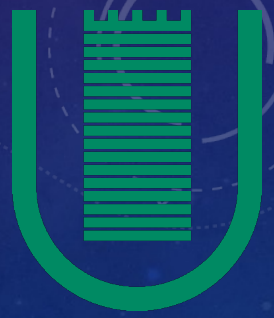
HYBRID THEORY-SIMULATIONS MACHINE LEARNING APPROACH TO BACKTRACING

Luca Tabarroni
luca.tabarroni@roma2.infn.it



PHISICALLY INFORMED NEURAL NETWORKS (PINNs)

- Automatic differentiation: Neural networks are able to keep track of the gradient of outputs with respect to inputs
- Training means tuning the parameters and therefore tuning the outputs and also their gradients
- PINNs are based on training a network in order to have specific output gradients to solve differential equations

HAMILTONIAN NEURAL NETWORKS

$$\mathbf{z} = \underbrace{(q_1, \ldots, q_d, p_1, \ldots, p_d)}_{D = d + d}^{\mathbf{T}}$$

$$\mathbf{J} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{-1} & \mathbf{0} \end{bmatrix}$$

D x D MATRIX

HAMILTONIAN NEURAL NETWORKS

$$\mathbf{z} = (\underbrace{q_1, \ldots, q_d, p_1, \ldots, p_d}_{D = d + d})^{\mathbf{T}}$$

$$\mathbf{J} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ -\mathbf{1} & \mathbf{0} \end{bmatrix}$$

D x D MATRIX

HAMILTON E.O.M. IN SYMPLECTIC NOTATION

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i} \qquad \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i} \longrightarrow \dot{\mathbf{z}} = \mathbf{J} \cdot \nabla_{\mathbf{z}} \mathcal{H}$$

HAMILTONIAN NEURAL NETWORKS

$$\mathbf{z} = (\underbrace{q_1, \ldots, q_d, p_1, \ldots, p_d}_{D = d + d})^{\mathbf{T}}$$

$$\mathbf{J} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ -\mathbf{1} & \mathbf{0} \end{bmatrix}$$

D x D MATRIX

HAMILTON E.O.M. IN SYMPLECTIC NOTATION

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i} \qquad \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i} \longrightarrow \dot{\mathbf{z}} = \mathbf{J} \cdot \nabla_{\mathbf{z}} \mathcal{H}$$

$$L = \frac{1}{K} \sum_{n=1}^{K} (\dot{\hat{\mathbf{z}}}^{(n)} - \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}^{(\mathbf{n})}} \mathcal{H}(\hat{\mathbf{z}}^{(n)}))^2 + \lambda C_{reg}$$

HAMILTONIAN NEURAL NETWORKS

$$\mathbf{z} = (\underbrace{q_1, \ldots, q_d, p_1, \ldots, p_d}_{D = d + d})^{\mathbf{T}}$$

$$\mathbf{J} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ -\mathbf{1} & \mathbf{0} \end{bmatrix}$$

D x D MATRIX

HAMILTON E.O.M. IN SYMPLECTIC NOTATION

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i} \qquad \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i} \longrightarrow \dot{\mathbf{z}} = \mathbf{J} \cdot \nabla_{\mathbf{z}} \mathcal{H}$$

LOSS FUNCTION

$$L = \frac{1}{K} \sum_{n=1}^{K} (\dot{\hat{\mathbf{z}}}^{(n)} - \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}^{(\mathbf{n})}} \mathcal{H}(\hat{\mathbf{z}}^{(n)}))^2 + \lambda C_{reg}$$

ENERGY CONSERVATION

$$C_{reg} = \frac{1}{K} \sum_{n=1}^{K} (\mathcal{H}(\hat{\mathbf{z}}^{(n)}) - E_0)^2$$

APPLICATION: GRAVITATIONAL FIELD

Hamiltonian Neural Networks approach to fuzzball geodesics

Andrea Cipriani,[1,2,*] Alessandro De Santis,[2,3,4,†] Giorgio Di Russo,[2,5,‡] Alfredo Grillo,[1,2,§] and Luca Tabarroni[1,2,¶]

[1]*Dipartimento di Fisica, Università di Roma Tor Vergata, Via della Ricerca Scientifica 1, 00133 Roma, Italia*
[2]*Sezione INFN Roma 2, Via della Ricerca Scientifica 1, 00133, Roma Italia*
[3]*Helmholtz-Institut Mainz, Johannes Gutenberg-Universität Mainz, 55099 Mainz, Germany*
[4]*GSI Helmholtz Centre for Heavy Ion Research, 64291 Darmstadt, Germany*
[5]*School of Fundamental Physics and Mathematical Sciences, Hangzhou Institute for Advanced Study, UCAS, Hangzhou 310024, China*

(Dated: November 26, 2025)

The recent increase in computational resources and data availability has led to a significant rise in the use of Machine Learning (ML) techniques for data analysis in physics. However, the application of ML methods to solve differential equations capable of describing even complex physical systems is not yet fully widespread in theoretical high-energy physics. Hamiltonian Neural Networks (HNNs) are tools that minimize a loss function defined to solve Hamilton equations of motion. In this work, we implement several HNNs trained to solve, with high accuracy, the Hamilton equations for a massless probe moving inside a smooth and horizonless geometry known as D1-D5 circular fuzzball. We study both planar (equatorial) and non-planar geodesics in different regimes according to the impact parameter, some of which are unstable. Our findings suggest that HNNs could eventually replace standard numerical integrators, as they are equally accurate but more reliable in critical situations.

I. INTRODUCTION

Physics-informed neural networks (PINNs) are a widely used tool in today's Machine Learning (ML) landscape. They consist of Neural Networks (NNs) that, during the training phase, learn to solve the differential equations governing the physical laws of a system in a model-independent way. When these differential equations correspond to Hamilton equations of motion, we refer to them as Hamiltonian Neural Networks (HNNs). The HNN paradigm was introduced in [1] and in the present work we closely follow the strategy proposed in [2]. The key advantages of HNNs over standard numerical integrators can be summarized as follows:

- the predicted solution is analytical in time and not limited to a discrete set of time steps;
- conservation laws, symmetries, constraints and prior knowledge of the system can be easily incorporated at the level of the architecture and of the loss function to improve the predictability of the HNN;
- the minimization process of the loss function occurs under the constraint that the solution satisfies the system of equations at all times simultaneously and independently, thus avoiding any iterative mechanism.

The last point is crucial for systems whose (effective) potential exhibits unstable critical points, for which small changes of the initial conditions lead to completely different behaviors of the solutions, similar to what happens in chaotic dynamical systems. As it is well known, any numerical integrator operating iteratively, building the solution based on the result of the previous time step, accumulates errors and inevitably loses accuracy over long time scales in such situations. HNNs have the potential to overcome this issue.

While HNNs have been applied in various physical contexts[1], their diffusion in high-energy physics remains limited. In this paper, we apply this technique for the first time in the field of String Theory, where ML has only begun to gain attraction in recent years [18–36]. Specifically, we focus on using HNNs to determine the non-critical and unstable critical geodesics of a massless particle moving in a D1-D5 circular fuzzball geometry by solving the associated equations of motion. The scattering properties, scalar perturbations and tidal deformations of this physical system has already been extensively studied in [37–39]. It serves as a well-suited test case, given that the exact trajectory, to which we will refer to as the *ground truth*, can be explicitly computed. Thus, the results presented in this paper do not add new physical insights into the system under study; rather, the work is methodological in nature and aims to assess the feasibility and performance of HNNs compared to standard numerical methods in a controlled setting. Our investigation, as clearly demonstrated by the numerical results,

* andrea.cipriani@roma2.infn.it
† desantia@uni-mainz.de
‡ gdr794@ucas.ac.cn
§ alfredo.grillo89@gmail.com
¶ luca.tabarroni@roma2.infn.it

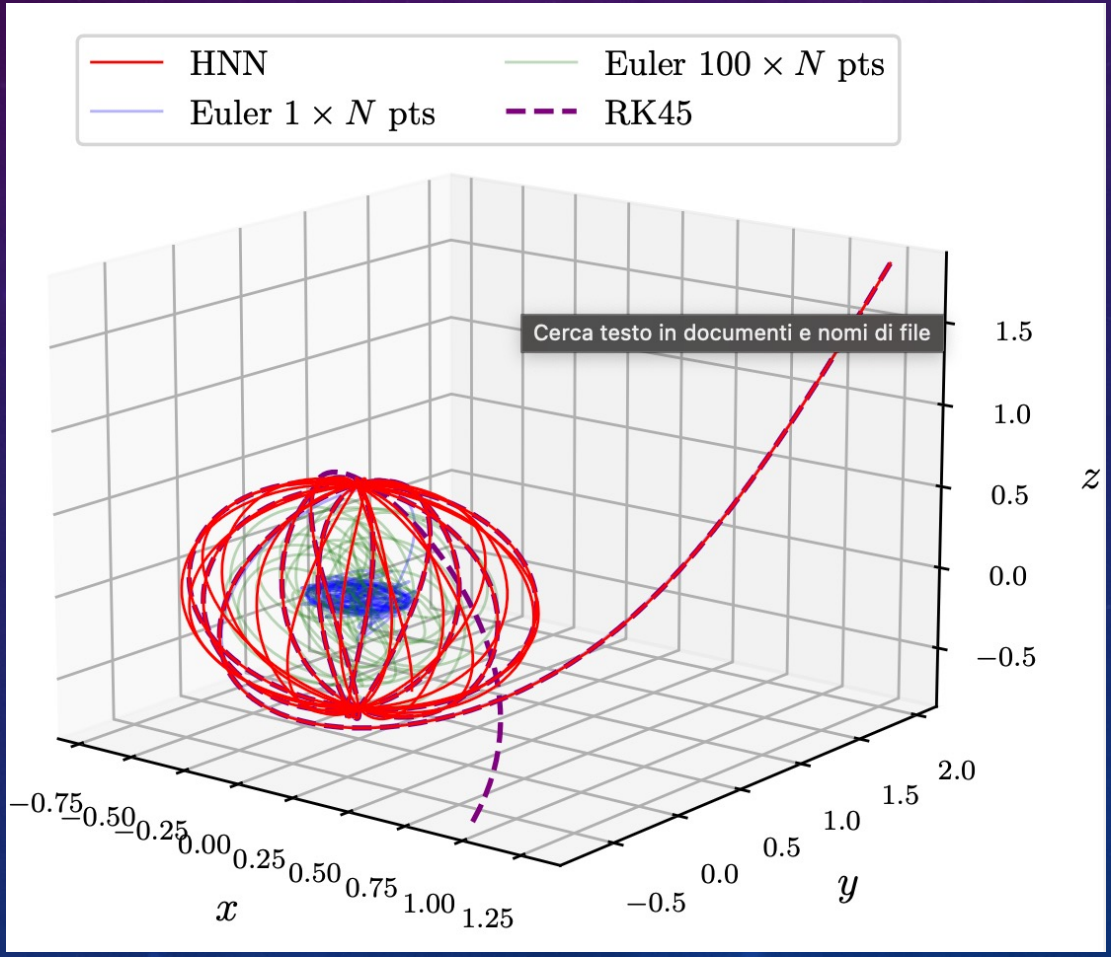
[1] Due to the rapidly increasing number of ML applications, the literature on this topic is vast, and providing a comprehensive review is beyond the scope of this article. An incomplete selection of applications includes [3–17].

HAMILTONIAN NEURAL NETWORK APPROACH TO FUZZBALL GEODESICS

A.Cipriani, A. De Santis, G. Di Russo, A. Grillo, L. Tabarroni

https://doi.org/10.1103/dssv-x49b

$$\mathcal{H} = \frac{1}{2\left(\rho^2 + a_f^2 \cos^2\theta\right) H}\left[P_\rho^2(\rho^2+a_f^2)+P_\theta^2+\frac{J_\phi^2}{\sin^2\theta}-\frac{(J_\phi a_f - E L_1 L_5)^2}{\rho^2 + a_f^2}-E^2(\rho^2+a_f^2+L_1^2+L_5^2-a_f^2\sin^2\theta)\right]$$



BACKTRACING

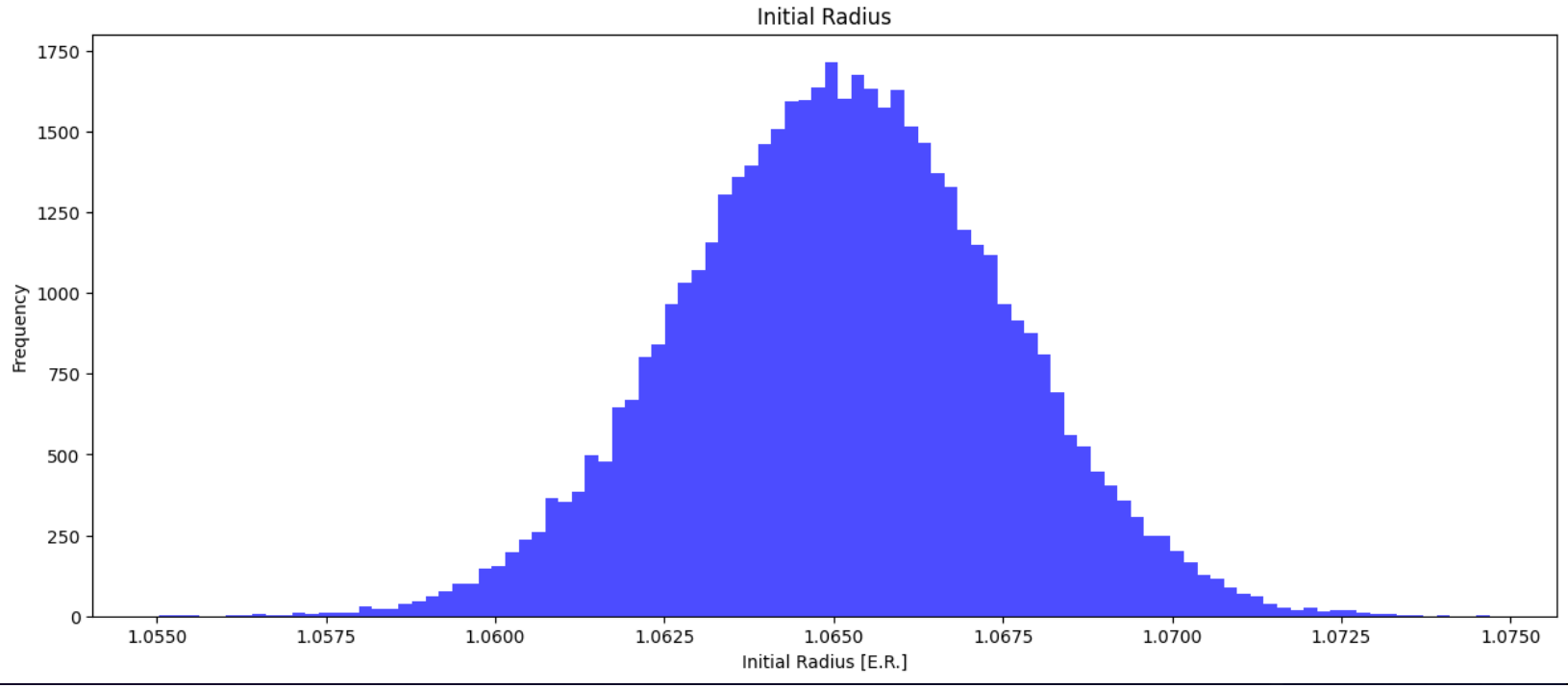
Starting point: runge-kutta TRAJECTORIES:

