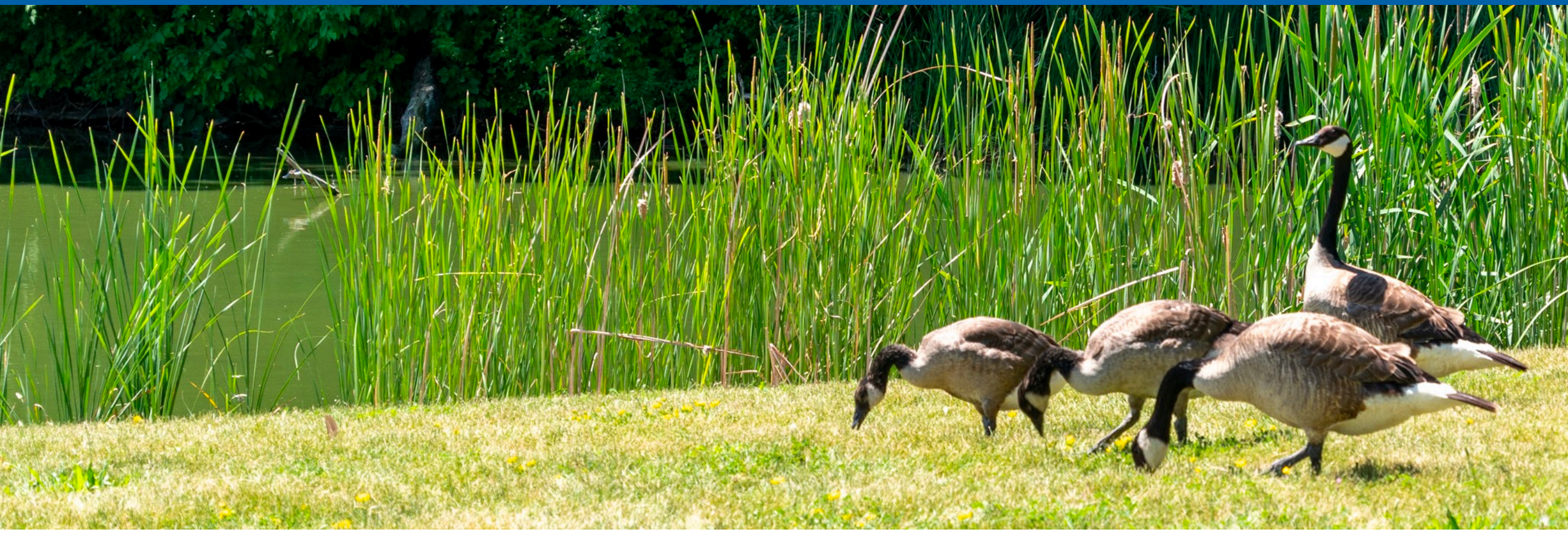


Image credit: Marguerite Tonjes

A Differentiable Simulation for Muon Cooling

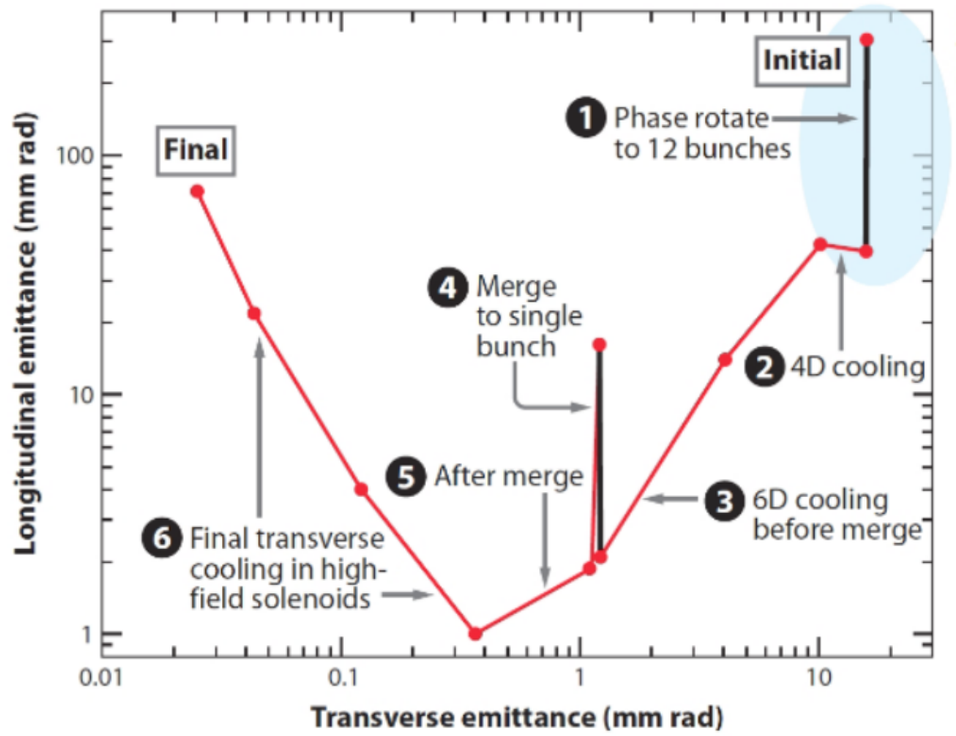
UTK accelerator meeting

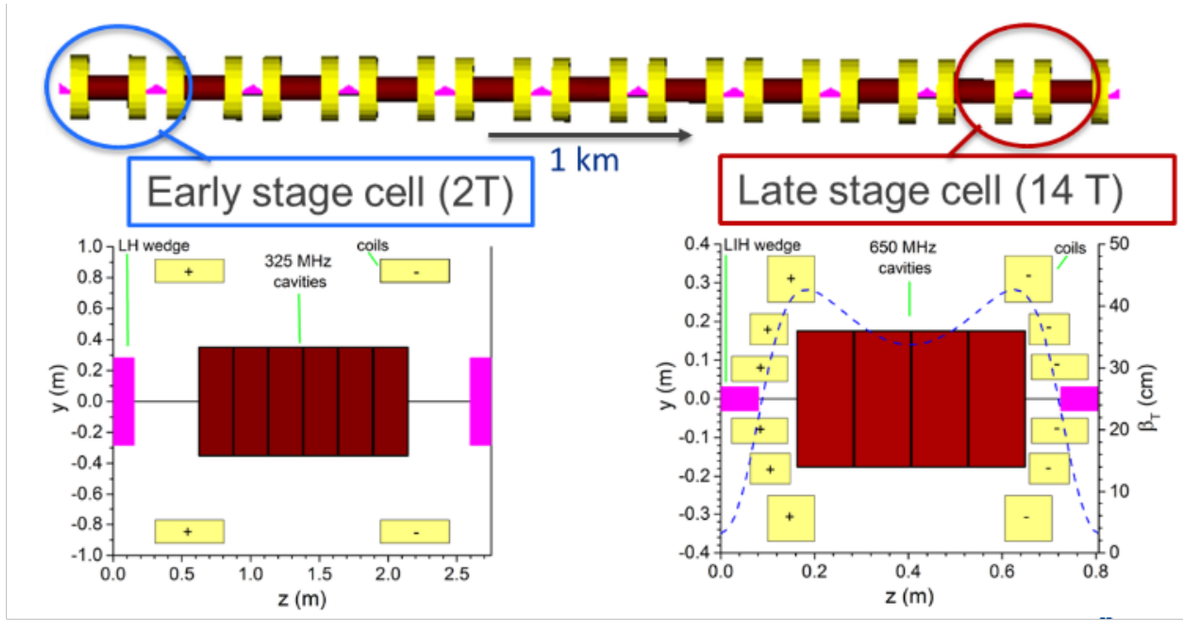
Sergo Jindariani, Nikita Kuklev, Prasanth Shyamsundar, **Nick Smith**, Diktys Stratakis, Nhan Tran (FNAL)
Philip Harris, Andrzej Novak, Holger Witte (MIT)
Eric Anderson, Corrinne Mills (UIC)

October 2, 2026

The challenge

- A muon collider at FNAL would be cool, but the beam must be cooled
- Rectilinear design: 10^5 reduction in emittance through several stages in a 1km channel
  - Conservative design, hand-optimization
- Each of the 12 stages in the rectilinear ionization cooling channel has 20-100 cells, each of which has several tunable parameters
  - Optimal working point may differ in each stage, and improvements *compound*
- Simulation requires atypical mix of robust beam transport optics, material interaction, and collective beam dynamics

