Mathematical Foundations of Neural Network Track Extrapolation in LHCb

G. Scriven

January 2026

Abstract

This document presents the complete mathematical derivations for neural network-based track extrapolation in the LHCb detector. We cover the physics of charged particle motion in magnetic fields, derive the governing equations in z-parameterization, and develop three neural network architectures: Multi-Layer Perceptron (MLP), Physics-Informed Neural Network (PINN), and Runge-Kutta Physics-Informed Neural Network (RK-PINN). We also provide a comprehensive treatment of Runge-Kutta numerical integration theory that motivates the RK-PINN architecture.

Contents

1 Introduction

Track reconstruction in the LHCb detector requires extrapolating particle trajectories through the magnetic field. The traditional approach uses numerical integration of the equations of motion (Runge-Kutta methods). We explore neural network alternatives that learn the extrapolation mapping directly from data or by incorporating physics constraints.

1.1 Notation

Throughout this document, we use the following notation:

- $\boldsymbol{x} = (x, y, z)^T$ - spatial position in mm
- $\boldsymbol{p} = (p_x, p_y, p_z)^T$ - momentum in MeV
- $\boldsymbol{B} = (B_x, B_y, B_z)^T$ - magnetic field in Tesla
- $t_x = \frac{dx}{dz}$, $t_y = \frac{dy}{dz}$ - track slopes (dimensionless)
- q/p - charge over momentum in MeV^{-1}
- $c_{\text{light}} = 2.99792458 \times 10^{-4}$ - speed of light factor

2 Physics of Charged Particle Motion

2.1 The Lorentz Force

A charged particle with charge q and velocity $\boldsymbol{v}$ in a magnetic field $\boldsymbol{B}$ experiences the Lorentz force:

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

Using Newton's second law and the relativistic momentum $\boldsymbol{p} = \gamma m \boldsymbol{v}$:

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

Remark 2.1. The Lorentz force is perpendicular to the velocity, so it does no work on the particle. Energy (and thus $|\boldsymbol{p}|$) is conserved in a pure magnetic field.

2.2 Z-Parameterization

In LHCb, tracks propagate predominantly in the $+z$ direction. It is therefore convenient to parameterize the trajectory by z rather than time. We define the **track state** at position z as:

$$\boldsymbol{y}(z) = \begin{pmatrix} x(z) \\ y(z) \\ t_x(z) \\ t_y(z) \end{pmatrix} \tag{3}$$

where $t_x = \frac{dx}{dz}$ and $t_y = \frac{dy}{dz}$ are the track slopes.
The relationship between time and z derivatives is:

$$\frac{d}{dt} = \frac{v_z}{1}\frac{d}{dz} = \frac{p_z}{\gamma m}\frac{d}{dz} \tag{4}$$

2.3 Derivation of the Equations of Motion

Theorem 2.1 (Equations of Motion in Z-Parameterization)**.** The evolution of the track state $\boldsymbol{y}(z)$ is governed by:

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

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

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

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

where:

$$\kappa = \frac{q}{p} \cdot c_{\text{light}}, \qquad N = \sqrt{1 + t_x^2 + t_y^2} \tag{9}$$

Proof. We derive the slope equations starting from the Lorentz force. The momentum components are related to the slopes by:

$$p_x = p \cdot \frac{t_x}{N}, \quad p_y = p \cdot \frac{t_y}{N}, \quad p_z = p \cdot \frac{1}{N} \tag{10}$$

where $N = \sqrt{1 + t_x^2 + t_y^2}$ ensures $|\boldsymbol{p}| = p$.

The velocity is $\boldsymbol{v} = \boldsymbol{p}/(\gamma m)$, so:

$$v_z = \frac{p_z}{\gamma m} = \frac{p}{N \gamma m} \tag{11}$$

From the Lorentz force $\frac{d\boldsymbol{p}}{dt} = q(\boldsymbol{v} \times \boldsymbol{B})$, the x-component is:

$$\frac{dp_x}{dt} = q(v_y B_z - v_z B_y) = \frac{q}{\gamma m}(p_y B_z - p_z B_y) \tag{12}$$

Converting to z-derivatives using $\frac{dt}{d=} v_z \frac{dz}{d}$:

$$\frac{dp_x}{dz} = \frac{1}{v_z}\frac{dp_x}{dt} = \frac{N\gamma m}{p} \cdot \frac{q}{\gamma m}(p_y B_z - p_z B_y) \tag{13}$$

Simplifying:

$$\frac{dp_x}{dz} = \frac{qN}{p}\left(\frac{p t_y}{N} B_z - \frac{p}{N} B_y\right) = q(t_y B_z - B_y) \tag{14}$$

Now, $t_x = p_x/p_z = p_x N/p$, so:

$$\frac{dt_x}{dz} = \frac{N}{p}\frac{dp_x}{dz} + \frac{p_x}{p}\frac{dN}{dz} \tag{15}$$

Since $|\boldsymbol{p}|$ is conserved and N depends on slopes:

$$\frac{dN}{dz} = \frac{t_x}{N}\frac{dt_x}{dz} + \frac{t_y}{N}\frac{dt_y}{dz} \tag{16}$$

After algebraic manipulation (similar derivation for t_y), we obtain:

$$\frac{dt_x}{dz} = \frac{q}{p} c_{\text{light}} \cdot N \cdot \left[t_x t_y B_x - (1 + t_x^2) B_y + t_y B_z\right] \tag{17}$$

$$\frac{dt_y}{dz} = \frac{q}{p} c_{\text{light}} \cdot N \cdot \left[(1 + t_y^2) B_x - t_x t_y B_y - t_x B_z\right] \tag{18}$$

where $c_{\text{light}} = 2.99792458 \times 10^{-4}$ accounts for unit conversions. □

2.4 Compact Form

We can write the system compactly as:

$$\frac{d\boldsymbol{y}}{dz} = \boldsymbol{f}(\boldsymbol{y}, z; q/p) \tag{19}$$

where $\boldsymbol{f} : \mathbb{R}^4 \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^4$ is defined by Equations (5)–(8).