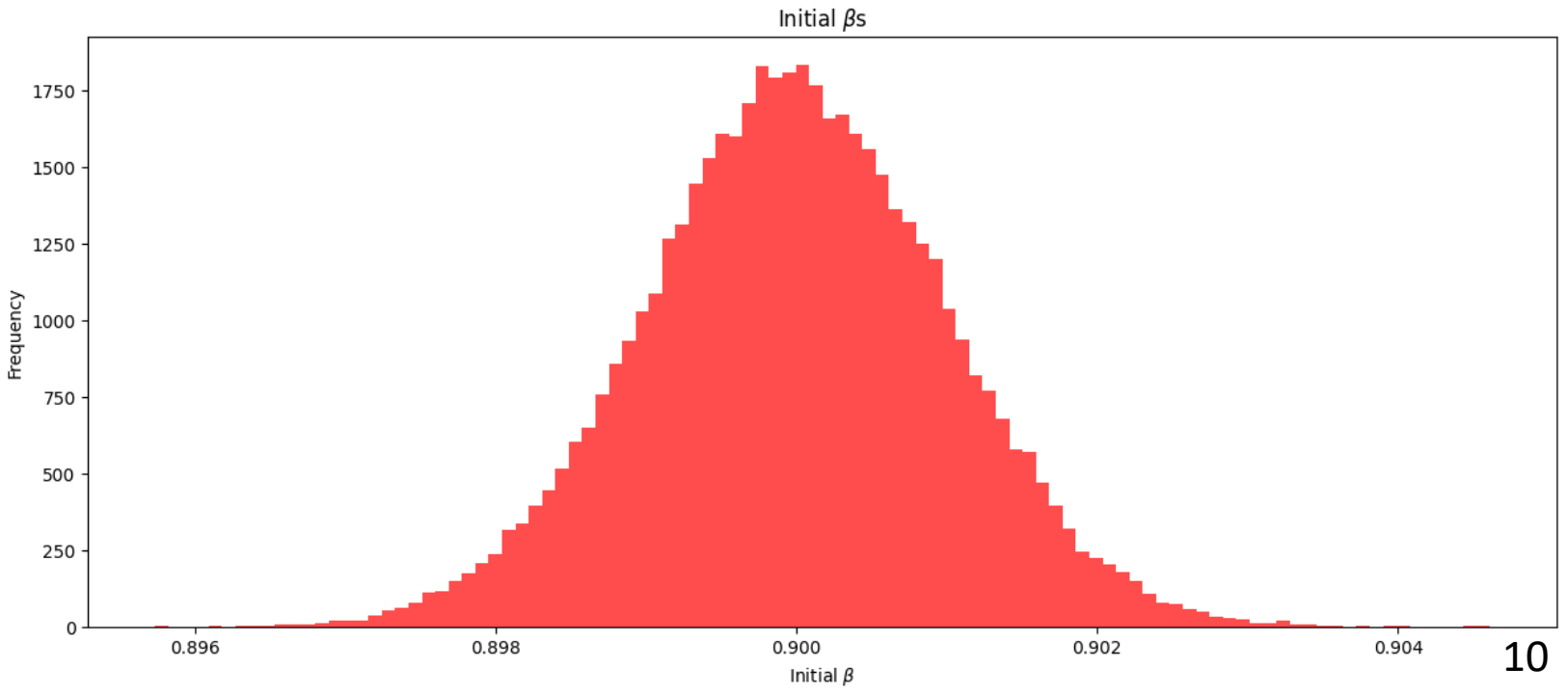
- Protons
- Earth magnetic dipole,
- Generated with RK45,
- Initial conditions generated with Guassian distributions for moduli of position and velocity, uniform distributions for theta and phi.

