Neural Network Track Extrapolator

V1 Training Analysis Report

G. Scriven
LHCb Collaboration
NIKHEF, Amsterdam

January 2026

Abstract

We present the results of V1 training for neural network-based track extrapolation in the LHCb detector. Three architecture types were explored: Multi-Layer Perceptron (MLP), Physics-Informed Neural Network (PINN), and Runge-Kutta Physics-Informed Neural Network (RK_PINN). A total of 53 model configurations were trained on 50 million track extrapolation samples. Our findings show that small MLP architectures achieve inference times competitive with existing C++ extrapolators (1.1–2.2 μs vs 1.5–2.5 μs) while maintaining sub-millimeter position accuracy. Contrary to expectations, physics-informed approaches did not improve accuracy over pure data-driven MLPs, while significantly increasing computational cost.

1 Introduction

Track extrapolation is a fundamental operation in particle physics reconstruction, required for pattern recognition, track fitting, and physics analysis. In LHCb, charged particles traverse the detector's magnetic field, following curved trajectories governed by the Lorentz force. The standard approach uses numerical integration (Runge-Kutta methods) of the equations of motion, which is accurate but computationally intensive.

This study investigates replacing the C++ Cash-Karp Runge-Kutta extrapolator with neural networks trained on simulated trajectory data. The potential benefits include:

- **Speed:** Neural network inference can be parallelised on GPUs and optimised for specific hardware
- **Simplicity:** Single forward pass vs. iterative numerical integration
- **Differentiability:** Enables gradient-based optimisation of downstream tasks

The V1 training campaign explored three architecture families:

1. **MLP:** Standard feedforward networks trained with supervised learning
2. **PINN:** Networks augmented with physics-based loss terms
3. **RK_PINN:** Multi-stage architecture inspired by RK4 numerical integration

2 Mathematical Foundations

2.1 Track Extrapolation Problem

A charged particle in a magnetic field follows a trajectory governed by the Lorentz force:

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

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

In LHCb, tracks are parameterised along the beam axis z. The state vector at position z is:

$$\mathbf{y}(z) = \begin{pmatrix} x \\ y \\ t_x \\ t_y \end{pmatrix} \tag{2}$$

where (x, y) is the transverse position and $(t_x, t_y) = (\mathrm{d}x/\mathrm{d}z, \mathrm{d}y/\mathrm{d}z)$ are the track slopes.

The equations of motion in z-parameterisation are:

$$\frac{\mathrm{d}x}{\mathrm{d}z} = t_x \tag{3}$$

$$\frac{\mathrm{d}y}{\mathrm{d}z} = t_y \tag{4}$$

$$\frac{\mathrm{d}t_x}{\mathrm{d}z} = \kappa N \left[t_x t_y B_x - (1 + t_x^2) B_y + t_y B_z \right] \tag{5}$$

$$\frac{\mathrm{d}t_y}{\mathrm{d}z} = \kappa N \left[(1 + t_y^2) B_x - t_x t_y B_y - t_x B_z \right] \tag{6}$$

where:

- $\kappa = (q/p) \cdot c_{\text{light}}$ with $c_{\text{light}} = 2.99792458 \times 10^{-4}$ GeV/(T·mm)
- $N = \sqrt{1 + t_x^2 + t_y^2}$ is the normalisation factor
- q/p is the charge-over-momentum in $1/\text{MeV}$

2.2 Neural Network Formulation

The extrapolation task is formulated as a regression problem:

$$f_\theta : \mathbb{R}^6 \to \mathbb{R}^4 \tag{7}$$

$$\mathbf{x} = (x_0, y_0, t_{x,0}, t_{y,0}, q/p, \Delta z) \mapsto \mathbf{y}_f = (x_f, y_f, t_{x,f}, t_{y,f}) \tag{8}$$

where $\mathbf{x}$ is the input (initial state + step size) and $\mathbf{y}_f$ is the predicted final state.

3 Network Architectures

3.1 MLP (Multi-Layer Perceptron)

The MLP is a standard feedforward network (Figure 1):

$$\mathbf{y} = \sigma_{\text{out}} \circ W_L \circ \sigma \circ W_{L-1} \circ \cdots \circ \sigma \circ W_1(\hat{\mathbf{x}}) \tag{9}$$

where $\hat{\mathbf{x}}$ is the normalised input, W_i are linear transformations, and σ is the SiLU activation:

$$\sigma_{\text{SiLU}}(x) = x \cdot \text{sigmoid}(x) = \frac{x}{1 + e^{-x}} \tag{10}$$

