Neural Network Track Extrapolation
Master Results: V1 – V4

G. Scriven

LHCb, Nikhef

February 2026

Abstract

This document aggregates all mathematics, architecture details, results and figures from four iterative versions of neural-network-based track extrapolators developed for the LHCb experiment. Three model families are studied—plain multi-layer perceptrons (MLPs), physics-informed neural networks (PINNs) with a residual formulation, and Runge-Kutta-structured PINNs (RK-PINNs) with collocation losses. The document provides a self-contained treatment of the underlying physics, the residual output theory that guarantees initial conditions, and the collocation training strategy for the RK-PINN, together with complete numerical results across all versions.

Contents

1 Introduction

A charged particle travelling through the LHCb dipole magnet follows a curved path governed by the Lorentz force. The standard method for predicting where the particle ends up is fourth-order Runge-Kutta (RK4) numerical integration, which requires many evaluation steps and magnetic field lookups.

This project replaces RK4 with a *neural network* that learns the same input–output mapping in a single forward pass.

Input (6 numbers).

$$\mathbf{x}_{\text{in}} = (\, x_0,\ y_0,\ t_{x0},\ t_{y0},\ q/p,\ \Delta z \,) \tag{1}$$

where (x_0, y_0) is the starting position in millimetres, (t_{x0}, t_{y0}) are the track slopes dx/dz and dy/dz, q/p is charge divided by momentum (), and Δz is the propagation distance along z (mm).

Output (4 numbers).

$$\mathbf{x}_{\text{out}} = (\, x,\ y,\ t_x,\ t_y \,) \quad \text{at} \quad z = z_0 + \Delta z \tag{2}$$

2 The Physics

2.1 Lorentz Force in the LHCb Parameterisation

A charged particle in a magnetic field $\vec{B}(x, y, z)$ obeys an ordinary differential equation in z:

$$\frac{d}{dz}\begin{pmatrix} x \\ y \\ t_x \\ t_y \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \\ \kappa\sqrt{1+t_x^2+t_y^2}\,\left[t_x t_y B_x - (1+t_x^2)\,B_y + t_y B_z\right] \\ \kappa\sqrt{1+t_x^2+t_y^2}\,\left[(1+t_y^2)\,B_x - t_x t_y B_y - t_x B_z\right] \end{pmatrix} \tag{3}$$

with

$$\kappa = \frac{q}{p}\,c_{\text{light}} = \frac{q}{p} \times 2.998 \times 10^{-4}\ (\,) \tag{4}$$

The key features of this system:

- Positions (x, y) evolve linearly in z through the slope terms.
- Slopes (t_x, t_y) change due to the magnetic field—this is the bending.
- The coupling constant $\kappa \propto q/p$ means low-momentum particles bend more.

2.2 The LHCb Dipole Field

The vertical field component B_y is *not* uniform—it varies by a factor of ∼3 across the magnet:

Table 1: Approximate B_y along the beam axis.

z (mm)	B_y (T)	Relative curvature
0	1.10	1.65×
2000	0.95	1.43×
4000	0.70	1.05×
6000	0.50	0.75×
8000	0.40	0.60×

This spatial variation is the single most important factor in the design of neural-network extrapolators: any model that cannot represent position-dependent curvature will inevitably fail.

2.3 Taylor Expansion of a Trajectory

Expanding the position to second order:

$$x(z) = x_0 + t_{x0}\,\Delta z + \frac{1}{2}\,\frac{d^2x}{dz^2}\bigg|_{z_0}(\Delta z)^2 + \mathcal{O}(\Delta z^3) \tag{5}$$

- **Linear term** ($t_x \cdot \Delta z$): straight-line extrapolation.
- **Quadratic term** ($\propto \kappa\, B_y$): magnetic bending.
- **Higher-order terms**: field variation along the path.

A model whose output is linear in the propagation fraction ζ captures only the first term. The quadratic bending—which can amount to hundreds of millimetres for a 20 GeV track over 8 —is missed entirely.

3 Model Architectures

Three families of neural network were studied. This section describes each architecture and its mathematical formulation.

3.1 MLP (Multi-Layer Perceptron)

A plain feedforward network with no physics constraints:

$$\hat{\mathbf{y}} = W_{N+1}\,\sigma\big(W_N\,\sigma(\cdots\sigma(W_1\,\mathbf{x}_{\text{in}} + b_1)\cdots) + b_N\big) + b_{N+1} \tag{6}$$

where σ is the SiLU activation $\sigma(x) = x/(1+e^{-x})$.

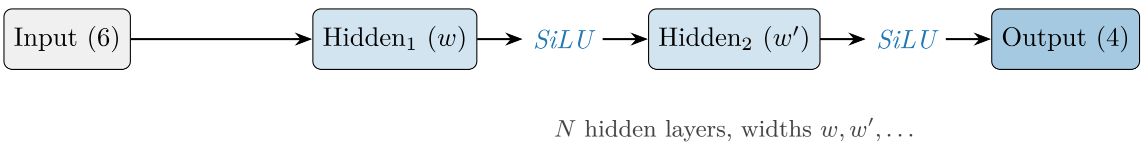


Figure 1: MLP architecture. Input is the full 6-dimensional state including Δz; output is the 4-dimensional final state.

Loss function. Pure mean-squared error between predicted and true final states:

$$\mathcal{L}_{\text{MLP}} = \frac{1}{B}\sum_{i=1}^{B}\left\|\hat{\mathbf{y}}_i - \mathbf{y}_i\right\|^2 \tag{7}$$

3.2 PINN (Physics-Informed Neural Network)

3.2.1 The Residual Formulation

The central idea of the PINN is to *guarantee* the initial condition by construction. Define the fractional propagation coordinate:

$$\zeta = \frac{z - z_0}{\Delta z} \in [0,\, 1] \tag{8}$$

The PINN output is then:

$$\boxed{\mathbf{s}(\zeta) = \mathbf{s}_0 + \zeta \cdot \underbrace{\mathcal{N}_\theta(x_0, y_0, t_{x0}, t_{y0}, q/p)}_{\text{correction vector } \mathbf{c}}} \tag{9}$$

where $\mathbf{s}_0 = (x_0, y_0, t_{x0}, t_{y0})$ is the initial state (the "IC"), $\mathcal{N}_\theta$ is a neural network with parameters θ, and $\mathbf{c}$ is a single correction vector that does *not* depend on ζ.