3 The LHCb Magnetic Field

3.1 Dipole Magnet Characteristics

The LHCb dipole magnet produces a field primarily in the y-direction (vertical):

- Peak field: $|B_y| \approx 1.03$ T at $z \approx 5000$ mm
- Field integral: $\int B_y \, dz \approx 4.44$ T·m
- Dominant component: B_y (causes bending in the x-z plane)

3.2 Field Map Representation

3.2.1 General Formulation

Throughout this work, we treat the magnetic field as a **callable function**:

$$\boldsymbol{B} : \mathbb{R}^3 \to \mathbb{R}^3, \quad (x, y, z) \mapsto (B_x, B_y, B_z) \tag{20}$$

This function may be implemented as:

1. An **analytical approximation** (e.g., Gaussian profile)
2. A **tabulated field map** with interpolation
3. A **neural network surrogate** trained on field map data

For physics-informed training, the field function must be **differentiable** with respect to position to support automatic differentiation.

3.2.2 Tabulated Field Map (twodip.rtf)

The LHCb field map is provided as a tabulated 3D grid:

- Format: (x, y, z, B_x, B_y, B_z) values on regular grid
- Grid extent: $x \in [-4000, 4000]$ mm, $y \in [-4000, 4000]$ mm, $z \in [-500, 14000]$ mm
- Grid spacing: $\Delta x = \Delta y = 100$ mm, $\Delta z = 100$ mm
- Total points: $\approx 958,000$

Field values at arbitrary positions are obtained via **trilinear interpolation**:

$$B_i(x, y, z) = \sum_{\alpha,\beta,\gamma \in \{0,1\}} w_\alpha(x) \, w_\beta(y) \, w_\gamma(z) \, B_i^{(\alpha\beta\gamma)} \tag{21}$$

where $w_0(\xi) = 1 - \frac{\xi - \xi_j}{\Delta\xi}$ and $w_1(\xi) = \frac{\xi - \xi_j}{\Delta\xi}$ are linear interpolation weights.

3.2.3 Interpolation Error Analysis

For trilinear interpolation on a grid with spacing h, the local truncation error is:

$$\epsilon_{\text{interp}} = \mathcal{O}(h^2 \cdot \max|\partial^2 B_i/\partial x_j^2|) \tag{22}$$

For the LHCb field map with $h = 100$ mm and typical field curvature:

$$\epsilon_{\text{interp}} \lesssim 10^{-5} \text{ T} \tag{23}$$

This is negligible compared to the peak field of ~ 1 T.

Remark 3.1 (Differentiability of Interpolated Field)**.** Trilinear interpolation is **piecewise linear** and therefore **not continuously differentiable** at grid cell boundaries. The gradient is discontinuous at these points:

$$\frac{\partial B_i}{\partial x_j} \text{ has jump discontinuities at } x_j = n \cdot \Delta x_j \tag{24}$$

For PINN training, this means:

- The physics residual may have small discontinuities at cell boundaries
- Using many collocation points averages over these discontinuities
- Alternative: use cubic spline or higher-order interpolation for C^1 continuity

In practice, with 10–20 random collocation points per sample, the effect is negligible.

3.3 Gaussian Field Approximation (Optional)

For rapid prototyping or when the field map is unavailable, we can use a Gaussian approximation:

$$B_y(z) = B_0 \exp\left(-\frac{1}{2}\left(\frac{z - z_c}{\sigma_z}\right)^2\right) \tag{25}$$

with fitted parameters:

$$B_0 = -1.0182 \text{ T}, \quad z_c = 5007 \text{ mm}, \quad \sigma_z = 1744 \text{ mm} \tag{26}$$

Approximation error: The Gaussian model has RMS error ≈ 0.013 T ($\sim 1.3\%$ of peak) when compared to the full field map along the beam axis. This error propagates to trajectory predictions and should be accounted for in error budgets.

Remark 3.2 (When to Use Each Field Model)**.**

- **Tabulated field map**: For production training and final evaluation. Provides highest accuracy.
- **Gaussian approximation**: For initial prototyping, debugging, and when field map access is restricted. Faster to evaluate.

Our training data is generated using the Gaussian approximation for simplicity, but the PINN can be retrained with the full field map for maximum accuracy.

4 Multi-Layer Perceptron (MLP)

4.1 Architecture

The MLP learns the extrapolation mapping directly from data without explicit physics. Given input features:

$$\boldsymbol{x} = (x_0, y_0, t_{x,0}, t_{y,0}, q/p, \Delta z)^T \in \mathbb{R}^6 \tag{27}$$

the network predicts the final state:

$$\hat{\boldsymbol{y}} = (x_f, y_f, t_{x,f}, t_{y,f})^T \in \mathbb{R}^4 \tag{28}$$

4.2 Network Definition

Definition 4.1 (Multi-Layer Perceptron)**.** An MLP with L hidden layers is defined as:

$$\text{MLP}(\boldsymbol{x}) = \mathbf{W}_{L+1}\sigma_L(\mathbf{W}_L\sigma_{L-1}(\cdots\sigma_1(\mathbf{W}_1\boldsymbol{x}+\boldsymbol{b}_1)\cdots)+\boldsymbol{b}_L)+\boldsymbol{b}_{L+1} \tag{29}$$

where $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$ are weight matrices, $\boldsymbol{b}_i$ are bias vectors, and σ_i are activation functions.