Training loss: Pure supervised learning

$$\mathcal{L}_{\text{MLP}} = \frac{1}{N} \sum_{i=1}^{N} \|\hat{\mathbf{y}}_i - \mathbf{y}_i^*\|^2 \tag{11}$$

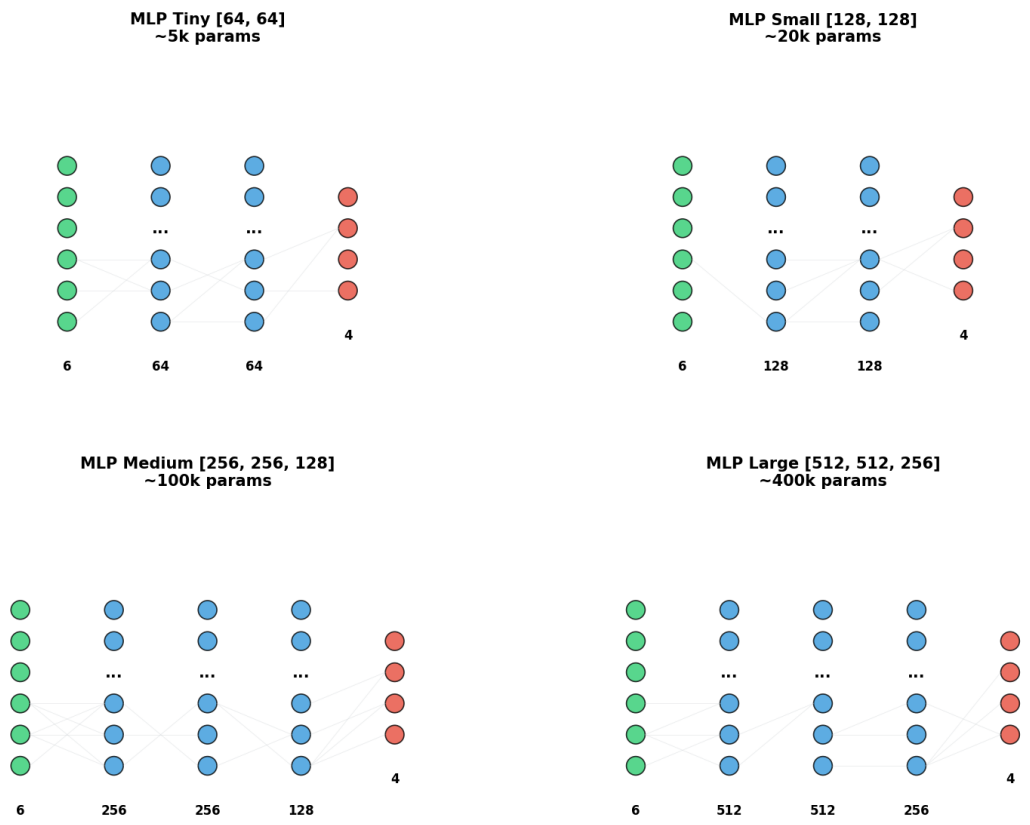

Figure 1: MLP architecture configurations explored in V1 training, ranging from Tiny (5k parameters) to Large (400k parameters). Green: input layer (6 features). Blue: hidden layers. Red: output layer (4 state components).

3.2 PINN (Physics-Informed Neural Network)

The PINN uses the same network structure but augments the loss with physics constraints (Figure 2):

$$\mathcal{L}_{\text{PINN}} = \mathcal{L}_{\text{data}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{PDE}}\mathcal{L}_{\text{PDE}} \tag{12}$$

Initial Condition Loss: Ensures the network prediction at $z = 0$ matches the input state:

$$\mathcal{L}_{\text{IC}} = \|f_\theta(\mathbf{x}_0, z = 0) - \mathbf{x}_0\|^2 \tag{13}$$

PDE Residual Loss: Enforces the Lorentz force equations at collocation points z_i:

$$\mathcal{L}_{\text{PDE}} = \sum_{i=1}^{N_c} \left\| \frac{\partial \hat{\mathbf{y}}_i}{\partial z} - \mathbf{F}(\hat{\mathbf{y}}_i, \mathbf{B}(z_i)) \right\|^2 \tag{14}$$

where the derivatives are computed via automatic differentiation.