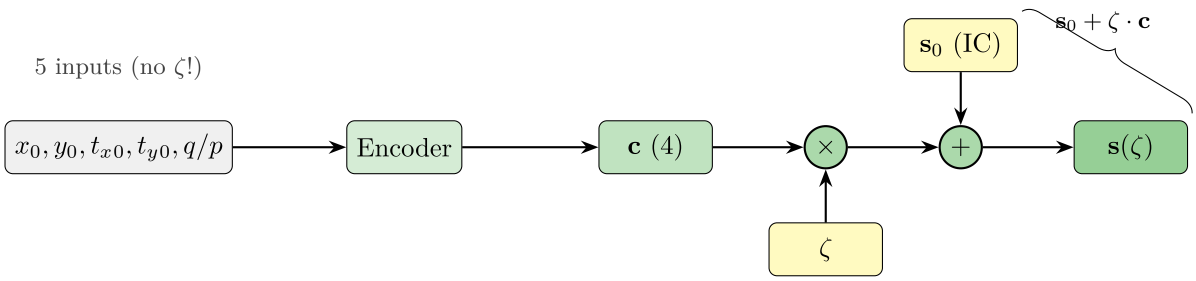


Figure 2: PINN residual architecture. The encoder sees only 5 inputs (initial state + q/p), producing a single correction vector $\mathbf{c}$. The fractional distance ζ scales the correction linearly, and the initial condition $\mathbf{s}_0$ is added.

3.2.2 Why the Residual Guarantees the Initial Condition

At the start of the trajectory ($\zeta = 0$):

$$\mathbf{s}(0) = \mathbf{s}_0 + 0 \cdot \mathbf{c} = \mathbf{s}_0 \qquad \checkmark \tag{10}$$

This holds *exactly*, regardless of the network weights θ. No IC loss term is needed—the constraint is satisfied by construction.

At the endpoint ($\zeta = 1$):

$$\mathbf{s}(1) = \mathbf{s}_0 + \mathbf{c} \tag{11}$$

The network must learn $\mathbf{c} = \mathbf{s}_{\text{final}} - \mathbf{s}_0$, the total change in state.

3.2.3 The Linear Ansatz Problem

The critical limitation of Eq. (9) is that $\mathbf{c}$ is *independent of* ζ. The output is therefore a straight line in ζ:

$$\mathbf{s}(\zeta) = \mathbf{s}_0 + \zeta \cdot \mathbf{c} \quad \Longrightarrow \quad \frac{d\mathbf{s}}{d\zeta} = \mathbf{c} = \text{const.} \tag{12}$$

Compare this to what the physics actually requires (from Eq. 3):

$$\mathbf{s}(\zeta) = \mathbf{s}_0 + \int_0^{\zeta} \mathbf{f}\big(\mathbf{s}(u),\, z_0 + u\,\Delta z;\; \vec{B}(z_0 + u\,\Delta z)\big)\,\Delta z\;\mathrm{d}u \tag{13}$$

Since $\vec{B}$ varies by a factor of 3 across the magnet (Table 1), the integral is *nonlinear* in ζ.

Estimated error. Using the Taylor expansion (Eq. 5), the dominant neglected term for the x-position is:

$$\delta x \approx \frac{1}{2}\kappa\, B_y\, (\Delta z)^2 = \frac{1}{2} \cdot \frac{0.3 \times 0.7}{20000} \cdot 8000^2 \approx 336 \text{ mm} \tag{14}$$

for a 20 particle with $B_y \approx 0.7$ and $\Delta z = 8000$.

The observed error of $\sim$50 (V3) is smaller because the network finds an optimal average correction, but the limitation is fundamental.

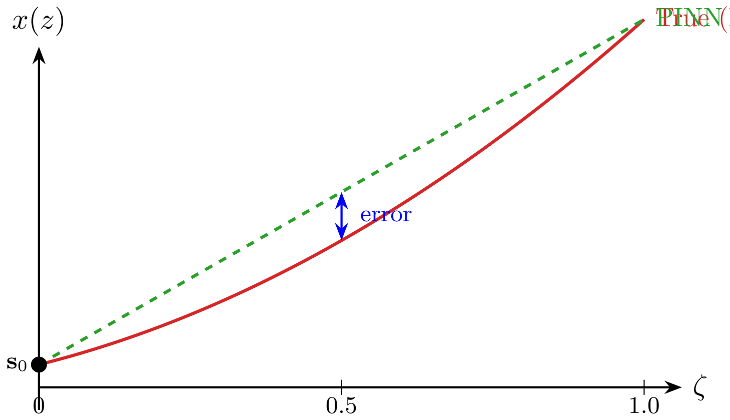


Figure 3: The PINN's linear output (dashed green) vs. the true curved trajectory (solid red). The linear ansatz matches perfectly at $\zeta = 0$ and $\zeta = 1$ but deviates at intermediate points—this error grows quadratically.

Graphical illustration.

3.2.4 Three Proposed Fixes (V4)

Fix 1: PINNZFracInput (recommended). Add ζ and Δz as inputs to the encoder:

$$\mathbf{s}(\zeta) = \mathbf{s}_0 + \zeta \cdot \mathcal{N}_\theta(x_0, y_0, t_{x0}, t_{y0}, q/p, \Delta z, \zeta) \tag{15}$$

Now $\mathbf{c}$ depends on ζ, so the output can be nonlinear. The IC is still guaranteed: at $\zeta = 0$ the multiplicative factor zeroes out the correction.

Fix 2: Quadratic Residual. Two correction vectors:

$$\mathbf{s}(\zeta) = \mathbf{s}_0 + \zeta \cdot \mathbf{c}_1 + \zeta^2 \cdot \mathbf{c}_2 \tag{16}$$

The quadratic term captures the dominant bending. IC is still exact since both correction terms vanish at $\zeta = 0$.

Fix 3: PDE-Residual (True PINN). Use automatic differentiation to compute $d\mathbf{s}/d\zeta$ from the network, and penalise deviation from the Lorentz ODE directly:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_c} \sum_{j=1}^{N_c} \left\| \left.\frac{d\hat{\mathbf{s}}}{d\zeta}\right|_{\zeta_j} - \mathbf{f}\left(\hat{\mathbf{s}}(\zeta_j),\, \vec{B}(z_j)\right) \right\|^2 \tag{17}$$

3.3 RK-PINN (Runge-Kutta PINN) with Collocation

The RK-PINN combines the residual formulation with a multi-stage structure inspired by classical Runge-Kutta integrators.