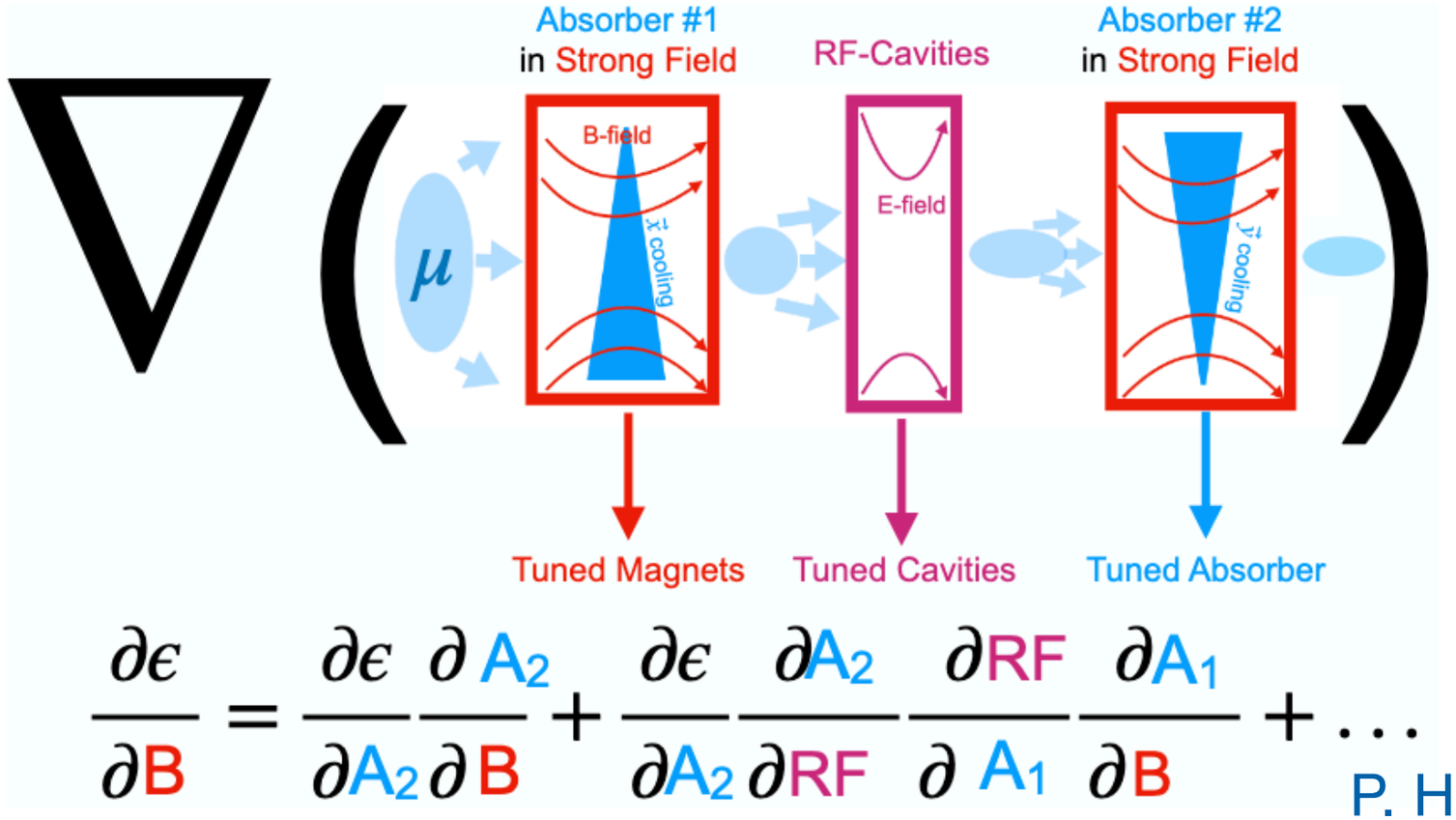




Fermilab

The opportunity

- Large community interested in applying differentiable programming techniques to HEP problems, such as detector design and beam tuning
  - See e.g. MODE workshops
- Differentiable simulation enables efficient high-dimensional optimization
  - For our problem, a reasonable set of building blocks is understood!
    - TM010 pillbox, thick solenoid, wedge absorber, … already a benchmarking activity



P. Harris

Fermilab

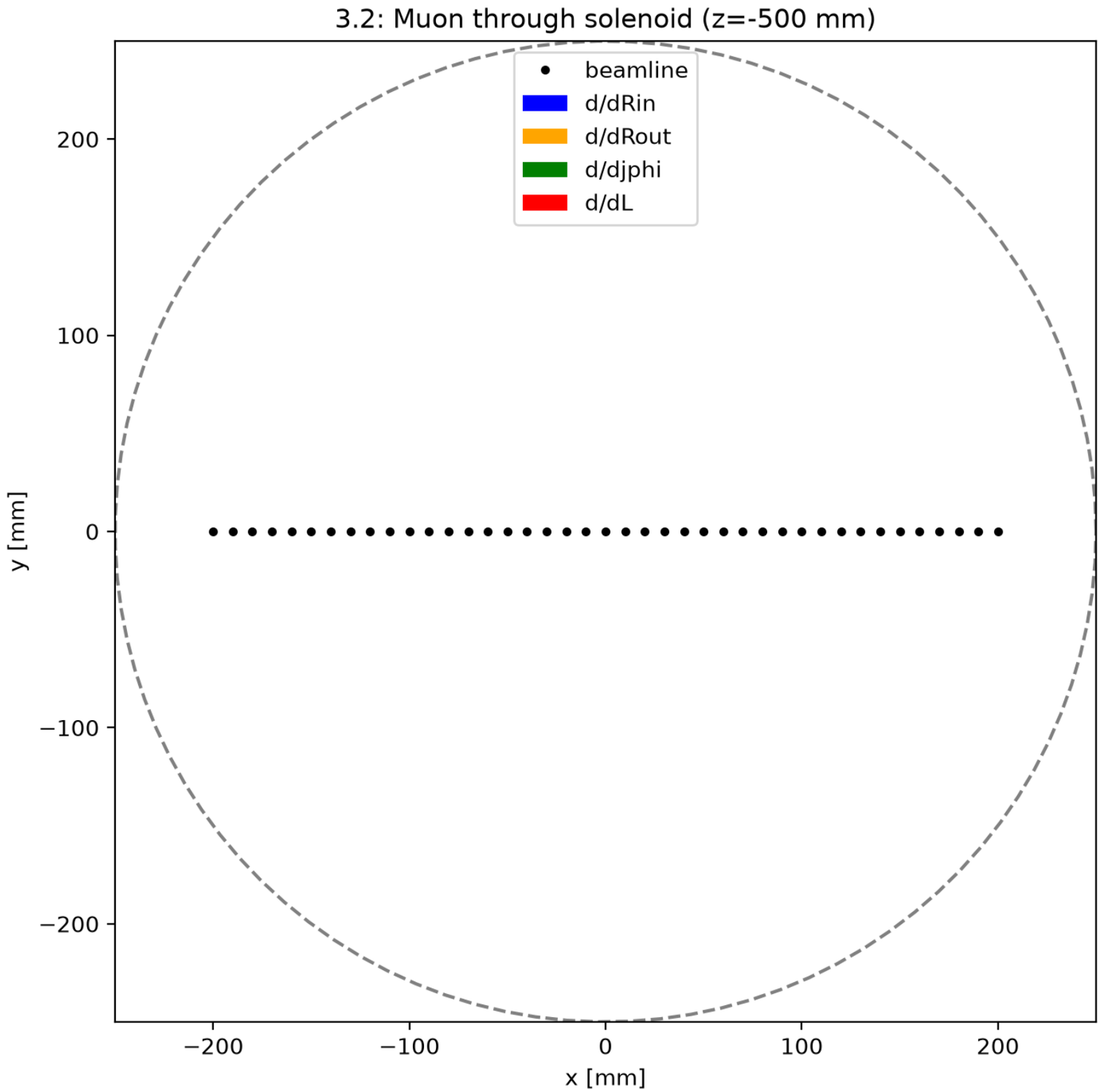
The plan

- There is plenty of prior art in both muon cooling simulators and differentiable beamline simulations to draw on
  - iCOOL, G4Beamline, BDSim, RF-Track, …
  - Cheetah, WarpX, Synergia, Bmad-Julia, …
- We are building a ground-up simulation in Jax
  - Lack of an existing tool at the intersection of all requirements
    - Optics + material interaction (+ collective effects) + ∂
  - Desire to leverage a computational framework with first-class (automatic*) differentiation support and GPU acceleration "for free”
  - **LLM coding tools change the calculus for refactor vs. rewrite**
    - At least for “Minimum Viable Product” stage
    - Once we understand the challenges, maybe we pivot?



* AD is not always correct nor always the fastest!