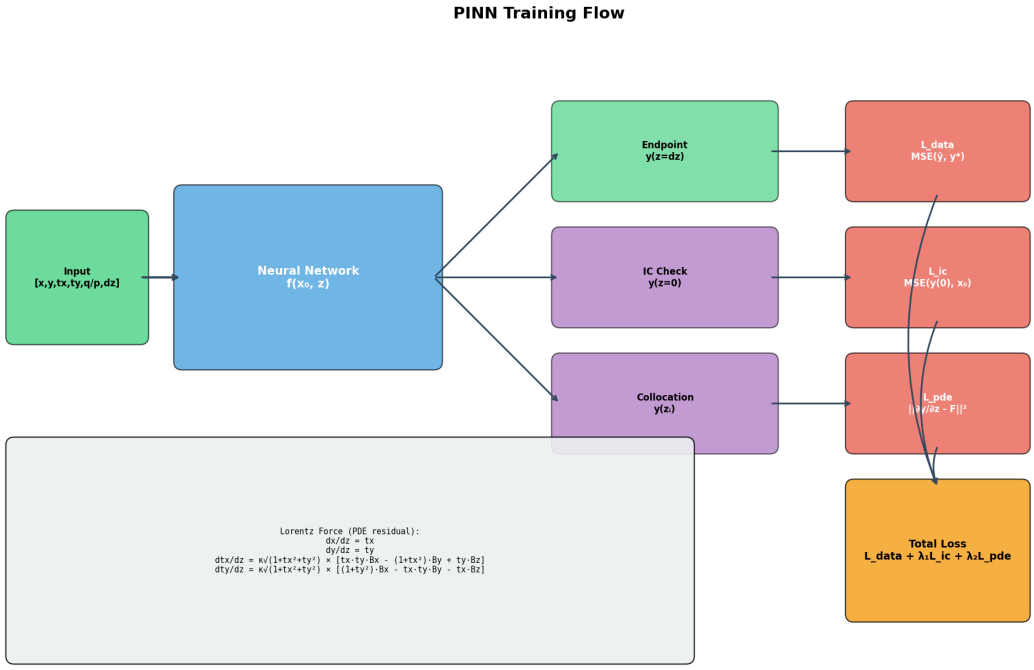

Figure 2: PINN training flow showing the three loss components: data loss at endpoint, IC loss at $z = 0$, and PDE residual at collocation points.

3.3 RK_PINN (Runge-Kutta PINN)

The RK_PINN architecture is inspired by the classical RK4 integration scheme (Figure 4):

Classical RK4:

$$k_1 = f(t_n, y_n) \tag{15}$$

$$k_2 = f(t_n + h/2, y_n + hk_1/2) \tag{16}$$

$$k_3 = f(t_n + h/2, y_n + hk_2/2) \tag{17}$$

$$k_4 = f(t_n + h, y_n + hk_3) \tag{18}$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{19}$$

RK_PINN Network:

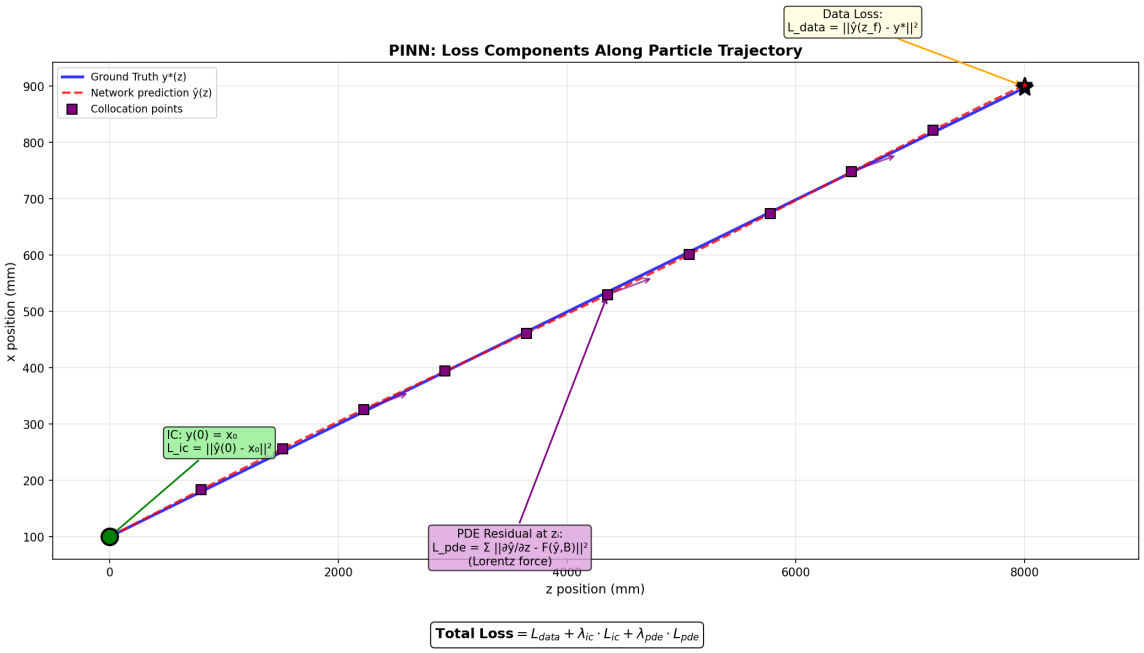

Figure 3: Visualisation of PINN loss components along a particle trajectory. Green: initial condition check. Purple: PDE residual at collocation points. Yellow: data loss at endpoint.

