Neural Network-Based Fast Track Extrapolation

Machine Learning for LHCb Track Reconstruction

G. Scriven

December 20, 2025

Abstract

This report documents the development and evaluation of a neural network-based approach for accelerating track extrapolation in the LHCb detector. We present a **data-driven Multi-Layer Perceptron (MLP)** trained on output from the traditional 4th-order Runge-Kutta integrator to learn the mapping from initial to final track states through the LHCb magnetic field. Our implementation achieves a $\sim$155$\times$ speedup compared to the reference RK4 method, with a mean radial position error of 6.97 mm. We discuss the theoretical foundations, clarify the distinction between our current data-driven approach and true Physics-Informed Neural Networks (PINNs), and propose extensions to incorporate physics constraints for improved accuracy.

Contents

1 Introduction

1.1 Motivation

Track extrapolation is a computationally intensive component of particle physics reconstruction at the LHCb experiment. The standard method uses Runge-Kutta numerical integration to solve the equations of motion for charged particles in the magnetic field. While highly accurate, this approach requires multiple field evaluations and iterations, making it a bottleneck in real-time trigger applications where millions of tracks must be processed per second.

Machine learning approaches offer the potential to dramatically accelerate track extrapolation by learning the transformation from initial to final track states. This work explores using neural networks to achieve this acceleration while maintaining sufficient accuracy for physics applications.

1.2 LHCb Detector and Magnetic Field

The LHCb detector features a large warm dipole magnet that provides bending power for momentum measurement. Key characteristics include:

- Integrated field: $\int B_y \, dz \approx 4$ T·m
- Peak field strength: $|B_y|_{\max} \approx 1.1$ T
- Effective length: ∼4 m (from $z \approx 3000$ mm to $z \approx 7000$ mm)
- Field orientation: Primarily vertical (B_y), causing horizontal (x) bending
- Polarity: Reversible (MagUp/MagDown) for systematic studies

2 Theoretical Foundations

2.1 Equations of Motion in a Magnetic Field

A charged particle moving through a magnetic field experiences the Lorentz force:

$$\boldsymbol{F} = q(\boldsymbol{v} \times \boldsymbol{B}) \tag{1}$$

where q is the particle charge, $\boldsymbol{v}$ is the velocity vector, and $\boldsymbol{B}$ is the magnetic field vector.

For relativistic particles, the equation of motion is:

$$\frac{d\boldsymbol{p}}{dt} = q(\boldsymbol{v} \times \boldsymbol{B}) \tag{2}$$

where the relativistic momentum is $\boldsymbol{p} = \gamma m \boldsymbol{v}$ with Lorentz factor $\gamma = 1/\sqrt{1 - v^2/c^2}$.

2.2 Track State Parameterization

In LHCb, tracks are parameterized at fixed z-planes using the state vector:

$$\boldsymbol{s} = (x, y, t_x, t_y, q/p)^T \tag{3}$$

where:

- x, y: Transverse position coordinates (mm)
- $t_x \equiv dx/dz$, $t_y \equiv dy/dz$: Track direction tangents
- q/p: Signed inverse momentum (MeV^{-1}), encodes both charge and momentum

The unit direction vector is:

$$\hat{\boldsymbol{n}} = \frac{1}{\sqrt{1 + t_x^2 + t_y^2}} \begin{pmatrix} t_x \\ t_y \\ 1 \end{pmatrix} \tag{4}$$

2.3 Equations of Motion in Track Parameters

Converting the Lorentz equation to track parameters with z as the independent variable:

$$\frac{dx}{dz} = t_x \tag{5}$$

$$\frac{dy}{dz} = t_y \tag{6}$$

$$\frac{dt_x}{dz} = \kappa\sqrt{1 + t_x^2 + t_y^2}\left[t_x t_y B_x - (1 + t_x^2)B_y + t_y B_z\right] \tag{7}$$

$$\frac{dt_y}{dz} = \kappa\sqrt{1 + t_x^2 + t_y^2}\left[(1 + t_y^2)B_x - t_x t_y B_y - t_x B_z\right] \tag{8}$$

$$\frac{d(q/p)}{dz} = 0 \quad \text{(no energy loss in vacuum)} \tag{9}$$

where $\kappa = c \cdot (q/p)$ with $c = 299.792458$ mm/ns is the speed of light.