4.3 Input/Output Normalization

For stable training, we apply z-score normalization:

$$\tilde{x}_i = \frac{x_i - \mu_i}{\sigma_i} \tag{30}$$

where μ_i and σ_i are the mean and standard deviation computed from training data.

The network operates on normalized inputs and outputs:

$$\hat{\boldsymbol{y}} = \sigma_y \odot \text{MLP}(\tilde{\boldsymbol{x}}) + \mu_y \tag{31}$$

4.4 Loss Function

The MLP is trained with Mean Squared Error (MSE) loss:

$$\mathcal{L}_{\text{data}} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i^*\|^2 \tag{32}$$

where $\boldsymbol{y}_i^*$ is the ground truth from Runge-Kutta integration.

4.5 Implicit Physics Learning

Remark 4.1. The MLP learns the physics *implicitly* from training data generated by the RK extrapolator. The network approximates the flow map:

$$\Phi_{\Delta z} : \boldsymbol{y}_0 \mapsto \boldsymbol{y}(\Delta z) \tag{33}$$

which is the solution operator for the ODE system (19).

5 Physics-Informed Neural Network (PINN)

5.1 Concept

Physics-Informed Neural Networks (PINNs) incorporate the governing physics equations directly into the loss function. Instead of learning only from data, the network is constrained to satisfy the differential equations.

5.2 Key Distinction from MLP

Remark 5.1 (MLP vs PINN: Fundamental Difference)**.** The MLP performs a **direct mapping**:

$$\text{MLP} : (x_0, y_0, t_{x,0}, t_{y,0}, q/p, z_0, z_{\text{end}}) \mapsto (x_{\text{end}}, y_{\text{end}}, t_{x,\text{end}}, t_{y,\text{end}}) \tag{34}$$

One forward pass produces one answer. There is no notion of the path taken between z_0 and z_{end}.

The PINN, in contrast, outputs a **continuous trajectory** parameterized by $\zeta \in [0, 1]$:

$$\text{PINN} : (x_0, y_0, t_{x,0}, t_{y,0}, q/p, z_0, z_{\text{end}}, \zeta) \mapsto (x, y, t_x, t_y) \text{ at that } \zeta \tag{35}$$

where:

- $\zeta = 0$ corresponds to the initial state at z_0
- $\zeta = 1$ corresponds to the final state at z_{end}
- $\zeta = 0.5$ corresponds to the state halfway through the propagation

This trajectory representation is essential because the physics loss requires computing derivatives *along* the trajectory.

5.3 Network as Trajectory Function

The PINN learns the continuous trajectory:

$$\boldsymbol{y}_\theta : [0, 1] \to \mathbb{R}^4, \quad \zeta \mapsto \boldsymbol{y}_\theta(\zeta) \tag{36}$$

where $\zeta = (z - z_0)/\Delta z \in [0, 1]$ is the normalized position and θ represents network parameters.

5.4 Training Data Requirements

Remark 5.2 (No Intermediate Trajectory Data Required)**.** A common misconception is that PINNs require ground truth trajectories for training. This is **not** the case.

Training data needed:

- Initial state: $(x_0, y_0, t_{x,0}, t_{y,0}, q/p)$ at z_0
- Final state: $(x_{\text{end}}, y_{\text{end}}, t_{x,\text{end}}, t_{y,\text{end}})$ at z_{end}

This is **identical** to the MLP training data. No intermediate trajectory points are needed.

How it works: The physics loss (Eq. 40) does not compare the network's intermediate predictions against any ground truth. Instead, it checks whether the *network's own predicted trajectory* satisfies the Lorentz force equations. The network learns the correct intermediate states by being forced to satisfy physics self-consistently.

The only external knowledge required beyond endpoint data is:

1. The magnetic field model $\boldsymbol{B}(z)$ (known analytically)
2. The Lorentz force equations (known from physics)

5.5 Loss Function Components

The total PINN loss consists of three components:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{PDE}}\mathcal{L}_{\text{PDE}} \tag{37}$$

5.5.1 Data Loss

At the endpoint $\zeta = 1$:

$$\mathcal{L}_{\text{data}} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{y}_\theta(\zeta = 1; \boldsymbol{x}_i) - \boldsymbol{y}_i^*\|^2 \tag{38}$$

5.5.2 Initial Condition Loss

At $\zeta = 0$, the trajectory must match the initial state:

$$\mathcal{L}_{\text{IC}} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{y}_\theta(\zeta = 0; \boldsymbol{x}_i) - \boldsymbol{y}_{0,i}\|^2 \tag{39}$$

where $\boldsymbol{y}_{0,i} = (x_{0,i}, y_{0,i}, t_{x,0,i}, t_{y,0,i})^T$.

5.5.3 PDE Residual Loss

The physics constraint is enforced at *collocation points* $\{\zeta_k\}_{k=1}^{K}$ sampled along the trajectory:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{NK} \sum_{i=1}^{N} \sum_{k=1}^{K} \left\| \left. \frac{d\boldsymbol{y}_\theta}{d\zeta} \right|_{\zeta_k} - \Delta z \cdot \boldsymbol{f}(\boldsymbol{y}_\theta(\zeta_k), z_k; q/p_i) \right\|^2 \tag{40}$$

5.6 Why Collocation Points Throughout the Trajectory?

Remark 5.3 (Endpoints Alone Are Insufficient)**.** One might ask: is it sufficient to enforce the physics only at the endpoints ($\zeta = 0$ and $\zeta = 1$)?