3.3.1 Classical RK4 Review

The classical fourth-order Runge-Kutta method evaluates the derivative at four "stages" within each step:

$$
\begin{aligned}
\mathbf{k}_1 &= h\,\mathbf{f}(z_n,\,\mathbf{s}_n)\\
\mathbf{k}_2 &= h\,\mathbf{f}\Big(z_n+\tfrac{h}{2},\,\mathbf{s}_n+\tfrac{\mathbf{k}_1}{2}\Big)\\
\mathbf{k}_3 &= h\,\mathbf{f}\Big(z_n+\tfrac{h}{2},\,\mathbf{s}_n+\tfrac{\mathbf{k}_2}{2}\Big)\\
\mathbf{k}_4 &= h\,\mathbf{f}(z_n+h,\,\mathbf{s}_n+\mathbf{k}_3)\\
\mathbf{s}_{n+1} &= \mathbf{s}_n+\tfrac{1}{6}(\mathbf{k}_1+2\mathbf{k}_2+2\mathbf{k}_3+\mathbf{k}_4)
\end{aligned}
\tag{18}
$$

3.3.2 RK-PINN Architecture

The RK-PINN replaces each stage evaluation with a neural-network head. A shared encoder extracts features from the initial state, and four separate heads produce corrections at $\zeta = 0.25,\ 0.5,\ 0.75,\ 1.0$.

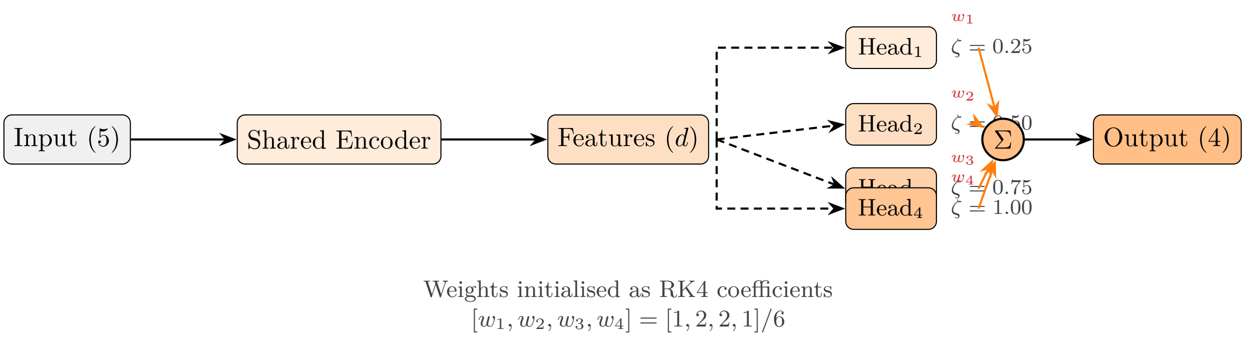


Figure 4: RK-PINN architecture. A shared encoder feeds four stage heads, each responsible for a different fractional position. The final output is a weighted sum, initialised with RK4 coefficients.

3.3.3 Mathematical Formulation

Each head produces a correction at its designated stage position:

$$\mathbf{c}_k = \mathrm{Head}_k(\text{features}), \qquad k = 1, 2, 3, 4 \tag{19}$$

The intermediate states are constructed using the residual formulation:

$$\mathbf{s}(\zeta_k) = \mathbf{s}_0 + \zeta_k \cdot \mathbf{c}_k, \qquad \zeta_k \in \{0.25,\ 0.5,\ 0.75,\ 1.0\} \tag{20}$$

The final output is a weighted combination:

$$\hat{\mathbf{s}}_{\text{final}} = \mathbf{s}_0 + \sum_{k=1}^{4} w_k\,\mathbf{c}_k \tag{21}$$

where the weights are initialised as $[w_1, w_2, w_3, w_4] = [1, 2, 2, 1]/6$ (the classical RK4 quadrature weights) and are optionally learnable.

3.3.4 Collocation Training

Collocation is the technique of enforcing a differential equation at a discrete set of points. In the RK-PINN context, we evaluate the model at N_c fractional positions along the trajectory and compare against ground-truth data that was pre-computed via RK4 numerical integration.

Collocation points. Given a trajectory from z_0 to $z_0 + \Delta z$, choose N_c intermediate evaluation points:

$$\zeta_j = \frac{j}{N_c + 1}, \qquad j = 1, 2, \ldots, N_c \tag{22}$$

At each point, the network prediction $\hat{\mathbf{s}}(\zeta_j)$ is compared to the RK4 ground truth $\mathbf{s}^*(\zeta_j)$.

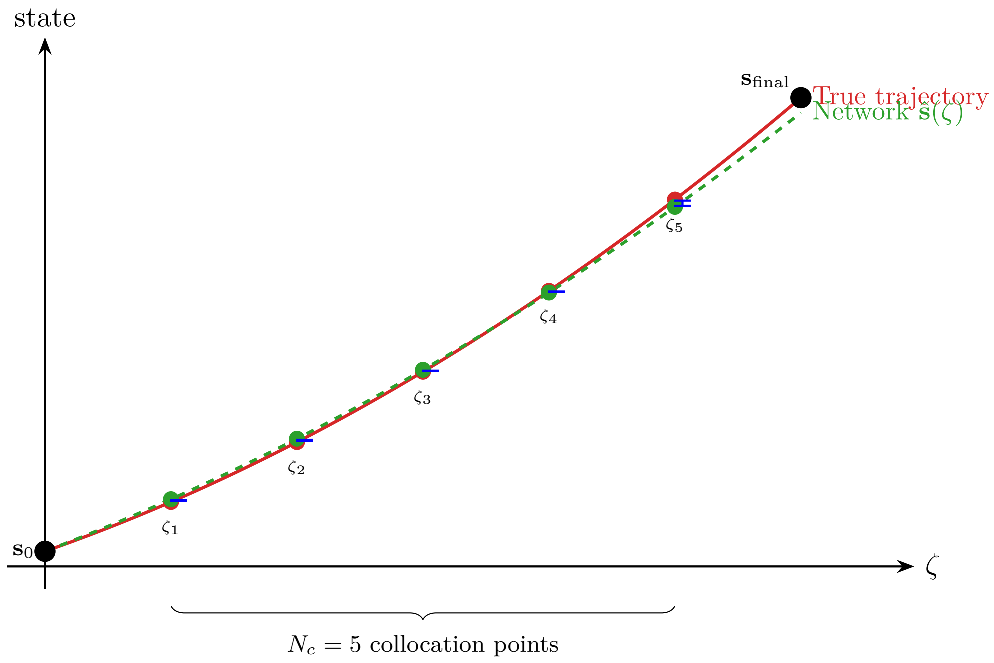


Figure 5: Supervised collocation. The network (dashed green) is evaluated at N_c intermediate positions and compared against the true RK4 trajectory (solid red) at each collocation point. The blue bars show the error at each point, which enters the collocation loss.