For the LHCb dipole with dominant B_y component and small B_x, B_z, the bending is primarily in the x-direction:

$$\frac{dt_x}{dz} \approx -\kappa\sqrt{1 + t_x^2 + t_y^2} \cdot B_y \tag{10}$$

2.4 Bending Angle and Momentum Measurement

The total bending angle in the x-z plane is:

$$\Delta t_x = t_x^{\text{final}} - t_x^{\text{initial}} \approx -c \cdot \frac{q}{p} \int_{z_i}^{z_f} B_y(z)\, dz \tag{11}$$

For the LHCb magnet with $\int B_y\, dz \approx -4$ T·m (for MagDown):

$$\Delta t_x \approx \frac{1.2 \text{ GeV/c}}{p} \cdot \text{sign}(q) \tag{12}$$

This "p_T kick" is the basis for momentum measurement in LHCb.

2.5 Runge-Kutta Integration

The standard numerical approach solves the system of ODEs (5)–(9) using Runge-Kutta methods. The classical 4th-order RK4 method for a single step:

$$\boldsymbol{k}_1 = h \cdot f(z_n, \boldsymbol{s}_n) \tag{13}$$

$$\boldsymbol{k}_2 = h \cdot f(z_n + h/2, \boldsymbol{s}_n + \boldsymbol{k}_1/2) \tag{14}$$

$$\boldsymbol{k}_3 = h \cdot f(z_n + h/2, \boldsymbol{s}_n + \boldsymbol{k}_2/2) \tag{15}$$

$$\boldsymbol{k}_4 = h \cdot f(z_n + h, \boldsymbol{s}_n + \boldsymbol{k}_3) \tag{16}$$

$$\boldsymbol{s}_{n+1} = \boldsymbol{s}_n + \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4) + O(h^5) \tag{17}$$

Each step requires 4 magnetic field evaluations. With adaptive step sizing, a typical extrapolation through the LHCb magnet requires ∼50–200 steps.

3 Neural Network Approaches

3.1 Data-Driven Neural Networks (Current Implementation)

Our **current implementation** uses a purely data-driven approach:

Model: Multi-Layer Perceptron (MLP)

$$\hat{\boldsymbol{s}}_{\text{final}} = f_\theta(\boldsymbol{s}_{\text{initial}}, \Delta z) \tag{18}$$

where f_θ is a neural network with parameters θ.

Loss Function: Pure supervised learning with Mean Squared Error

$$\mathcal{L}_{\text{data}} = \frac{1}{N}\sum_{i=1}^{N} \left\|\hat{\boldsymbol{s}}_i - \boldsymbol{s}_i^{\text{RK4}}\right\|^2 \tag{19}$$

This approach learns to mimic the RK4 integrator output without explicit knowledge of the underlying physics equations.

3.2 Physics-Informed Neural Networks (PINNs)

A **true PINN** incorporates the governing physics equations directly into the loss function. For track extrapolation, this would be:

Total Loss:

$$\mathcal{L}_{\text{PINN}} = \mathcal{L}_{\text{data}} + \lambda_{\text{phys}}\mathcal{L}_{\text{physics}} \tag{20}$$

Physics Loss (Lorentz Force Residual):

$$\mathcal{L}_{\text{physics}} = \frac{1}{N_c}\sum_{j=1}^{N_c} \left\|\mathcal{R}(\boldsymbol{s}(z_j))\right\|^2 \tag{21}$$

where the residual $\mathcal{R}$ enforces the equations of motion at collocation points:

$$\mathcal{R} = \begin{pmatrix} \frac{\partial x}{\partial z} - t_x \\ \frac{\partial y}{\partial z} - t_y \\ \frac{\partial t_x}{\partial z} - \kappa\sqrt{1+t_x^2+t_y^2}[t_x t_y B_x - (1+t_x^2)B_y + t_y B_z] \\ \frac{\partial t_y}{\partial z} - \kappa\sqrt{1+t_x^2+t_y^2}[(1+t_y^2)B_x - t_x t_y B_y - t_x B_z] \end{pmatrix} \tag{22}$$

The physics loss requires computing gradients of the network output with respect to z, typically using automatic differentiation.