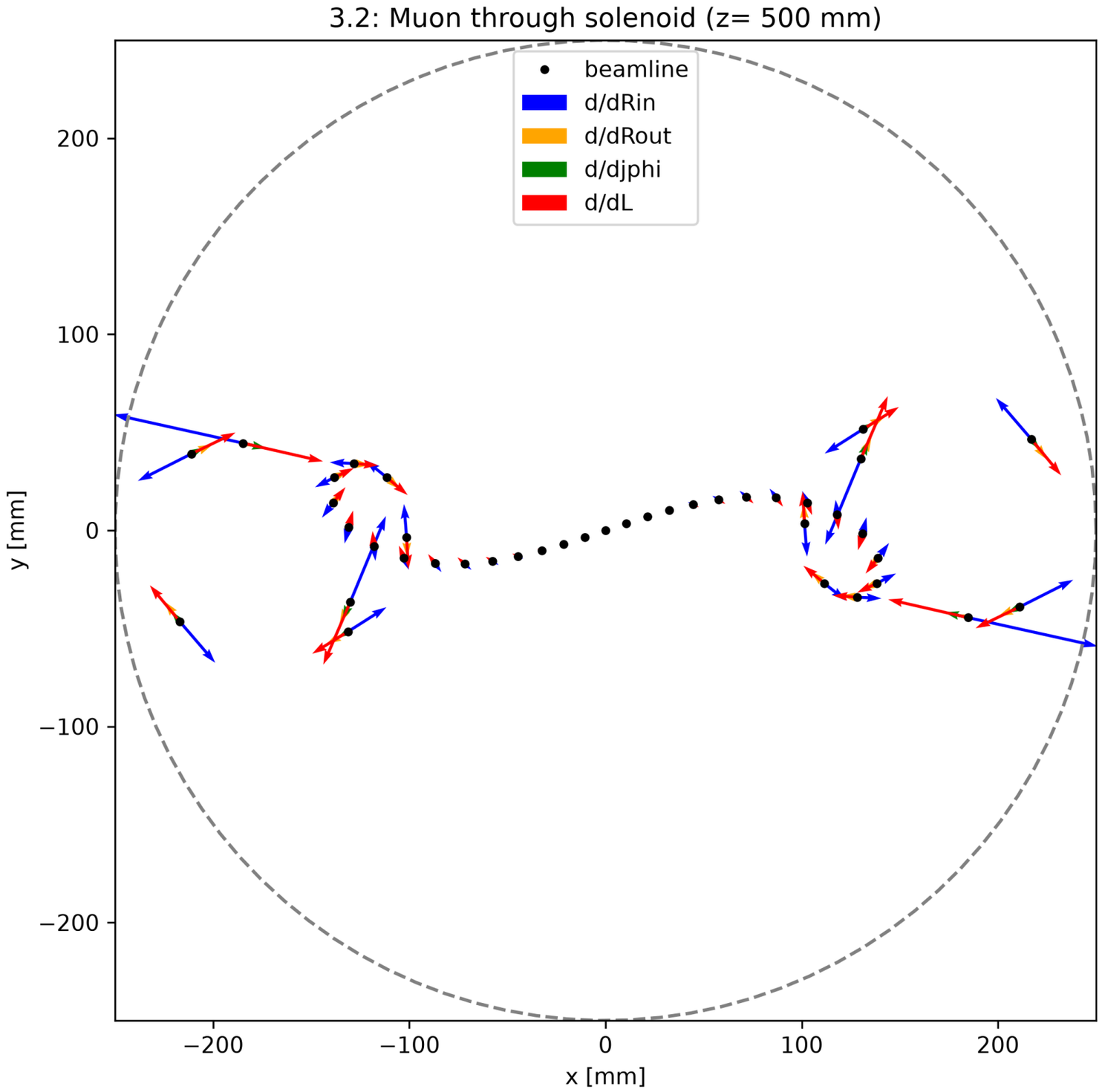
Progress



[Benchmark 3.2: thick solenoid]

Fermilab

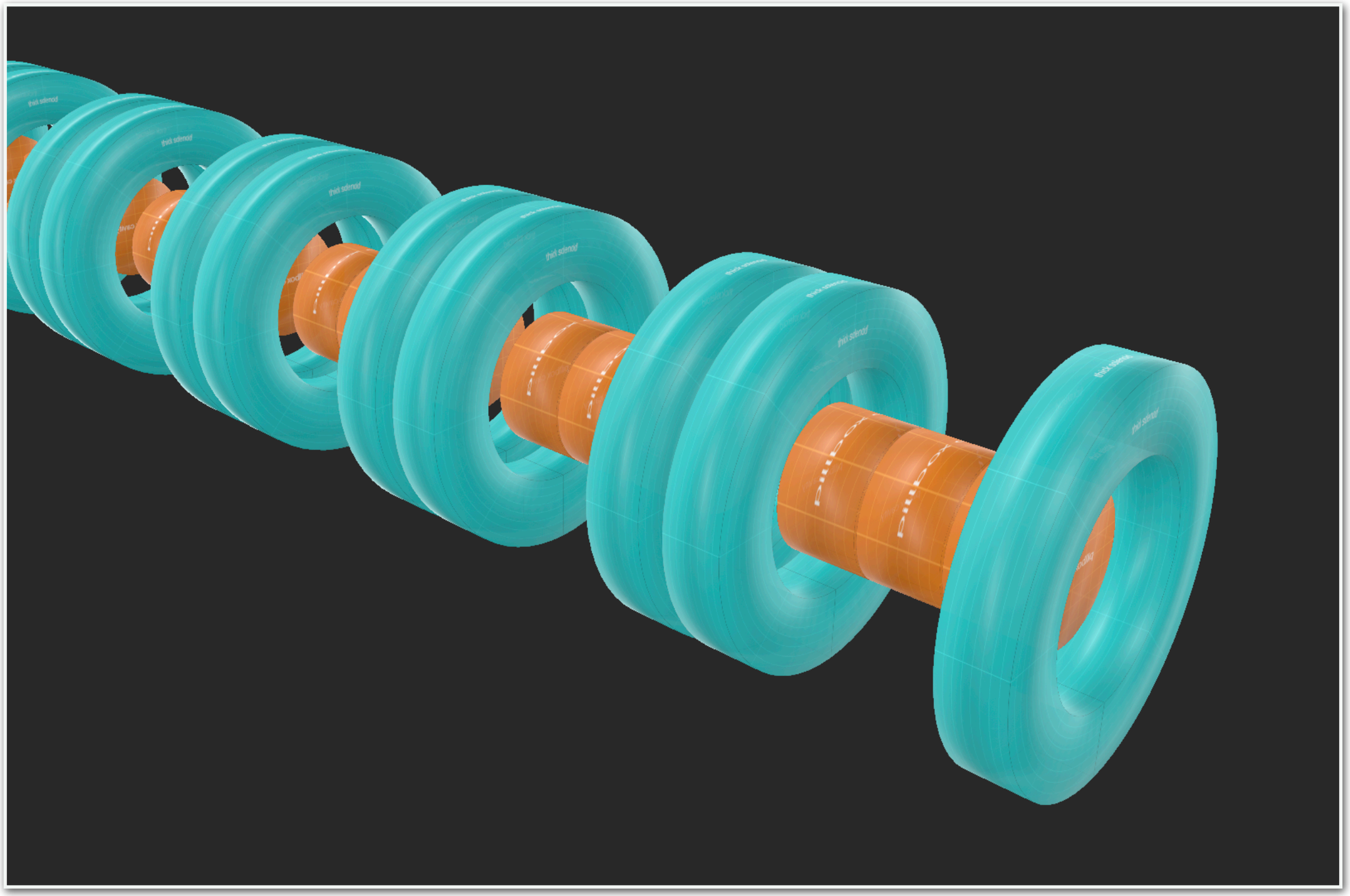
Progress



[Benchmark 3.2: thick solenoid]

