Sparse Neutrino Detector Simulation on GPUs

An Example from the DUNE Near Detector

Yousen Zhang[1], Brett Viren[1], Mary Bishai[1], Sergey Martynenko[1],

Xin Qian[1,2], Rado Razakamiandra[2], Brooke Russel[3]

[1]Brookhaven National Laboratory, [2]Stony Brook University, [3]Massachusetts Institute of Technology

arXiv:2602.12052

https://github.com/wirecell/tred

@BrookhavenLab

Brookhaven National Laboratory

DUNE: Large Detectors, Sparse Activity

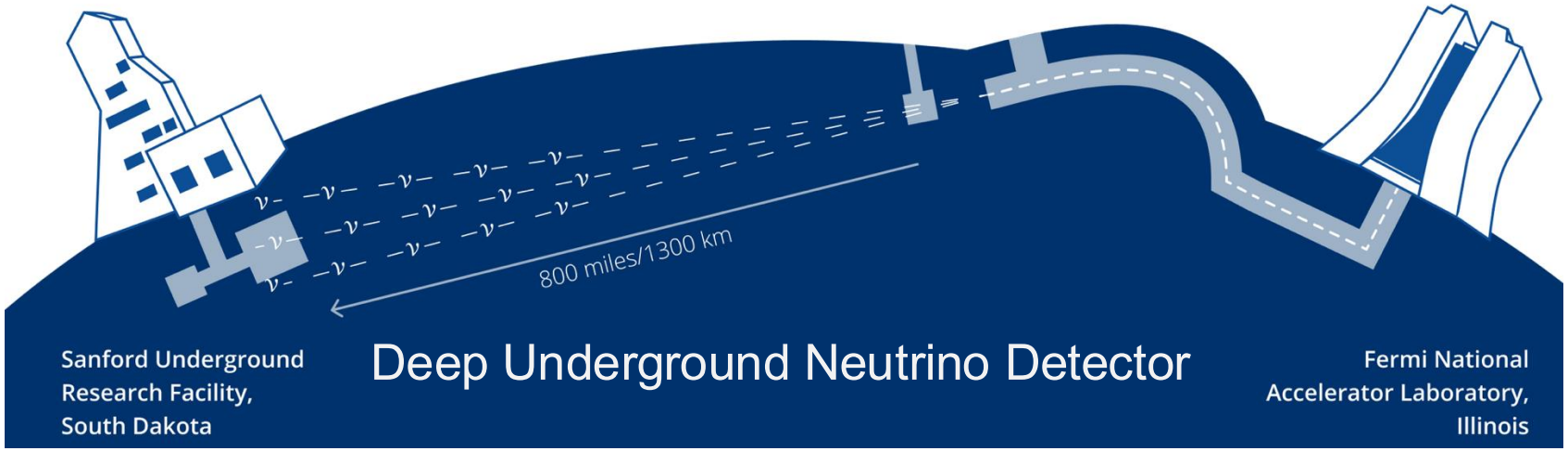
- Next-generation long-baseline neutrino experiment
- Near Detector (Fermilab) + Far Detector (SURF)
- Large-scale liquid argon TPC detectors

Computational challenges
- Large detector volumes
- Fine-granularity readout
- Millions of channels

Opportunities
- Highly localized charge deposition

Large detector volume + sparse activity

→ **Exploit sparsity without losing GPU efficiency**



Brookhaven National Laboratory

ND-LAr at DUNE

Pixelated charge readout

- ~ 14 million channels
- ~ 4 mm pixel pitch over a ~7 × 3 × 5 m³ active volume

Beam environment

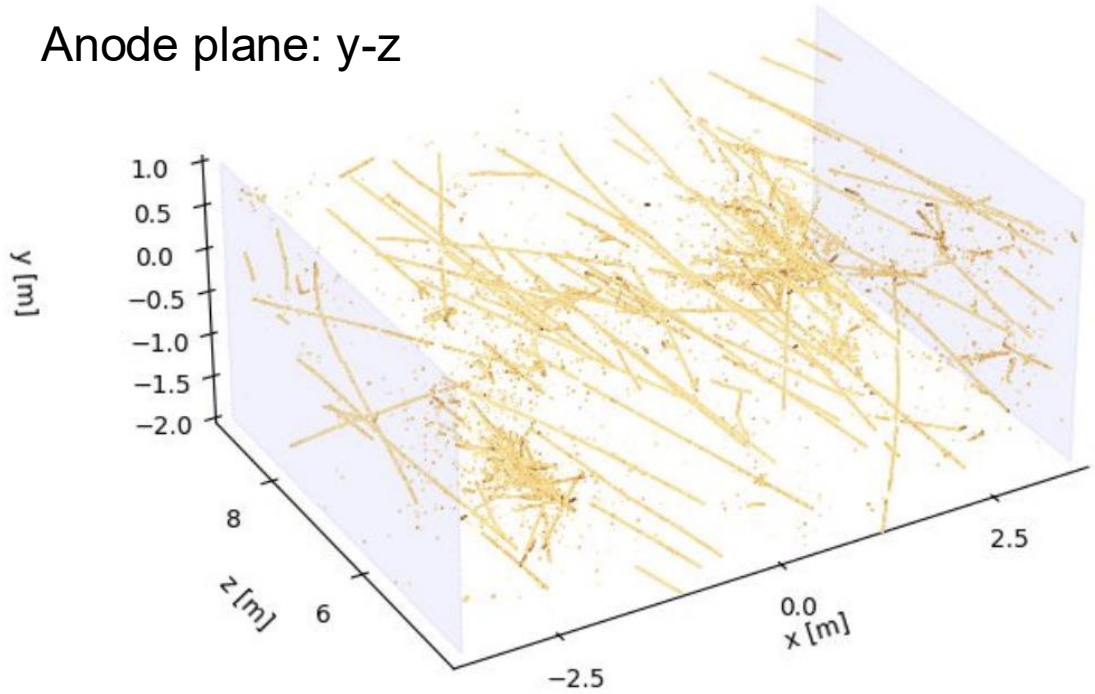
- O(50) neutrino interactions inside NDLAr per beam spill
- Additional activity from upstream interactions

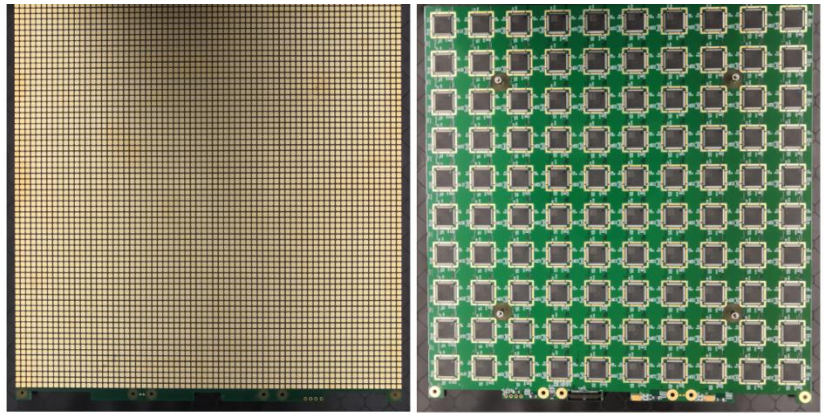
Sparse and irregular detector response

- Charge localized along particle trajectories
- Tracks may appear anywhere in the detector volume
- Most channels remain inactive