1 E.R. = 6371 km
ISS altitude ∽ (400 + 6371)km∽ 1.06 E.R.

- Trajectories generated at CNAF
- 10’000 points for each trajectory

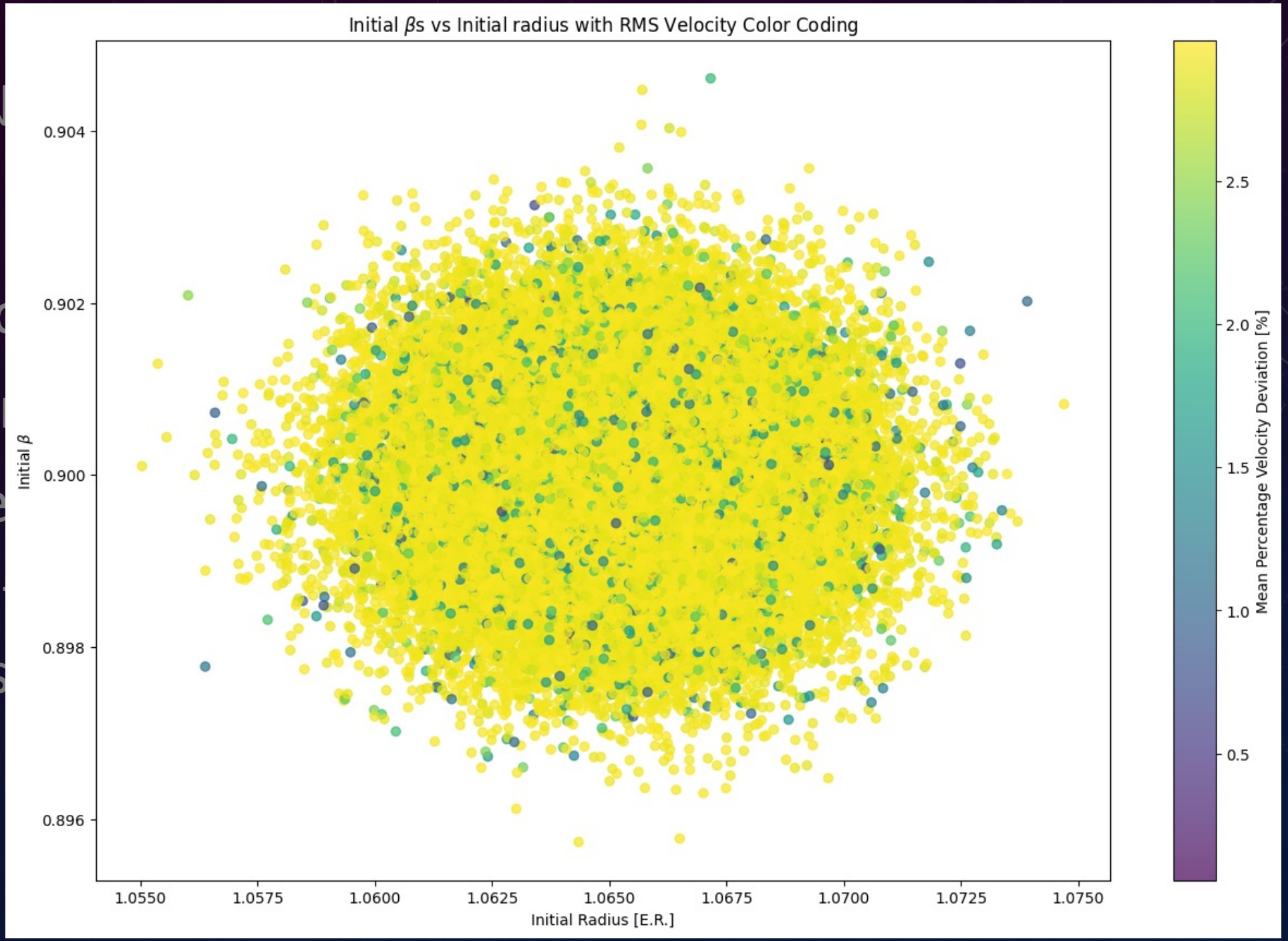
c ∽ 47.05 E.R./s





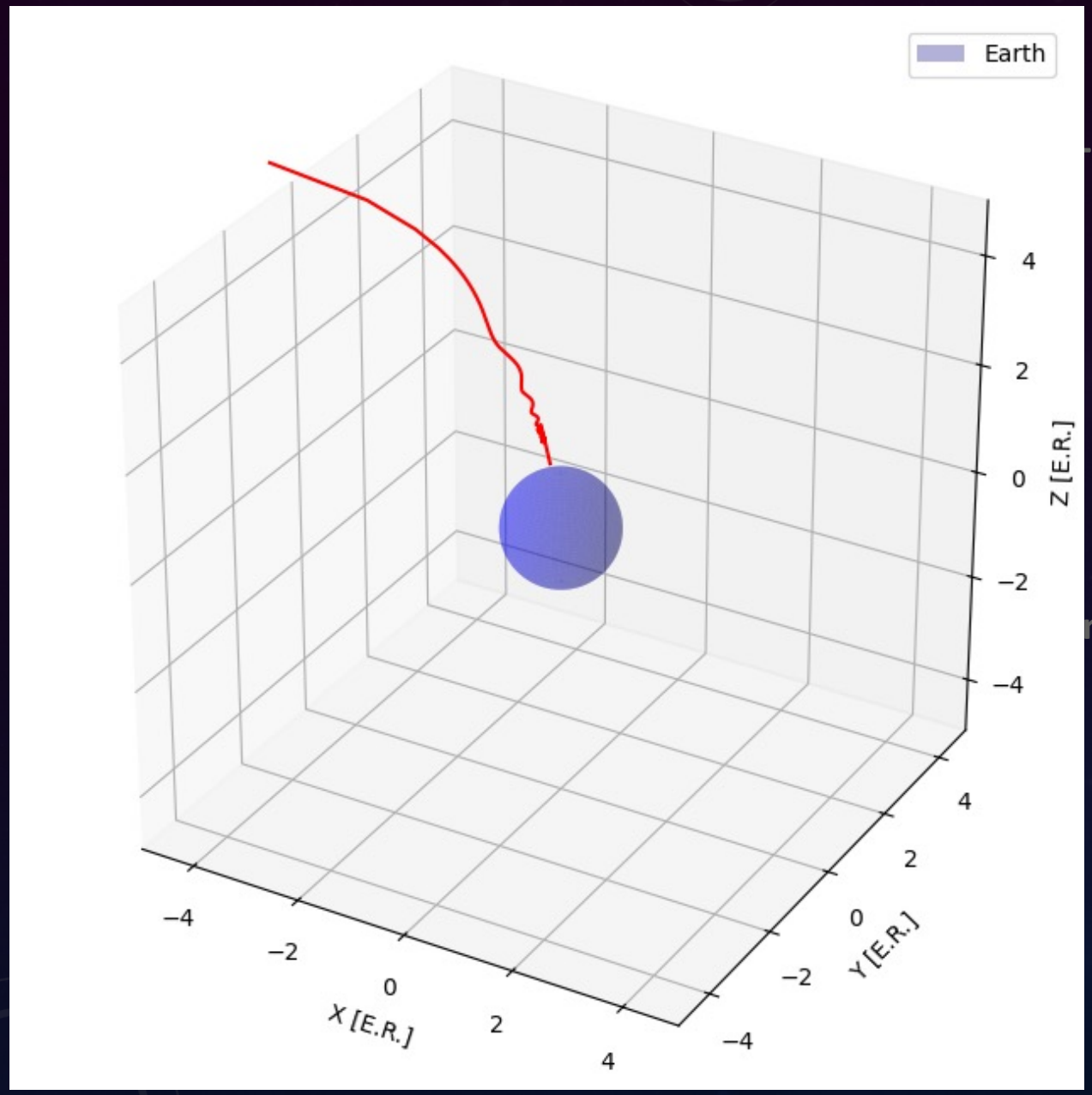
STARTIN

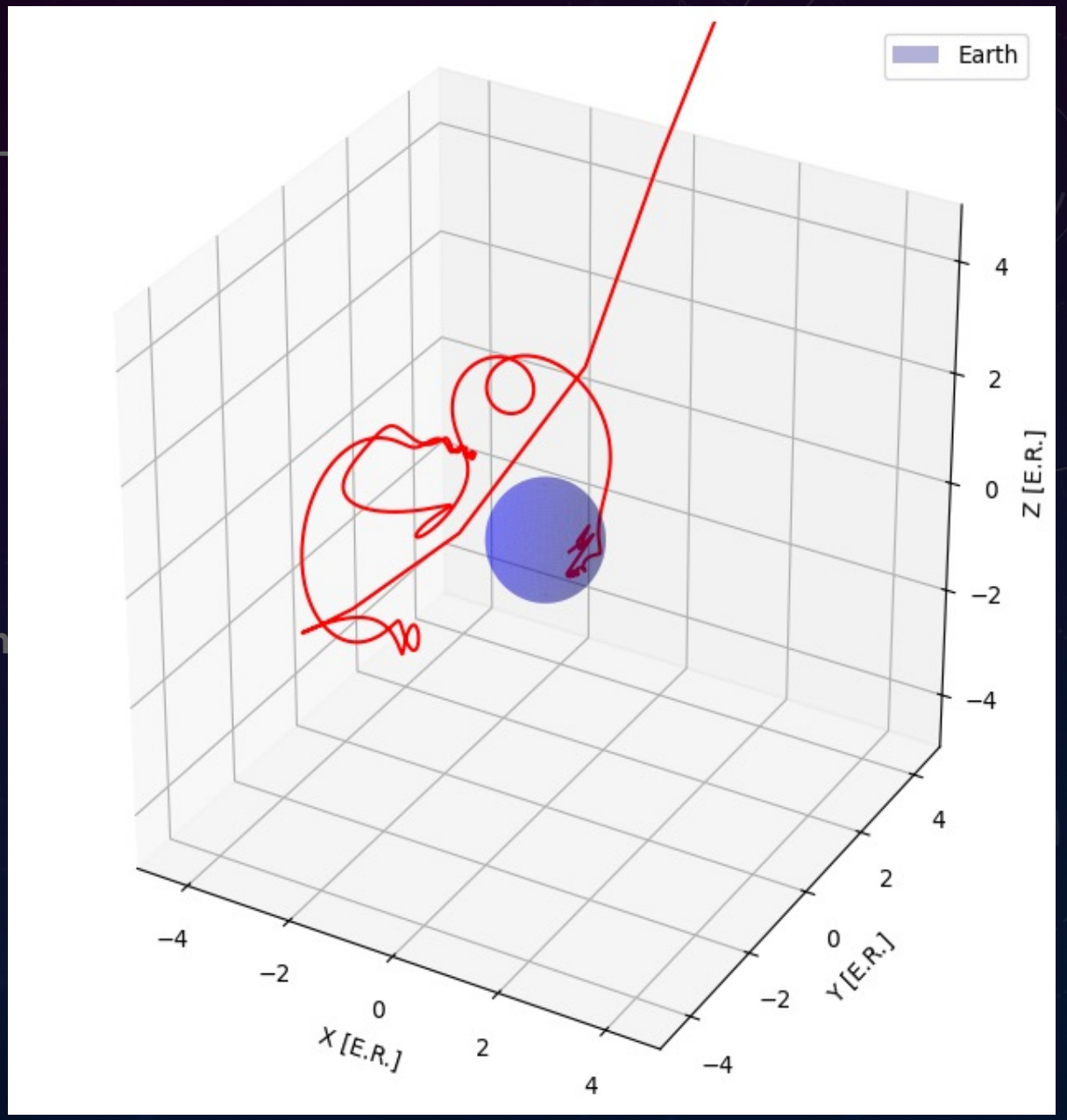
- Pro
- Ea
- Ge
- Ini
  dis
  un



STARTING POINT: RUNGE-KUTTA

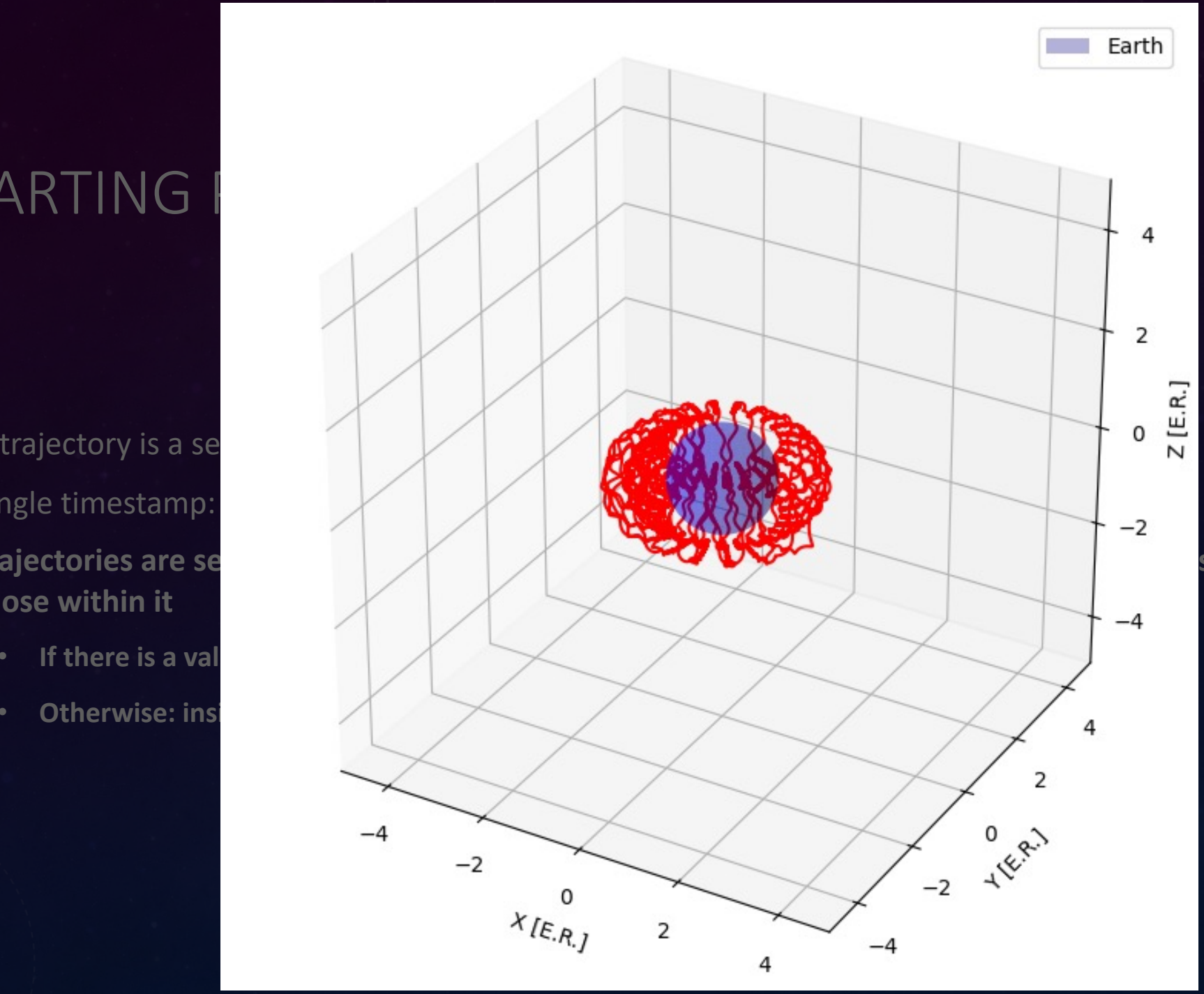
- A trajectory is a sequence of timestamps,
- Single timestamp: $[t, x, y, z, v^x, v^y, v^z, a^x, a^y, a^z]$,
- **Trajectories are separated into two groups: those originated outside the Earth's magnetosphere and those within it**
  - **If there is a value of r>10E. R. : outside**
  - **Otherwise: inside**


Earth
X [E.R.]
Y [E.R.]
Z [E.R.]
−4
−2
0
2
4


Earth
X [E.R.]
Y [E.R.]
Z [E.R.]
−4
−2
0
2
4

STARTING P

- A trajectory is a se
- Single timestamp:
- **Trajectories are se** **sphere and those within it**
  - **If there is a val**
  - **Otherwise: insi**



PHISICALLY INFORMED NEURAL NETWORKS (PINNs)

- Automatic differentiation: Neural networks are able to keep track of the gradient of outputs with respect to inputs
- Training means tuning the parameters and therefore tuning the outputs and also their gradients
- PINNs ar order to have specific output equations

$$\mathcal{L}_i = \alpha\{[\overrightarrow{\text{grad}}(\vec{r}_i, t_i) - \vec{v}_i] + [\overrightarrow{\text{grad}}(\vec{v}_i, t_i) - \vec{v}_i \times \vec{B}_i]\} + \beta\{|\vec{v}_i| - |\vec{v}_0|\}$$

TRAIN MINIMIZING THIS TERM
MEANS SOLVING
THE EQUATIONS OF MOTION

PHISICALLY INFORMED NEURAL NETWORKS (PINNs)

- Automatic differentiation: Neural networks are able to keep track of the gradient of outputs with respect to inputs
- Training means tuning the parameters and therefore tuning the outputs and also their gradients
- PINNs are based on training a netwo
  output gradients to solve differe

$$\mathcal{L}_i = \alpha\left\{\left[\overrightarrow{\text{grad}}(\vec{r}_i, t_i) - \vec{v}_i\right] + \left[\overrightarrow{\text{grad}}(\vec{v}_i, t_i) - \vec{v}_i \times \vec{B}_i\right]\right\} + \beta\left\{|\vec{v}_i| - |\vec{v}_0|\right\}$$

TRAIN MINIMIZING THIS TERM
MEANS CONSERVING
THE ENERGY

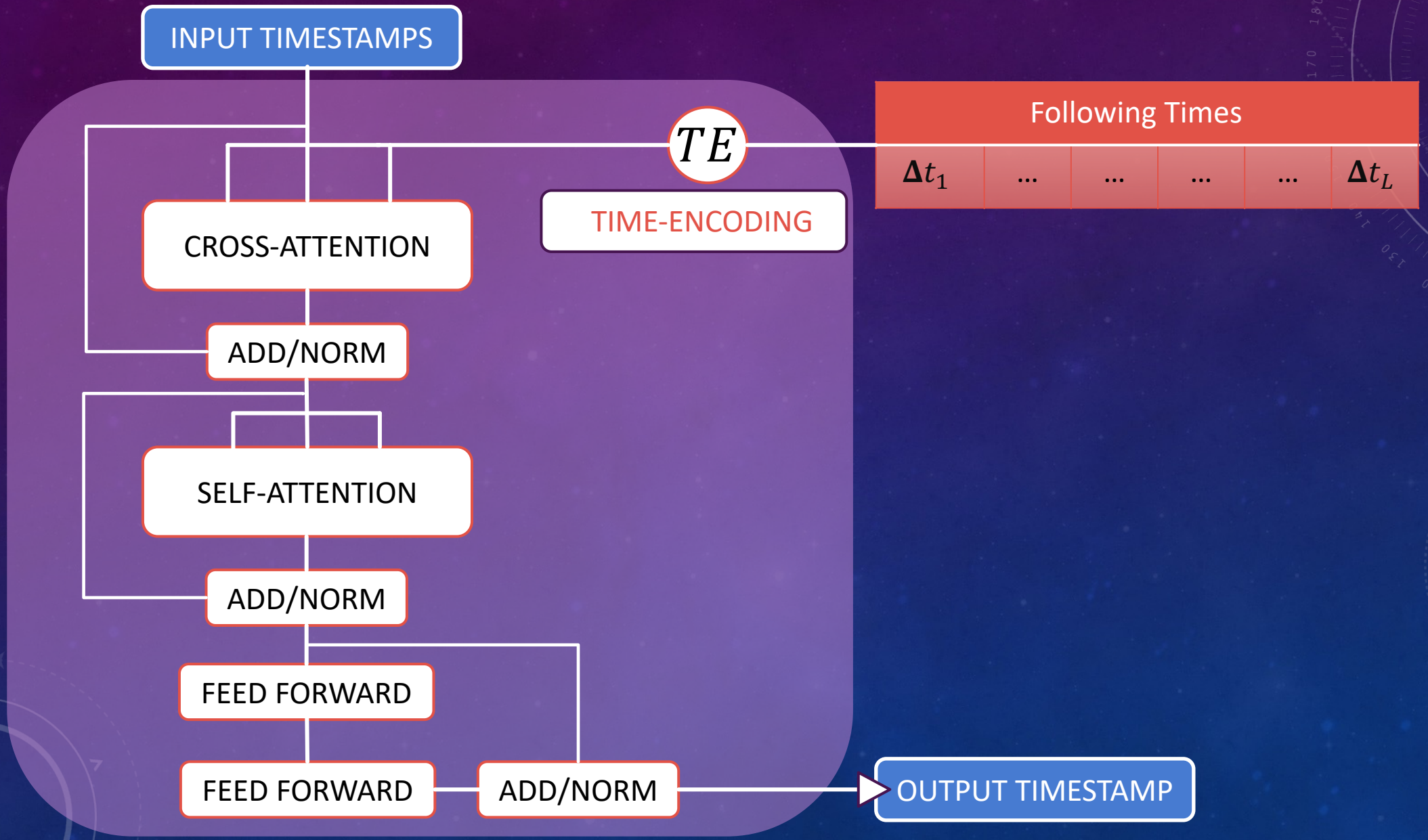
ML DATASET: FRAGMENT OF TRAJECTORY

$$\Delta t_k = t_{k+1} - t_k \qquad \vec{a}_k = \vec{v}_k \times \vec{B}(\vec{r}_k)$$