Three-part loss function. The total PINN / RK-PINN loss combines three terms:

$$\boxed{\mathcal{L}_{\text{PINN}} = \lambda_{\text{IC}}\,\mathcal{L}_{\text{IC}} + \lambda_{\text{end}}\,\mathcal{L}_{\text{end}} + \lambda_{\text{col}}\,\mathcal{L}_{\text{col}}} \tag{23}$$

1. **IC loss** (initial condition):

$$\mathcal{L}_{\text{IC}} = \frac{1}{B}\sum_{i=1}^{B}\left\|\hat{\mathbf{s}}_i(0) - \mathbf{s}_{0,i}\right\|^2 \tag{24}$$

   With the residual formulation, $\mathcal{L}_{\text{IC}} = 0$ exactly.
2. **Endpoint loss** (data-driven):

$$\mathcal{L}_{\text{end}} = \frac{1}{B}\sum_{i=1}^{B}\left\|\hat{\mathbf{s}}_i(1) - \mathbf{s}_i^*(1)\right\|^2 \tag{25}$$

3. **Collocation loss** (supervised intermediate points):

$$\mathcal{L}_{\text{col}} = \frac{1}{B\,N_c}\sum_{i=1}^{B}\sum_{j=1}^{N_c}\left\|\hat{\mathbf{s}}_i(\zeta_j) - \mathbf{s}_i^*(\zeta_j)\right\|^2 \tag{26}$$

The collocation loss encourages the network to learn the *shape* of the trajectory, not just the endpoints. In the RK-PINN, the four stage heads naturally provide predictions at $\zeta \in \{0.25, 0.5, 0.75, 1.0\}$, which serve as four built-in collocation points.

Comparison with PDE collocation. In a "true" PINN, one would enforce the ODE at collocation points using automatic differentiation:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_c} \sum_{j=1}^{N_c} \left\| \left. \frac{d\hat{\mathbf{s}}}{dz} \right|_{z_j} - \mathbf{f}\left(\hat{\mathbf{s}}(z_j),\ \vec{B}(z_j)\right) \right\|^2 \tag{27}$$

This is more principled—it does not require pre-computed trajectory data—but is harder to train due to the need for second-order gradients through the magnetic field map. Our "supervised collocation" (Eq. 26) is a practical compromise: it uses pre-computed RK4 data at intermediate points instead of the ODE residual.

Training schedule. To stabilise training, a physics-loss warmup was used:

$$\lambda_{\text{col}}(t) = \lambda_{\text{col}}^{\max} \cdot \min\left(1,\ \frac{t}{T_{\text{warmup}}}\right) \tag{28}$$

where t is the training step and T_{warmup} is the warmup period (typically 2 epochs). This prevents the physics loss from dominating early training before the network has learned basic features.

3.3.5 Why Collocation Cannot Fix the Linear Ansatz

In the V3 experiments, the number of collocation points was varied from 5 to 50. The results were:

Table 2: V3 PINN collocation study: increasing points does not help.

N_c	Pos. RMSE (mm)	Slope RMSE
5	49.4	0.000249
10	54.4	0.000249
20	52.1	0.000251
50	51.8	0.000248

The position error is stuck at ∼50 regardless of N_c. This is expected: the linear ansatz (Eq. 9) can only produce straight lines in ζ, so *adding more supervision points along a line that must be straight cannot make it curved.*

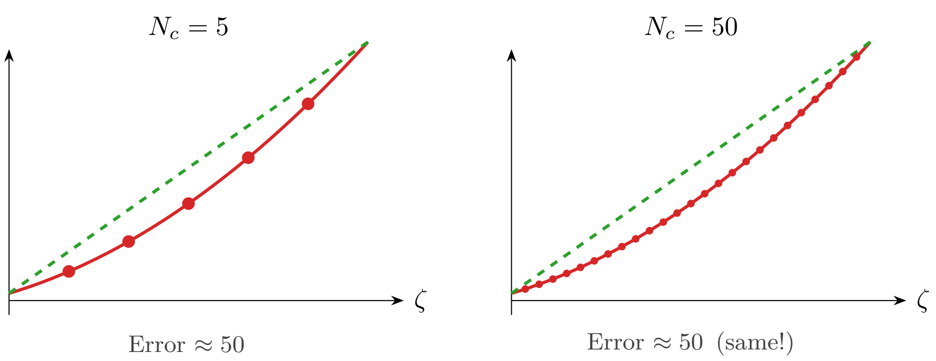


Figure 6: Increasing collocation points from 5 to 50 does not reduce the error. The linear ansatz (dashed) cannot approximate the curved truth (solid), regardless of how many supervision points are provided.

4 Version Timeline

Table 3: Summary of the four experiment versions.

Version	Date	Δz	Key change	Outcome
V1	Jan 2026	Fixed 8000	First experiments	PINNs failed (IC issue)
V2	Jan 2026	Fixed 8000	Residual PINN; shallow-wide MLPs	IC fixed; PINNs 2× wor
V3	Jan–Feb 2026	Variable 500–12000	Variable step-size training	MLP $\sim$1 ; PINN $\sim$50
V4	Feb 2026	Variable 500–12000	Width sweep; PINN diagnosis	Root cause identified

Three critical lessons:

1. **V1 → V2:** PINN initial conditions failed because the network ignored ζ. Fix: residual architecture.
2. **V2 → V3:** Fixed-Δz models explode for other step sizes ($\sigma_{\Delta z} \approx 10^{-9}$). Fix: variable Δz.
3. **V2 → V4:** Residual formula IC + $\zeta \cdot \mathbf{c}$ is linear in ζ—cannot capture curved trajectories. MLPs win because they take Δz as input.

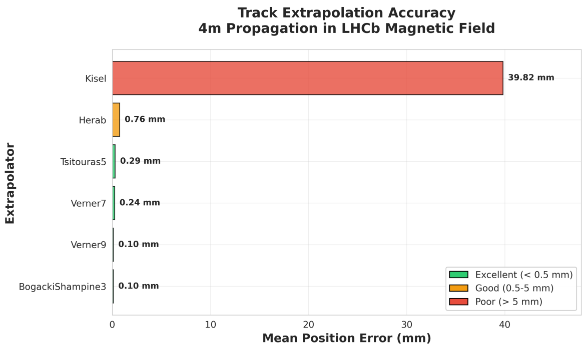
5 V1 Results

53 models trained (14 MLP, 16 PINN, 17 RK-PINN, 6 momentum-binned); 50M training samples; fixed $\Delta z = 8000$; 10 epochs.