Simulated LBNF beam spill

Anode plane: y-z





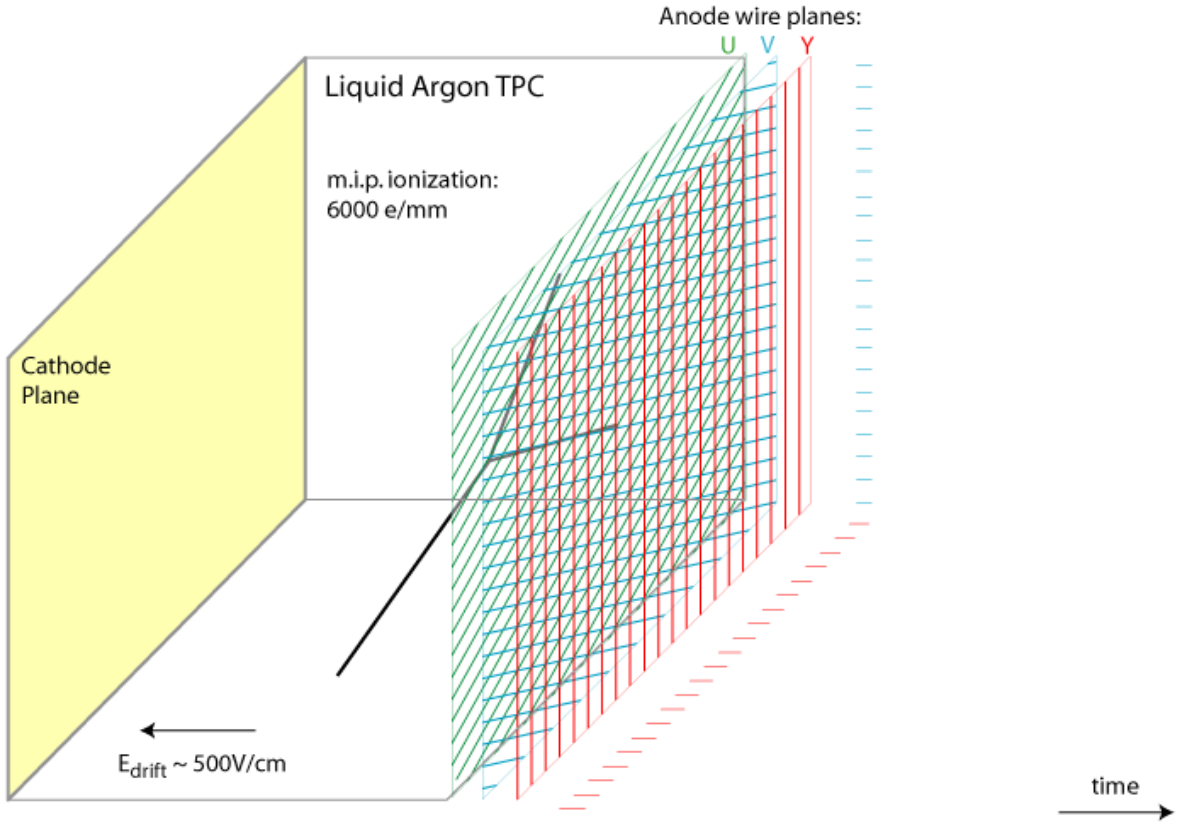
Front and back of an anode pixel tile.

Brookhaven National Laboratory

Ionization charge signal in LArTPC

- Drift under the electric field.
- Recombination.
- Diffusion effect.
- Attenuation due to impurities in liquid argon.

Credit: Bo Yu



Computational challenge

Charge transport

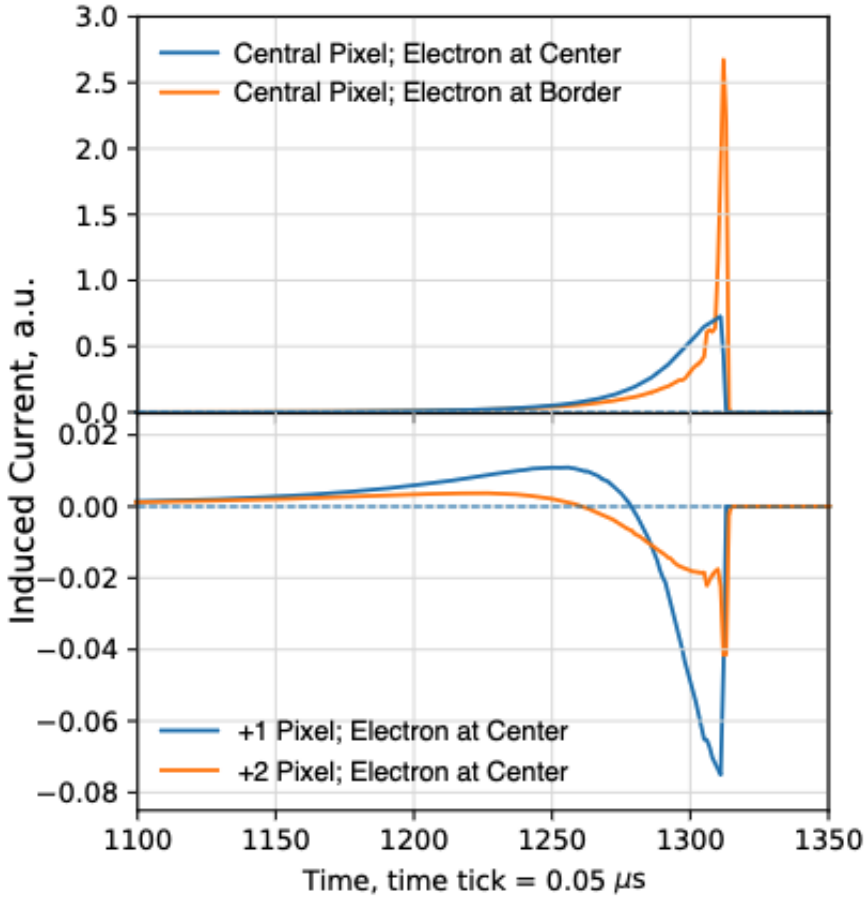
- Charge deposition is localized
- Diffusion spreads charge across multiple pixels
- Fine pixel granularity → large channel count

Signal formation

- Drift times are O(ms)
- Long waveforms are induced
- Neighboring pixels also receive induced current

Computational impact

- Large memory footprint
- Expensive charge–pixel evaluations
- Difficult to scale



Existing solution: larnd-sim

Existing framework: larnd-sim

- Official DUNE ND-LAr simulation framework
- GPU-accelerated sparse detector simulation
- Detector-specific implementation optimized for ND-LAr

Motivation for package TRED in this talk

- General sparse-tensor formulation
- Reusable operators across detector simulations
- Closer integration with AI/HPC software ecosystems

Highly-parallelized simulation of a pixelated LArTPC on a GPU

The DUNE Collaboration

E-mail: roberto@lbl.gov

larnd-sim
JINST 18 (2023) P04034

IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. XX, NO. XX, XXXX 2026

GPU-Accelerated Analytic Simulation of Sparse Signals in Pixelated Time Projection Detector

Yousen Zhang, Brett Viren, Mary Bishai, Sergey Martynenko, Xin Qian, Rado Razakamiandra, Brooke Russell

TRED

Analytic solution

The factorized solution: $I(\mathrm{x}_p, t) = \int \mathrm{dr'}\, Q(\mathrm{r'}; t) * i(\mathrm{x}_p, \mathrm{r'}; t)$ with

- $Q(\mathrm{r'})$: charge distribution at the anode
  - Transverse diffusion width is comparable to readout pitch
  - Longitudinal diffusion width is much smaller than drift time and signal is only sizable near anode.
- $i(\mathrm{x}_p\ , \mathrm{r'})$: pixel response to a point charge.
  - Decided only by detector geometry and operation condition.
  - Compute numerically according to Ramo theorem.

Ramo theorem

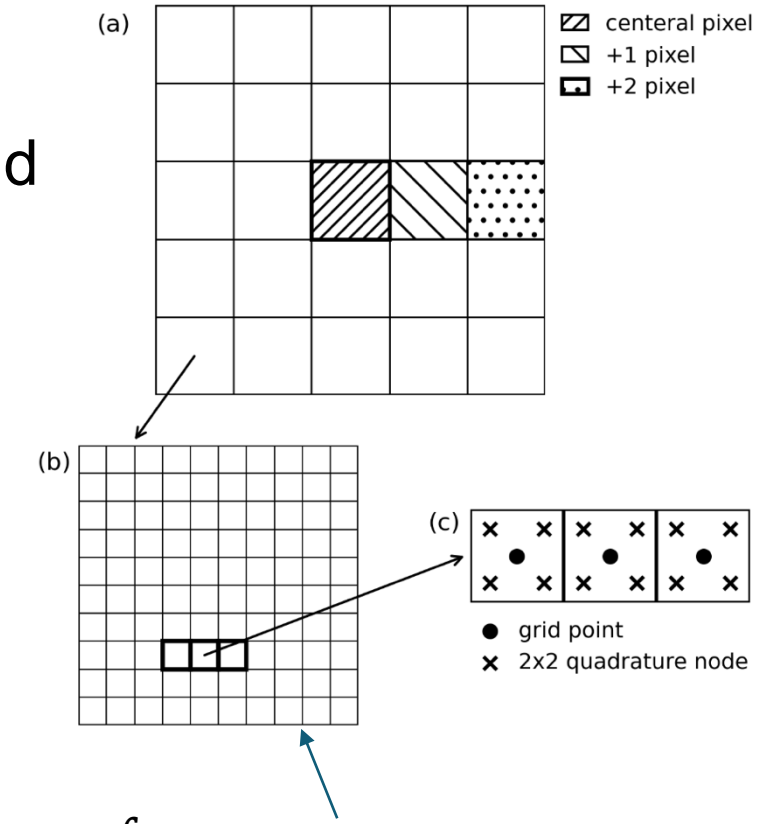
$$I = -q \cdot \vec{E}_w \cdot \vec{v}_q$$

v_q: drift velocity
E_w: weighting field
q: source charge

Discretization and Effective Charge

Continuous integral → quadrature sum on a regular grid

Field responses are evaluated at grid points and interpolated to quadrature nodes.