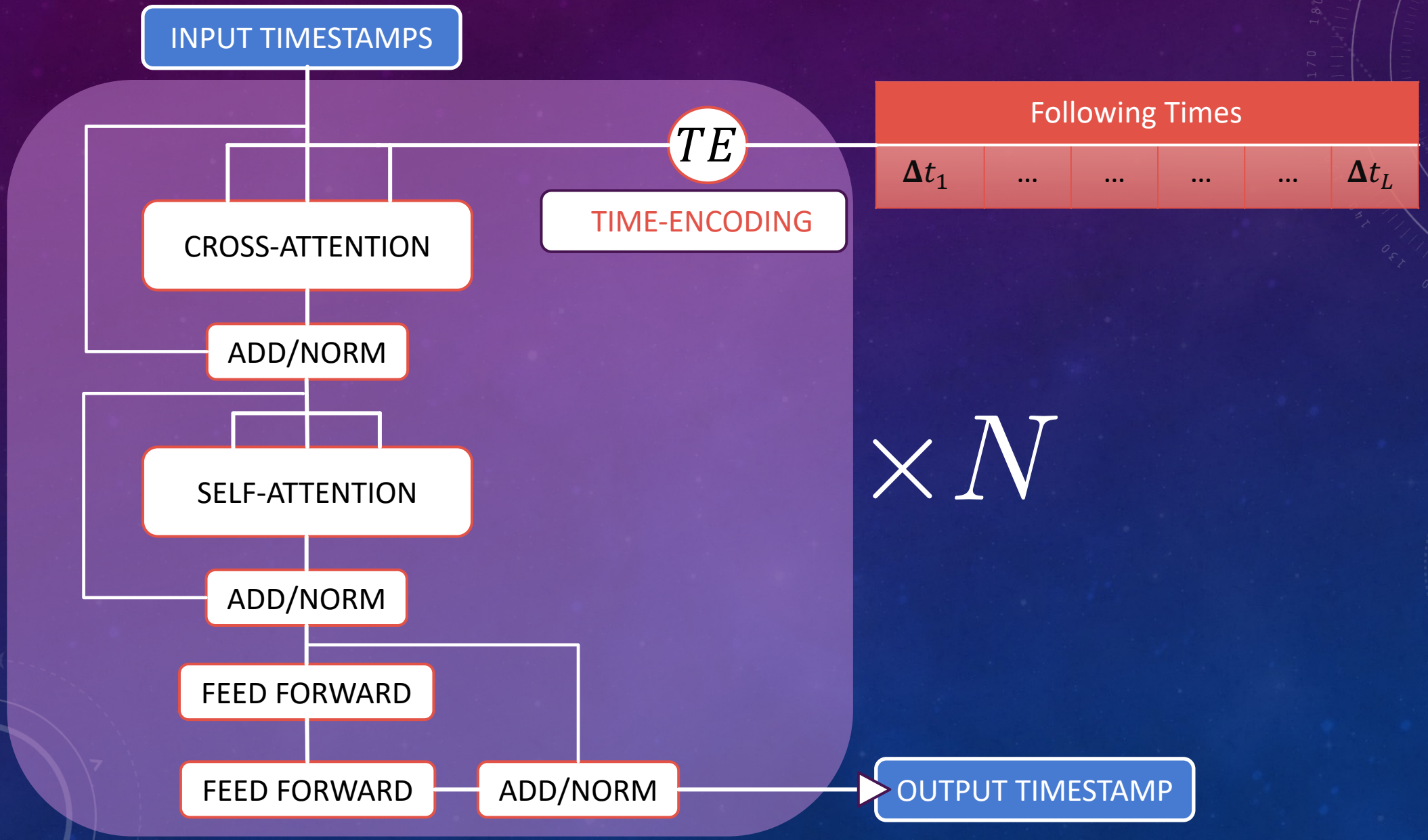
NORMALIZED MASS AND CHARGE:
m=1
q=1

Δt_{i-N}	…	…	…	…	…	…	t_i	…	…	…	…	…	…	Δt_{i+N}
x_{i-N}	…	…	…	…	…	…	x_i	…	…	…	…	…	…	x_{i+N}
y_{i-N}	…	…	…	…	…	…	y_i	…	…	…	…	…	…	y_{i+N}
z_{i-N}	…	…	…	…	…	…	z_i	…	…	…	…	…	…	z_{i+N}
v^x_{i-N}	…	…	…	…	…	…	v^x_i	…	…	…	…	…	…	v^x_{i+N}
v^y_{i-N}	…	…	…	…	…	…	v^y_i	…	…	…	…	…	…	v^y_{i+N}
v^z_{i-N}	…	…	…	…	…	…	v^z_i	…	…	…	…	…	…	v^z_{i+N}
a^x_{i-N}	…	…	…	…	…	…	a^x_i	…	…	…	…	…	…	a^x_{i+N}
a^y_{i-N}	…	…	…	…	…	…	a^y_i	…	…	…	…	…	…	a^y_{i+N}
a^z_{i-N}	…	…	…	…	…	…	a^z_i	…	…	…	…	…	…	a^z_{i+N}

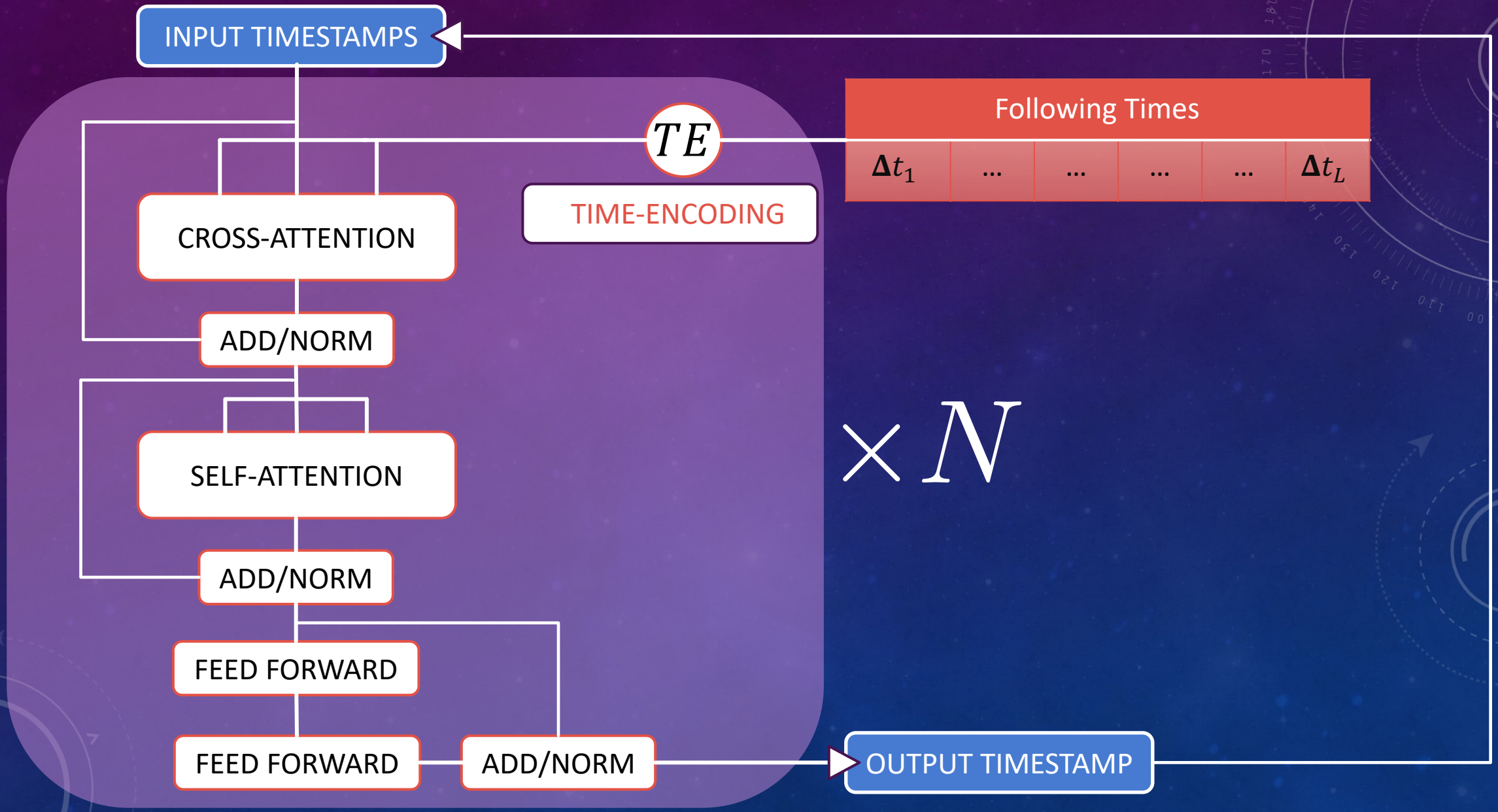
TRANSFORMER DECODER TRAINING



TRANSFORMER DECODER TRAINING



TRANSFORMER DECODER INFERENCE



POSITIONAL ENCODING

$$I(P) = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline x_i & y_i & z_i & v_i^x & v_i^y & v_i^z & a_i^x & a_i^y & a_i^z \\ \hline \end{array}$$

- P: token position in the sequence P=1,…,L
- i: feature position i=0,…,8

PROBLEMS

- Discrete and fixed position **Δ**P=1
- Relative positions cannot be learned

$$PE(P, 2i) = \cos\left[\frac{P}{10000^{2i/d_m}}\right]$$

$$PE(P, 2i+1) = \sin\left[\frac{P}{10000^{2i/d_m}}\right]$$

$$I(P) \rightarrow I(P) + PE(P)$$

RELATIVE LEARNABLE TIME ENCODING

Following Times					
Δt_1	…	…	…	…	Δt_L

RELATIVE LEARNABLE TIME ENCODING

Following Times					
Δt_1	…	…	…	…	Δt_L

p is an hyperparameter

$$\Delta = \begin{pmatrix} \Delta t_1 & \Delta t_1^2 & \dots & \Delta t_1^p \\ \Delta t_2 & \Delta t_2^2 & \dots & \Delta t_2^p \\ \vdots & \vdots & \ddots & \vdots \\ \Delta t_L & \Delta t_L^2 & \dots & \Delta t_L^p \end{pmatrix}$$

N.B. $\Delta t_i = t_{i+1} - t_i$

RELATIVE LEARNABLE TIME ENCODING

Following Times					
Δt_1	...	...	...	...	Δt_L

p is an hyperparameter

$$\Delta = \begin{pmatrix} \Delta t_1 & \Delta t_1^2 & \dots & \Delta t_1^p \\ \Delta t_2 & \Delta t_2^2 & \dots & \Delta t_2^p \\ \vdots & \vdots & \ddots & \vdots \\ \Delta t_L & \Delta t_L^2 & \dots & \Delta t_L^p \end{pmatrix}$$

$$W_t = \begin{pmatrix} w_{11}^t & w_{12}^t & \dots & w_{19}^t \\ w_{21}^t & w_{22}^t & \dots & w_{29}^t \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1}^t & w_{k2}^t & \dots & w_{k9}^t \end{pmatrix}$$

LEARNABLE

N.B. $\Delta t_i = t_{i+1} - t_i$

RELATIVE LEARNABLE TIME ENCODING

Following Times					
Δt_1	...	...	...	...	Δt_L

p is an hyperparameter

$$\Delta = \begin{pmatrix} \Delta t_1 & \Delta t_1^2 & \dots & \Delta t_1^p \\ \Delta t_2 & \Delta t_2^2 & \dots & \Delta t_2^p \\ \vdots & \vdots & \ddots & \vdots \\ \Delta t_L & \Delta t_L^2 & \dots & \Delta t_L^p \end{pmatrix}$$

$$W_t = \begin{pmatrix} w_{11}^t & w_{12}^t & \dots & w_{19}^t \\ w_{21}^t & w_{22}^t & \dots & w_{29}^t \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1}^t & w_{k2}^t & \dots & w_{k9}^t \end{pmatrix}$$

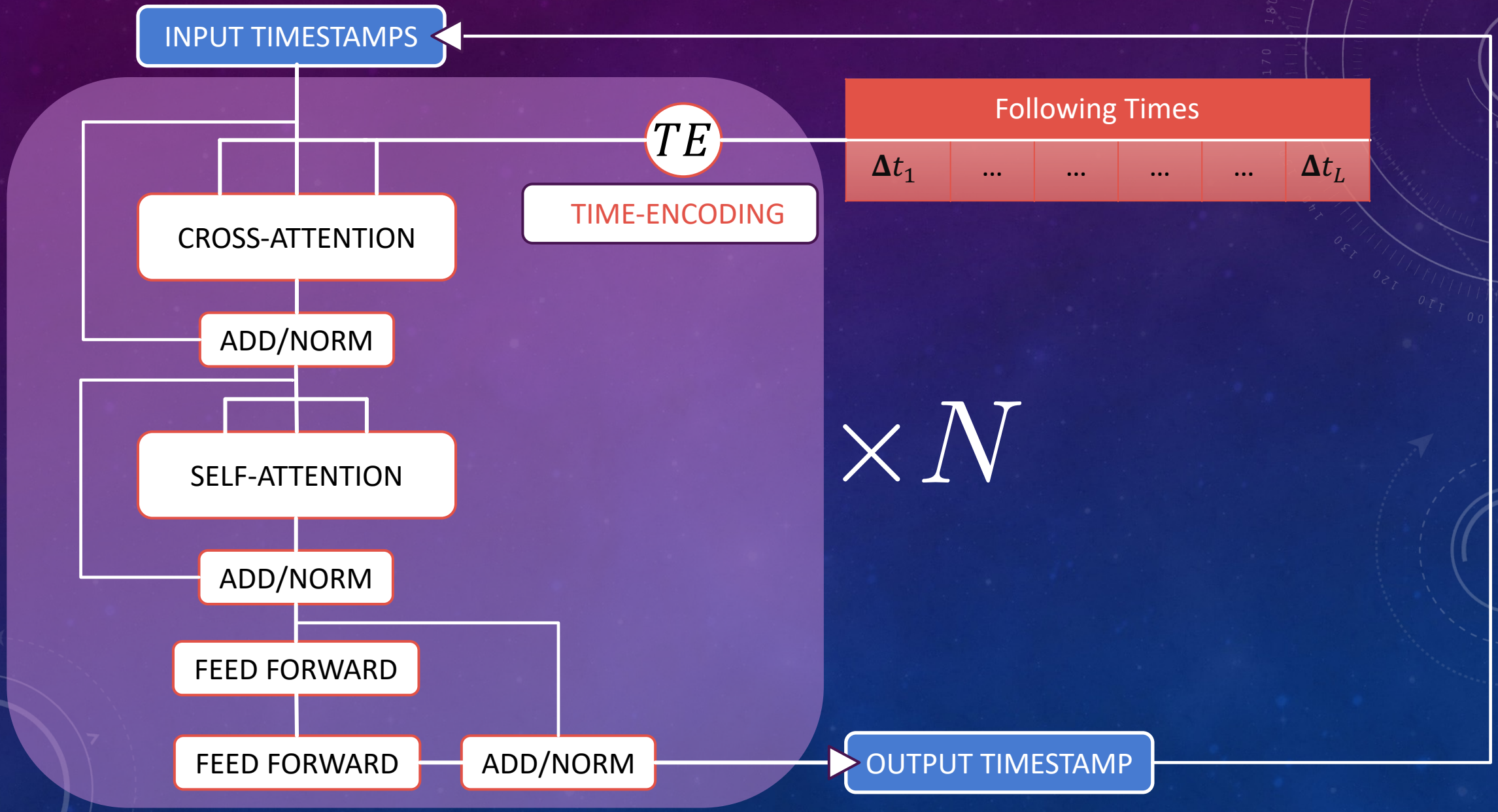
LEARNABLE

$$\Delta \times W_t = T \in \mathbb{R}^{L \times d_m}$$

$d_m = 9$

N.B. $\Delta t_i = t_{i+1} - t_i$

TRANSFORMER DECODER INFERENCE



CROSS ATTENTION LAYER

$$I = \begin{pmatrix} x_1 & y_1 & \dots & a_1^z \\ x_2 & y_2 & \dots & a_2^z \\ \vdots & \vdots & \ddots & \vdots \\ x_L & y_L & \dots & a_L^z \end{pmatrix}$$