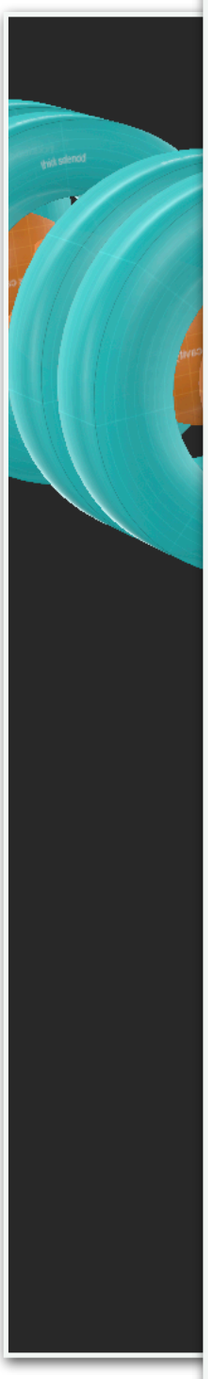
Fermilab

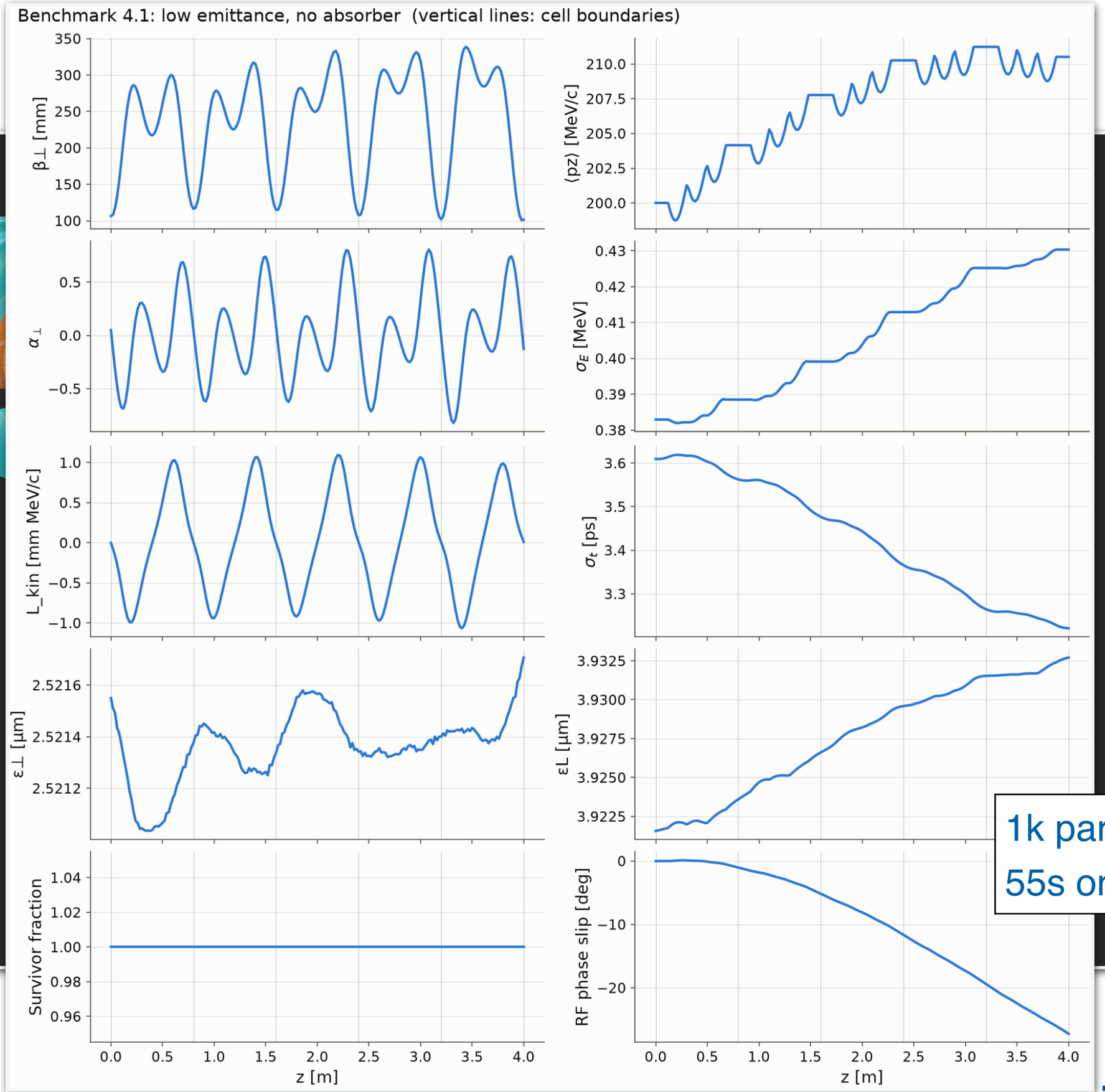
Progress



[Benchmark 4.1: low-emittance channel, no absorber]

Progress



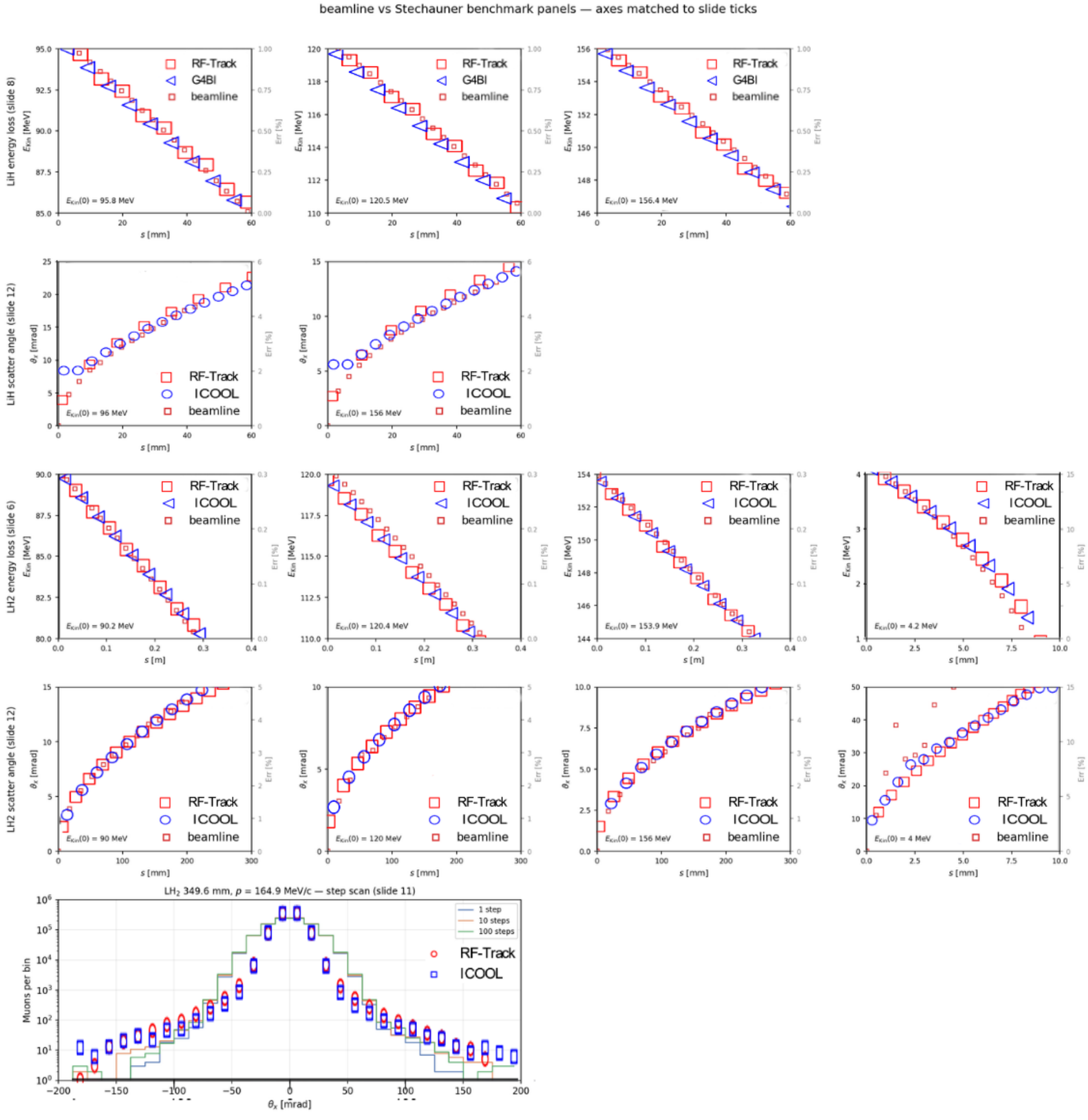


1k particles
55s on one core

Fermilab

Progress

- First look at absorber
- Comparing via plot overlays
- Reaching out to get data for G4Beamline, RF-Track and ICOOL