The answer is **no**, and the LHCb magnetic field illustrates why. The field varies significantly along z:

$$z = 2500 \text{ mm}: \quad B_y \approx 0 \text{ (entering magnet)} \tag{41}$$

$$z = 5000 \text{ mm}: \quad B_y \approx -1.0 \text{ T (peak field)} \tag{42}$$

$$z = 7500 \text{ mm}: \quad B_y \approx 0 \text{ (exiting magnet)} \tag{43}$$

If we only enforced physics at the endpoints where $B \approx 0$, the network could learn a **straight-line trajectory**:

- At $\zeta = 0$: $d\boldsymbol{y}/d\zeta \approx 0$ (✓satisfies physics, no field)
- At $\zeta = 1$: $d\boldsymbol{y}/d\zeta \approx 0$ (✓satisfies physics, no field)
- At $\zeta = 0.5$: Completely ignores the strong bending from $B_y = -1.0$ T

By sampling collocation points **throughout** $[0, 1]$, we force the network to learn a trajectory that respects the Lorentz force *everywhere*, including where the field is strongest. The trajectory "discovers" the correct curved path by satisfying the physics constraints at many intermediate points.

5.7 Automatic Differentiation

The key insight of PINNs is that the derivative $\frac{\partial \boldsymbol{y}_\theta}{\partial \zeta}$ can be computed exactly using automatic differentiation:

$$\frac{\partial y_j}{\partial \zeta} = \frac{\partial}{\partial \zeta} \text{NN}_j(\boldsymbol{x}, \zeta; \theta) \tag{44}$$

This is computed via backpropagation through the network graph.

5.8 Concrete Example of Physics Enforcement

To illustrate concretely how the physics is enforced without trajectory data, consider a collocation point at $\zeta = 0.5$ (middle of the magnet, $z \approx 5000$ mm):

1. **Network prediction:** Query $\boldsymbol{y}_\theta(\zeta = 0.5)$, suppose it returns $(x, y, t_x, t_y) = (10.2, 3.1, 0.15, 0.02)$
2. **Autodiff derivative:** Compute $d\boldsymbol{y}_\theta/d\zeta$ at $\zeta = 0.5$ via backpropagation, suppose it gives $(0.15, 0.02, -0.003, 0.0001)$
3. **Field lookup:** At $z = 5000$ mm, the field model gives $B_y = -1.0$ T
4. **Physics calculation:** Using the Lorentz force equations (7)–(8) with the network's predicted state and the known field, compute what $d\boldsymbol{y}/d\zeta$ *should* be: $(0.15, 0.02, -0.0028, 0.0001)$

5. **Residual:** $\|(0.15, 0.02, -0.003, 0.0001) - (0.15, 0.02, -0.0028, 0.0001)\|^2$ penalizes the mismatch

Note that step 4 uses only the magnetic field model and physics equations—no ground truth intermediate state is required. The network is trained to make its own predictions self-consistent with physics.

5.9 Algorithm

The PINN training procedure follows these steps:

1. **Input:** Batch of initial states $\{\boldsymbol{x}_i\}$, ground truth endpoints $\{\boldsymbol{y}_i^*\}$
2. Sample collocation points $\{\zeta_k\}_{k=1}^K \sim \text{Uniform}(0, 1)$
3. For each sample i:
   - Compute $\boldsymbol{y}_\theta(\zeta = 0)$, $\boldsymbol{y}_\theta(\zeta = 1)$, and $\boldsymbol{y}_\theta(\zeta_k)$ for all k
   - Compute $\frac{\partial \boldsymbol{y}_\theta}{\partial \zeta}$ at each ζ_k via autodiff
   - Evaluate physics residual using Eqs. (5)–(8)
4. Compute $\mathcal{L}_{\text{total}}$ using Eq. (37)
5. Update θ via gradient descent

At inference time: Simply evaluate the network at $\zeta = 1$ to obtain the final state, identical to MLP inference. The intermediate trajectory capability is only used during training.

6 Runge-Kutta Numerical Integration

6.1 General Framework

Runge-Kutta methods approximate the solution of an initial value problem:

$$\frac{d\boldsymbol{y}}{dz} = \boldsymbol{f}(\boldsymbol{y}, z), \quad \boldsymbol{y}(z_0) = \boldsymbol{y}_0 \tag{45}$$

Definition 6.1 (Explicit Runge-Kutta Method)**.** An s-stage explicit RK method advances the solution from $\boldsymbol{y}_n$ to $\boldsymbol{y}_{n+1}$ over step h by:

$$\boldsymbol{k}_1 = h \cdot \boldsymbol{f}(z_n, \boldsymbol{y}_n) \tag{46}$$
$$\boldsymbol{k}_2 = h \cdot \boldsymbol{f}(z_n + c_2 h, \boldsymbol{y}_n + a_{21}\boldsymbol{k}_1) \tag{47}$$
$$\boldsymbol{k}_3 = h \cdot \boldsymbol{f}(z_n + c_3 h, \boldsymbol{y}_n + a_{31}\boldsymbol{k}_1 + a_{32}\boldsymbol{k}_2) \tag{48}$$
$$\vdots \tag{49}$$
$$\boldsymbol{k}_s = h \cdot \boldsymbol{f}(z_n + c_s h, \boldsymbol{y}_n + \sum_{j=1}^{s-1} a_{sj}\boldsymbol{k}_j) \tag{50}$$
$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \sum_{i=1}^{s} b_i \boldsymbol{k}_i \tag{51}$$