The input dimension L changes during training and inference

$$W_q = \begin{pmatrix} w_{11}^q & w_{12}^q & \dots & w_{19}^q \\ w_{21}^q & w_{22}^q & \dots & w_{29}^q \\ \vdots & \vdots & \ddots & \vdots \\ w_{91}^q & w_{92}^q & \dots & w_{99}^q \end{pmatrix} \quad W_k = \begin{pmatrix} w_{11}^k & w_{12}^k & \dots & w_{19}^k \\ w_{21}^k & w_{22}^k & \dots & w_{29}^k \\ \vdots & \vdots & \ddots & \vdots \\ w_{91}^k & w_{92}^k & \dots & w_{99}^k \end{pmatrix}$$

The dimensions of the weight matrices are fixed at the beginning

CROSS ATTENTION LAYER

QUERIES

$$I \times W_q = Q \in \mathbb{R}^{L \times d_m}$$

KEYS

$$I \times W_k = K \in \mathbb{R}^{L \times d_m}$$

TIMES

$$\Delta \times W_t = T \in \mathbb{R}^{L \times d_m}$$

$$\text{score}(Q, K, T) = \text{softmax}\left[\frac{Q \times (K + T)^\top}{\sqrt{d_m}}\right]$$

CROSS ATTENTION LAYER

QUERIES

$$I \times W_q = Q \in \mathbb{R}^{L \times d_m}$$

KEYS

$$I \times W_k = K \in \mathbb{R}^{L \times d_m}$$

TIMES

$$\Delta \times W_t = T \in \mathbb{R}^{L \times d_m}$$

$$\text{score}(Q, K, T) = \text{softmax}\left[\frac{Q \times (K + T)^\top}{\sqrt{d_m}}\right]$$

$$W_v = \begin{pmatrix} w_{11}^v & w_{12}^v & \cdots & w_{19}^v \\ w_{21}^v & w_{22}^v & \cdots & w_{29}^v \\ \vdots & \vdots & \ddots & \vdots \\ w_{91}^v & w_{92}^v & \cdots & w_{99}^v \end{pmatrix}$$

CROSS ATTENTION LAYER

QUERIES

$$I \times W_q = Q \in \mathbb{R}^{L \times d_m}$$

KEYS

$$I \times W_k = K \in \mathbb{R}^{L \times d_m}$$

TIMES

$$\Delta \times W_t = T \in \mathbb{R}^{L \times d_m}$$

$$\text{score}(Q, K, T) = \text{softmax}\left[\frac{Q \times (K + T)^\top}{\sqrt{d_m}}\right]$$

$$W_v = \begin{pmatrix} w_{11}^v & w_{12}^v & \cdots & w_{19}^v \\ w_{21}^v & w_{22}^v & \cdots & w_{29}^v \\ \vdots & \vdots & \ddots & \vdots \\ w_{91}^v & w_{92}^v & \cdots & w_{99}^v \end{pmatrix}$$

VALUES

$$I \times W_V = V \in \mathbb{R}^{L \times d_m}$$

CROSS ATTENTION LAYER

QUERIES

$$I \times W_q = Q \in \mathbb{R}^{L \times d_m}$$

KEYS

$$I \times W_k = K \in \mathbb{R}^{L \times d_m}$$

TIMES

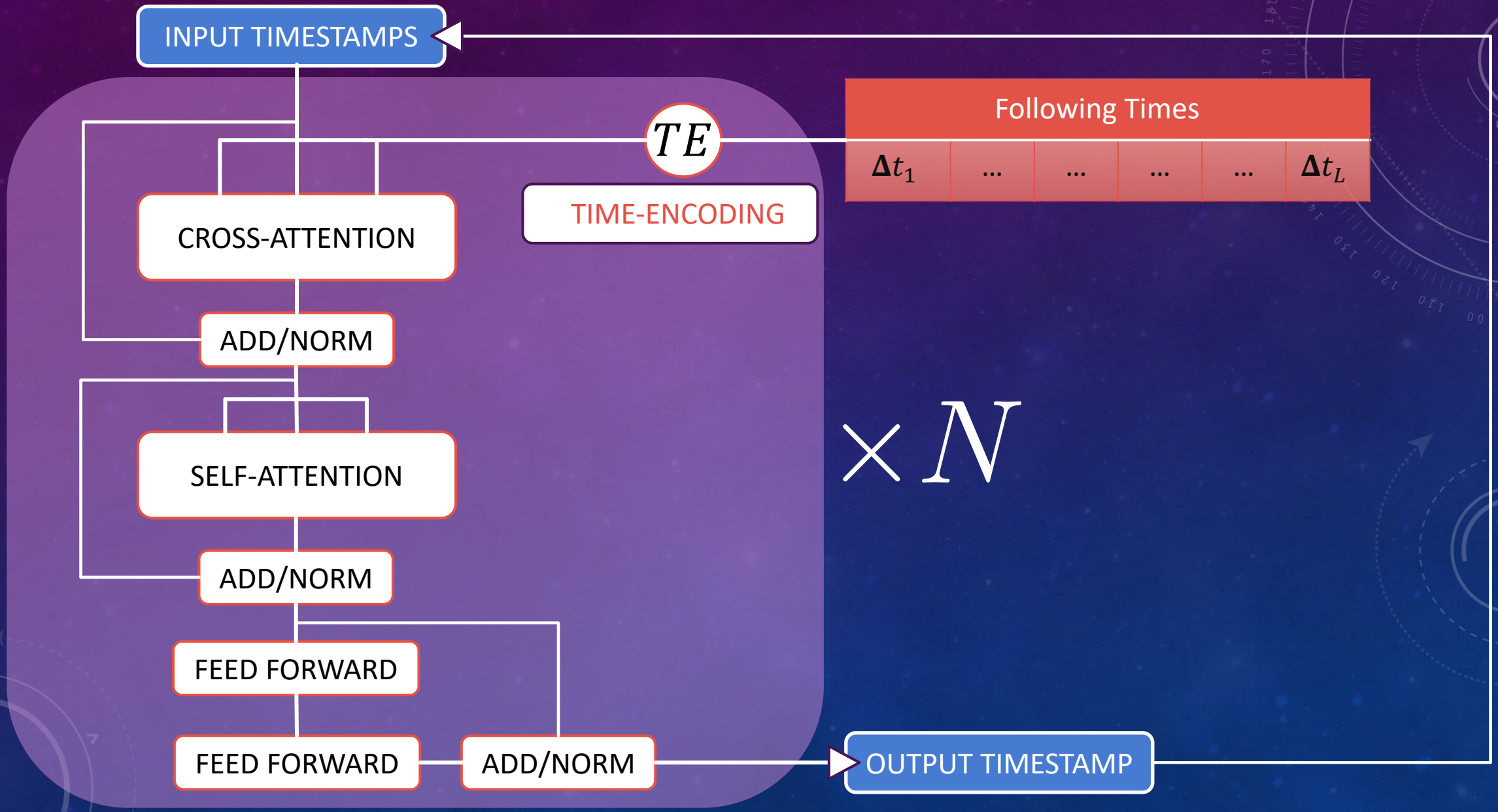
$$\Delta \times W_t = T \in \mathbb{R}^{L \times d_m}$$

VALUES

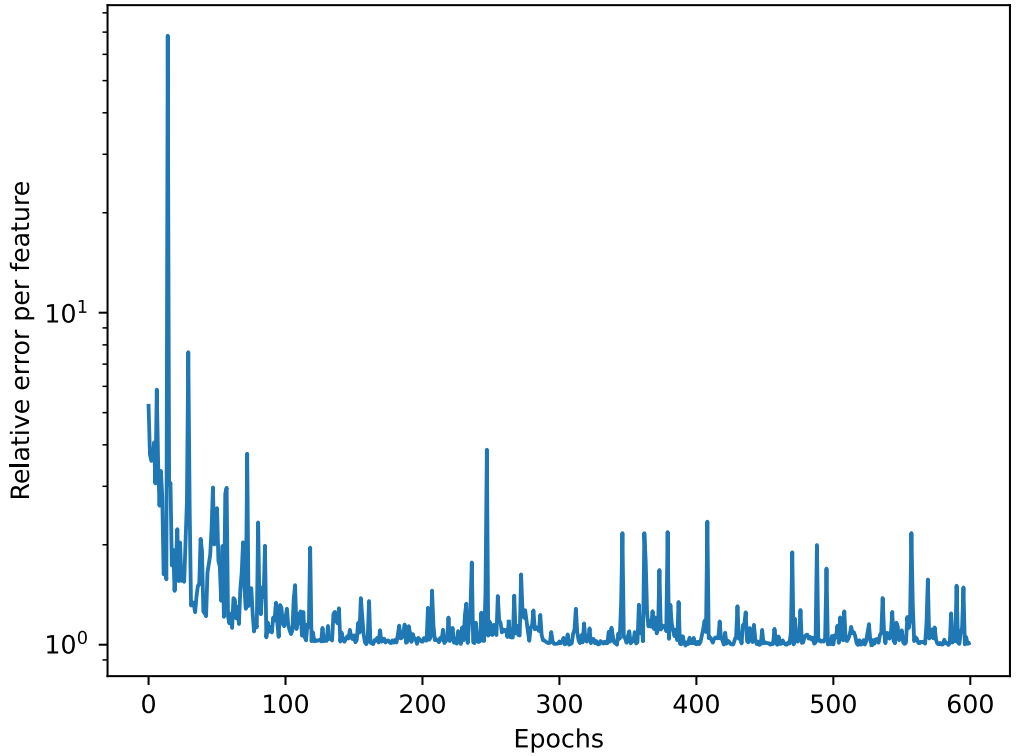
$$I \times W_V = V \in \mathbb{R}^{L \times d_m}$$

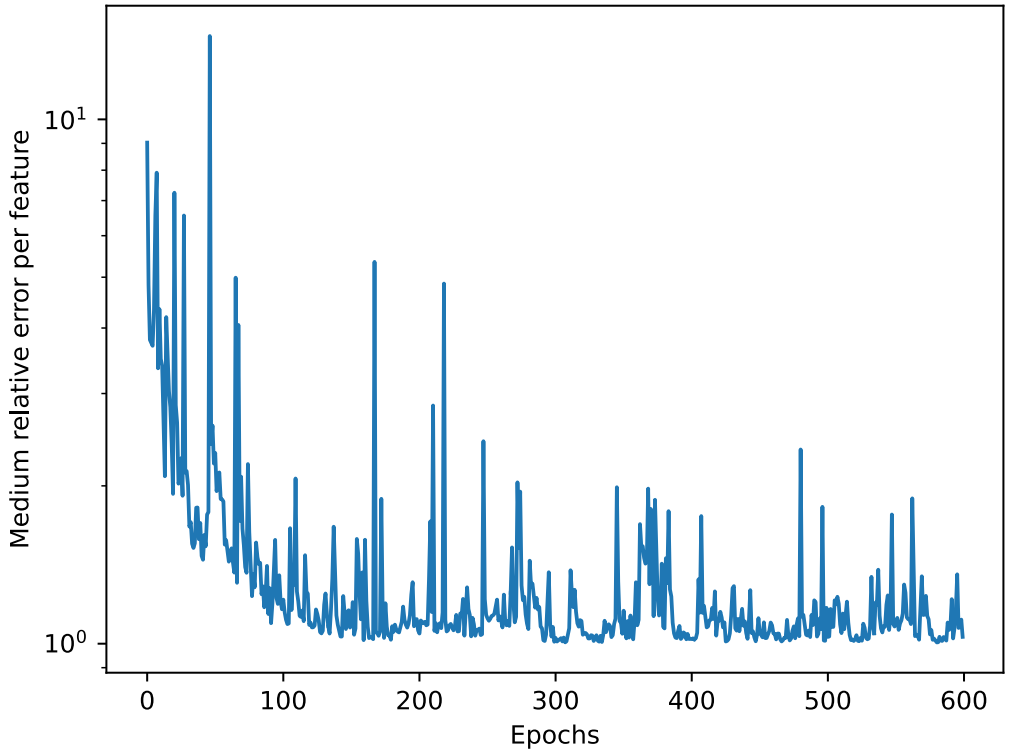
$$\mathrm{att}(Q, K, T, V) = \mathrm{softmax}\left[\frac{Q \times (K + T)^\top}{\sqrt{d_m}}\right] V$$