3.3 Comparison: Data-Driven vs PINN

Table 1: Comparison of neural network approaches

Aspect	Data-Driven MLP	True PINN
Training data required	Large amount	Can be smaller
Physics knowledge	Implicit (in data)	Explicit (in loss)
Generalization	Limited to training distribution	Better extrapolation
Training complexity	Simple (standard backprop)	Complex (autodiff for physics)
Interpretability	Black box	Physics-constrained
Our implementation	✓	Future work

Important Clarification: Our current model is a **data-driven MLP**, not a true PINN. We use the term "physics-informed" loosely to indicate that the training data comes from physics simulations, but the network itself does not explicitly enforce physics constraints during training.

4 Implementation

4.1 Network Architecture

We use a fully-connected feedforward network:

$$\text{Input} : (x, y, t_x, t_y, q/p, \Delta z) \in \mathbb{R}^6 \to \text{Output} : (x', y', t'_x, t'_y) \in \mathbb{R}^4 \tag{23}$$

Architecture:

- Input layer: 6 neurons
- Hidden layer 1: 128 neurons + tanh activation
- Hidden layer 2: 128 neurons + tanh activation
- Hidden layer 3: 64 neurons + tanh activation
- Output layer: 4 neurons (linear)

Total trainable parameters:

$$N_{\text{params}} = 6 \times 128 + 128 + 128 \times 128 + 128 + 128 \times 64 + 64 + 64 \times 4 + 4 = 26,180 \tag{24}$$

4.2 Input/Output Normalization

All inputs and outputs are standardized:

$$\tilde{x} = \frac{x - \mu_x}{\sigma_x} \tag{25}$$

where μ_x and σ_x are computed from training data. This ensures all features have similar scales for stable training.

4.3 Training Data

Training data generated from C++ RK4 reference extrapolator:

Table 2: Training data parameter grid

Parameter	Range	Grid Points	Physical Meaning
x (mm)	$[-900, 900]$	6	Horizontal position
y (mm)	$[-750, 750]$	6	Vertical position
t_x	$[-0.3, 0.3]$	6	Horizontal slope
t_y	$[-0.25, 0.25]$	6	Vertical slope
q/p (MeV^{-1})	$\pm 4 \times 10^{-4}$	2	Corresponds to $p = 2.5$ GeV/c
Δz (mm)	4000	1	Fixed extrapolation distance

Total samples: $6^4 \times 2 = 2592$ grid points, reduced to 1,210 after filtering invalid tracks.

4.4 Training Procedure

- Loss: Mean Squared Error (Eq. 19)
- Optimizer: Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$)
- Initial learning rate: 10^{-3}

- Scheduler: ReduceLROnPlateau (factor=0.5, patience=50)
- Batch size: 64
- Epochs: 2000
- Train/validation split: 80%/20%

5 Results

5.1 Accuracy Metrics

Table 3: Neural network extrapolation accuracy

Metric	Mean	Std Dev	95th Percentile
Radial error $\sqrt{\Delta x^2 + \Delta y^2}$ (mm)	6.97	8.86	23.64
x position error (mm)	1.14	8.19	–
y position error (mm)	-0.41	4.95	–
t_x slope error	0.0009	0.058	–
t_y slope error	0.0006	0.007	–