6.2 Butcher Tableau

The RK method is characterized by its Butcher tableau:

$$\begin{array}{c|cccc} c_1 & 0 & & & \\ c_2 & a_{21} & 0 & & \\ c_3 & a_{31} & a_{32} & 0 & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & 0 \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{52}$$

6.3 Classical RK4

The classical fourth-order Runge-Kutta method (RK4) has the tableau:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{53}$$

Explicitly:

$$\boldsymbol{k}_1 = h \cdot \boldsymbol{f}(z_n, \boldsymbol{y}_n) \tag{54}$$

$$\boldsymbol{k}_2 = h \cdot \boldsymbol{f}\left(z_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{\boldsymbol{k}_1}{2}\right) \tag{55}$$

$$\boldsymbol{k}_3 = h \cdot \boldsymbol{f}\left(z_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{\boldsymbol{k}_2}{2}\right) \tag{56}$$

$$\boldsymbol{k}_4 = h \cdot \boldsymbol{f}(z_n + h, \boldsymbol{y}_n + \boldsymbol{k}_3) \tag{57}$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4) \tag{58}$$

6.4 Order Conditions

Theorem 6.1 (Local Truncation Error)**.** The local truncation error of RK4 is $O(h^5)$, making it a fourth-order method with global error $O(h^4)$.

The order conditions are derived by matching Taylor series coefficients. For order p, the method must satisfy:

$$\boldsymbol{y}(z_n + h) - \boldsymbol{y}_{n+1} = O(h^{p+1}) \tag{59}$$

6.5 Geometric Interpretation

RK4 can be interpreted as:

1. $\boldsymbol{k}_1$: Slope at the starting point
2. $\boldsymbol{k}_2$: Slope at the midpoint using $\boldsymbol{k}_1$
3. $\boldsymbol{k}_3$: Slope at the midpoint using $\boldsymbol{k}_2$
4. $\boldsymbol{k}_4$: Slope at the endpoint using $\boldsymbol{k}_3$

The final update is a weighted average: $(1 + 2 + 2 + 1)/6 = 1$.

7 RK-PINN: Runge-Kutta Physics-Informed Neural Network

7.1 Motivation

The RK-PINN combines the structure of Runge-Kutta methods with neural network learning. The key insight is that the intermediate stages of RK methods provide natural positions for physics constraints.

7.2 Architecture

Definition 7.1 (RK-PINN)**.** The RK-PINN consists of:

1. **Shared backbone** $\boldsymbol{h}_\theta : \mathbb{R}^6 \to \mathbb{R}^d$ that extracts features
2. **Stage heads** $\{g_{\phi_i}\}_{i=1}^s$ that predict states at stage positions
3. **Learnable weights** $\{w_i\}_{i=1}^s$ for combining stage outputs

For a 4-stage RK-PINN (analogous to RK4):

$$\boldsymbol{h} = \text{Backbone}(\boldsymbol{x}) \tag{60}$$

$$\hat{\boldsymbol{y}}_1 = g_{\phi_1}(\boldsymbol{h}, \zeta = 0.25) \quad (\text{at } z_0 + 0.25\Delta z) \tag{61}$$

$$\hat{\boldsymbol{y}}_2 = g_{\phi_2}(\boldsymbol{h}, \zeta = 0.50) \quad (\text{at } z_0 + 0.50\Delta z) \tag{62}$$

$$\hat{\boldsymbol{y}}_3 = g_{\phi_3}(\boldsymbol{h}, \zeta = 0.75) \quad (\text{at } z_0 + 0.75\Delta z) \tag{63}$$

$$\hat{\boldsymbol{y}}_4 = g_{\phi_4}(\boldsymbol{h}, \zeta = 1.00) \quad (\text{at } z_0 + \Delta z) \tag{64}$$

$$\hat{\boldsymbol{y}}_{\text{final}} = \sum_{i=1}^{4} \tilde{w}_i \hat{\boldsymbol{y}}_i \tag{65}$$

where $\tilde{w}_i = \text{softmax}(\boldsymbol{w})_i$ ensures weights sum to 1.

7.3 Weight Initialization

The weights are initialized to match RK4:

$$w_1^{(0)} = \frac{1}{6}, \quad w_2^{(0)} = \frac{2}{6}, \quad w_3^{(0)} = \frac{2}{6}, \quad w_4^{(0)} = \frac{1}{6} \tag{66}$$

During training, these weights can adapt to better fit the data.

7.4 Loss Function

The RK-PINN loss combines data and physics terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda_{\text{IC}} \mathcal{L}_{\text{IC}} + \lambda_{\text{PDE}} \sum_{i=1}^{s} \mathcal{L}_{\text{PDE}}^{(i)} \tag{67}$$

where the PDE loss at each stage is:

$$\mathcal{L}_{\text{PDE}}^{(i)} = \frac{1}{N} \sum_{j=1}^{N} \left\| \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \zeta} - \Delta z \cdot \boldsymbol{f}(\hat{\boldsymbol{y}}_i, z_i; q/p_j) \right\|^2 \tag{68}$$

Property	PINN	RK-PINN
Collocation points	Random sampling	Fixed at RK positions
Architecture	Single network	Backbone + stage heads
Stage weights	N/A	Learnable (init: RK4)
Physics structure	Soft constraint	Structured like RK

Table 1: Comparison of PINN and RK-PINN architectures.