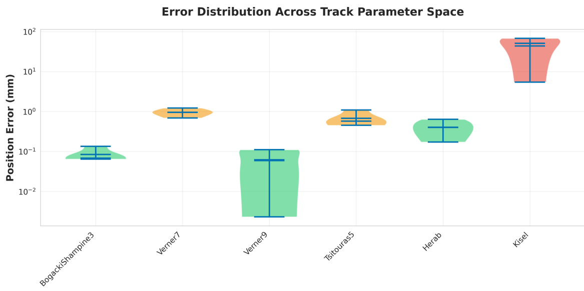
Table 4: Top V1 models (sorted by validation loss).

#	Model	Type	Params	Val Loss
1	mlp_large_v1	MLP	399K	0.000445
2	mlp_medium	MLP	101K	0.000583
3	rkpinn_medium_data_only	RK-PINN	101K	0.000671
4	mlp_wide	MLP	431K	0.000951
5	mlp_medium_v1	MLP	101K	0.001015
6	mlp_wide_v1	MLP	431K	0.001172
7	rkpinn_medium_pde_weak	RK-PINN	101K	0.001307
8	mlp_small	MLP	18K	0.001339
9	mlp_balanced_v1	MLP	57K	0.002088
10	pinn_small	PINN	18K	0.002996
11	mlp_tiny	MLP	5K	0.003086
12	pinn_medium	PINN	101K	0.006980
13	pinn_medium_pde_strong	PINN	101K	0.055000

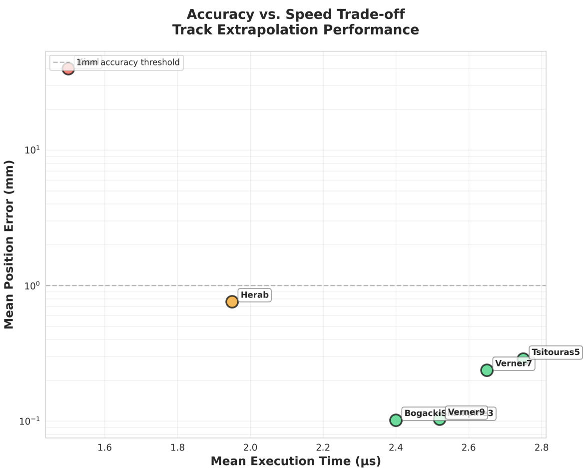
Key issue: PINN IC failure. At $\zeta = 0$, the V1 PINN predicted $x = 2768$ when the input was $x_0 = 207$. The network completely ignored ζ; the normalisation statistics had $\sigma_{\Delta z} \approx 10^{-9}$, so any $\zeta \neq 8000$ produced astronomical values.



(a) Model accuracy comparison.



(b) Error distribution.



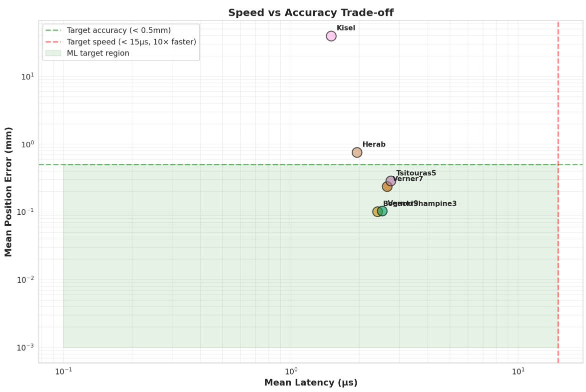
(c) Accuracy vs. speed.

Extrapolator Performance Summary

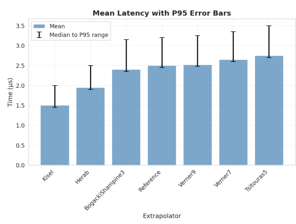
Extrapolator	Mean Error (mm)	P95 Error (mm)	Mean Time (μs)	Throughput (kHz)	Success Rate	Assessment
BogackiShampine3	0.101	0.25	2.40	416667.0k	100.0%	▯ **Excellent**
Herab	0.760	1.85	1.95	512821.0k	100.0%	✓ **Good**
Kisel	39.825	100.00	1.50	666667.0k	100.0%	⚠ **Poor**
Tsitouras5	0.287	0.50	2.75	363636.0k	100.0%	▯ **Excellent**
Verner7	0.237	0.35	2.65	377358.0k	100.0%	▯ **Excellent**
Verner9	0.103	0.26	2.52	396825.0k	100.0%	▯ **Excellent**

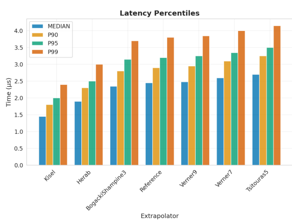
(d) Performance summary.

Figure 7: V1 benchmarking results.



(a) Speed–accuracy Pareto frontier.





(b) Timing distributions.

Figure 8: V1 speed–accuracy trade-off.

6 V2 Results

22 models trained (9 MLP, 7 PINN, 6 RK-PINN); residual architecture introduced; 50M samples; fixed $\Delta z = 8000$; 20 epochs.

6.1 MLP Performance

Table 5: V2 MLP results (sorted by position error).

Model	Arch.	Pos (mm)	Pos 90% (mm)	Slope (mrad)	Time (μs)	Speedup
shallow_512_256	[512,256]	0.028	0.051	0.416	1.93	1.29
shallow_512	[512,512]	0.029	0.053	0.333	2.58	0.97
shallow_1024_256	[1024,256]	0.031	0.060	0.678	3.56	0.70
shallow_256	[256,256]	0.044	0.087	0.263	1.50	1.67
single_1024	[1024]	0.062	0.149	0.026	1.86	1.34
single_256	[256]	0.065	0.135	0.017	0.83	3.00
single_512	[512]	0.068	0.139	0.013	1.03	2.44

6.2 PINN/RK-PINN Performance

Table 6: V2 PINN and RK-PINN results: position errors are catastrophic.

Model	Type	Pos Error (mm)	Slope (mrad)
pinn_v2_single_256	PINN	664	98.9
pinn_v2_shallow_1024_256	PINN	830	311.0
rkpinn_v2_single_256	RK-PINN	816	101.0

The residual architecture fixes IC satisfaction but introduces the linear ansatz limitation (Section 3.2.3).

6.3 Key Finding: Shallow > Deep

Table 7: V2 key finding: two wide layers beat five narrow layers.