5.2 Error Distribution

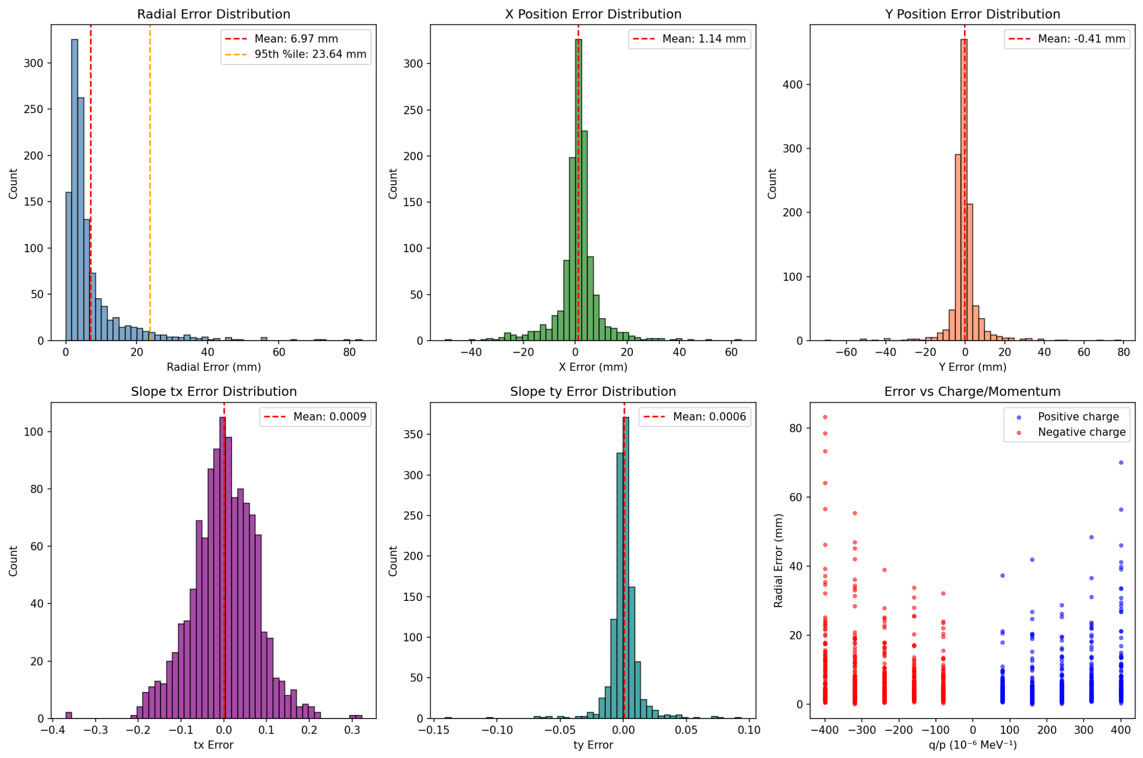

Figure 1: Distribution of extrapolation errors comparing neural network predictions to C++ RK4 reference.

5.3 Phase Space Analysis

Error correlations with input parameters:

- Correlation with y: $r = 0.21$ (strongest)
- Correlation with t_y: $r = 0.21$
- Correlation with q/p: $r = -0.10$
- Correlation with x, t_x: $r \approx -0.03$ (weak)

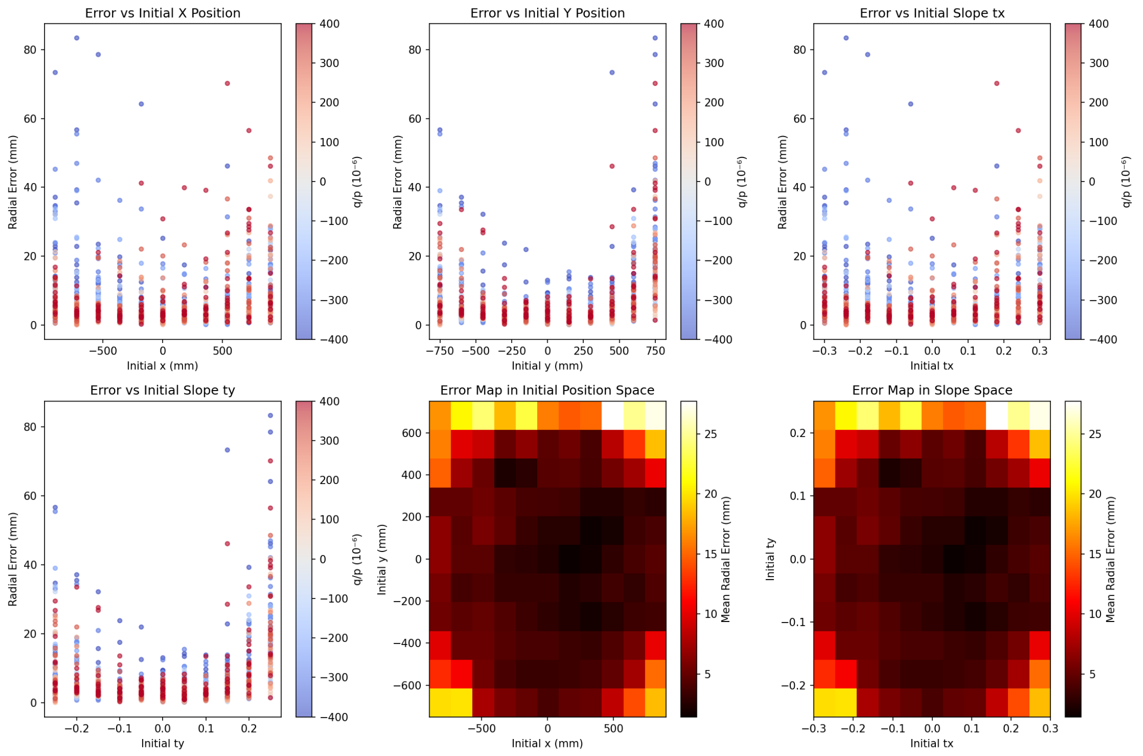

Figure 2: Phase space analysis showing error correlations and 2D error heatmaps.