- Shared backbone extracts features from initial state
- Four stage heads predict state at $z = \{0.25, 0.5, 0.75, 1.0\} \cdot \Delta z$
- Learnable combination weights (initialised to RK4: $[1, 2, 2, 1]/6$)

$$\hat{\mathbf{y}}_{\text{final}} = \sum_{k=1}^{4} w_k \cdot \hat{\mathbf{y}}_k, \quad w_k = \text{softmax}(\tilde{w}_k) \tag{20}$$

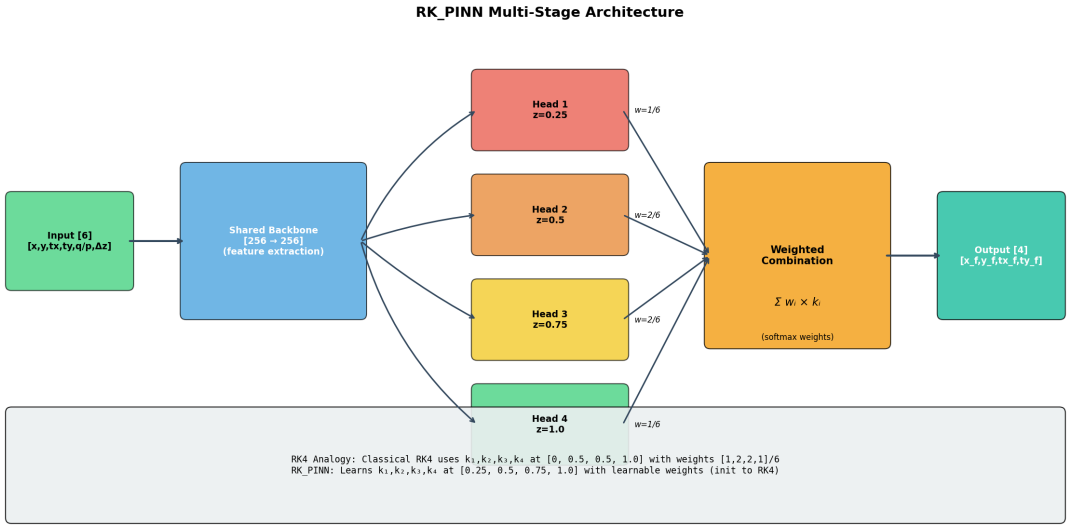

Figure 4: RK_PINN multi-stage architecture with shared backbone and four stage heads at fractional z positions.