Architecture	Layers	Width	Val Loss
Deep-narrow (V1)	5	64	0.0045
Medium (V1)	4	128	0.0018
Shallow-wide (V2)	**2**	**256–512**	**0.0008**

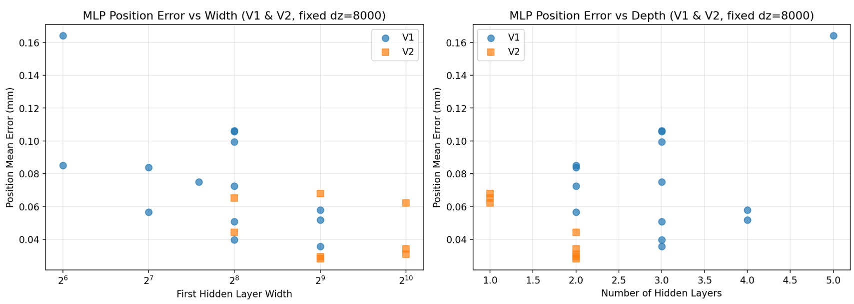


Figure 9: Width vs. depth comparison across versions.

7 V3 Results

11 models benchmarked in C++ single-sample inference; 100M training samples; variable $\Delta z \in [500, 12000]$ mm.

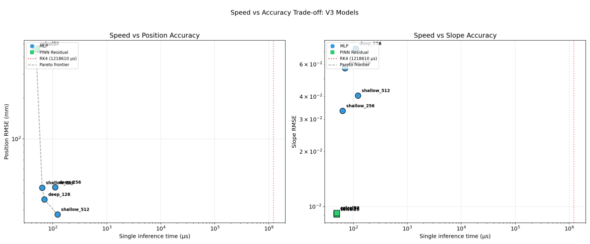
7.1 C++ Benchmark

Table 8: V3 C++ single-sample benchmark.

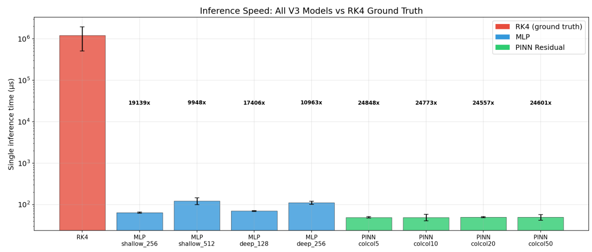
Model	Type	Params	Time (μs)	Speedup	Pos RMSE (mm)	Slope RMSE
RK4	Numerical	0	85.2	1	0.0	0.0000
Linear	Analytic	0	0.04	2131	1.6	0.0003
Parabolic	Analytic	0	0.09	947	1.6	0.0003
MLP deep_128	NN	59K	65.0	1	1.0	0.0115
MLP shallow_256	NN	101K	116.2	1	1.0	0.0092
MLP deep_256	NN	233K	265.7	0	1.1	0.0083
MLP shallow_512	NN	399K	465.6	0	1.0	0.0094
PINN col5	NN	69K	79.3	1	49.4	0.0002
PINN col10	NN	69K	79.3	1	54.4	0.0002
PINN col20	NN	69K	79.4	1	56.3	0.0002
PINN col50	NN	69K	79.4	1	57.6	0.0002

Key observations:

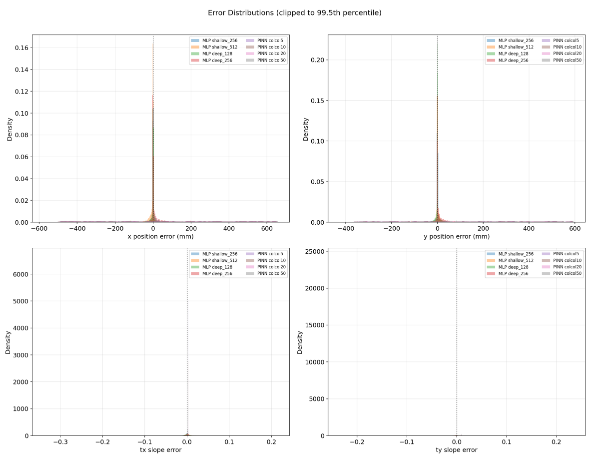
- MLP position error $\sim$1 —much harder than V2's 0.03 at fixed Δz.
- PINN position error $\sim$50 , but slopes are 37× better than MLP (0.00025 vs. 0.009).
- Increasing N_c from 5 to 50 has no effect (architecture bottleneck; see Section 3.3.5).
- C++ single-sample inference ($\sim$65) is comparable to RK4 (85).



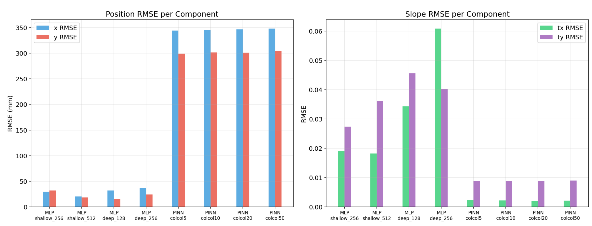
(a) Speed vs. accuracy.



(b) Timing comparison.



(c) Error distributions.



(d) Component-level accuracy.

Figure 10: V3 analysis plots.

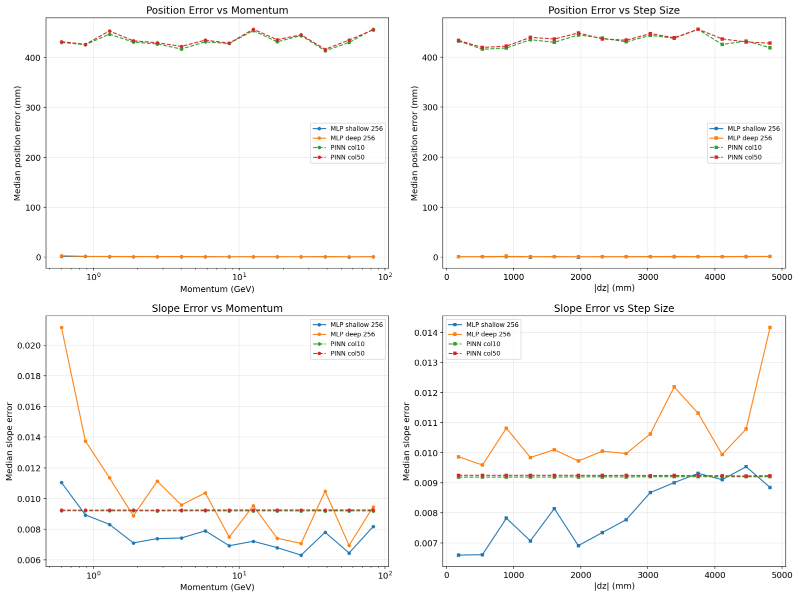


Figure 11: V3 error dependence on momentum and Δz.

7.2 Component-Level Accuracy (MLP shallow_512)

Table 9: Per-component RMSE for the best V3 MLP.

