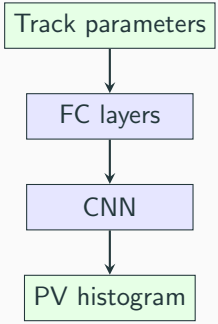


PVFinder Concept

Evolution from traditional PV-finding:

- Conventional: Heuristic track clustering
- PVFinder: End-to-end deep learning solution
- Hybrid approach:
  - FC layers + CNN architecture
  - Loss combines classification + regression
- Adaptable across LHCb and ATLAS

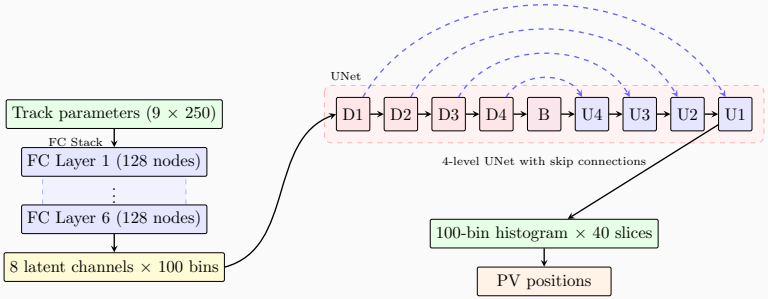
PVFinder paper (arXiv:2309.12417)



Two-stage process:

1. Track parameters → FC → latent "KDE-like" features
2. → CNN → predicted target histogram → PV positions

Network Architecture



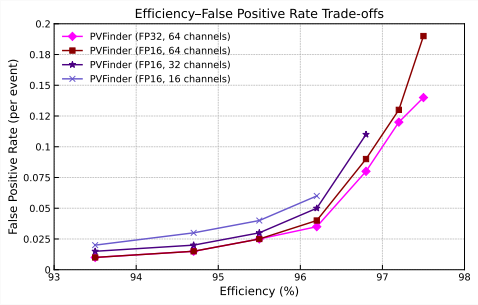
PVFinder algorithm steps:

1. **Spatial slicing:** Split detector z-region (4000 bins) into 40 intervals of 100 bins each
2. **FC Network:** Process track parameters within each slice to create histogram-like features
3. **CNN Network:** Process 1D histogram features with spatial context to predict PVs

LHCb Performance

Key metrics:

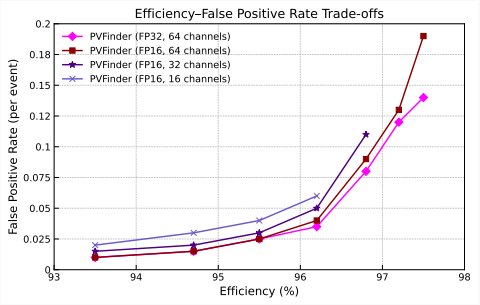
- **Efficiency:** >96% (exceeds heuristic)
- **False positive rate:** 0.03 per event
- **Z-resolution:** ∼12 μm (competitive)
- **FP16 quantization:** minimal impact
- **CNN channels:** 16-64 explored



LHCb Performance

Key metrics:

- **Efficiency:** >96% (exceeds heuristic)
- **False positive rate:** 0.03 per event
- **Z-resolution:** ∼12 μm (competitive)
- **FP16 quantization:** minimal impact
- **CNN channels:** 16-64 explored



Achievement: End-to-end tracks-to-hist model outperforms KDE-to-hist models

Embedding PVFinder in Allen: Requirements

Integration constraints:

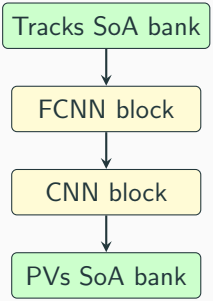
- 30 MHz input (33 μs/event)
- Latency budget: 150 μs kernel time
- Deterministic memory footprint
- Minimal external dependencies
- SoA data format compatibility

Allen processing model:

- Raw data copied to GPU once
- Full algorithm sequence on GPU
- Only selection results returned to CPU

Two-part implementation challenge:

1. Custom FCNN implementation **Completed**
2. CNN block with cuDNN integration **In progress**



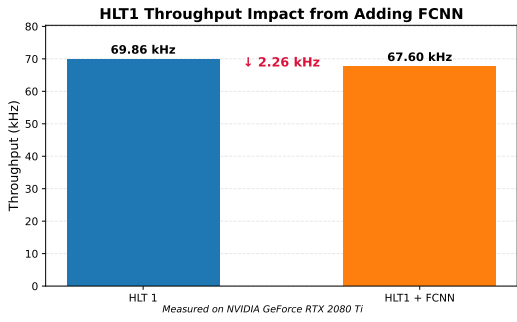
FCNN Inference Engine (implemented)

Implementation details:

- Hand-written CUDA matrix-vector ops
- Zero external dependencies
- Native support for Allen SoA format
- Integration with Allen memory pools

Performance:

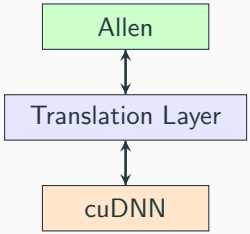
- **Impact:** Only 2.26 kHz reduction in throughput
- **Latency added:** 0.5 μs/event on NVIDIA RTX 2080
- **Throughput maintained:** 96.8% of baseline



CNN Block Integration Challenges

Key integration issues:

- **Format mismatch:**
  - cuDNN: NCHW tensor format
  - Allen: SoA flat arrays
- **Resource management:**
  - Workspace memory ownership
  - CUDA stream coordination
- **Execution model:**
  - Allen: async event-parallel
  - cuDNN: blocking API calls