5.4 Timing Performance

Table 4: Computational performance comparison

Method	Time per track	Speedup
C++ RK4 (Reference)	189.0 μs	1×
Neural Network (PyTorch, batched)	1.2 μs	155×

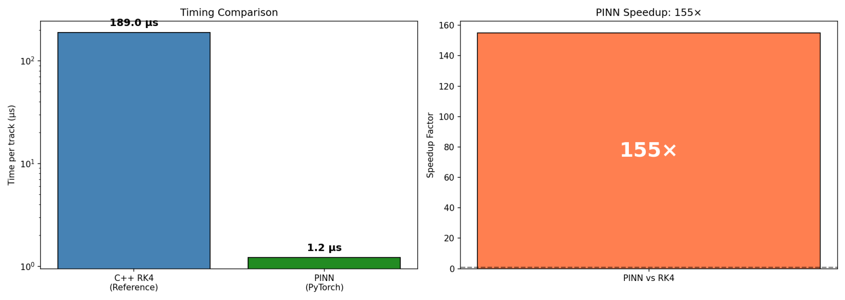

Figure 3: Timing comparison between C++ RK4 and neural network extrapolation.

6 Physics Analysis

6.1 Magnetic Field Profile

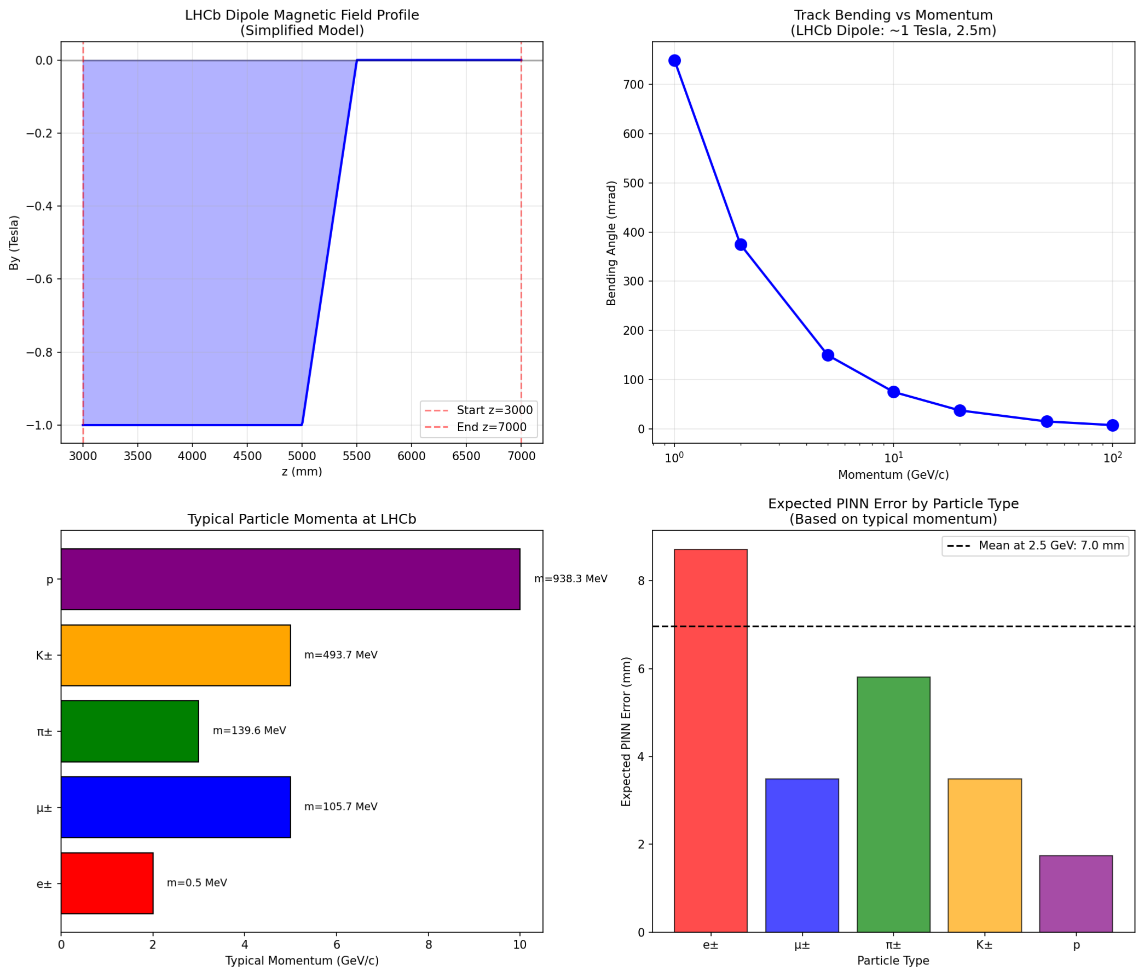

Figure 4: LHCb magnetic field profile, bending vs momentum, and expected errors by particle type.

6.2 Momentum Dependence

The bending angle scales inversely with momentum (Eq. 11):

$$|\Delta t_x| \propto \frac{1}{p} \tag{26}$$

Our model was trained at $p = 2.5$ GeV/c. Expected behavior at other momenta:

- $p = 1$ GeV/c: 2.5× larger bending, likely **worse** accuracy
- $p = 10$ GeV/c: 4× smaller bending, possibly **better** accuracy
- $p = 100$ GeV/c: Nearly straight tracks, should be **very good**

6.3 Particle Types at LHCb

Table 5: Typical particle momenta at LHCb

Particle	Mass (MeV/c^2)	Typical p (GeV/c)	Bending (mrad)
$e^{\pm}$	0.511	1–5	240–1200
$\mu^{\pm}$	105.7	2–100	12–600
$\pi^{\pm}$	139.6	1–50	24–1200
$K^{\pm}$	493.7	2–100	12–600
p	938.3	5–200	6–240

7 True Physics-Informed Neural Networks

7.1 Fundamentals of PINNs

Physics-Informed Neural Networks (PINNs), introduced by Raissi et al. (2019), embed governing physical laws directly into the neural network training process. Unlike standard data-driven networks that only minimize prediction error, PINNs add physics residual terms to the loss function.

Key Insight: Rather than viewing the neural network as a black-box function approximator, PINNs treat it as a *mesh-free solution* to the underlying differential equation.