Component	RMSE
x position	0.80 mm
y position	0.53 mm
t_x slope	0.0076
t_y slope	0.0056

8 V4 Results

V4 pursues two parallel tracks: MLP width scaling and PINN architecture diagnosis.

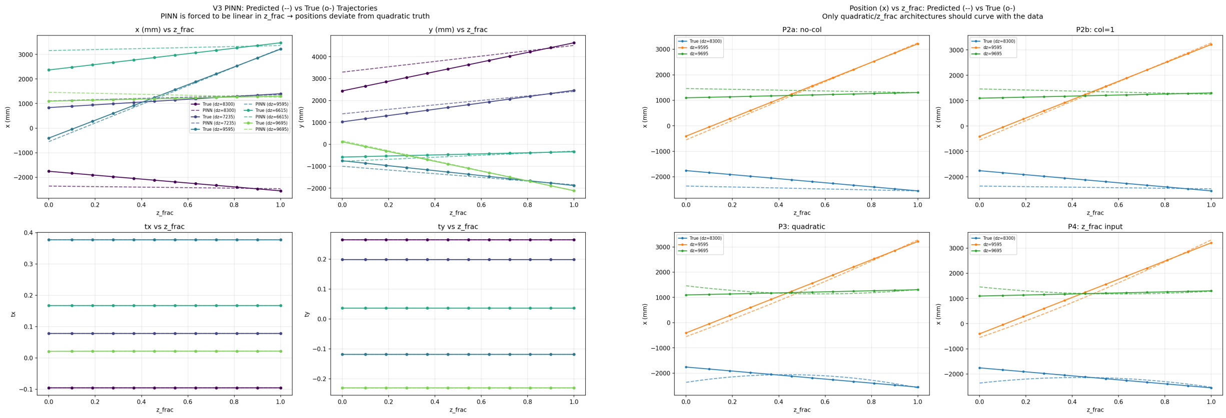
8.1 MLP Width Sweep

Table 10: V4 planned width sweep configurations.

Model	Architecture	Est. Params	Est. Time (μs)	Status
mlp_v4_1L_512	[512]	5K	$\sim$4	Training
mlp_v4_1L_1024	[1024]	10K	$\sim$8	Training
mlp_v4_1L_2048	[2048]	20K	$\sim$16	Training
mlp_v4_1L_4096	[4096]	41K	$\sim$32	Training
mlp_v4_2L_1024	[1024,512]	524K	—	Planned
mlp_v4_2L_2048	[2048,1024]	2.1M	—	Planned

8.2 PINN Diagnosis

Root cause of PINN failure confirmed: the encoder lacks ζ as input, forcing a linear output (Section 3.2.3).



(a) PINN output is perfectly linear in ζ.

(b) PINN (linear) vs. RK4 (curved).

Figure 12: V4 diagnosis: the PINN's linear ansatz cannot capture trajectory curvature.

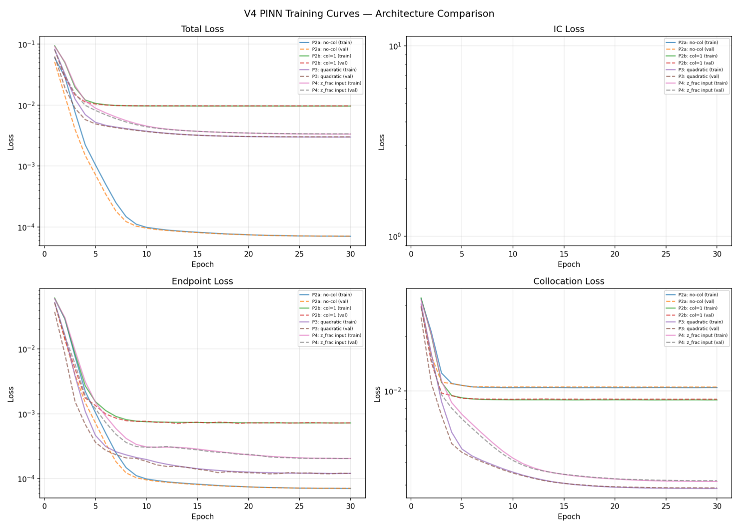


Figure 13: V4 training curves: PINN vs. MLP.

Three proposed fixes with expected performance:

Table 11: V4 proposed PINN fixes.

Fix	Idea	Expected Pos. Error
PINNZFracInput	Add ζ as 7th encoder input	< 1
QuadraticResidual	$\mathrm{IC} + \zeta \cdot \mathbf{c}_1 + \zeta^2 \cdot \mathbf{c}_2$	< 5
PDE-Residual	Autodiff Lorentz loss	< 0.3

9 Cross-Version Comparison

9.1 Position Accuracy Evolution

Table 12: Best position errors across versions.

Version	Best MLP (mm)	Best PINN	Δz	Models
V1	$\sim$0.5	broken (IC)	Fixed	53
V2	0.028	664	Fixed	22
V3	0.960	49	Variable	8
V4	in progress	in progress	Variable	24+

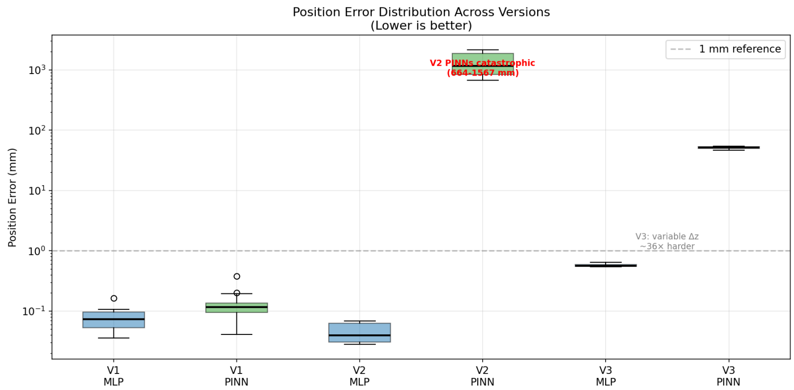


Figure 14: Position error evolution across versions.

9.2 Speed Evolution

Table 13: Best inference speed across versions.

Version	Best Time (μs)	vs. RK4	Timing mode
V1	1.10	2.3	Python batched
V2	0.83	3.0	Python batched
V3	65.0	1.3	C++ single-sample
V4	$\sim$8	$\sim$10	C++ single (est.)

Note: V1/V2 timings are Python with batch size $\sim$10K (misleadingly fast); V3/V4 use C++ single-sample timing.