TRANSFORMER DECODER INFERENCE

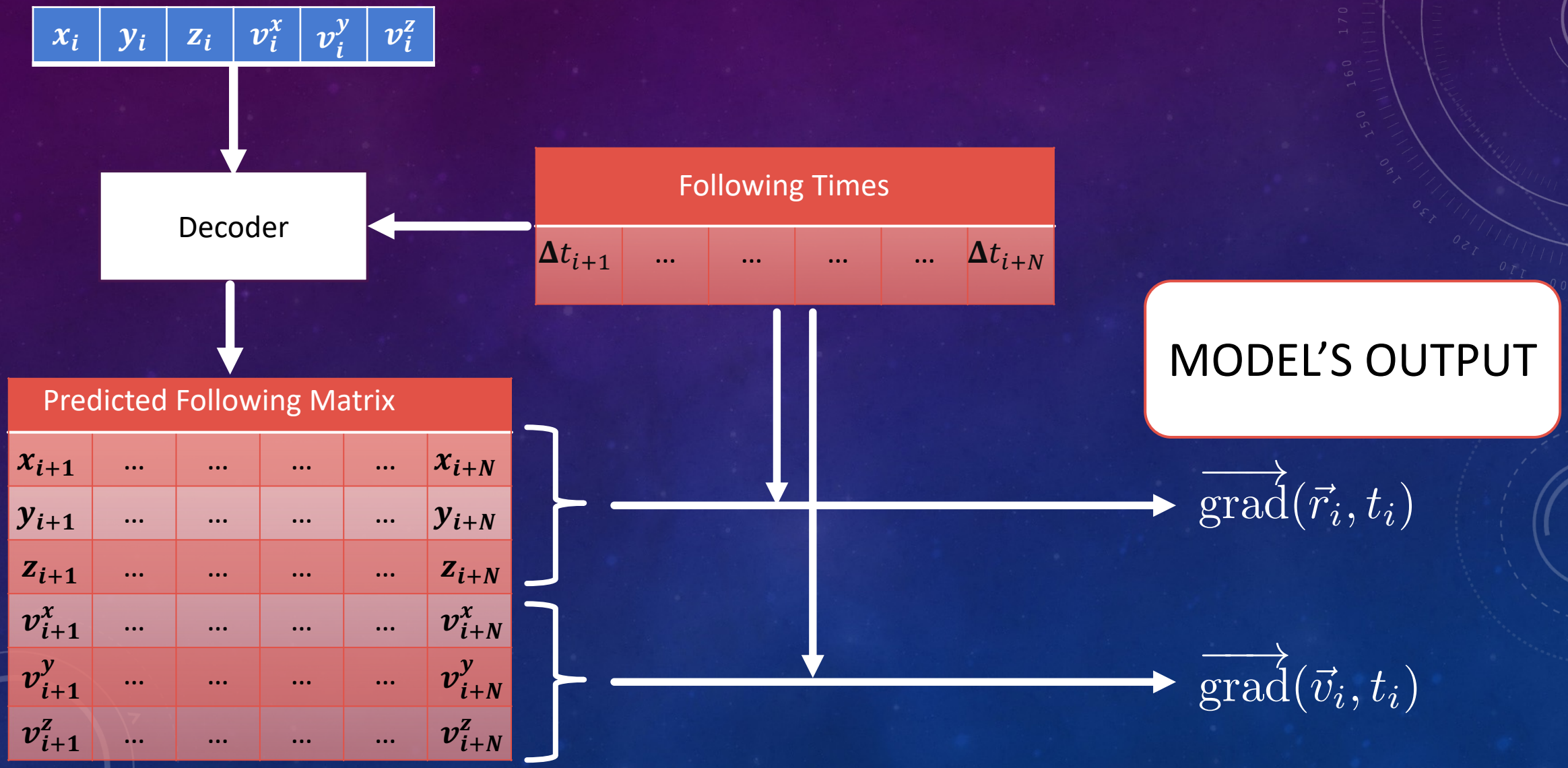


PRELIMINARY RESULTS





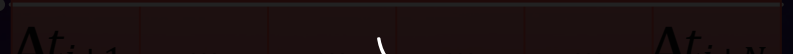
RE-TRAINING: PINN



RE-TRAINING: PINN

Decoder

Following Times



$$\mathcal{L}_i = \alpha\left\{\left[\overrightarrow{\text{grad}}(\vec{r}_i, t_i) - \vec{v}_i\right] + \left[\overrightarrow{\text{grad}}(\vec{v}_i, t_i) - \vec{v}_i \times \vec{B}_i\right]\right\} + \beta\left\{|\vec{v}_i| - |\vec{v}_0|\right\}$$

MODEL'S OUTPUT

Predicted Following Matrix

x_{i+1}	...	...	...	...	x_{i+N}
y_{i+1}	...	...	...	...	y_{i+N}
z_{i+1}	...	...	...	...	z_{i+N}
v_{i+1}^x	...	...	...	...	v_{i+N}^x
v_{i+1}^y	...	...	...	...	v_{i+N}^y
v_{i+1}^z	...	...	...	...	v_{i+N}^z

$\overrightarrow{\text{grad}}(\vec{r}_i, t_i)$

$\overrightarrow{\text{grad}}(\vec{v}_i, t_i)$

THANK YOU
FOR
YOUR ATTENTION

BACKUP SLIDES

ML DATASET: FRAGMENT OF TRAJECTORY

$$\Delta t_k = t_k - t_i$$

$$\vec{a}_k = \vec{v}_k \times \vec{B}(\vec{r}_k)$$

NORMALIZED MASS AND CHARGE:
m=1
q=1

Δt_{i-N}	…	…	…	…	…	…	t_i	…	…	…	…	…	…	Δt_{i+N}
x_{i-N}	…	…	…	…	…	…	x_i	…	…	…	…	…	…	x_{i+N}
y_{i-N}	…	…	…	…	…	…	y_i	…	…	…	…	…	…	y_{i+N}
z_{i-N}	…	…	…	…	…	…	z_i	…	…	…	…	…	…	z_{i+N}
v^x_{i-N}	…	…	…	…	…	…	v^x_i	…	…	…	…	…	…	v^x_{i+N}
v^y_{i-N}	…	…	…	…	…	…	v^y_i	…	…	…	…	…	…	v^y_{i+N}
v^z_{i-N}	…	…	…	…	…	…	v^z_i	…	…	…	…	…	…	v^z_{i+N}
a^x_{i-N}	…	…	…	…	…	…	a^x_i	…	…	…	…	…	…	a^x_{i+N}
a^y_{i-N}	…	…	…	…	…	…	a^y_i	…	…	…	…	…	…	a^y_{i+N}
a^z_{i-N}	…	…	…	…	…	…	a^z_i	…	…	…	…	…	…	a^z_{i+N}

ML DATASET

CENTRAL TIMESTAMP

Δt_{i-N}	…	…	…	…	…	…	t_i	…	…	…	…	…	…	Δt_{i+N}
x_{i-N}	…	…	…	…	…	…	x_i	…	…	…	…	…	…	x_{i+N}
y_{i-N}	…	…	…	…	…	…	y_i	…	…	…	…	…	…	y_{i+N}
z_{i-N}	…	…	…	…	…	…	z_i	…	…	…	…	…	…	z_{i+N}
v^x_{i-N}	…	…	…	…	…	…	v^x_i	…	…	…	…	…	…	v^x_{i+N}
v^y_{i-N}	…	…	…	…	…	…	v^y_i	…	…	…	…	…	…	v^y_{i+N}
v^z_{i-N}	…	…	…	…	…	…	v^z_i	…	…	…	…	…	…	v^z_{i+N}
a^x_{i-N}	…	…	…	…	…	…	a^x_i	…	…	…	…	…	…	a^x_{i+N}
a^y_{i-N}	…	…	…	…	…	…	a^y_i	…	…	…	…	…	…	a^y_{i+N}
a^z_{i-N}	…	…	…	…	…	…	a^z_i	…	…	…	…	…	…	a^z_{i+N}

ML DATASET

PREVIOUS MATRIX

Δt_{i-N}	…	…	…	…	…	…	t_i	…	…	…	…	…	…	Δt_{i+N}
x_{i-N}	…	…	…	…	…	…	x_i	…	…	…	…	…	…	x_{i+N}
y_{i-N}	…	…	…	…	…	…	y_i	…	…	…	…	…	…	y_{i+N}
z_{i-N}	…	…	…	…	…	…	z_i	…	…	…	…	…	…	z_{i+N}
v^x_{i-N}	…	…	…	…	…	…	v^x_i	…	…	…	…	…	…	v^x_{i+N}
v^y_{i-N}	…	…	…	…	…	…	v^y_i	…	…	…	…	…	…	v^y_{i+N}
v^z_{i-N}	…	…	…	…	…	…	v^z_i	…	…	…	…	…	…	v^z_{i+N}
a^x_{i-N}	…	…	…	…	…	…	a^x_i	…	…	…	…	…	…	a^x_{i+N}
a^y_{i-N}	…	…	…	…	…	…	a^y_i	…	…	…	…	…	…	a^y_{i+N}
a^z_{i-N}	…	…	…	…	…	…	a^z_i	…	…	…	…	…	…	a^z_{i+N}

ML DATASET

FOLLOWING MATRIX

Δt_{i-N}	...	...	...	...	...	...	t_i	...	...	...	...	...	...	Δt_{i+N}
x_{i-N}	...	...	...	...	...	...	x_i	...	...	...	...	...	...	x_{i+N}
y_{i-N}	...	...	...	...	...	...	y_i	...	...	...	...	...	...	y_{i+N}
z_{i-N}	...	...	...	...	...	...	z_i	...	...	...	...	...	...	z_{i+N}
v^x_{i-N}	...	...	...	...	...	...	v^x_i	...	...	...	...	...	...	v^x_{i+N}
v^y_{i-N}	...	...	...	...	...	...	v^y_i	...	...	...	...	...	...	v^y_{i+N}
v^z_{i-N}	...	...	...	...	...	...	v^z_i	...	...	...	...	...	...	v^z_{i+N}
a^x_{i-N}	...	...	...	...	...	...	a^x_i	...	...	...	...	...	...	a^x_{i+N}
a^y_{i-N}	...	...	...	...	...	...	a^y_i	...	...	...	...	...	...	a^y_{i+N}
a^z_{i-N}	...	...	...	...	...	...	a^z_i	...	...	...	...	...	...	a^z_{i+N}

ML DATASET

Δt_{i-N}	…	…	…	…	…	…	t_i	…	…	…	…	…	…	Δt_{i+N}
x_{i-N}	…	…	…	…	…	…	x_i	…	…	…	…	…	…	x_{i+N}
y_{i-N}	…	…	…	…	…	…	y_i	…	…	…	…	…	…	y_{i+N}
z_{i-N}	…	…	…	…	…	…	z_i	…	…	…	…	…	…	z_{i+N}
v^x_{i-N}	…	…	…	…	…	…	v^x_i	…	…	…	…	…	…	v^x_{i+N}
v^y_{i-N}	…	…	…	…	…	…	v^y_i	…	…	…	…	…	…	v^y_{i+N}
v^z_{i-N}	…	…	…	…	…	…	v^z_i	…	…	…	…	…	…	v^z_{i+N}
a^x_{i-N}	…	…	…	…	…	…	a^x_i	…	…	…	…	…	…	a^x_{i+N}
a^y_{i-N}	…	…	…	…	…	…	a^y_i	…	…	…	…	…	…	a^y_{i+N}
a^z_{i-N}	…	…	…	…	…	…	a^z_i	…	…	…	…	…	…	a^z_{i+N}

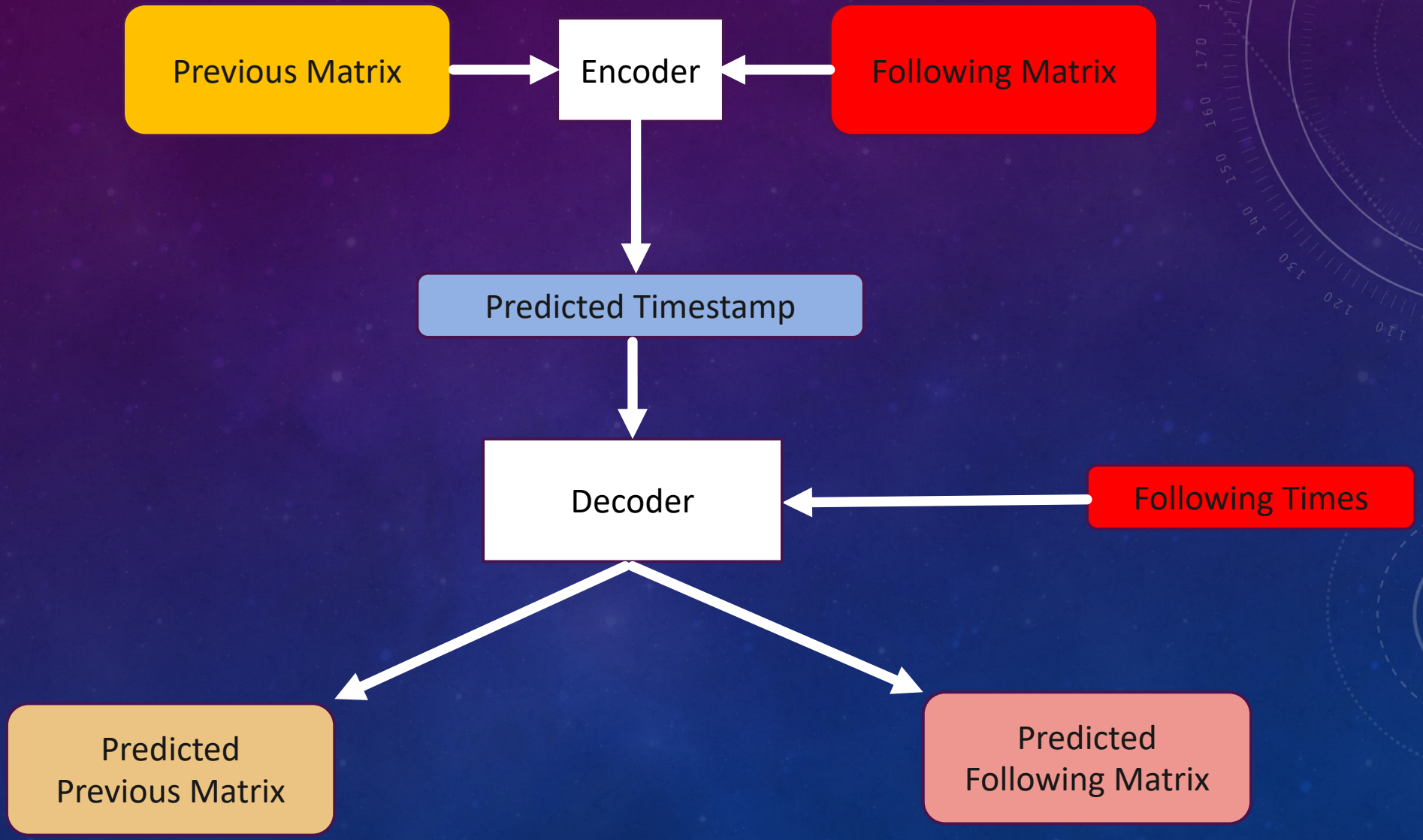
NETWORK INPUT

Previous Matrix					
Δt_{i-N}	…	…	…	…	Δt_{i-1}
x_{i-N}	…	…	…	…	x_{i-1}
y_{i-N}	…	…	…	…	y_{i-1}
z_{i-N}	…	…	…	…	z_{i-1}
v^x_{i-N}	…	…	…	…	v^x_{i-1}
v^y_{i-N}	…	…	…	…	v^y_{i-1}
v^z_{i-N}	…	…	…	…	v^z_{i-1}
a^x_{i-N}	…	…	…	…	a^x_{i-1}
a^y_{i-N}	…	…	…	…	a^y_{i-1}
a^z_{i-N}	…	…	…	…	a^z_{i-1}