7.1.1 General PINN Framework

For a system governed by:

$$\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}, t) = 0, \quad \boldsymbol{x} \in \Omega, \quad t \in [0, T] \tag{27}$$

where $\mathcal{N}$ is a (possibly nonlinear) differential operator, a PINN approximates the solution $\boldsymbol{u}(\boldsymbol{x}, t)$ via a neural network $\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t)$.

The total loss combines:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{BC}}\mathcal{L}_{\text{BC}} + \lambda_{\text{PDE}}\mathcal{L}_{\text{PDE}} \tag{28}$$

- $\mathcal{L}_{\text{data}}$: Supervised loss at observation points
- $\mathcal{L}_{\text{IC}}$: Initial condition enforcement
- $\mathcal{L}_{\text{BC}}$: Boundary condition enforcement
- $\mathcal{L}_{\text{PDE}}$: Physics residual at collocation points

7.2 Track Extrapolation as a PINN Problem

For track propagation through a magnetic field, the governing equations are:

$$\frac{d\boldsymbol{s}}{dz} = \boldsymbol{f}(\boldsymbol{s}, z, \boldsymbol{B}(x, y, z)) \tag{29}$$

where $\boldsymbol{s} = (x, y, t_x, t_y, q/p)^T$ and $\boldsymbol{f}$ encodes the Lorentz force (Eqs. 7–8).

7.2.1 Network Architecture

The PINN network takes:

$$\text{Input}: \boldsymbol{\xi} = (\underbrace{x_0, y_0, t_{x,0}, t_{y,0}, q/p}_{\text{initial conditions}}, \underbrace{z}_{\text{query position}}) \in \mathbb{R}^6 \tag{30}$$

$$\text{Output}: \hat{\boldsymbol{s}}(\boldsymbol{\xi}) = (\hat{x}(z), \hat{y}(z), \hat{t}_x(z), \hat{t}_y(z)) \in \mathbb{R}^4 \tag{31}$$

The network learns the *trajectory* $\boldsymbol{s}(z)$ given initial conditions, not just the endpoint.