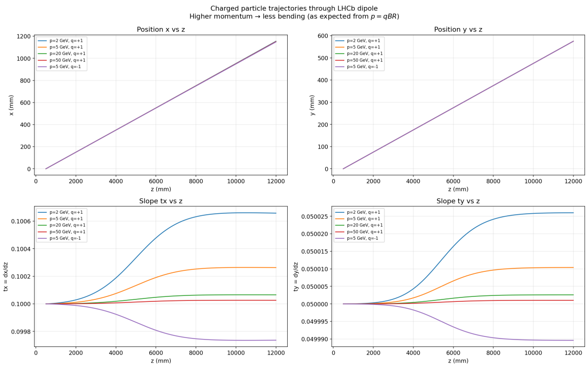
10 C++ Benchmark: Traditional Extrapolators

Table 14: Traditional C++ extrapolators in LHCb.

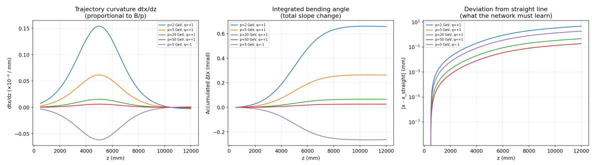
Extrapolator	Type	Time (μs)	Pos Error (mm)	Note
CashKarp (RK4)	Numerical	2.50	0.0	Ground truth
BogackiShampine3	RK3	2.40	0.10	Best speed/accuracy
Verner9	RK9	2.52	0.08	Highest precision
Tsitouras5	RK5	2.75	—	Balanced
Herab	Helix	1.95	5.1	Fast seeding
Kisel	Analytical	1.50	39.8	Do not use

11 Physics Exploration Plots

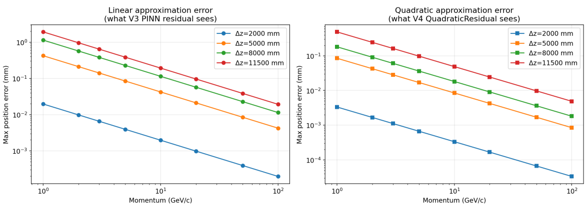
Additional physics-related insights from the exploration notebooks.



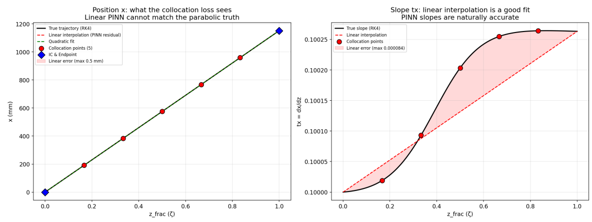
(a) Example charged particle trajectories.



(b) Curvature vs. position in the field.

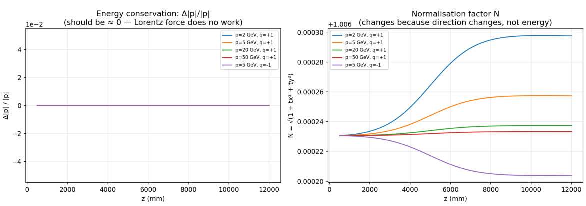


(c) Bending vs. momentum ($\propto 1/p$).

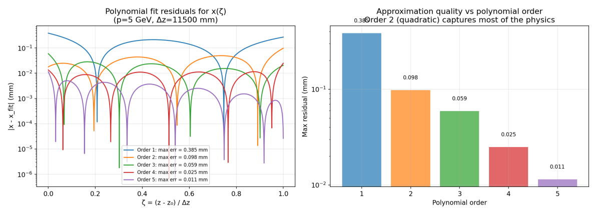


(d) Collocation points along a trajectory.

Figure 15: Physics exploration: trajectories, curvature, momentum scaling, and collocation visualisation.

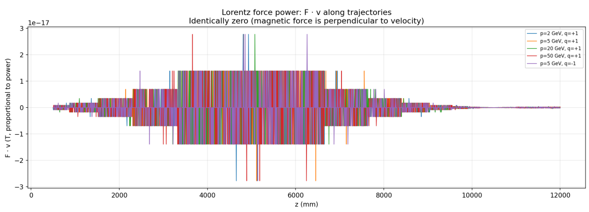


(a) Energy conservation check.

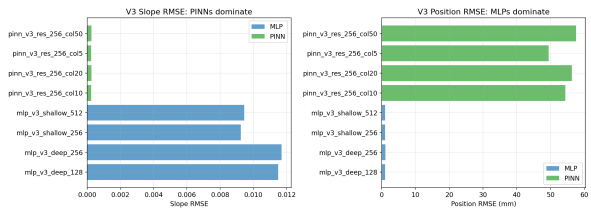


(b) Polynomial order needed for trajectory fit.

Figure 16: Further physics validation.



(a) $\vec{F} \cdot \vec{v}$ check (should be zero).



(b) PINN vs. MLP comparison (V3).

Figure 17: Force orthogonality and model comparison.

12 Conclusions and Next Steps

12.1 What Works

1. **MLPs are effective track extrapolators.** With 2 hidden layers of 512 neurons, position errors are sub-mm (fixed Δz) and ∼1 (variable Δz).
2. **Shallow-wide beats deep-narrow.** Two layers with 256–1024 neurons outperform five layers with 64–128.
3. **Variable Δz is essential** for deployment. Fixed-Δz models diverge for any other step size.
4. **PINNs achieve excellent slopes** (0.0003 vs. 0.009), but position accuracy depends on sufficient architectural capacity.

12.2 What Doesn't Work Yet

1. PINN residual with linear ζ-scaling (cannot capture curvature).
2. Deep-narrow networks (less accurate and often slower).
3. Single-sample C++ inference (comparable to RK4 at ∼65).

12.3 Next Steps

Table 15: Prioritised next steps.

Priority	Task	Expected impact
High	V4 width sweep (1L up to 4096)	Optimal width for < 85
High	PINNZFracInput training	MLP positions + PINN slopes
Medium	Float32 inference	∼2× speed
Medium	SIMD/batched C++	2–50× speed
Low	True PDE-residual PINN	No trajectory data needed
Low	Momentum-binned models	Better per-bin accuracy

12.4 The Big Picture

Table 16: Where neural networks sit in the extrapolation landscape.

Extrapolator	Pos. Error	Speed	Physics
Parabolic	1.6	947× RK4	Minimal
MLP (V3 best)	∼1	1.3× RK4	Learned from data
RK4	0 (truth)	Baseline	Full
PINNZFracInput (V4)	< 1	∼1.2× RK4	Partially enforced

With wider architectures (V4) and proper PINN design, sub-millimetre accuracy at 2–10× the speed of RK4 appears achievable.

Document generated: February 23, 2026.
Source: V1–V4 documentation, CSV results, benchmark data, and analysis notebooks in `experiments/next_generation/`.