7.5 Comparison with Standard PINN

8 Summary of Loss Functions

8.1 MLP

$$\mathcal{L}_{\text{MLP}} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i^*\|^2 \tag{69}$$

8.2 PINN

$$\mathcal{L}_{\text{PINN}} = \underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i^*\|^2}_{\mathcal{L}_{\text{data}}} + \lambda_{\text{IC}} \underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\hat{\boldsymbol{y}}_{0,i} - \boldsymbol{y}_{0,i}\|^2}_{\mathcal{L}_{\text{IC}}} + \lambda_{\text{PDE}} \underbrace{\frac{1}{NK}\sum_{i,k} \|\mathcal{R}_{ik}\|^2}_{\mathcal{L}_{\text{PDE}}} \tag{70}$$

where the residual is:

$$\mathcal{R}_{ik} = \left.\frac{\partial \boldsymbol{y}_\theta}{\partial \zeta}\right|_{\zeta_k} - \Delta z \cdot \boldsymbol{f}(\boldsymbol{y}_\theta(\zeta_k), z_k; q/p_i) \tag{71}$$

8.3 RK-PINN

$$\mathcal{L}_{\text{RK-PINN}} = \mathcal{L}_{\text{data}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{PDE}} \sum_{s=1}^{4} \mathcal{L}_{\text{PDE}}^{(s)} \tag{72}$$

9 Implementation Notes

9.1 Critical Constants

$$c_{\text{light}} = 2.99792458 \times 10^{-4} \quad [\text{mm/ns} \times \text{unit conversions}] \tag{73}$$

This constant converts:

$$\kappa = \frac{q}{p} \cdot c_{\text{light}} \quad [\text{MeV}^{-1}\ \text{T}^{-1}\ \text{mm}^{-1}] \tag{74}$$

9.2 Normalization Factor

The path length factor:

$$N = \sqrt{1 + t_x^2 + t_y^2} = \frac{|\boldsymbol{p}|}{p_z} = \frac{ds}{dz} \tag{75}$$

accounts for the fact that the particle travels a distance $ds = N \cdot dz$ when advancing by dz in z.

9.3 Dominant Field Component

In LHCb, B_y dominates, causing bending in the x-z plane:

$$\frac{dt_x}{dz} \approx -\kappa \cdot N \cdot (1 + t_x^2) \cdot B_y \tag{76}$$

For small slopes ($t_x, t_y \ll 1$) and positive particles ($q > 0$) in a negative field ($B_y < 0$):

$$\frac{dt_x}{dz} > 0 \quad \Rightarrow \quad \text{track bends to positive } x \tag{77}$$

10 Error Analysis and Uncertainty Quantification

A rigorous understanding of the error sources is essential for deploying neural network extrapolators in a physics experiment. We categorize errors into several distinct types.

10.1 Training Data Errors

10.1.1 Ground Truth Generation Error

The training targets are generated using numerical integration (RK4). The local truncation error of RK4 is:

$$\epsilon_{\text{RK4}}^{\text{local}} = \mathcal{O}(h^5) \tag{78}$$

where h is the step size. For n steps, the global error accumulates:

$$\epsilon_{\text{RK4}}^{\text{global}} = \mathcal{O}(h^4) \tag{79}$$

With our step size $h = 10$ mm over $\Delta z \approx 2500$ mm ($n = 250$ steps), the RK4 error is typically $\mathcal{O}(10^{-8})$ in normalized coordinates, well below our target precision.

10.1.2 Magnetic Field Evaluation Error

The magnetic field function $\boldsymbol{B}(x, y, z)$ introduces error depending on its implementation:

Option 1: Interpolated Field Map (Recommended) When using trilinear interpolation from the tabulated field map:

$$\epsilon_{\text{field}}^{\text{interp}} = \mathcal{O}(h_{\text{grid}}^2) \approx 10^{-5} \text{ T} \tag{80}$$

where $h_{\text{grid}} = 100$ mm is the field map grid spacing.

This error is **negligible** compared to the peak field (~ 1 T) and does not significantly impact trajectory accuracy.

Option 2: Gaussian Approximation When using the analytical Gaussian model:

$$B_y^{\text{approx}}(z) = B_0 \exp\left(-\frac{(z - z_c)^2}{2\sigma_z^2}\right) \tag{81}$$

The approximation error relative to the true field map is:

$$\epsilon_{\text{field}}^{\text{Gaussian}}(z) = B_y^{\text{true}}(z) - B_y^{\text{approx}}(z) \tag{82}$$

with RMS ≈ 0.013 T (about 1.3% of peak field).

This propagates to trajectory error as:

$$\delta t_x \sim \kappa \int_{z_0}^{z_{\text{end}}} \epsilon_{\text{field}}(z)\, dz \tag{83}$$

For high-precision applications, the interpolated field map should be used.

Field Model	Error Magnitude	Differentiable?
Tabulated + trilinear interp.	$\sim 10^{-5}$ T	Piecewise (discontinuous gradient)
Tabulated + cubic spline	$\sim 10^{-7}$ T	C^1 continuous
Gaussian approximation	$\sim 10^{-2}$ T	C^∞ (smooth)

Table 2: Comparison of magnetic field model accuracy and differentiability.