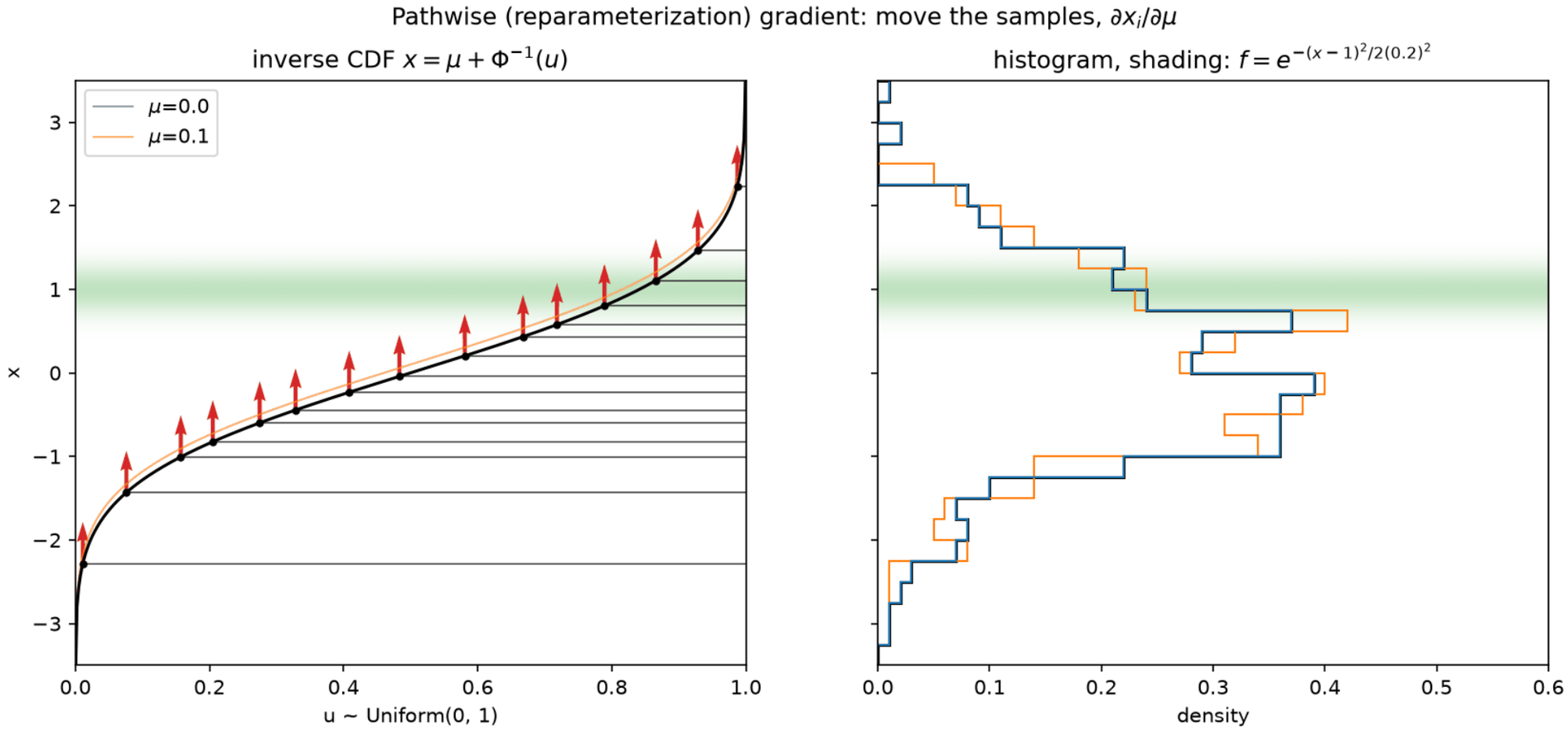
[Benchmark 3.1: absorber]

Fermilab

Stochastic effects

- The hard part is taking gradients through stochastic elements
  - Absorber: energy straggling, multiple scattering
- The objective is emittance, extracted from moments of 6D distribution $p(x, \dot{x})$
  - $\epsilon_\theta(t) \sim \int dx d\dot{x}\, f(x,\dot{x}) p_\theta(x,\dot{x};t) = \int dx d\dot{x}\, f(x,\dot{x}) \int^t dt' \frac{dp_\theta(x,\dot{x})}{dt'}$
  - Sample $(x, \dot{x}) \sim p(x, \dot{x}; t = 0)$ and push forward ⇒ weighted samples $(x_\theta, \dot{x}_\theta, w_\theta)$
  - Two paths for θ gradients: through the final positions, or through the sample weights

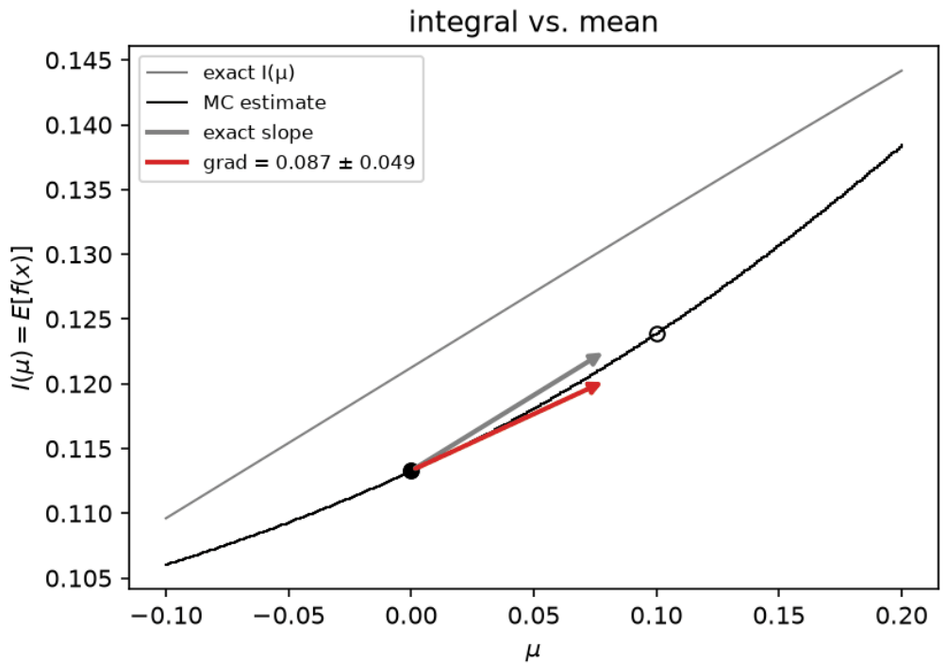
Sample (path) gradients



$$I(\mu) = \int f(x)\, p(x;\mu)\, dx = \int_0^1 f(\mu + \Phi^{-1}(u))\, du$$
$$\hat{I} = \frac{1}{N}\sum_i f(x_i), \quad x_i = \mu + \Phi^{-1}(u_i)$$
$$\partial_\mu \hat{I} = \frac{1}{N}\sum_i f'(x_i)\, \partial_\mu x_i$$

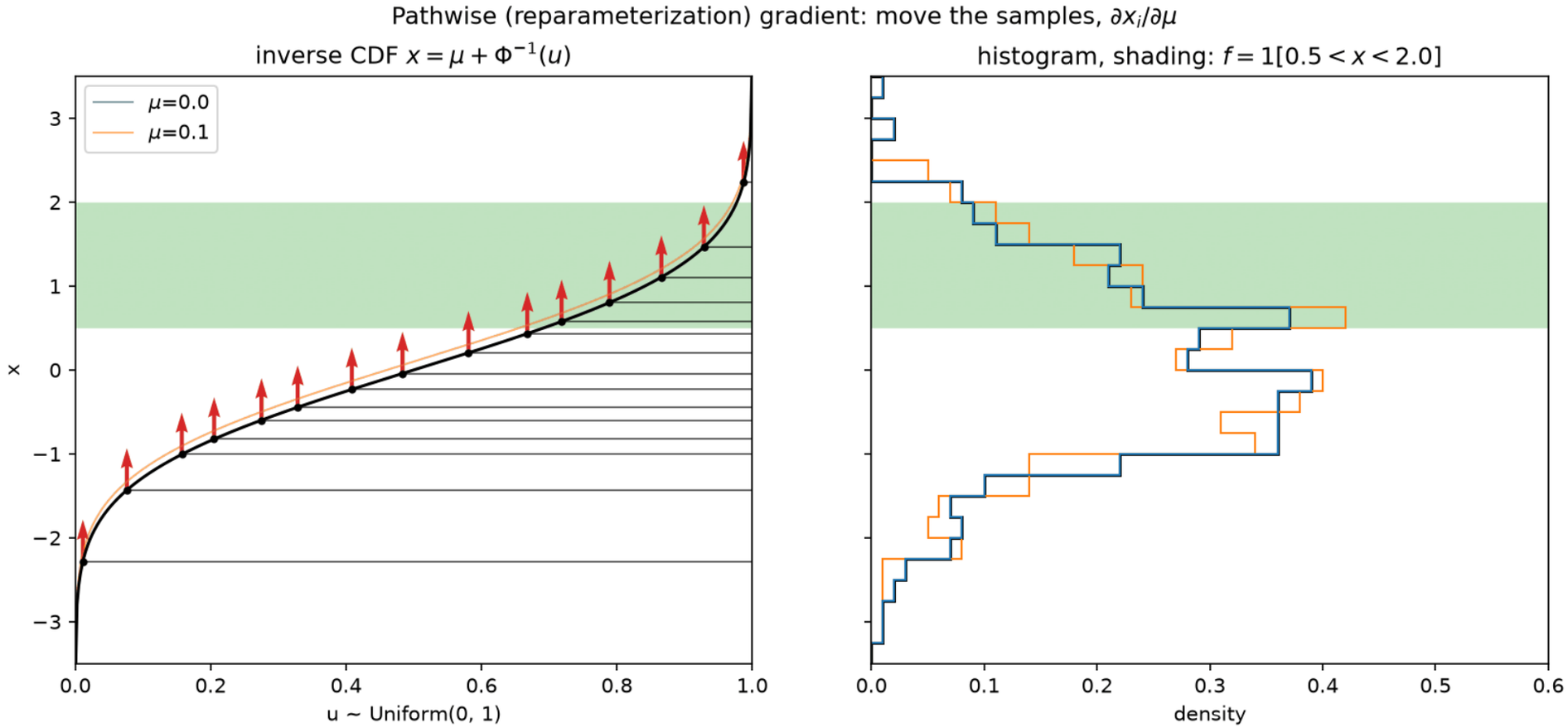
Samples move (red arrows, top left: $\partial x_i/\partial\mu = 1$)
and carry their weight across bin edges (top right).
f is smooth, so each sample contributes $f'(x_i)$: the MC
estimate is smooth in μ and autodiff gives an unbiased slope.
But $f' \sim 1/\text{width}$, so a narrow f means a few
samples near its peak carry large, noisy contributions.