Effective charge

Q^{eff} absorbs quadrature weights and linear interpolation factors.

$$\int \mathrm{dr'}\, Q(\mathrm{r'};t)\, i\left(\mathrm{x}_p, r';t\right) =$$

$$\sum_{i\in grid} \int \mathrm{dr'}\, Q_i I_i = \sum_{i\in grid} Q_i^{\mathrm{eff}} I_i$$

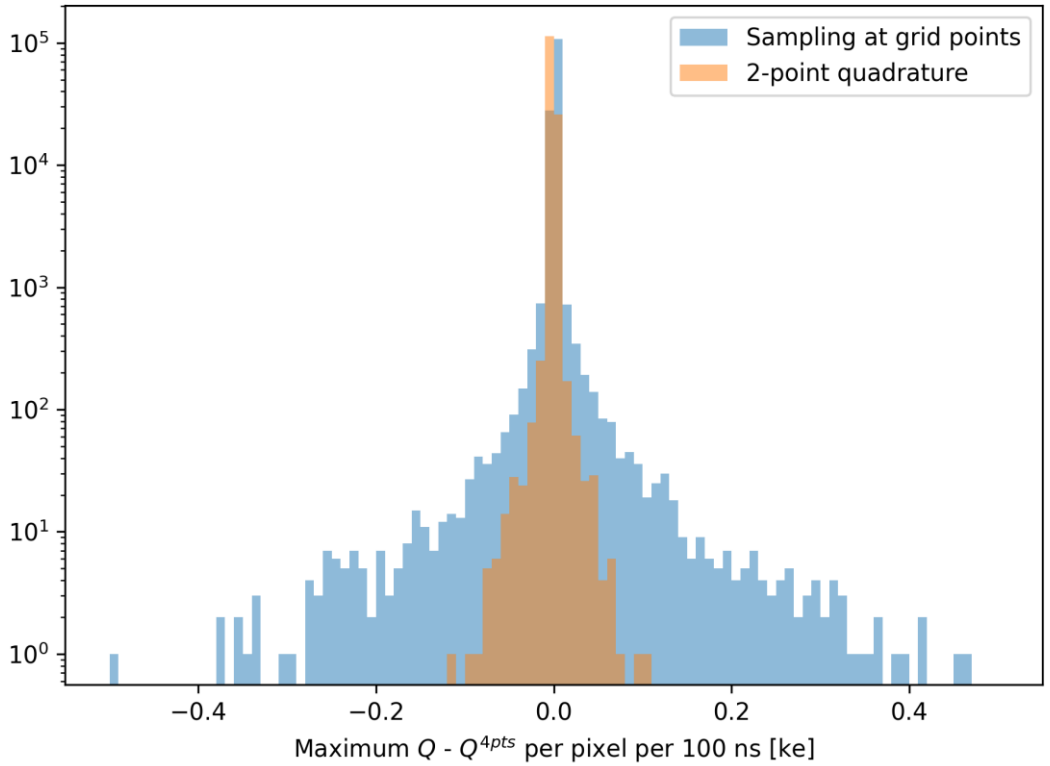
Brookhaven National Laboratory

Is the discretization accurate?

- Sampling at grid points ignores sub-grid structure.
- 2-point quadrature captures intra-pixel variation.
- Only a few quadrature nodes are required.

Result

- 2-point quadrature is already close to the 4-point reference.
- **Sub-grid accuracy at low computational cost.**



For reference, frontend noises in LAr is <O(500e-)

Computing Q^{eff} as a Convolution Problem

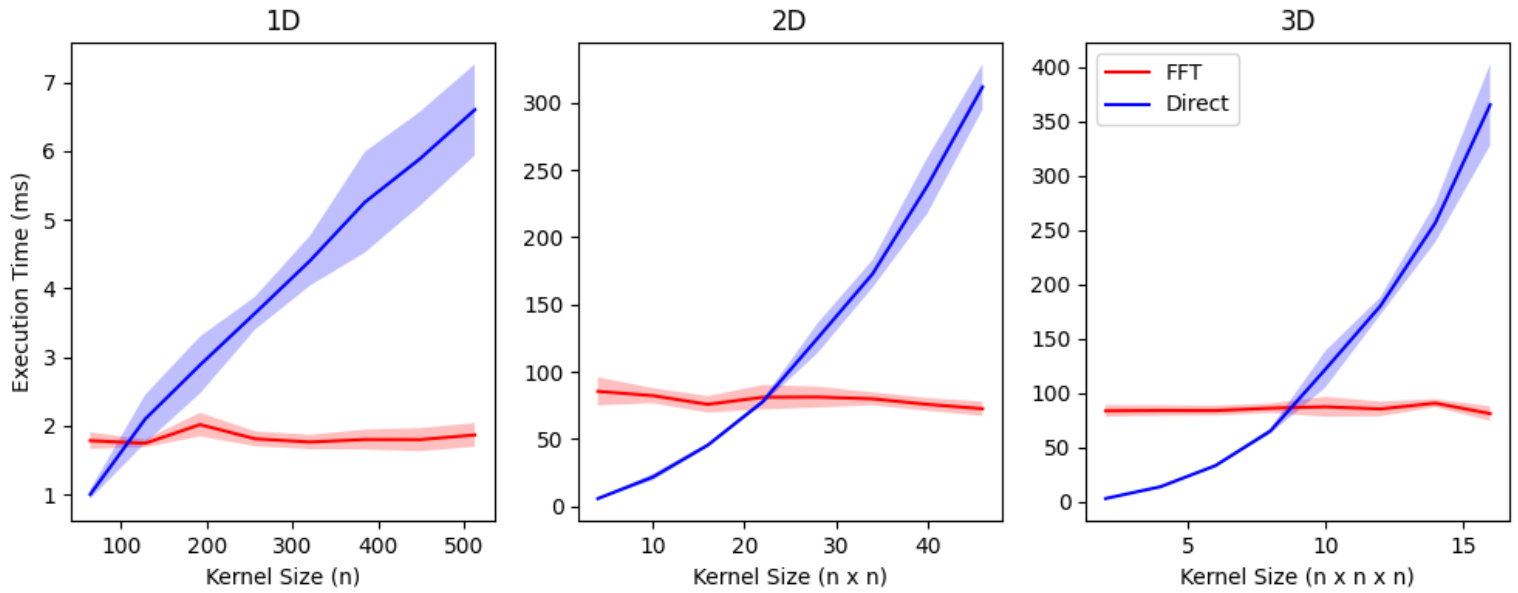
- $Q^{\mathrm{eff}} = Q * K$
  - Q obtained by analytic evaluation at irregular quadrature nodes, assembled into regular dense tensor blocks
  - K contains quadrature weights and interpolation coefficients
  - Block size limited by diffusion scale.

- **GPU-friendly implementation**
  - Dense local tensor blocks
  - Sparsity tracked through block coordinates
  - Convolution kernels from PyTorch
  - Native GPU acceleration

Sparse globally, dense locally:
Detector charge is represented as coordinate-indexed dense tensor blocks.

Convolution with Field Response: Direct or FFT

- **Challenge**
  - Field responses extend over long drift times
  - Large convolution kernels in time and space
  - Direct convolution scales poorly, ~O(NK)
- **FFT Acceleration ~** O(Nlog N)
  - Better scaling for large kernels
  - Efficient GPU implementations available



Brookhaven National Laboratory

FFT is Fast, but What About Sparsity

- Sparse structures in the time domain often lead to dense frequency-domain representations.
  - An intuition analogue to Heisenberg's **uncertainty principle**, $\boldsymbol{\Delta x \Delta p \geq \hbar/2}$

- **FFT operates on dense tensor blocks**
  - Sparse tensor frameworks (COO/CSR) do not support FFT
  - Coordinate indexed dense blocks are compatible with FFT

Coordinate-indexed dense blocks enable FFT-based convolution. The remaining challenge is detector-scale data organization.