7.2.2 Loss Components for Track PINN

1. Initial Condition Loss (at $z = z_0$):

$$\mathcal{L}_{\text{IC}} = \frac{1}{N} \sum_{i=1}^{N} \|\hat{\boldsymbol{s}}(\boldsymbol{\xi}_i^{z=z_0}) - \boldsymbol{s}_{0,i}\|^2 \tag{32}$$

This enforces that the network output matches the initial state at $z = z_0$.

2. Data Loss (at $z = z_{\text{final}}$):

$$\mathcal{L}_{\text{data}} = \frac{1}{N} \sum_{i=1}^{N} \left\|\hat{\boldsymbol{s}}(\boldsymbol{\xi}_i^{z=z_f}) - \boldsymbol{s}_{\text{RK},i}\right\|^2 \tag{33}$$

This matches RK4 reference outputs at the final z plane.

3. Physics Residual Loss (Lorentz force equations):

At *collocation points* $\{z_j\}$ sampled throughout the propagation domain:

$$\mathcal{L}_{\text{physics}} = \frac{1}{N_c} \sum_{j=1}^{N_c} \|\mathcal{R}(\hat{\boldsymbol{s}}(z_j))\|^2 \tag{34}$$

where the residual vector is:

$$\mathcal{R} = \begin{pmatrix} \frac{\partial \hat{x}}{\partial z} - \hat{t}_x \\ \frac{\partial \hat{y}}{\partial z} - \hat{t}_y \\ \frac{\partial \hat{t}_x}{\partial z} - \kappa\sqrt{1 + \hat{t}_x^2 + \hat{t}_y^2}\left[\hat{t}_x\hat{t}_y B_x - (1 + \hat{t}_x^2)B_y + \hat{t}_y B_z\right] \\ \frac{\partial \hat{t}_y}{\partial z} - \kappa\sqrt{1 + \hat{t}_y^2 + \hat{t}_y^2}\left[(1 + \hat{t}_y^2)B_x - \hat{t}_x\hat{t}_y B_y - \hat{t}_x B_z\right] \end{pmatrix} \tag{35}$$

The derivatives $\partial\hat{\boldsymbol{s}}/\partial z$ are computed via **automatic differentiation** through the neural network.

7.3 Computing Physics Gradients

A key technical requirement for PINNs is computing $\partial\hat{\boldsymbol{s}}/\partial z$. Since z is an input to the network, we use the chain rule:

$$\frac{\partial \hat{x}}{\partial z} = \frac{\partial \hat{x}}{\partial \tilde{z}} \cdot \frac{\partial \tilde{z}}{\partial z} = \frac{\partial \hat{x}}{\partial \tilde{z}} \cdot \frac{1}{\sigma_z} \tag{36}$$

where $\tilde{z} = (z - \mu_z)/\sigma_z$ is the normalized input.

In PyTorch, this is implemented as:

```python
# Forward pass with gradient tracking
x_input.requires_grad_(True)
output = model(x_input)

# Compute d(output)/d(z) via autograd
for i in range(output_dim):
    grad = torch.autograd.grad(
        outputs=output[:, i].sum(),
        inputs=x_input,
        create_graph=True  # For second-order gradients
    )[0]
    doutput_dz[:, i] = grad[:, 5]  # z is input index 5
```

7.4 Total PINN Loss for Track Extrapolation

Combining all components:

$$\boxed{\mathcal{L}_{\text{PINN}} = \underbrace{\mathcal{L}_{\text{IC}}}_{\text{initial conditions}} + \underbrace{\mathcal{L}_{\text{data}}}_{\text{final state}} + \lambda_{\text{phys}} \underbrace{\mathcal{L}_{\text{physics}}}_{\text{Lorentz force}}} \tag{37}$$

Typical values: $\lambda_{\text{phys}} \in [0.01, 1.0]$, tuned based on relative magnitudes.