N = 400; arrows show the change for $\Delta\mu = 0.5$



Fermilab

Sample (path) gradients



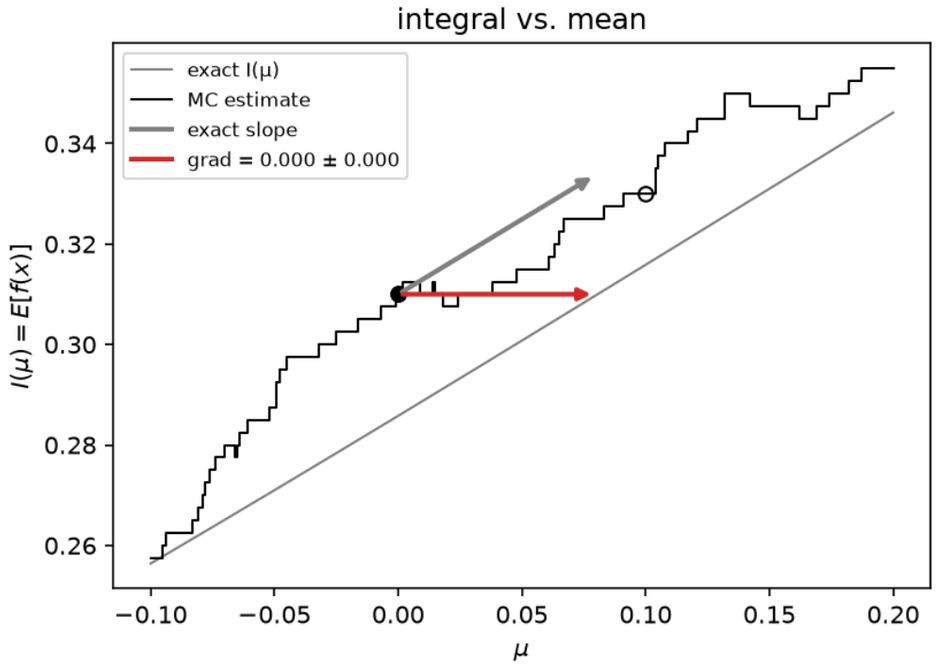
$$I(\mu) = \int f(x)\, p(x;\mu)\, dx = \int_0^1 f(\mu + \Phi^{-1}(u))\, du$$

$$\hat{I} = \frac{1}{N}\sum_i f(x_i), \quad x_i = \mu + \Phi^{-1}(u_i)$$

$$\partial_\mu \hat{I} = \frac{1}{N}\sum_i f'(x_i)\, \partial_\mu x_i$$

Samples move (red arrows, top left: $\partial x_i/\partial\mu = 1$)
and carry their weight across bin edges (top right).
But f is a step: $f'(x_i) = 0$ for every sample, so autodiff
returns exactly 0. The gradient is the flux of samples through
the window edges, $p(a) - p(b)$, only seen once the edges are
smoothed (width h, $O(h^2)$ bias).

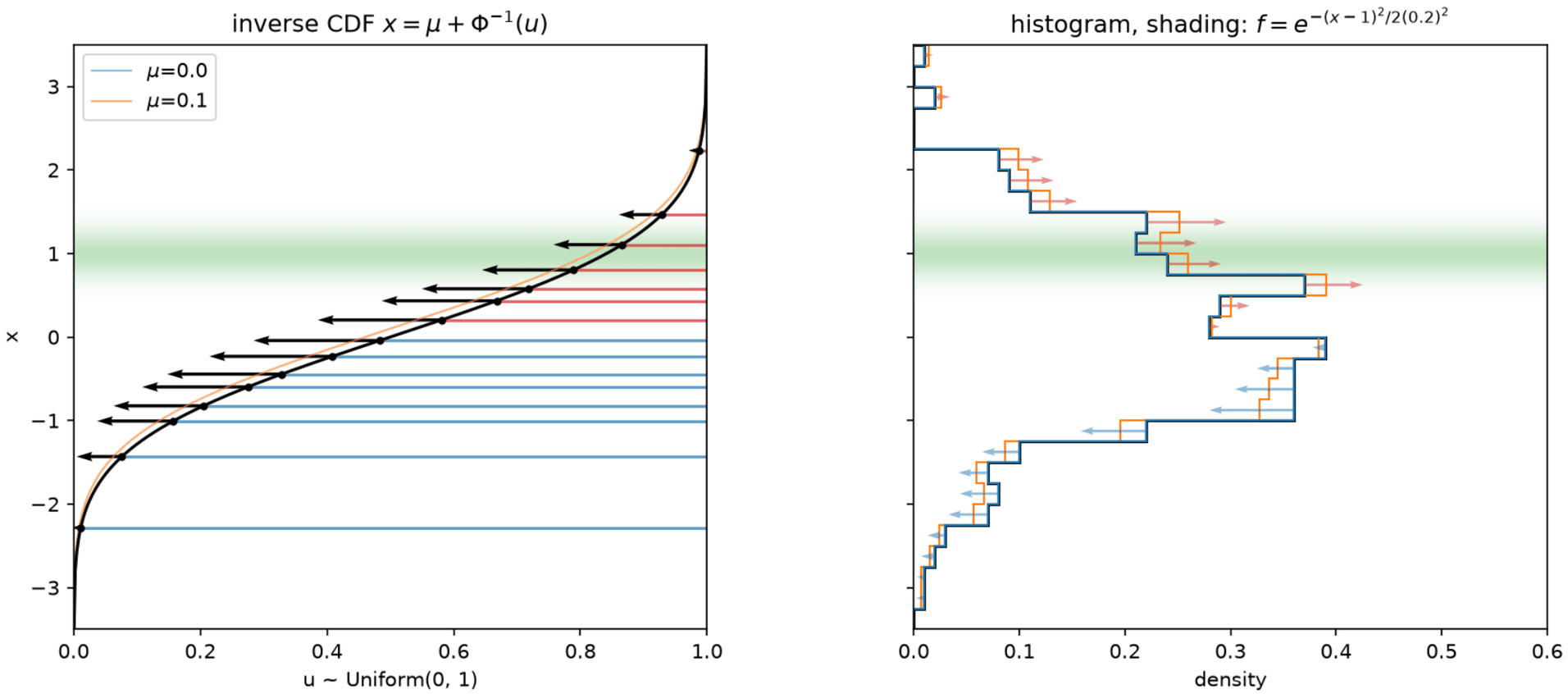
N = 400; arrows show the change for $\Delta\mu = 0.5$