Comparison of Field Models

10.1.3 Sampling Bias

If the training distribution $P_{\text{train}}(\boldsymbol{x})$ does not match the operational distribution $P_{\text{physics}}(\boldsymbol{x})$, systematic biases emerge:

$$\text{Bias} = \mathbb{E}_{P_{\text{physics}}}[\hat{y}(\boldsymbol{x})] - \mathbb{E}_{P_{\text{physics}}}[y^*(\boldsymbol{x})] \tag{84}$$

Critical distributions to match:

- Momentum spectrum (especially low-p tails)
- Angular distribution (t_x, t_y)
- Charge ratio (q/p sign)
- Spatial distribution (x, y at entry)

10.2 Model Approximation Errors

10.2.1 Network Capacity Error

A neural network with finite parameters θ cannot represent arbitrary functions. The approximation error is:

$$\epsilon_{\text{approx}} = \inf_{\theta} \mathbb{E}\left[\|f_\theta(\boldsymbol{x}) - f^*(\boldsymbol{x})\|^2\right] \tag{85}$$

This depends on:

- Network depth L and width W
- Activation function smoothness
- Complexity of the true mapping f^*

Universal approximation theorems guarantee $\epsilon_{\text{approx}} \to 0$ as capacity $\to \infty$, but practical networks have non-zero error.

10.2.2 Optimization Error

Gradient descent may not find the global optimum:

$$\epsilon_{\text{opt}} = \mathcal{L}(\theta_{\text{final}}) - \mathcal{L}(\theta^*) \tag{86}$$

where θ^* is the global minimum. Sources include:

- Local minima and saddle points
- Learning rate selection
- Early stopping (intentional regularization)
- Batch size effects on gradient noise

10.2.3 Generalization Error

The gap between training and test performance:

$$\epsilon_{\text{gen}} = \mathbb{E}_{\text{test}}[\mathcal{L}] - \mathbb{E}_{\text{train}}[\mathcal{L}] \tag{87}$$

Controlled by:

- Training set size N
- Model complexity (number of parameters)
- Regularization (dropout, weight decay)
- Data augmentation

10.3 PINN-Specific Errors

10.3.1 Collocation Discretization Error

The continuous physics constraint is approximated by discrete sampling:

$$\mathcal{L}_{\text{PDE}}^{\text{continuous}} = \int_0^1 \left\| \frac{d\boldsymbol{y}_\theta}{d\zeta} - \boldsymbol{f}(\boldsymbol{y}_\theta, B) \right\|^2 d\zeta \tag{88}$$

$$\mathcal{L}_{\text{PDE}}^{\text{discrete}} = \frac{1}{K} \sum_{k=1}^{K} \left\| \left. \frac{d\boldsymbol{y}_\theta}{d\zeta} \right|_{\zeta_k} - \boldsymbol{f}(\boldsymbol{y}_\theta(\zeta_k), B(\zeta_k)) \right\|^2 \tag{89}$$

The discretization error is:

$$\epsilon_{\text{colloc}} = \left| \mathcal{L}_{\text{PDE}}^{\text{continuous}} - \mathcal{L}_{\text{PDE}}^{\text{discrete}} \right| \tag{90}$$

For K uniformly distributed points, by standard quadrature theory:

$$\epsilon_{\text{colloc}} = \mathcal{O}(K^{-1}) \quad \text{(random sampling)} \tag{91}$$

Validation procedure: After training with K points, evaluate the residual on a much finer grid ($K' \gg K$):

$$R(\zeta) = \left\| \frac{d\boldsymbol{y}_\theta}{d\zeta} - \boldsymbol{f}(\boldsymbol{y}_\theta, B) \right\| \tag{92}$$

If $\max_\zeta R(\zeta) \gg \frac{1}{K}\sum_k R(\zeta_k)$, the network is "cheating" between collocation points.

10.3.2 Inter-Point Oscillation Error

Between collocation points, the network is unconstrained and may develop spurious oscillations:

$$\boldsymbol{y}_\theta(\zeta) = \boldsymbol{y}_{\text{physical}}(\zeta) + \delta\boldsymbol{y}(\zeta) \tag{93}$$

where $\delta\boldsymbol{y}(\zeta_k) \approx 0$ at collocation points but $\delta\boldsymbol{y}(\zeta) \neq 0$ between them.

Mitigation strategies:

1. Increase K (more collocation points)
2. Use smooth activation functions (SiLU, Tanh vs ReLU)
3. Add higher-order derivative penalties
4. Randomize collocation points each batch (prevents overfitting to fixed ζ)

10.3.3 Loss Balancing Error

The total PINN loss combines multiple terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda_{\text{PDE}}\mathcal{L}_{\text{PDE}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} \tag{94}$$

Poor choice of λ values leads to:

- $\lambda_{\text{PDE}} \ll 1$: Physics constraints ignored, network behaves like MLP
- $\lambda_{\text{PDE}} \gg 1$: Data fit sacrificed, poor endpoint accuracy

The optimal λ depends on the relative scales and noise levels:

$$\lambda_{\text{PDE}}^{\text{optimal}} \sim \frac{\text{Var}(\mathcal{L}_{\text{data}})}{\text{Var}(\mathcal{L}_{\text{PDE}})} \tag{95}$$

Adaptive strategies:

- GradNorm: Balance gradient magnitudes
- Learning rate annealing per loss term
- Curriculum learning: Start with data loss, gradually add physics

10.3.4 Automatic Differentiation Precision

Computing $\frac{\partial \boldsymbol{y}_\theta}{\partial \zeta}$ via backpropagation introduces floating-point errors:

$$\left(\frac{\partial \boldsymbol{y}}{\partial \zeta}\right)_{\text{computed}} = \left(\frac{\partial \boldsymbol{y}}{\partial \zeta}\right)_{\text{true}} + \epsilon_{\text{AD}} \tag{96}$$

For well-conditioned networks, $\epsilon_{\text{AD}} \sim 10^{-7}$ (float32) or 10^{-15} (float64). This is typically negligible compared to other errors.