7.5 Advantages of True PINNs

1. **Data Efficiency**: Can train with fewer labeled examples since physics provides regularization
2. **Generalization**: Physics constraints help extrapolate to unseen parameter regions
3. **Interpretability**: Residuals indicate where physics is violated
4. **Conservation Laws**: Can explicitly enforce energy conservation, symplectic structure

7.6 Implementation Status

We have implemented the true PINN training infrastructure in `train_true_pinn.py`:

- Differentiable magnetic field model (Gaussian approximation to LHCb dipole)
- Physics loss computation with automatic differentiation
- Collocation point sampling throughout propagation domain
- Configurable λ_{phys} weighting

Comparison with MLP: See `compare_models.py` for side-by-side evaluation of data-driven MLP vs. true PINN.

8 Future Work

8.1 Additional Physics Constraints

1. **Energy conservation**: $|\boldsymbol{p}|$ constant (no material)

$$\mathcal{L}_{\text{energy}} = \left| \frac{1}{q/p_{\text{final}}} - \frac{1}{q/p_{\text{initial}}} \right|^2 \tag{38}$$

2. **Symplectic structure**: Preserve phase space volume (Liouville's theorem)
3. **Boundary conditions**: Continuity at material interfaces

8.2 Architecture Improvements

1. **Residual connections**: $\boldsymbol{s}_{\text{out}} = \boldsymbol{s}_{\text{in}} + \Delta\boldsymbol{s}_{\text{NN}}$
2. **Multi-step prediction**: Learn $z \to z + \delta z$ for small δz, iterate
3. **Neural ODE**: $d\boldsymbol{s}/dz = f_\theta(\boldsymbol{s}, z)$, integrate with ODE solver
4. **Fourier features**: Better learning of high-frequency field variations

8.3 Quantitative Targets

Table 6: Performance targets for production use

Metric	Current	Target	Improvement
Radial error	6.97 mm	<1 mm	7×
t_x error	0.058	<0.001	58×
Momentum range	2.5 GeV	0.5–200 GeV	Full coverage
Speedup	155×	> 100×	Maintain

9 Conclusions

We have developed and evaluated a **data-driven neural network** for accelerating track extrapolation in the LHCb detector. Key findings:

1. **Speedup**: The neural network achieves 155× speedup over RK4
2. **Accuracy**: Current mean radial error of 6.97 mm requires improvement for physics applications
3. **Nomenclature**: Our current model is a data-driven MLP, not a true PINN—it does not explicitly enforce physics constraints in the loss function
4. **Path forward**: Converting to a true PINN with Lorentz force residual loss could improve generalization and accuracy

The approach demonstrates the potential for ML-accelerated track extrapolation. Future work will focus on incorporating physics constraints, expanding momentum coverage, and validating on full Monte Carlo simulation.

Acknowledgments

This work uses the LHCb software framework (Gaudi/Rec) and detector simulation infrastructure.

A Derivation of Track Equations of Motion

Starting from the Lorentz force:

$$\frac{d\boldsymbol{p}}{dt} = q\boldsymbol{v} \times \boldsymbol{B} \tag{39}$$

With $\boldsymbol{p} = p\hat{\boldsymbol{n}}$ where $\hat{\boldsymbol{n}}$ is the unit direction vector:

$$\hat{\boldsymbol{n}} = \frac{1}{\sqrt{1 + t_x^2 + t_y^2}} \begin{pmatrix} t_x \\ t_y \\ 1 \end{pmatrix} \tag{40}$$

The velocity is $\boldsymbol{v} = v\hat{\boldsymbol{n}}$ where $v = |\boldsymbol{v}|$. Converting to z as the independent variable using $dt = dz/(v \cdot n_z)$:

$$\frac{d\boldsymbol{p}}{dz} = \frac{q}{n_z}(\hat{\boldsymbol{n}} \times \boldsymbol{B}) \tag{41}$$

Expanding the cross product and using $n_z = 1/\sqrt{1 + t_x^2 + t_y^2}$:

$$\frac{dt_x}{dz} = \frac{c \cdot q/p}{\sqrt{1 + t_x^2 + t_y^2}} \left[(t_x t_y B_x - (1 + t_x^2) B_y + t_y B_z) \sqrt{1 + t_x^2 + t_y^2} \right] \tag{42}$$

$$= c \cdot (q/p) \sqrt{1 + t_x^2 + t_y^2} \left[t_x t_y B_x - (1 + t_x^2) B_y + t_y B_z \right] \tag{43}$$

Similarly for dt_y/dz. The factor $c = 299.792458$ mm/ns converts units when B is in Tesla, (q/p) in MeV^{-1}, and z in mm.