Fermilab

Weight gradients

Weighted (score function) gradient: fix the samples, $\partial w_i/\partial\mu$



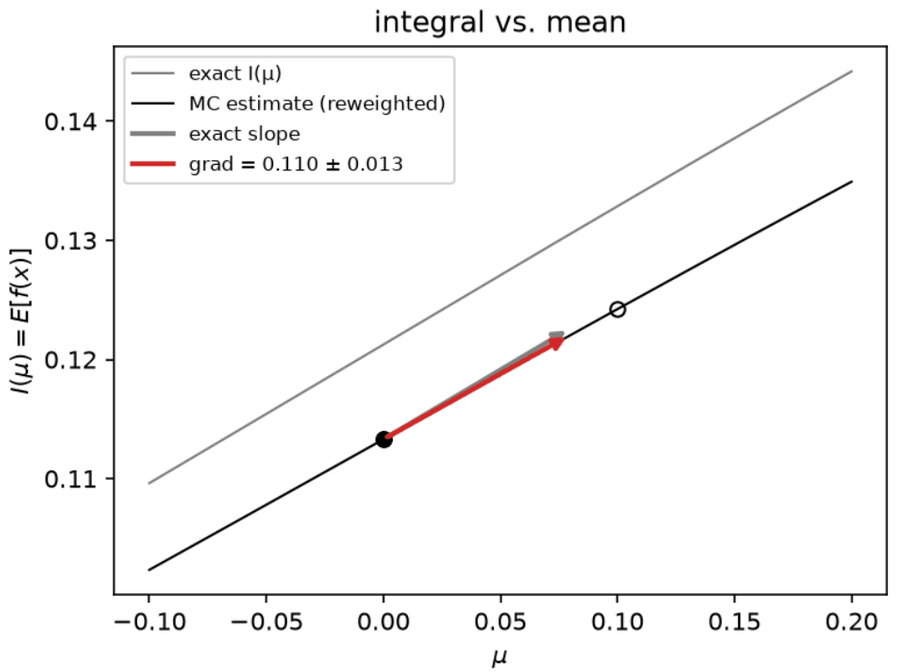
$$I(\mu) = \int f(x)\,\frac{p(x;\mu)}{p(x;\mu_0)}\,p(x;\mu_0)\,dx$$

$$\hat{I} = \frac{1}{N}\sum_i w_i(\mu) f(x_i), \quad x_i \sim p(x;\mu_0)$$

$$\partial_\mu \hat{I} = \frac{1}{N}\sum_i f(x_i)\,\partial_\mu \log p(x_i;\mu) = \frac{1}{N}\sum_i f(x_i)\,\frac{x_i-\mu}{\sigma^2}$$

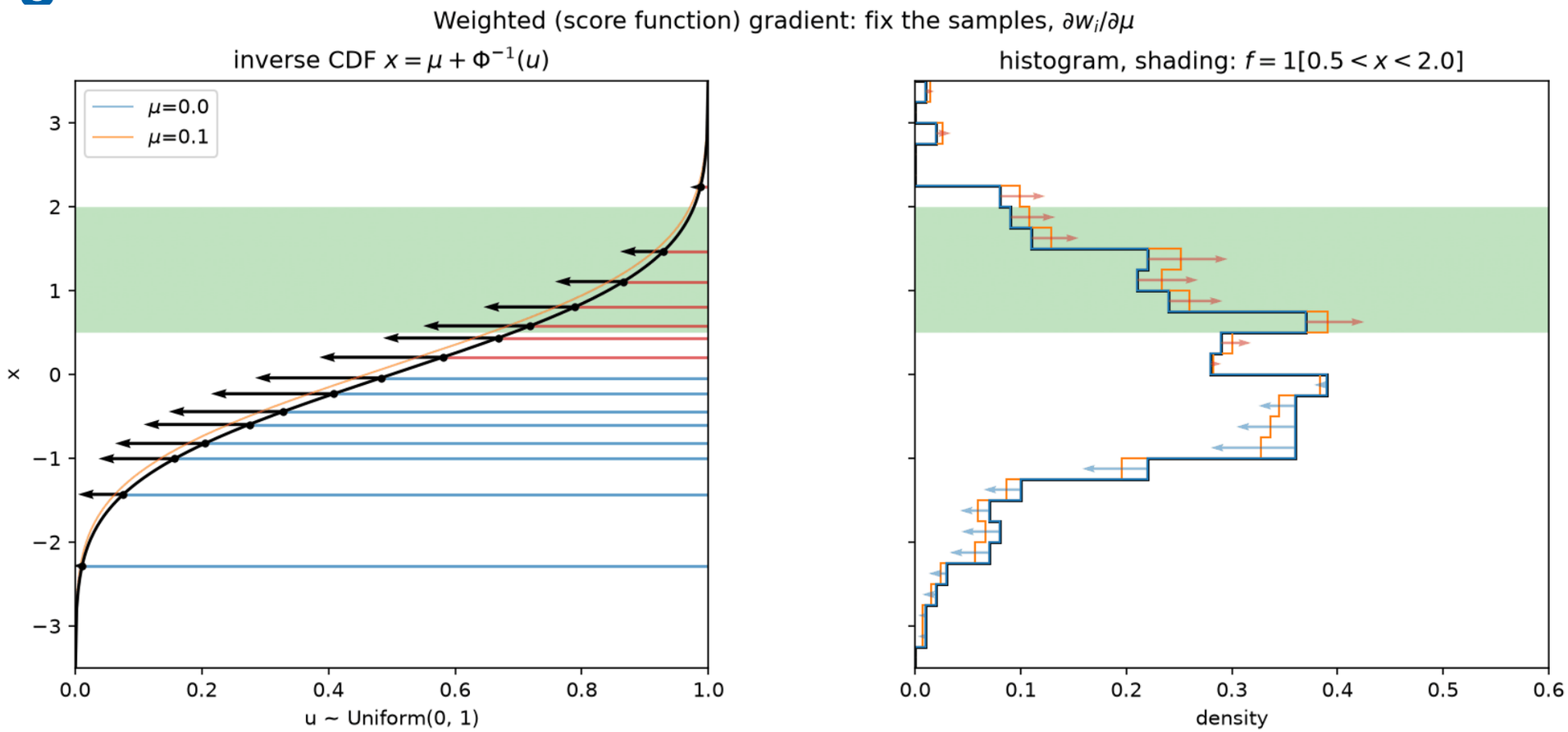
Samples stay put. As the curve shifts, each sample's quantile slides by $\partial u/\partial\mu = -p(x_i)$ (arrows, top left) and its weight is the ratio of curve slopes du/dx (line width, w^4).
Bins change height, not membership: red/blue = weight up/down.
f is never differentiated, so its edges or width don't matter; the noise comes from the score $(x_i-\mu)/\sigma^2$ instead.

N = 400; arrows show the change for $\Delta\mu$ = 0.5 (top left), 0.25 (bins)



Fermilab

Weight gradients



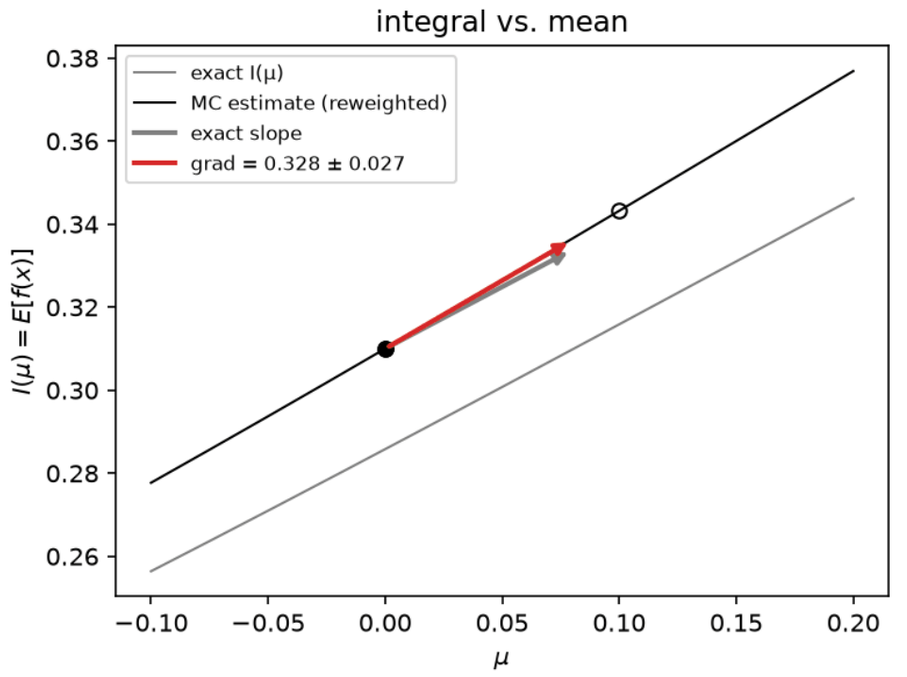
$$I(\mu) = \int f(x)\,\frac{p(x;\mu)}{p(x;\mu_0)}\,p(x;\mu_0)\,dx$$

$$\hat{I} = \frac{1}{N}\sum_i w_i(\mu) f(x_i), \quad x_i \sim p(x;\mu_0)$$

$$\partial_\mu \hat{I} = \frac{1}{N}\sum_i f(x_i)\,\partial_\mu \log p(x_i;\mu) = \frac{1}{N}\sum_i f(x_i)\,\frac{x_i-\mu}{\sigma^2}$$

Samples stay put. As the curve shifts, each sample's quantile slides by $\partial u/\partial\mu = -p(x_i)$ (arrows, top left) and its weight is the ratio of curve slopes du/dx (line width, w^4). Bins change height, not membership: red/blue = weight up/down. f is never differentiated, so its edges or width don't matter; the noise comes from the score $(x_i-\mu)/\sigma^2$ instead.