10.4 RK-PINN Specific Errors

10.4.1 Butcher Tableau Approximation

The RK-PINN uses fixed RK4 coefficients. If the learned force $\hat{\boldsymbol{f}}_\theta$ differs from the true force:

$$\hat{\boldsymbol{f}}_\theta(\boldsymbol{y}, z) = \boldsymbol{f}_{\text{true}}(\boldsymbol{y}, z) + \delta\boldsymbol{f}(\boldsymbol{y}, z) \tag{97}$$

The RK4 integration amplifies this error over the step:

$$\delta\boldsymbol{y}_{\text{step}} \approx \frac{h}{6}(\delta\boldsymbol{f}_1 + 2\delta\boldsymbol{f}_2 + 2\delta\boldsymbol{f}_3 + \delta\boldsymbol{f}_4) \tag{98}$$

10.4.2 Single-Step vs Multi-Step Error

The RK-PINN typically takes one large step ($h = \Delta z$) rather than many small steps. The truncation error is:

$$\epsilon_{\text{RK-PINN}} = \mathcal{O}(h^5) \quad \text{vs} \quad \epsilon_{\text{RK4}} = \mathcal{O}((h/n)^4 \cdot n) = \mathcal{O}(h^4/n^3) \tag{99}$$

For the same total Δz, multi-step RK4 is more accurate. The RK-PINN trades accuracy for speed by learning to correct the single-step error.

10.5 Inference-Time Errors

10.5.1 Extrapolation Beyond Training Domain

If the input $\boldsymbol{x}$ falls outside the training distribution:

$$\boldsymbol{x} \notin \text{support}(P_{\text{train}}) \tag{100}$$

Neural networks can produce arbitrarily wrong predictions. Critical boundaries:

- Momentum: $p < p_{\text{min}}^{\text{train}}$ or $p > p_{\text{max}}^{\text{train}}$
- Angles: $|t_x|, |t_y| >$ training range
- Position: (x, y) outside training envelope

Mitigation: Runtime bounds checking, uncertainty estimation, or ensemble disagreement detection.

10.5.2 Numerical Precision at Inference

Production deployment may use reduced precision:

$$\boldsymbol{y}_{\text{float16}} = \boldsymbol{y}_{\text{float32}} + \epsilon_{\text{quant}} \tag{101}$$

For typical track parameters, float16 error is $\sim 10^{-3}$ relative, which may be acceptable depending on requirements.

10.6 Error Budget Summary

Error Source	**Typical Magnitude**	**Control Method**
Training Data Errors		
RK4 ground truth	10^{-8} (normalized)	Decrease step size
Field interpolation	10^{-5} T	Use finer grid or cubic splines
Gaussian field approx.	10^{-2} T (1.3%)	Use interpolated field map
Sampling bias	Variable	Match physics distributions
Model Errors		
Network capacity	10^{-4}–10^{-3}	Increase depth/width
Optimization	10^{-5}–10^{-4}	Longer training, tuning
Generalization	10^{-4}–10^{-3}	More data, regularization
PINN-Specific Errors		
Collocation discretization	$\mathcal{O}(K^{-1})$	Increase K
Inter-point oscillation	Network-dependent	Smooth activations, more K
Loss balancing (λ)	Variable	Adaptive λ, tuning
Field gradient discontinuity	$\sim 10^{-5}$	Cubic spline interpolation
Inference-Time Errors		
Domain extrapolation	Unbounded	Input validation
Numerical precision (fp16)	$\sim 10^{-3}$ relative	Use fp32 if needed

Table 3: Comprehensive summary of error sources and mitigation strategies.

10.7 Recommended Validation Protocol

To ensure robust deployment, we recommend the following validation checks:

1. **Convergence check**: Verify loss curves have plateaued; early stopping should trigger.
2. **Train/validation/test split**: Report metrics on held-out test set, not training data.
3. **Physics residual analysis** (PINN): Evaluate $R(\zeta)$ on fine grid; ensure no inter-point violations.
4. **Momentum-binned metrics**: Report accuracy separately for low-p, mid-p, high-p tracks.
5. **Comparison to RK4**: On test set, compare against high-precision RK4 integration.
6. **Trajectory validation**: For PINN, compare predicted intermediate states against RK4 trajectories.
7. **Edge case testing**: Evaluate on extreme angles, minimum/maximum momentum, boundary positions.
8. **Charge symmetry**: Verify equal performance for $q > 0$ and $q < 0$ tracks.
9. **Reproducibility**: Train multiple times with different seeds; report mean $\pm$ std of metrics.

11 Conclusion

We have derived the complete mathematical framework for neural network-based track extrapolation:

1. **MLP**: Learns the extrapolation mapping implicitly from data
2. **PINN**: Enforces Lorentz force equations via automatic differentiation
3. **RK-PINN**: Combines RK structure with learnable physics constraints

The key physics are encoded in Equations (7)–(8), with the critical constant $c_{\text{light}} = 2.99792458 \times 10^{-4}$.

A comprehensive error analysis (Section 3) identifies the dominant error sources: training data quality, collocation discretization for PINNs, and domain extrapolation at inference. Rigorous validation protocols are essential before deployment in production.