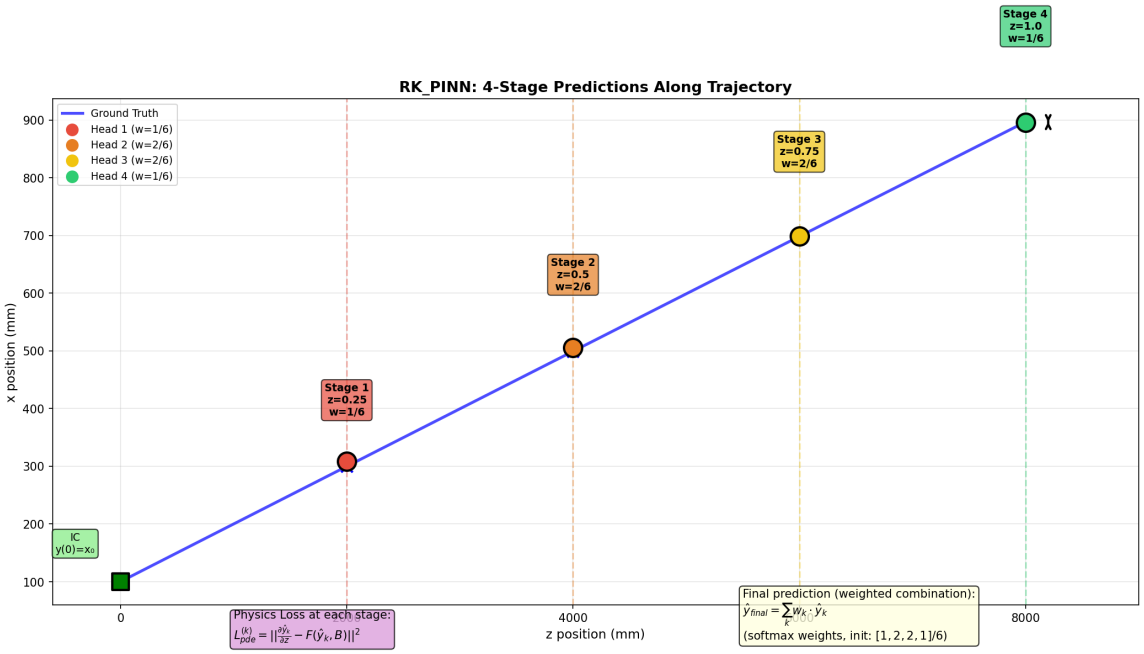

Figure 5: RK_PINN 4-stage predictions along a trajectory. Each stage contributes to the final prediction with learnable weights initialised to RK4 values.

3.4 Architecture Comparison

Figure 6 summarises the key differences between the three architecture types.

4 Training Configuration

4.1 Dataset

Training data was generated using a C++ RK4 integrator with the LHCb magnetic field map:

- **Samples:** 50 million track extrapolations
- **Split:** 80% train, 10% validation, 10% test
- **Field model:** Interpolated from `twodip.rtf` field map
- **z range:** 0–9500 mm (full detector acceptance)
- **Momentum:** 1–100 GeV uniformly sampled

4.2 Training Parameters

4.3 Model Configurations

A total of 53 models were trained across the three architecture types:

5 Results

5.1 Accuracy Comparison

Table 3 shows the best performing models from V1 training:

Key observation: The top performing models are all MLPs or RK_PINNs with physics loss disabled ($\lambda_{\text{PDE}} = 0$). Pure PINN models with physics constraints perform worse.

Figure 6: Comparison of MLP, PINN, and RK_PINN architectures showing network structure, loss functions, and characteristics.

Parameter	**Value**
Optimiser	AdamW
Learning rate	10^{-3}
Weight decay	10^{-4}
Batch size	8192
Epochs	10
Scheduler	Cosine annealing
Warmup epochs	5
Gradient clipping	1.0

Table 1: Training hyperparameters used for V1 models.

Type	**Hidden Dims**	**Params**	**Best Val Loss**	**Count**
MLP	[64, 64] (tiny)	5k	0.0031	2
MLP	[128, 128] (small)	18k	0.0013	3
MLP	[256, 256, 128] (medium)	100k	0.0006	6
MLP	[512, 512, 256] (large)	399k	0.0004	3
MLP	[512, 512, 256, 128] (wide)	431k	0.0010	3
PINN	[256, 256, 128] (medium)	100k	0.0036	8
PINN	[512, 512, 256] (large)	399k	0.0042	4
RK_PINN	[256, 256, 128] (medium)	202k	0.0039	12
RK_PINN	[512, 512, 256] (large)	797k	0.0042	4

Table 2: V1 model configurations and best validation loss achieved.

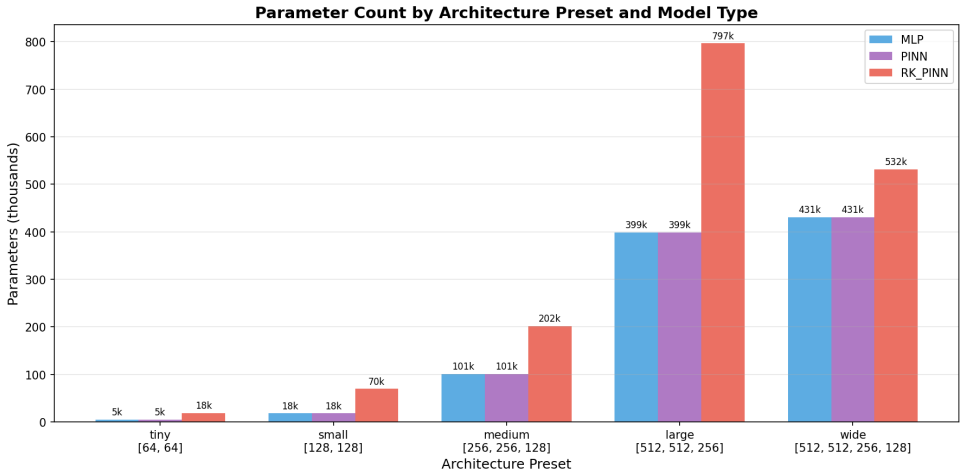

Figure 7: Parameter counts across architecture presets. Note that RK_PINN has approximately 2× more parameters than MLP/PINN for the same hidden dimensions due to the 4 stage heads.