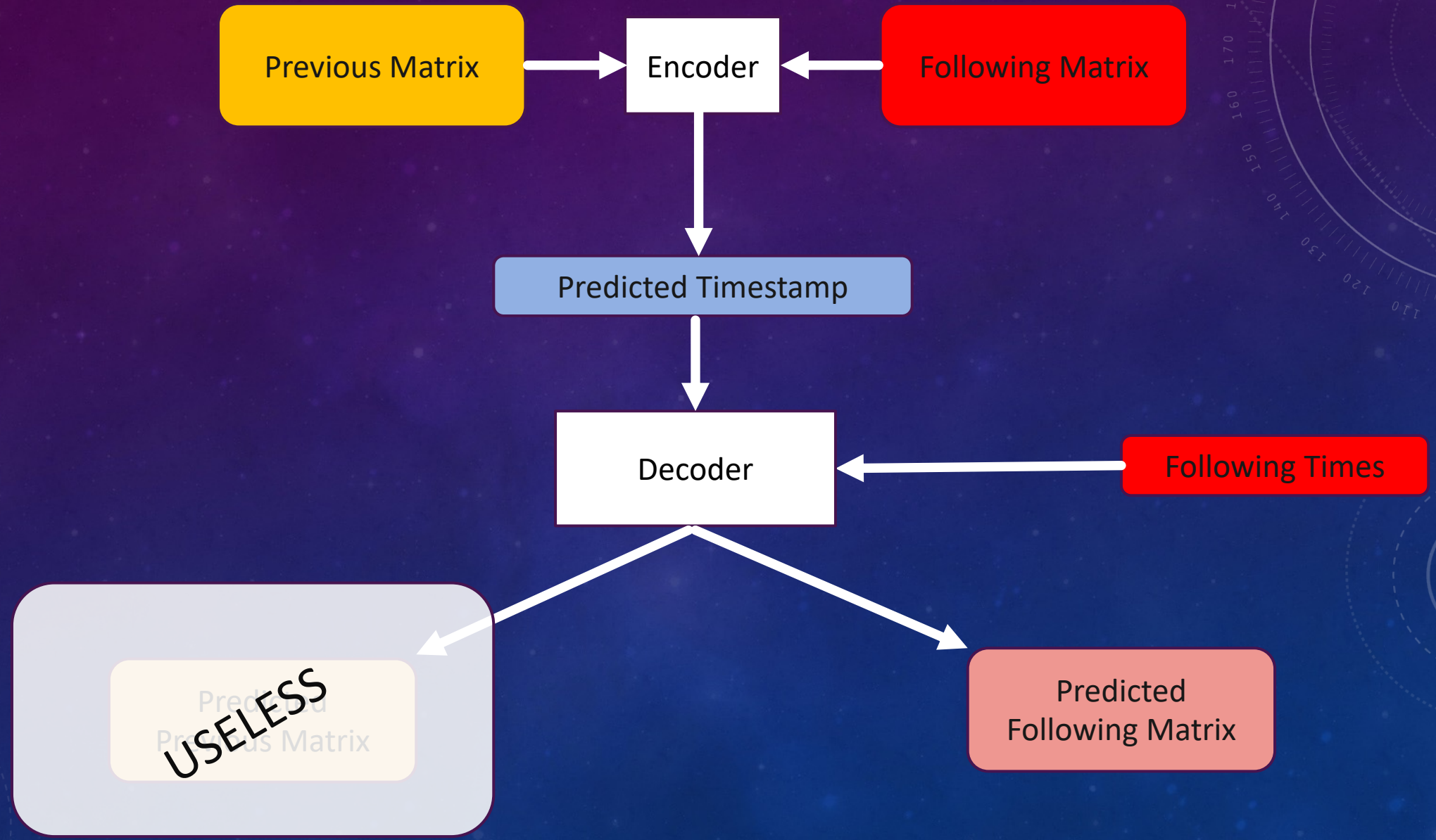
Following Matrix					
Δt_{i+1}	…	…	…	…	Δt_{i+N}
x_{i+1}	…	…	…	…	x_{i+N}
y_{i+1}	…	…	…	…	y_{i+N}
z_{i+1}	…	…	…	…	z_{i+N}
v^x_{i+1}	…	…	…	…	v^x_{i+N}
v^y_{i+1}	…	…	…	…	v^y_{i+N}
v^z_{i+1}	…	…	…	…	v^z_{i+N}
a^x_{i+1}	…	…	…	…	a^x_{i+N}
a^y_{i+1}	…	…	…	…	a^y_{i+N}
a^z_{i+1}	…	…	…	…	a^z_{i+N}

Following Times					
Δt_{i+1}	…	…	…	…	Δt_{i+N}

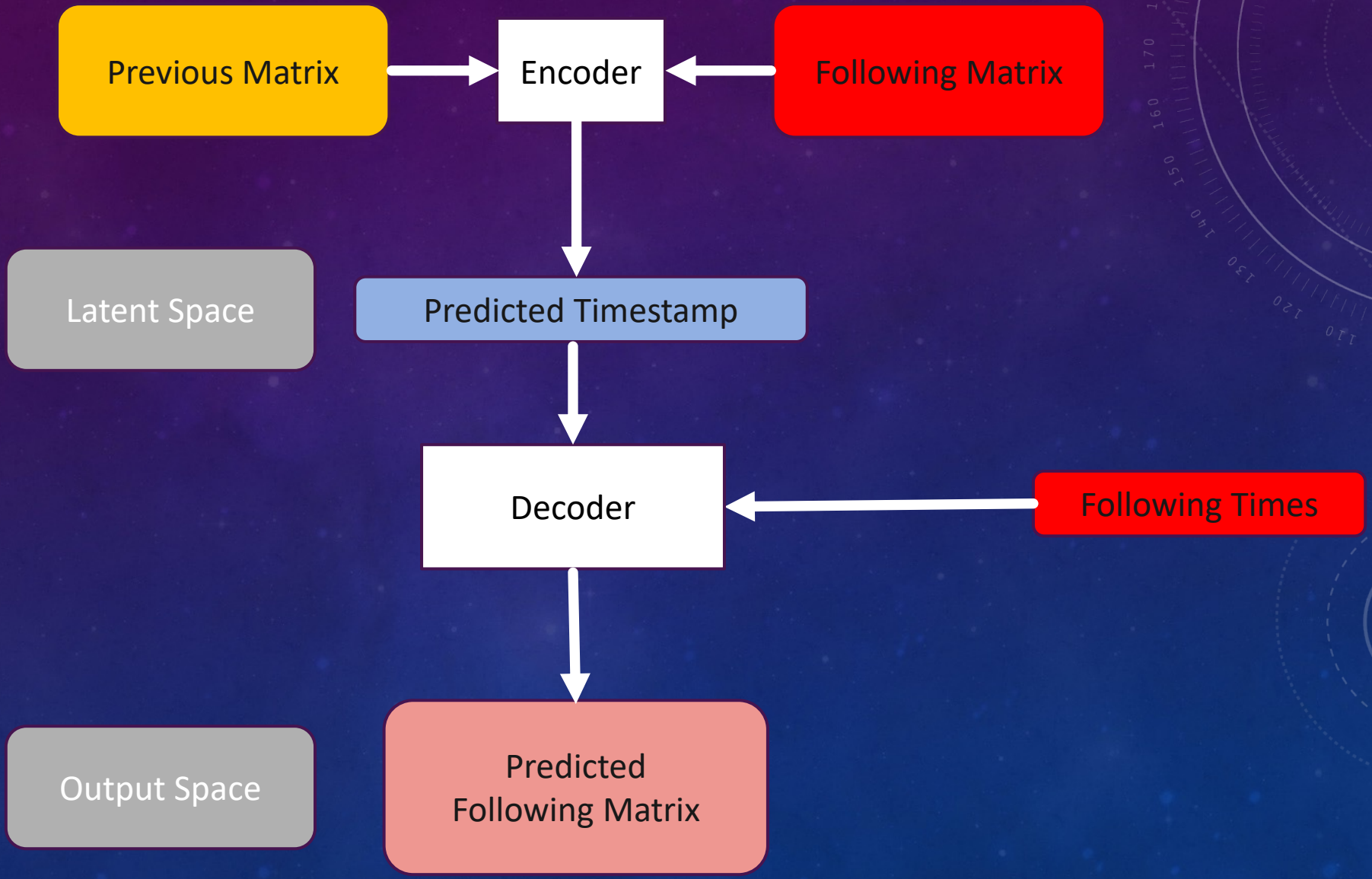
NETWORK ARCHITECTURE: VARIATIONAL AUTO-ENCODER



NETWORK ARCHITECTURE: VARIATIONAL AUTO-ENCODER



NETWORK ARCHITECTURE: VARIATIONAL AUTO-ENCODER



NETWORK OUTPUT

Predicted Timestamp
x_i
y_i
z_i
v_i^x
v_i^y
v_i^z

Predicted Following Matrix					
x_{i+1}	…	…	…	…	x_{i+N}
y_{i+1}	…	…	…	…	y_{i+N}
z_{i+1}	…	…	…	…	z_{i+N}
v_{i+1}^x	…	…	…	…	v_{i+N}^x
v_{i+1}^y	…	…	…	…	v_{i+N}^y
v_{i+1}^z	…	…	…	…	v_{i+N}^z

TRAINING EVENT

Previous Matrix					
Δt_{i-N}	…	…	…	…	Δt_{i-1}
x_{i-N}	…	…	…	…	x_{i-1}
y_{i-N}	…	…	…	…	y_{i-1}
z_{i-N}	…	…	…	…	z_{i-1}
v^x_{i-N}	…	…	…	…	v^x_{i-1}
v^y_{i-N}	…	…	…	…	v^y_{i-1}
v^z_{i-N}	…	…	…	…	v^z_{i-1}
a^x_{i-N}	…	…	…	…	a^x_{i-1}
a^y_{i-N}	…	…	…	…	a^y_{i-1}
a^z_{i-N}	…	…	…	…	a^z_{i-1}

t_i
x_i
y_i
z_i
v^x_i
v^y_i
v^z_i
a^x_i
a^y_i
a^z_i

Following Matrix					
Δt_{i+1}	…	…	…	…	Δt_{i+N}
x_{i+1}	…	…	…	…	x_{i+N}
y_{i+1}	…	…	…	…	y_{i+N}
z_{i+1}	…	…	…	…	z_{i+N}
v^x_{i+1}	…	…	…	…	v^x_{i+N}
v^y_{i+1}	…	…	…	…	v^y_{i+N}
v^z_{i+1}	…	…	…	…	v^z_{i+N}
a^x_{i+1}	…	…	…	…	a^x_{i+N}
a^y_{i+1}	…	…	…	…	a^y_{i+N}
a^z_{i+1}	…	…	…	…	a^z_{i+N}

CURRENT EVENT

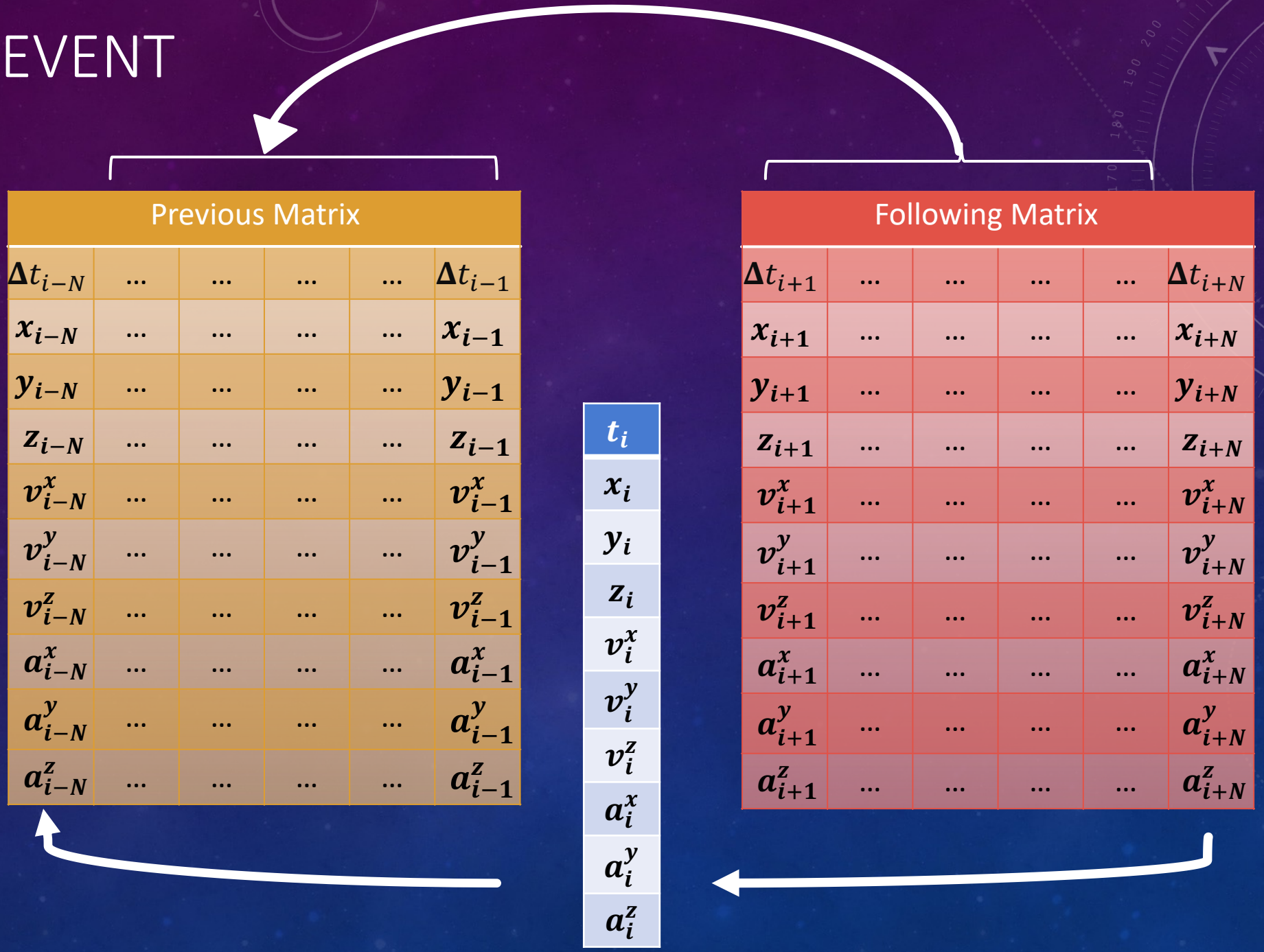
OUTPUT

Predicted Timestamp
x_i
y_i
z_i
v_i^x
v_i^y
v_i^z

Predicted Following Matrix					
x_{i+1}	…	…	…	…	x_{i+N}
y_{i+1}	…	…	…	…	y_{i+N}
z_{i+1}	…	…	…	…	z_{i+N}
v_{i+1}^x	…	…	…	…	v_{i+N}^x
v_{i+1}^y	…	…	…	…	v_{i+N}^y
v_{i+1}^z	…	…	…	…	v_{i+N}^z