Solution: Translation layer with batch-amortized memory operations

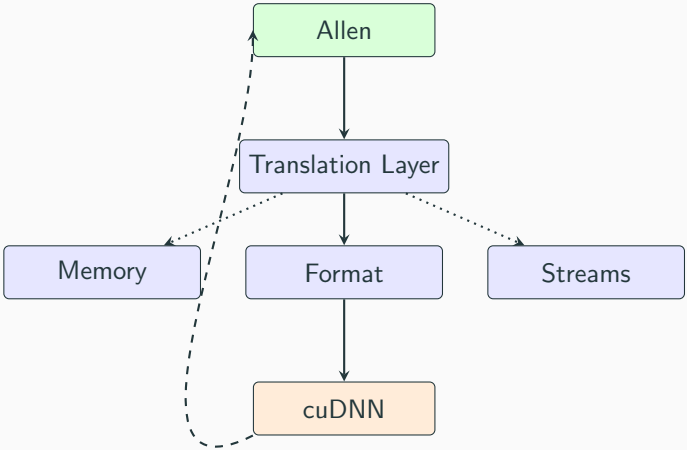
Translation Layer: Bridging Allen and cuDNN

The integration challenge:

- Allen and cuDNN have fundamentally different:
  - Memory models
  - Data layouts
  - Execution patterns

Our solution:

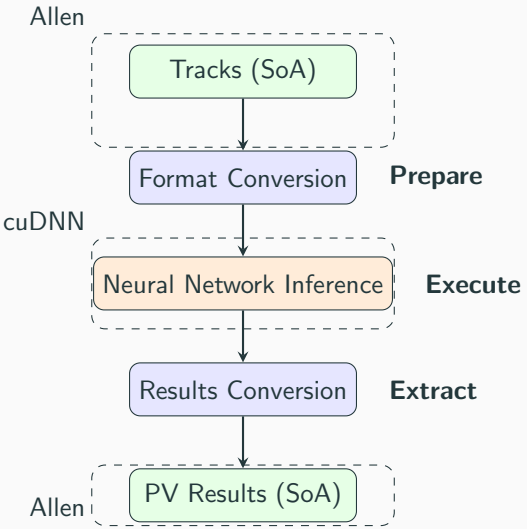
- Create an abstraction layer that acts as a bridge
- Handle all conversions and synchronization internally
- Present Allen-compatible interfaces to both sides



Translation Layer: Implementation Details

Three-phase implementation:

1. **Prepare Phase:**
   - SoA track data → NCHW tensors
   - Pre-allocated workspace setup
   - Batch dimension mapping
2. **Execute Phase:**
   - cuDNN neural network inference
   - Event-parallel processing
   - Non-blocking stream execution
3. **Extract Phase:**
   - NCHW results → SoA format
   - Memory cleanup and recycling
   - Data transfer optimization



Summary

Key Achievements:

- **Algorithm Development:** End-to-end tracks-to-hist PVFinder with $> 96\%$ efficiency
- **Performance:** False positive rate of 0.03 per event, competitive z-resolution
- **FCNN Implementation:** Custom CUDA engine integrated in Allen with minimal latency impact
- **Translation Layer:** Novel bridge architecture for Allen-cuDNN integration

Current Status:

- FCNN inference engine: **Completed**
- Translation layer design: **Completed**
- Full CNN integration: **Target Q3 2025**

Impact:

- First hybrid DNN approach for real-time PV reconstruction at 30 MHz
- Scalable framework for ML integration in high-throughput triggers

Thank you, Questions?

Backup Slides

Roadmap to Full Deployment

1. **Finalize translation layer API** (Q2 2025)
   - Complete abstraction shim between Allen and cuDNN
   - Validate memory management approach
2. **Fuse histogram maker + CNN** (Q2-Q3 2025)
   - Remove intermediate data copies
   - End-to-end slice processing
3. **Harmonize memory pools** (Q3 2025)
   - Integrate with Allen's allocation strategy
   - Optimize for GPU cache usage
4. **End-to-end benchmark** (Q4 2025)
   - Target: < 120 μs total latency
   - Measure throughput at scale
5. **Validation on 2025 data** (Q4 2025-Q1 2026)

Backup: Translation Layer API Sketch

```
class TranslationLayer  public:
TranslationLayer( AllenMemoryPool pool, const
AllenConfig config);
// Convert Allen SoA to cuDNN format
void prepareInputTensor( const AllenBank
input_bank);
// Run CNN forward pass void
forward(cudaStream_t stream);
// Copy results back to Allen format void
extractResults( AllenBank output_bank);
private: CudnnWorkspaceHandler
m_workspace; TensorConverterm_converter; UNetConfigm_config; ;
// namespace allen_cudnn//namespacepvfinder
```

Key design principles:

- Clean separation of concerns
- Explicit memory ownership
- Non-blocking operations
- Minimal data copies
- Thread-safe execution

Memory contract:

- Pre-allocated workspace
- No dynamic allocations
- Fixed tensor shapes
- Reuse conversion buffers

Backup: Detailed Latency Budget

Target latency breakdown:

Component & **Budget (μs)**
Input preparation & 10
FCNN inference & 7
Format conversion & 15
CNN forward pass & 70
Output conversion & 10
Peak finding & 8
Total & **120**

Performance targets:

- 97%+ efficiency
- $<$ 5% false positive rate
- $<$ 150 μs average latency
- $<$ 200 μs worst-case latency

Allen context:

- 30 MHz input rate
- 33 μs wall-time per event
- 150 μs kernel time budget
- Must maintain deterministic latency