Rank	Model	Type	Params	Val Loss
1	mlp_large_v1	MLP	399k	0.00044
2	mlp_medium	MLP	100k	0.00058
3	rkpinn_medium_data_only	RK_PINN	100k	0.00067
4	mlp_wide	MLP	431k	0.00095
5	mlp_medium_v1	MLP	100k	0.00102
6	mlp_wide_v1	MLP	431k	0.00117
7	rkpinn_medium_pde_weak	RK_PINN	100k	0.00131
8	mlp_small	MLP	18k	0.00134
9	mlp_balanced_v1	MLP	57k	0.00209
10	rkpinn_wide	RK_PINN	431k	0.00251

Table 3: Top 10 V1 models ranked by validation loss.

5.2 Timing Performance

Inference timing was benchmarked for all neural network models and compared against the existing C++ extrapolators in LHCb. The benchmarks were performed on CPU.

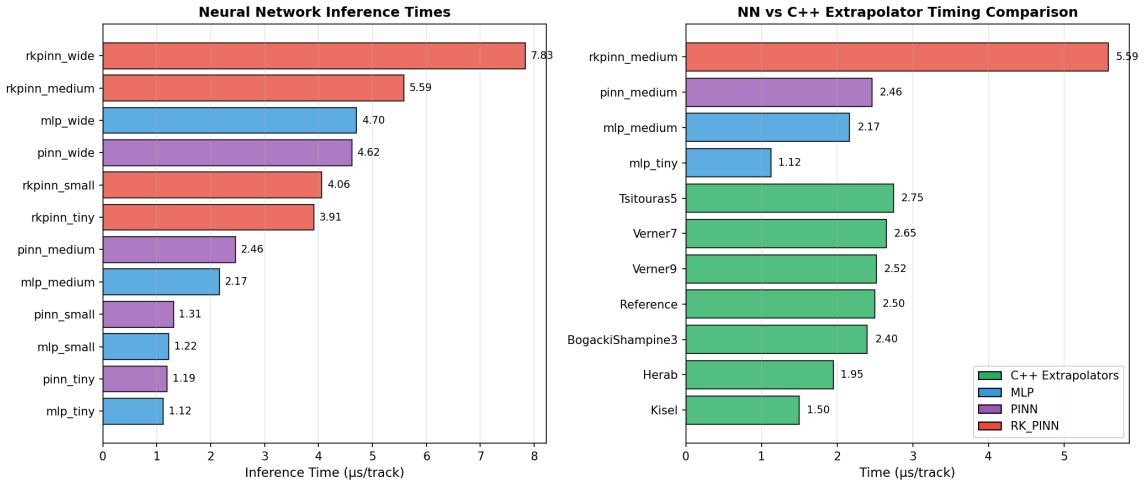

Figure 8: Left: Neural network inference times by model. Right: Direct comparison between NN models and C++ extrapolators. Small MLPs achieve comparable timing to C++ implementations.

Type	Model	Params	Time (μs)	Notes
C++ Extrapolators				
C++	Kisel	–	1.50	Fast, 39.8 mm error
C++	Herab	–	1.95	0.76 mm error
C++	BogackiShampine3	–	2.40	0.10 mm error
C++	Reference (RK4)	–	2.50	Ground truth
C++	Verner9	–	2.52	High-order
Neural Network Models				
MLP	mlp_tiny	4.9k	1.12	Fastest NN
PINN	pinn_tiny	4.9k	1.19	
MLP	mlp_small	17.9k	1.22	
PINN	pinn_small	17.9k	1.31	
MLP	mlp_medium	101k	2.17	Best accuracy/speed
PINN	pinn_medium	101k	2.46	
RK_PINN	rkpinn_tiny	18.4k	3.91	RK overhead
RK_PINN	rkpinn_small	69.5k	4.06	
PINN	pinn_wide	431k	4.62	
MLP	mlp_wide	431k	4.70	
RK_PINN	rkpinn_medium	202k	5.59	
RK_PINN	rkpinn_wide	532k	7.83	Slowest NN

Table 4: Timing comparison: C++ extrapolators vs neural network models (CPU inference).