From Local Blocks to Detector Signals

Sparse induced current signal in detector are represented as local blocks.

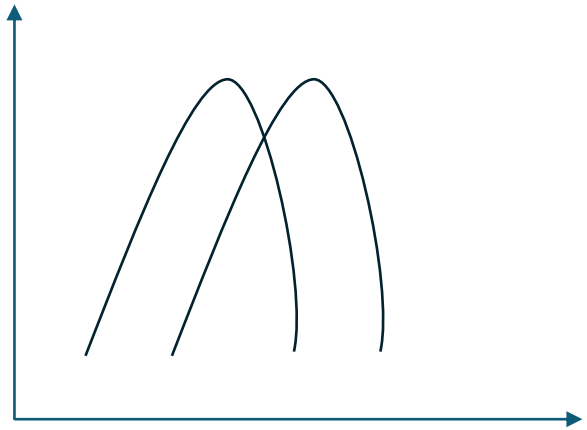
- Multiple blocks may contribute to the same detector region.
- The same merging problem appears for charge, Q^{eff}

Challenge

- Local sparse blocks are not naturally aligned on the detector grid.
- Efficient merging requires both alignment and accumulation.

Benefit

- Early alignment and accumulation reduces memory footprint and downstream computation.



Q1+Q2 waveform

Block Sparse Binned (BSB) tensor

Inspired by histogramming and scatter-add operations, **BSB generalizes binning and accumulation from scalars to sparse tensor blocks.**

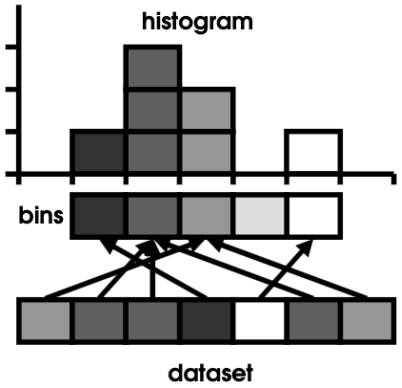
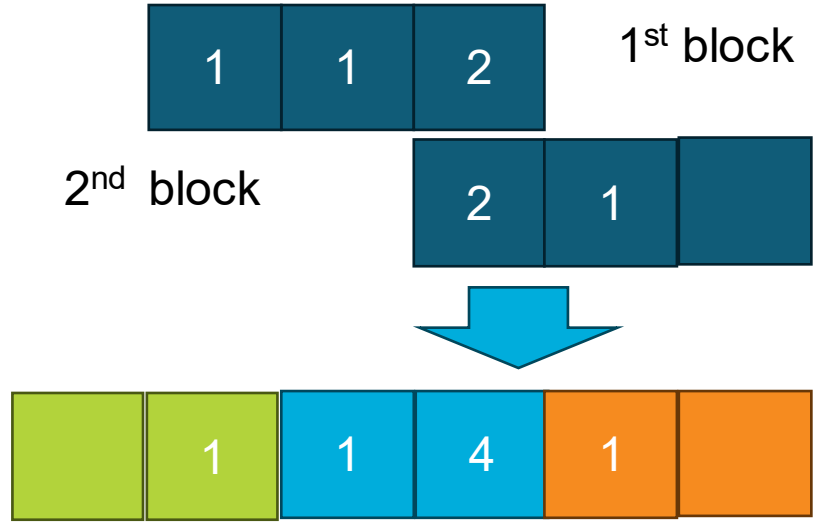


Figure 1: Parallel histogram computation leads to memory collision, when multiple elements of the dataset update the same histogram bin.

```
scatterAdd(<T> a[m], int b[n], <T> c[n]) {
  forall i = 1..n
    ATOMIC{a[b[i]] = (a[b[i]] + c[i])};
}

scatterAdd(<T> a[m], int b[n], <T> c) {
  forall i = 1..n
    ATOMIC{a[b[i]] = (a[b[i]] + c)};
}
```



Output: binned blocks or a binned envelop

- Input: size A block
- Output envelop: size NxB binned block with k non-empty bins saved.
- Note: B does not need to divide or align with A.
- Example:
  - A=3 blocks, B=2 blocks, N=3 bins, k=3 bins, 4 non-empety blocks

Histogram and scatter-add adapted from: J. H. Ahn, et al., HPCA 2005.

Why BSB

BSB provides

- **Flexible envelope size**
  - Independent block and bin sizes
- **Efficient sparse accumulation**
  - Alignment + accumulation in one abstraction
  - Applicable to Q^{eff}, and induced currents