N = 400; arrows show the change for $\Delta\mu$ = 0.5 (top left), 0.25 (bins)



The package itself

- https://github.com/nsmith-/beamline

Getting Started

This package is pip-installable. We recommend using uv for virtual environment management. With it, you can set up the environment and install dependencies using:

```
uv sync
```

All results are curated through the test infrastructure while we are in development mode. The eventual goal is a library and command-line interface. To produce all test artifacts, you can run:

```
uv run pytest
```

which will produce a `test_artifacts` directory containing figures and data files from the test results. This may take a few minutes.

Building a cell

```python
def coil(z: float, jphi: float) -> EMTensorField:
    return TransformEMField(
        transform=Transform.make_translation(z=z),
        field=ThickSolenoid(
            Rin=COIL_RIN,
            Rout=COIL_ROUT,
            jphi=jphi,
            L=COIL_LENGTH,
        ),
    )
```

Building benchmark 4.1:

- A thick solenoid
- An RF cavity
- A sum of two coils and 3 cavities

```python
def cavity(index: int) -> EMTensorField:
    return TransformEMField(
        transform=Transform.make_translation(z=CELL_LENGTH / 2 + index * RF_PITCH),
        field=PillboxCavity(
            length=RF_LENGTH,
            frequency=RF_FREQUENCY,
            E0=RF_E0,
            mode="TM",
            m=0,
            n=1,
            p=0,
            phase=phase_center + index * jnp.pi,
        ),
    )
```

```python
return SumField(
    [
        coil(COIL_Z, COIL_JPHI),
        coil(CELL_LENGTH - COIL_Z, -COIL_JPHI),
        cavity(-1),
        cavity(0),
        cavity(1),
    ]
)
```

Building a scene

```python
def place(index: jax.Array) -> EMTensorField:
    z0 = index * CELL_LENGTH
    return TransformEMField(
        transform=Transform.make_translation(z=z0, ct=z0 / BETA_REF),
        field=setup_cell(),
    )

indices = jnp.arange(-num_pad, num_cells + num_pad)
return StackedField(fields=eqx.filter_vmap(place)(indices))
```

```python
num_cells = 5
scene = setup_scene(num_cells=num_cells)
num_particles = 1000
initial_particles = sample_particles(
    rng=jax.random.PRNGKey(0), num_particles=num_particles
)
zvals = jnp.linspace(0.0, num_cells * CELL_LENGTH, num_cells * 40 + 1)

aperture = BeamPipe(radius=BEAM_PIPE_RADIUS)

@eqx.filter_jit
@eqx.filter_vmap(in_axes=(None, 0, None))
def track(field, start, cts):
    return diffrax_solve(field, start, cts, aperture=aperture)

trajectories, stats = track(scene, initial_particles, zvals)
```

Building benchmark 4.1:

- Tiling several cells
- Running the tracking

Fermilab

Derivatives

```python
@jax.jit
def run(fieldobj: ThickSolenoid, xstart: SFloat) -> MuonStateDz:
    start = MuonStateDz.make(
        position=Cartesian4.make(x=xstart, z=-500.0 * u.mm),
        momentum=Cartesian3.make(z=200 * u.MeV),
        q=1,
    )
    sol, _ = diffrax_solve(fieldobj, start, zs, forward_mode=True)
    return sol

track: MuonStateDz = jax.vmap(run, in_axes=(None, 0))(SOLENOID, xpos)
end: MuonStateDz = jax.tree.map(lambda x: x[:, -1], track)
# TODO: understand why forward_mode=True fails here (when using jacfwd) at x=0.0
grad: MuonStateDz = jax.vmap(jax.jacfwd(run), in_axes=(None, 0))(SOLENOID, xpos)
assert track.kin.p.x.shape == (len(xpos), len(zs))

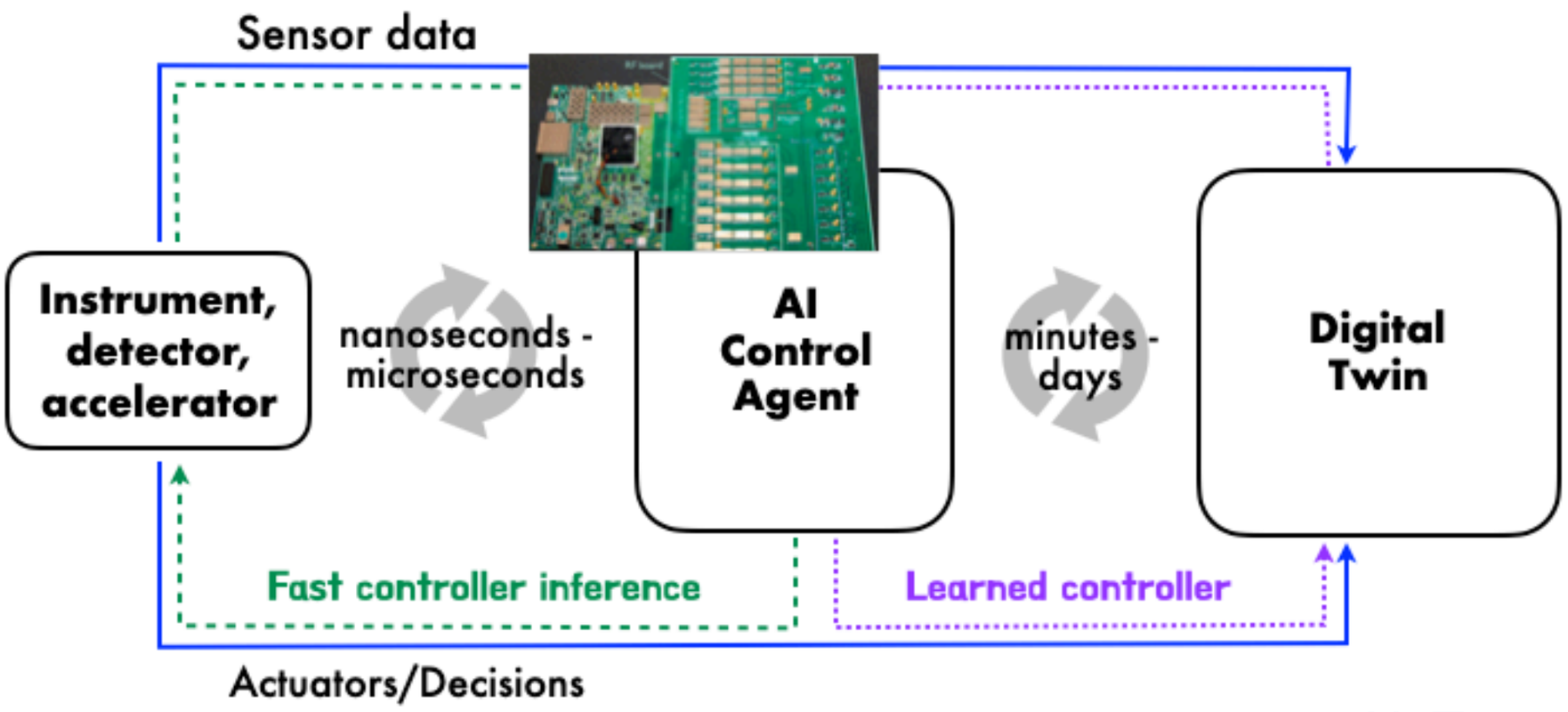
def extract( ...
    )

grad_Rin = extract(grad, lambda f: f.Rin)
grad_Rout = extract(grad, lambda f: f.Rout)
grad_jphi = extract(grad, lambda f: f.jphi)
grad_L = extract(grad, lambda f: f.L)
```

Gradients for benchmark 3.2
This generates the arrow animation from slide 5

Future tie-ins

- A differentiable simulator becomes a very useful component of a digital twin for online beam controls
  - Easier to train NN to capture delta between simulation and reality than a black box!
  - To do so effectively, “classical component” needs high-dimensional gradients
  - Here we can tie in with MOAT project



N. Tran

Summary

- A differentiable simulator for muon cooling channels is being developed
- Stay tuned for future updates