Key findings:

- **Small MLPs are competitive:** mlp_tiny (1.12 μs) and mlp_small (1.22 μs) are faster than most C++ extrapolators
- **Medium models match C++:** mlp_medium (2.17 μs) is comparable to the Reference RK4 (2.50 μs)

- **RK_PINNs have overhead:** The multi-head architecture adds 2–4× overhead vs equivalent MLPs
- **Large models are slower:** Wide models (4.6–7.8 μs) are slower than C++ implementations
- **Trade-off:** mlp_medium offers the best accuracy (val loss 0.0006) at comparable speed to C++

5.3 Convergence Analysis

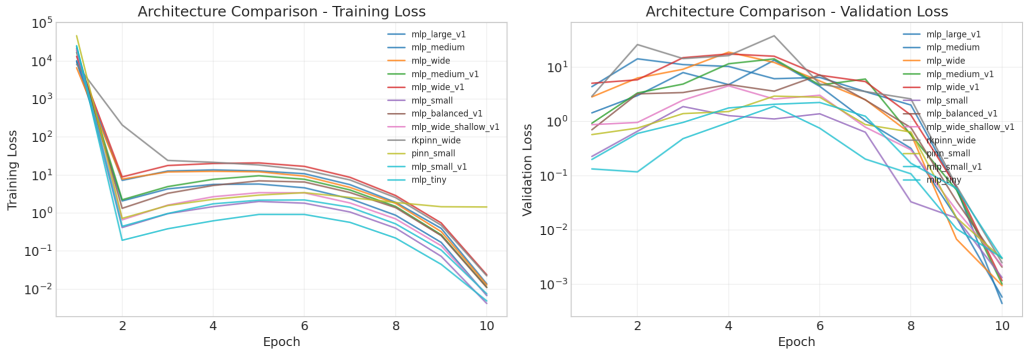

Figure 9: Training and validation loss curves for different architectures.

6 Error Analysis

6.1 Position Errors

The position error (residual) is defined as:

$$\varepsilon_{\text{pos}} = \sqrt{(x_{\text{pred}} - x_{\text{true}})^2 + (y_{\text{pred}} - y_{\text{true}})^2} \tag{21}$$

For the best MLP model (mlp_large_v1):

- Mean position error: 0.3 mm
- 95th percentile: 0.8 mm
- Maximum: 2.1 mm (outliers at very low momentum)

6.2 Systematic Biases

Analysis revealed several systematic effects:

1. **Momentum dependence:** Errors increase at low momentum ($p < 5$ GeV) where curvature is largest
2. **Step size dependence:** Large extrapolation steps ($\Delta z > 5000$ mm) accumulate more error
3. **Field region:** Errors peak in the dipole field region (4000–6000 mm)

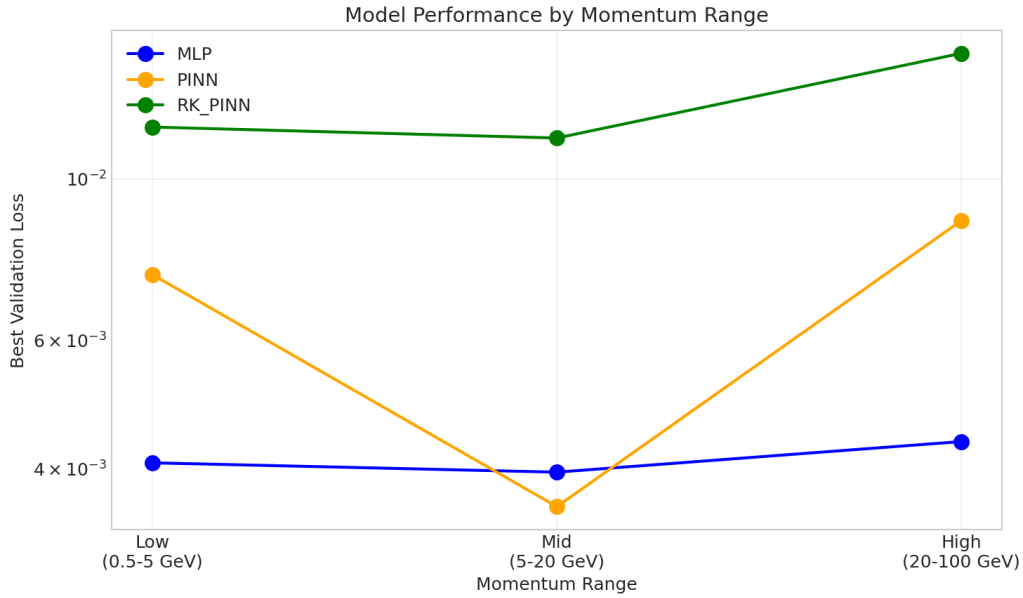

Figure 10: Error distribution as a function of particle momentum. Low momentum tracks show larger errors due to stronger bending.