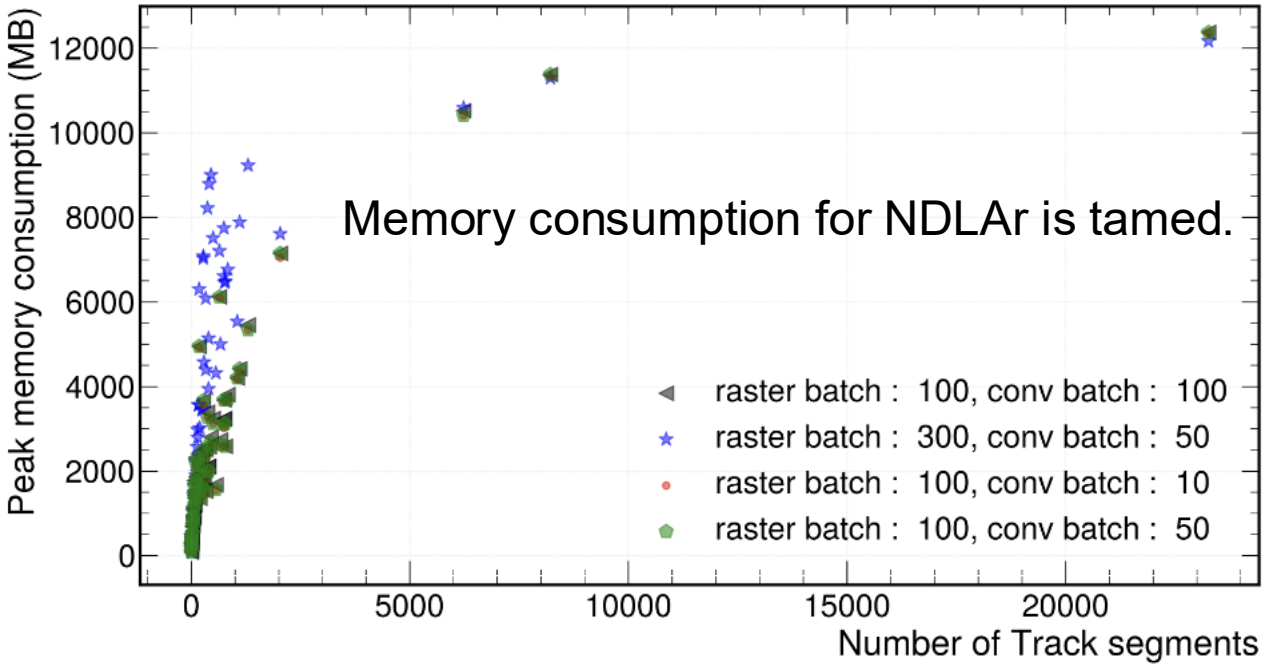
BSB provides

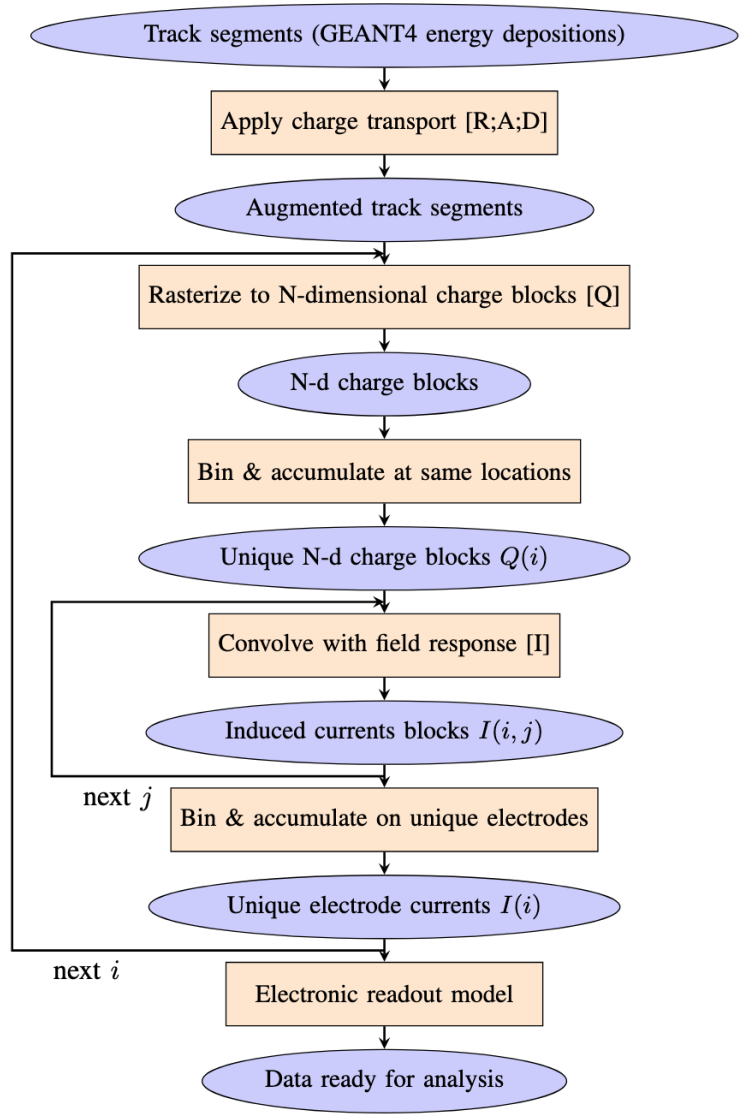
- **Reduced memory footprint**
  - Early merging of overlapping contributions
  - Only non-empty bins are stored

Sparse data representation and processing at detector scale

Practical Impact: Memory Scaling

- Hierarchical batching enabled by BSB
- Peak memory converges to the output footprint



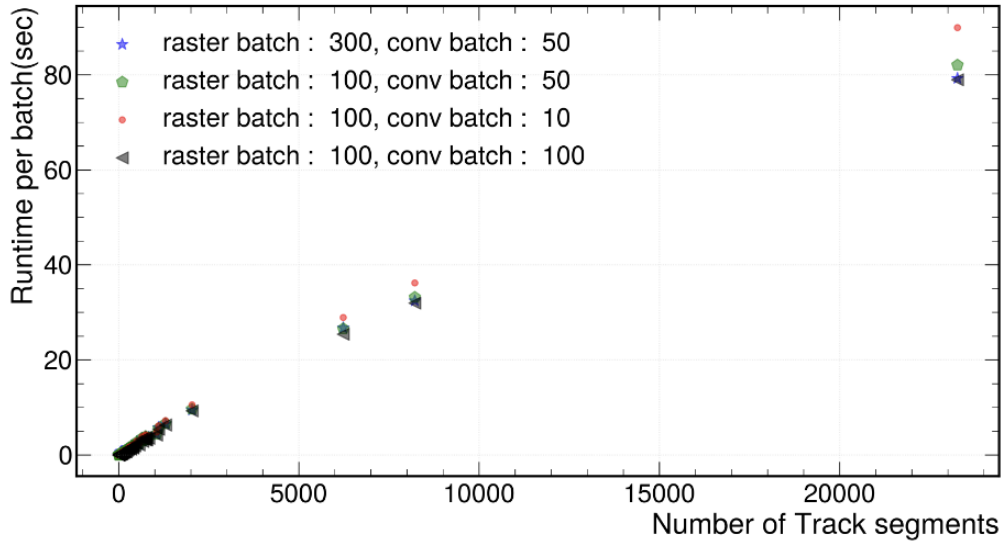


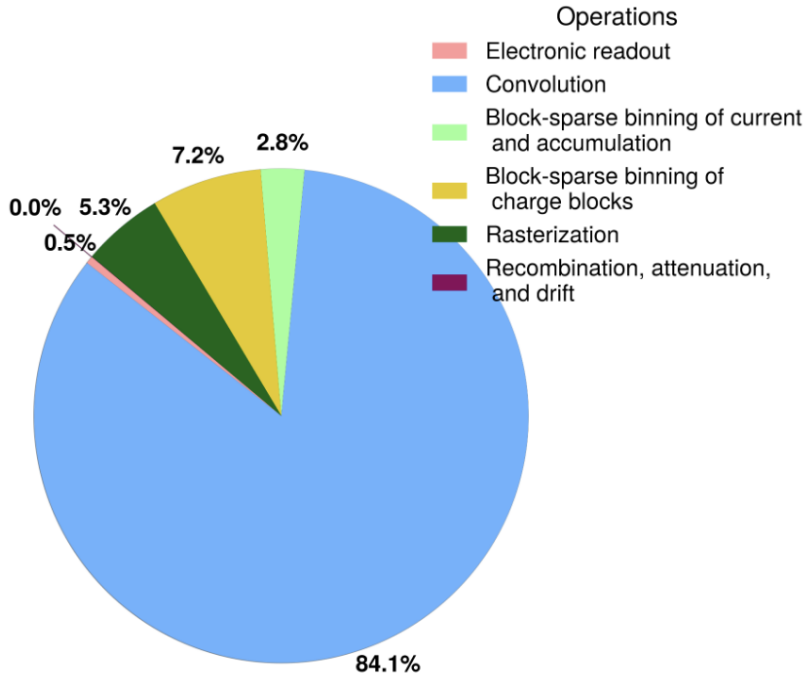
Runtime benchmark

FFT convolution dominates the runtime, as expected

Runtime scales approximately linearly with spill activity

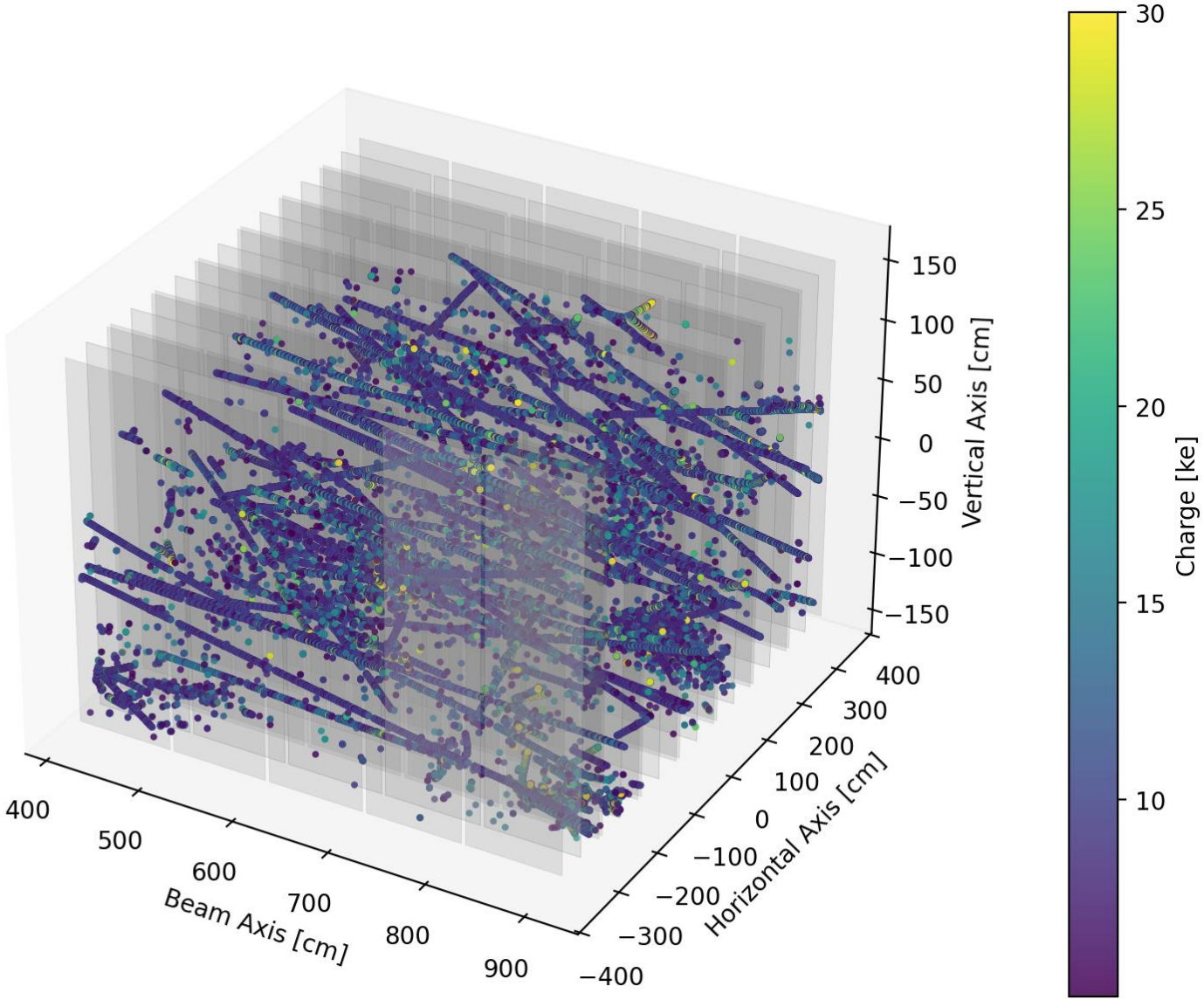
Full ND-LAr beam spill simulated in < 10 minutes on RTX 4090





Brookhaven National Laboratory

Our simulation package can model full ND-LAr beam-spill activity



The package handles detector-scale geometry, charge response, and complex event activity in one workflow.