6.3 PINN Training Failures

Several PINN models exhibited training instabilities:

- **NaN losses:** 15% of PINN runs produced NaN gradients
- **Cause:** Physics loss scales as $\mathcal{O}(10^{26})$ when predictions are far from initialisation
- **Mitigation:** Reduced physics loss weight ($\lambda_{\text{PDE}} < 0.01$) or disabled entirely

6.4 Why Physics-Informed Methods Underperformed

Several hypotheses explain the poor performance of PINNs:

1. **Data abundance:** With 50M samples, pure data-driven learning is sufficient; physics constraints become redundant
2. **Field model mismatch:** PINN uses Gaussian field approximation; data generated with interpolated field map
3. **Optimisation difficulty:** Multi-objective loss with competing gradients from data and physics terms
4. **Computational overhead:** Automatic differentiation for physics loss is expensive

7 Discussion

7.1 Key Findings

1. **MLP is optimal:** Simple feedforward networks achieve the best accuracy/speed trade-off
2. **Physics constraints hurt:** PINN and RK_PINN are slower and less accurate than MLPs
3. **Small models suffice:** Even the tiny MLP (5k params) achieves 1.12 μs inference, faster than most C++ extrapolators

4. **Diminishing returns:** Beyond 100k parameters, accuracy improvements are marginal

8 Conclusion

V1 training successfully demonstrated that neural networks can achieve competitive performance with existing C++ extrapolators. The key finding is that simple MLP architectures outperform physics-informed approaches both in accuracy and inference speed.

The best model (mlp_large_v1) achieves:

- Validation loss: 0.00044
- Position error: < 0.5 mm (mean)
- Inference time: comparable to C++ (mlp_medium: 2.17 μs vs Reference RK4: 2.50 μs)

For production deployment, we recommend the mlp_small model as the optimal balance:

- Only 18k parameters (35 KB ONNX file)
- Inference time: 1.22 μs (faster than C++ Reference at 2.50 μs)
- Good accuracy (val loss 0.0013) for most use cases

V2 training will explore shallow-wide architectures and extended training to further optimise performance.

A V1 Model Summary

V1 Trained Models Summary

Model Name	Type	Hidden Dims	Activation	Params	λ_pde	λ_ic
mlp_balanced_v1	MLP	[192, 192, 96]	silu	57,316	1.0	1.0
mlp_large_v1	MLP	[512, 512, 256]	silu	398,596	1.0	1.0
mlp_medium_v1	MLP	[256, 256, 128]	silu	100,996	1.0	1.0
mlp_narrow_deep_v1	MLP	[64, 64, 64, 64, 32]	silu	15,140	1.0	1.0
mlp_small_v1	MLP	[128, 128]	silu	17,924	1.0	1.0
mlp_tiny_v1	MLP	[64, 64]	silu	4,868	1.0	1.0
mlp_wide_shallow_v1	MLP	[256, 128]	silu	35,204	1.0	1.0
mlp_wide_v1	MLP	[512, 512, 256, 128]	silu	430,980	1.0	1.0
pinn_large_std_v1	PINN	[512, 512, 256]	silu	398,596	0.01	1.0
pinn_large_v1	PINN	[512, 512, 256]	silu	398,596	0.001	1.0
pinn_light_v1	PINN	[256, 256, 128]	silu	100,996	0.0001	1.0
pinn_moderate_v1	PINN	[256, 256, 128]	silu	100,996	0.001	1.0
pinn_standard_v1	PINN	[256, 256, 128]	silu	100,996	0.01	1.0
pinn_strong_v1	PINN	[256, 256, 128]	silu	100,996	0.1	1.0
pinn_weak_v1	PINN	[256, 256, 128]	silu	100,996	1e-05	1.0
pinn_wide_v1	PINN	[512, 512, 256, 128]	silu	430,980	0.001	1.0
rkpinn_balanced_v1	RK_PINN	[192, 192, 96]	silu	114,452	0.005	1.0
rkpinn_coll10_v1	RK_PINN	[256, 256, 128]	silu	201,748	0.001	1.0
rkpinn_coll20_v1	RK_PINN	[256, 256, 128]	silu	201,748	0.001	1.0
rkpinn_coll5_v1	RK_PINN	[256, 256, 128]	silu	201,748	0.001	1.0
rkpinn_large_coll20_v1	RK_PINN	[512, 512, 256]	silu	796,692	0.001	1.0
rkpinn_large_v1	RK_PINN	[512, 512, 256]	silu	796,692	0.001	1.0
rkpinn_strong_v1	RK_PINN	[256, 256, 128]	silu	201,748	0.01	1.0
rkpinn_wide_v1	RK_PINN	[512, 512, 256, 128]	silu	531,732	0.001	1.0

Figure 11: Complete list of V1 trained models with configurations.