HEP Software in the AI Era

- **Traditional HEP software**
  - Powerful and highly configurable
  - Documentation contains the answers
- But:
  - steep learning curve
  - expert knowledge required
  - trial-and-error configuration

- **Can we just ask an AI assistant?**
  - LLMs do not understand detector workflows
  - Missing execution context
  - No access to internal states and logs
  - Can generate plausible but incorrect actions

The challenge is exposing domain knowledge in a structured way.

Exposing Detector Expertise to AI Systems

```
you› Provide recommended configurtion and run a test on this machine. Input file is
 workspaces/data/tred/tred-assitant/MicroProdN1p1_NDLAr_1E18_RHC.convert2h5.nu.0000
001.EDEPSIM.hdf5 and response file is workspaces/data/tred/tred-assitant/response_f
or_tred.npz

Recommended config (auto-generated)

Config YAML:

 • workspaces/runs/tred/configs/generated/config.yaml

Key recommendation info from the helper:

 • Auto-detected GPU budget: 10.195 GB usable (of 11.99 GB total)
 • Peak stage (worst event): event_id 1110
 • Per-stage peak GPU use (GB):
    • effq: 10.155 GB (fits; tunable via batch_scheme[0], currently 242)
    • chunksum_q: 0.739 GB (fits; tunable via batch_scheme[0])
    • convo: 1.891 GB (fits; tunable via batch_scheme[1])
    • chunksum_i: 0.028 GB (fits)

Test run (single event)

I launched a test on event_id=1110 (the peak event for GPU memory).

 • Job log: workspaces/runs/tred/logs/test.log
 • Job id: job-20260616-010306

Ask me “check status” and I’ll poll the test job until it finishes.

you› check status

Test is still running on event_id=1110.

 • Job id: job-20260616-010306
 • Elapsed: ~108 s
 • Current log tail (workspaces/runs/tred/logs/test.log): processing continues
   (effq/raster, chunksum_charge, convo, readout stages), no errors shown yet.
```

Encoded expertise

- Configuration recommendation
- Resource estimation
- Workflow orchestration
- Runtime inspection and monitoring

Model-agnostic interface

- No detector-specific model training
- No fine-tuning required
- Compatible with multiple LLM backends
- Reusable across interfaces

Access methods

- Lightweight conversational client
- MCP interfaces
- Skill interfaces

Domain knowledge is encoded in tools and workflows, not in model weights.

Brookhaven National Laboratory

Conclusion

- Sparse detector simulation can be reformulated as **GPU tensor operations**.
- Analytic factorization + quadrature achieve **sub-grid accuracy**.
- FFT solves convolution complexity.
- BSB provides **detector-scale sparse alignment and accumulation**.
- Can extend to DUNE Far Detector simulation.
- AI-assisted workflows for HEP software and detector operations.

Backup

Discretization and Effective Charge

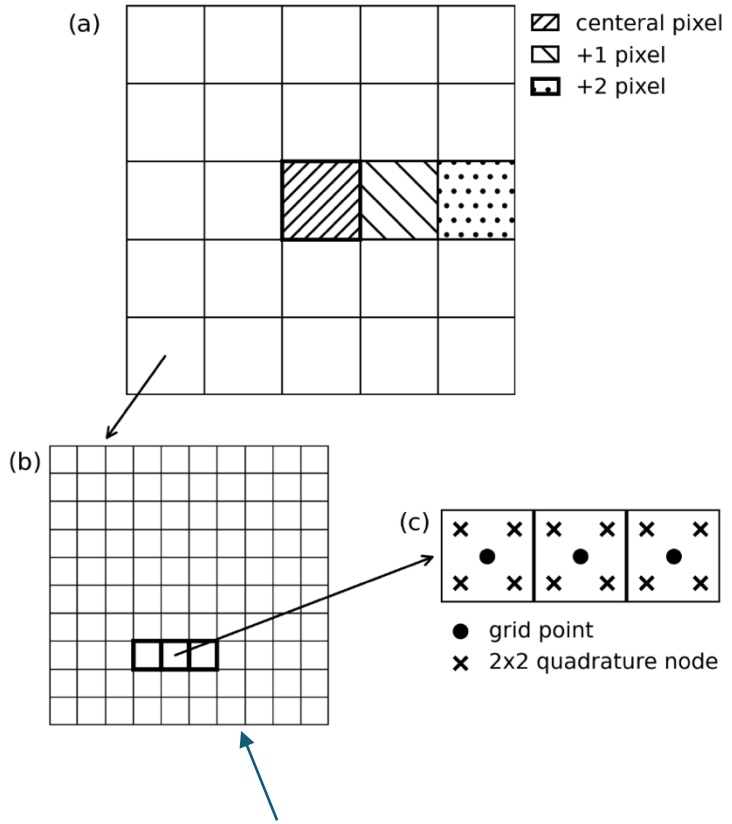
$$\mathcal{I}(\mathbf{x}_p) = \sum_{i,j,k} Q^{\text{eff}}_{ijk}\, I_{ijk}$$

Continuous charge integration is replaced by quadrature sum on a discrete grid.

Quadrature preserves accuracy on a discrete field-response grid. $\int_{-1}^{1} \mathrm{d}x\, h(x) \approx \sum_i w_i h(\xi_i)$

Field response are evaluated at grid points. FR at quadrature nodes from linear interpolations (u_{rst}).

Q^{eff} absorbs quadrature weights and linear interpolation factors.

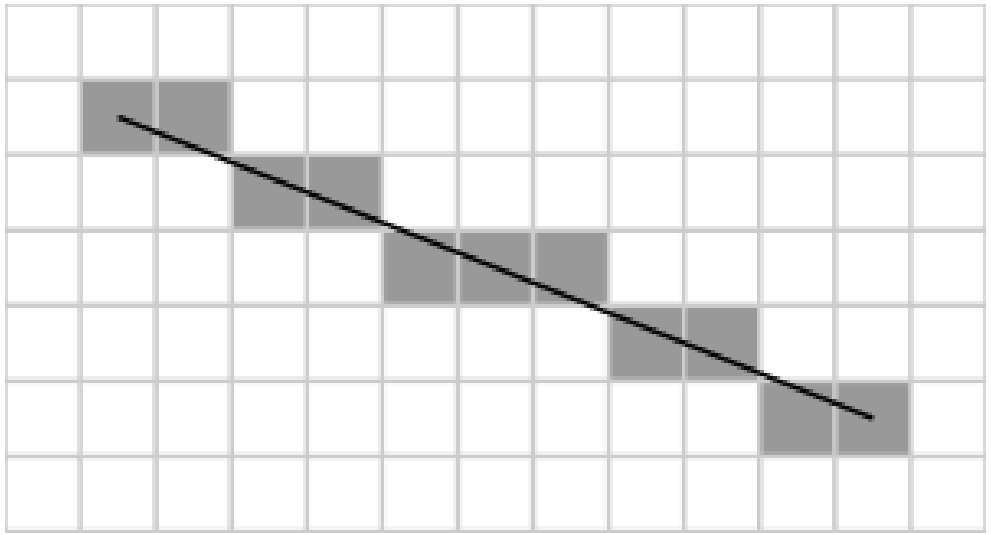


$$\int \mathrm{d}\mathbf{r}'\, Q(\mathbf{r}')I(\mathbf{r}';\mathbf{x}_p) = \sum_C \int_C \mathrm{d}\mathbf{r}'_C\, Q(\mathbf{r}'_C)\, I(\mathbf{r}'_C;\mathbf{x}_p)$$

$$Q^{\text{eff}}_{ijk} = \sum_{l,m,n}\sum_{r,s,t} w_{lmn}\, Q_{lmn,i-r,j-s,k-t}\, u_{lmn,rst}$$

Extend to far detector simulation

- The far detector simulation uses a global region of interests in FFT.
- TRED is more sparse.
- Preliminary tests shows 2x speed up using sparse signal, on a single CPU core.



White: FD simulated area
Grey: TRED simulated area

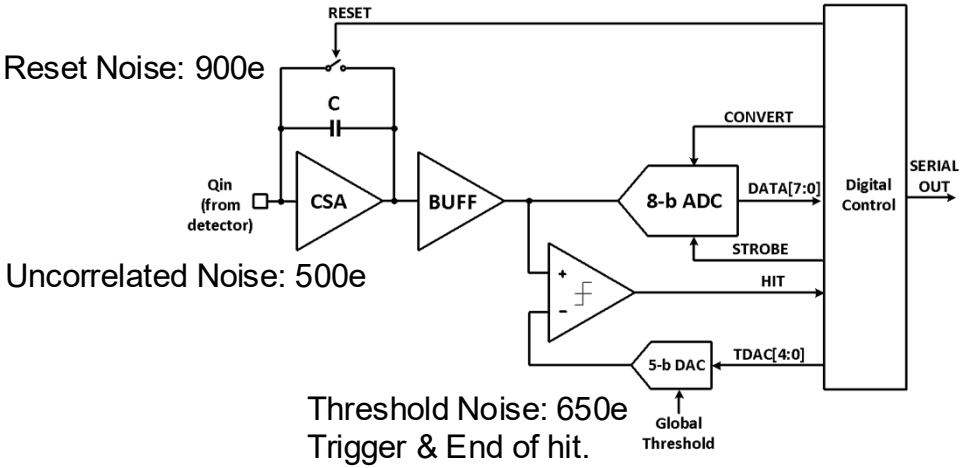
The signal simulation, i.e., the analog-to-digital converter (ADC) waveform on a given channel,

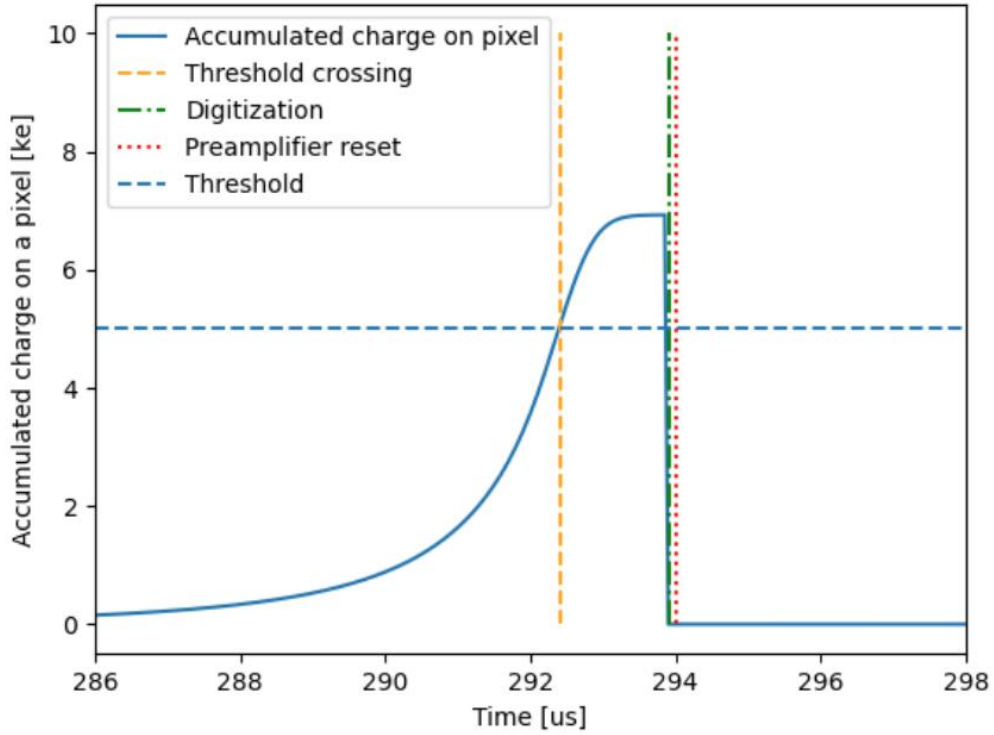
$$M = (Depo \otimes Drift \otimes Duct + Noise) \otimes Digit, \tag{4.2}$$

is conceptually a convolution of five functions:

Readout as the last piece of simulation

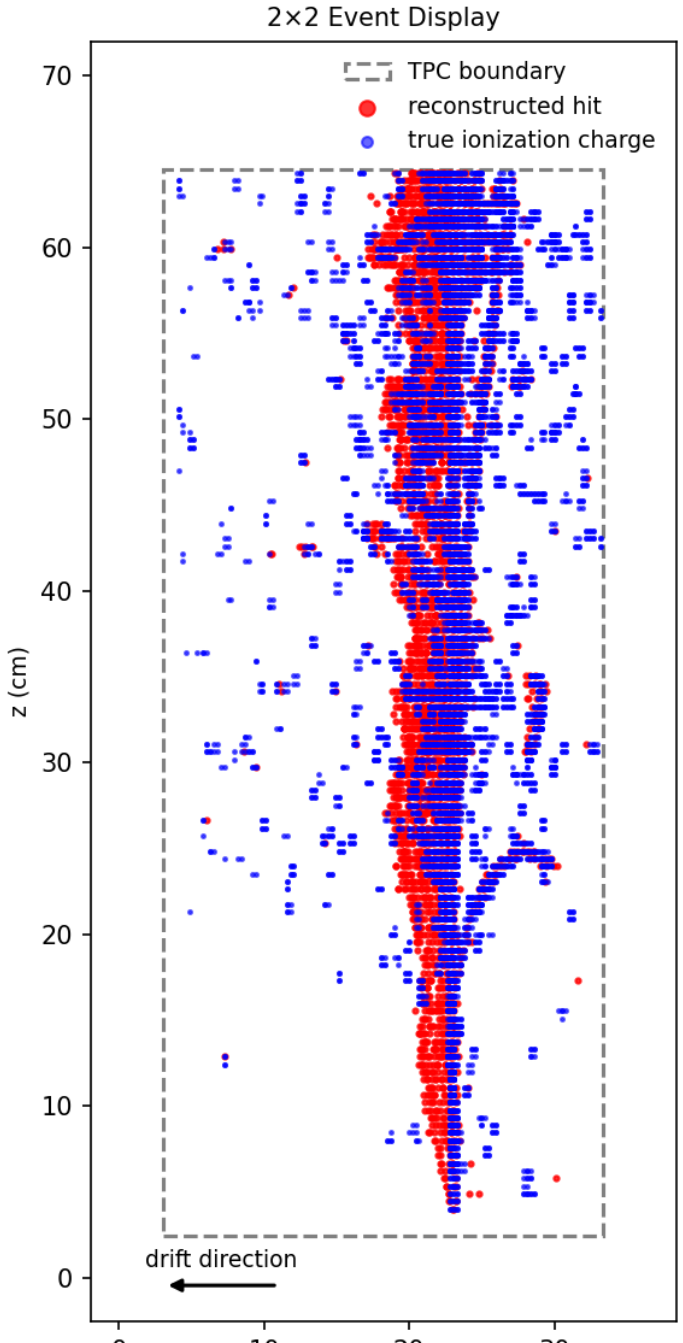
- Self-trigger readout logic to zero suppress data
  - Charge ≅ Charge @ Trigger Time + Charge in ADC_HOLD_DELAY window





Signal processing in zero-suppressed detector

- **Issue:** Ionization charge signals can be split into multiple hits due to long-range induced coherent signals from showers.
- Signal processing is needed to recover the original charge information.
- **Limitation:** Traditional signal processing requires full waveforms, fast digitization, and bandwidth shaping.
  - However, full waveforms are often zero-suppressed.

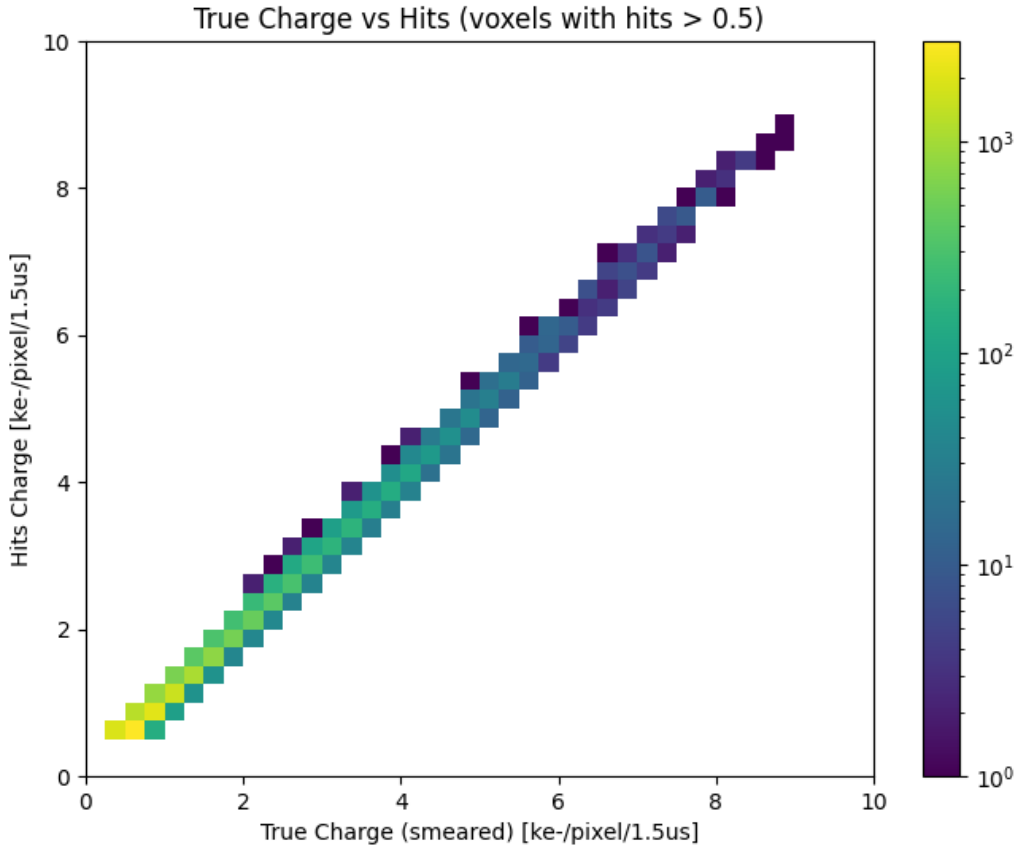


Brookhaven National Laboratory

Physics guided deconvolution

Solution: LArPix provides multiple operation modes for flexible signal readout.

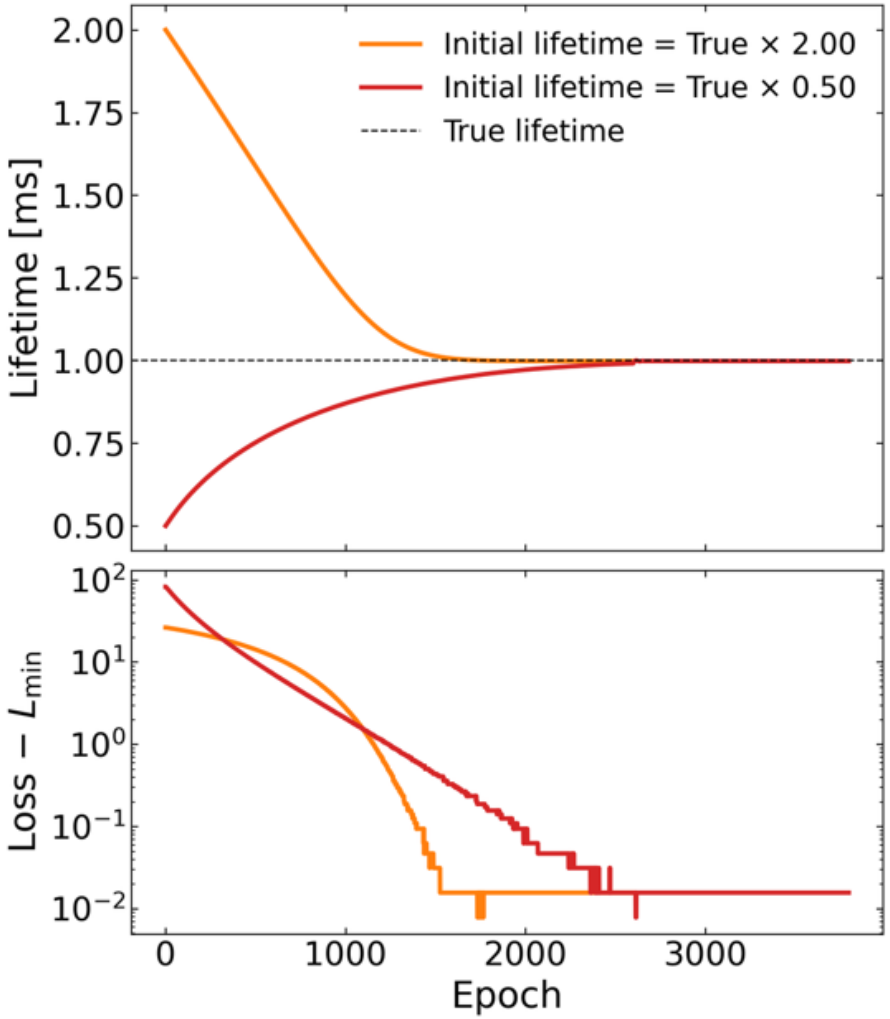
- **Burst mode:** continuously records charge data after a single trigger, enabling targeted waveform capture for signal recovery..
- **Deconvolution approach:** Recover zero-suppressed waveforms using physics-guided signal templates.



Brookhaven National Laboratory

Differentiable simulation

- Noisy true waveform is taken as the reference in loss minimization.
- Lifetime tuned to converge to true lifetime in true waveform.



How we bridge gaps

```
SCAN_HARDWARE
    ↓
PROBE_INPUTS  ──>  InputManifest  (events + sizes)
    ↓
SELECT_LARGEST event for testing
    ↓
ESTIMATE_RESOURCES  ──>  ResourcePlan  (batch size, device, …)
    ↓
RENDER_CONFIG  ──>  config.yaml  from Jinja2 template
    ↓
TEST_LARGEST
    ↓
success ──>  RUN_FULL ──>  COMPARE_IDS ──>  DONE
    ↓
fail    ──>  PARSE_LOG ──>  rule retry?
                        ──>  LLM_ANALYSIS ──>  APPLY ──>  RETEST

The LLM navigates this graph; the scripts execute each node.
```

Workflow for this package.

```
User message
  ↓
LLMClient.chat_complete(messages, tool_specs)
  ↓
LLM response:  { tool_calls: [ {name, arguments} ] }
  ↓
DISPATCH[name](**arguments, deps)  →  result dict
  ↓
to_tool_message(call_id, result)  →  appended to
messages
  ↓
LLMClient.chat_complete(messages, tool_specs)  ← next
round
  ↓
… until no tool_calls in response

The message list is the only shared state between LLM
and tools.
```

Workflow for the assistant.

Encapsulated in SKILL and MCP for generic AI agent.

Brookhaven National Laboratory

Effective charge

- Charge distribution of Gaussian-smeared track segment after diffusion

$$q(\mathbf{x}) = \frac{Q}{(2\pi)^{3/2}\sigma_x\sigma_y\sigma_z}\int_0^1 e^{-\frac{1}{2}\left(\frac{(x-(x_0+v(x_1-x_0)))^2}{\sigma_x^2}+\frac{(y-(y_0+v(y_1-y_0)))^2}{\sigma_y^2}+\frac{(z-(z_0+v(z_1-z_0)))^2}{\sigma_z^2}\right)}\,\mathrm{d}v$$

Analytic solution: $$q(x,y,z) = Q\frac{\exp(-B/\Delta^2)}{4\pi\Delta}\left(\mathrm{erf}\,\frac{A_2}{\sqrt{2}\Delta\sigma_x\sigma_y\sigma_z} - \mathrm{erf}(\frac{A_1}{\sqrt{2}\Delta\sigma_x\sigma_y\sigma_z})\right)$$

- Ionization charge at x_i is NOT the $q(x_i)$ but $w_i q(x_i) * u_i$ to improve accuracy of induced current on pixel $i(x) * q(x) = w_i q_i * u_i i_i$.
  - w_i is obtained using the Gauss-Legendre quadrature rule, while the field response $i(x)$ (induced current of a single electron) is evaluated via trilinear interpolation u_i.
  - Details in this talk.