CURRENT OUTPUT

TRAINING EVENT



TRAINING EVENT

Previous Matrix					
Δt_i	…	…	…	…	Δt_{i+N-1}
x_i	…	…	…	…	x_{i+N-1}
y_i	…	…	…	…	y_{i+N-1}
z_i	…	…	…	…	z_{i+N-1}
v_i^x	…	…	…	…	v_{i+N-1}^x
v_i^y	…	…	…	…	v_{i+N-1}^y
v_i^z	…	…	…	…	v_{i+N-1}^z
a_i^x	…	…	…	…	a_{i+N-1}^x
a_i^y	…	…	…	…	a_{i+N-1}^y
a_i^z	…	…	…	…	a_{i+N-1}^z

t_{i+N}
x_{i+N}
y_{i+N}
z_{i+N}
v_{i+N}^x
v_{i+N}^y
v_{i+N}^z
a_{i+N}^x
a_{i+N}^y
a_{i+N}^z

Following Matrix					
Δt_{i+N+1}	…	…	…	…	Δt_{i+2N}
x_{i+N+1}	…	…	…	…	x_{i+2N}
y_{i+N+1}	…	…	…	…	y_{i+2N}
z_{i+N+1}	…	…	…	…	z_{i+2N}
v_{i+N+1}^x	…	…	…	…	v_{i+2N}^x
v_{i+N+1}^y	…	…	…	…	v_{i+2N}^y
v_{i+N+1}^z	…	…	…	…	v_{i+2N}^z
a_{i+N+1}^x	…	…	…	…	a_{i+2N}^x
a_{i+N+1}^y	…	…	…	…	a_{i+2N}^y
a_{i+N+1}^z	…	…	…	…	a_{i+2N}^z

NEXT EVENT

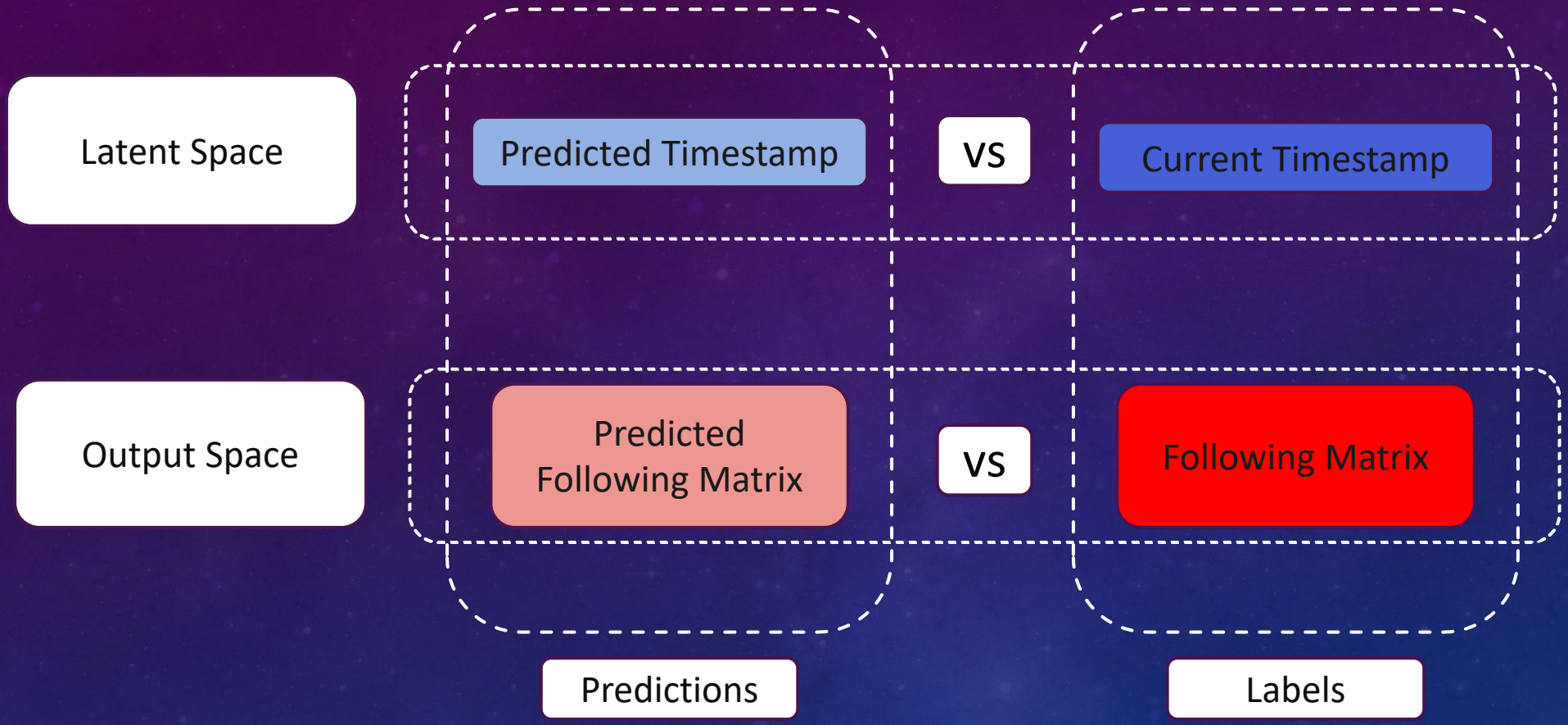
OUTPUT

Predicted Timestamp
x_{i+N}
y_{i+N}
z_{i+N}
v^x_{i+N}
v^y_{i+N}
v^z_{i+N}

PredictedFollowing Matrix					
x_{i+N+1}	…	…	…	…	x_{i+2N}
y_{i+N+1}	…	…	…	…	y_{i+2N}
z_{i+N+1}	…	…	…	…	z_{i+2N}
v^x_{i+N+1}	…	…	…	…	v^x_{i+2N}
v^y_{i+N+1}	…	…	…	…	v^y_{i+2N}
v^z_{i+N+1}	…	…	…	…	v^z_{i+2N}

NEXT OUTPUT

TRAINING LOSS

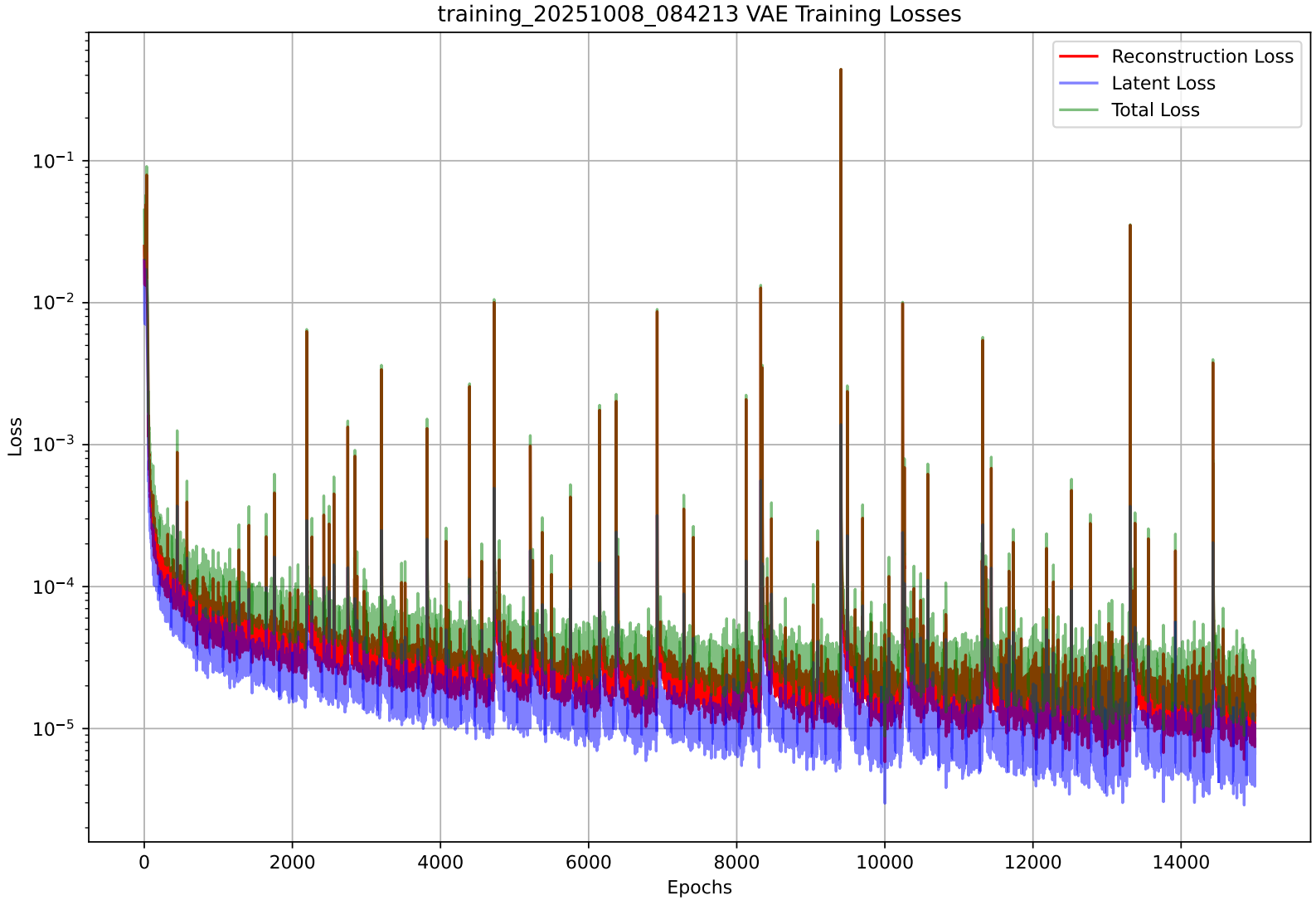


$$\mathcal{L}_{\mathrm{tot}} = \mathrm{MSE}(\mathrm{Timestamps}) + \mathrm{MSE}(\mathrm{Following\ Matrices})$$

TRAININ

Latent Spa

Output Sp

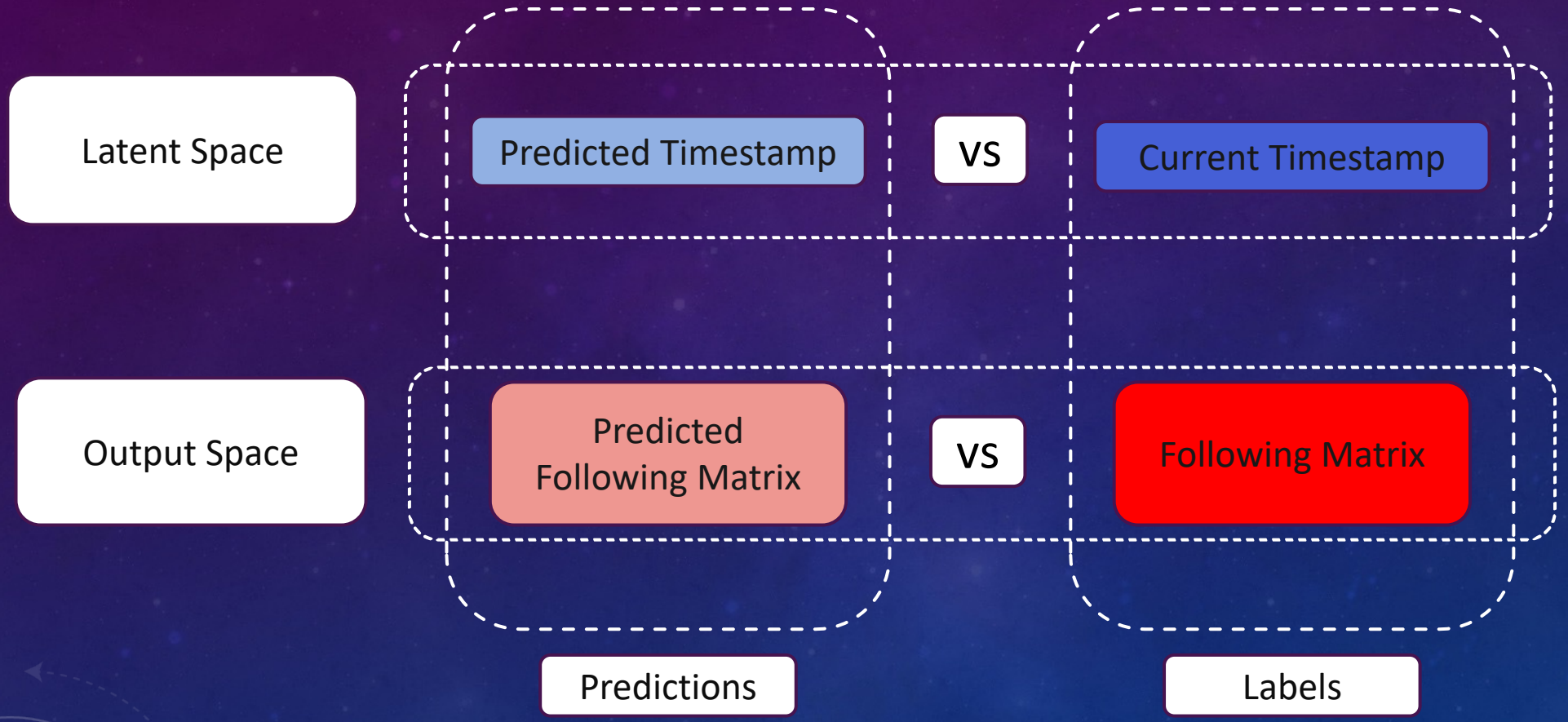


Timestamps)

owing Matrices)

$$\mathcal{L}_{\text{tot}} = \text{MSE}(\text{Timestamps}) + \text{MSE}(\text{Following Matrices})$$

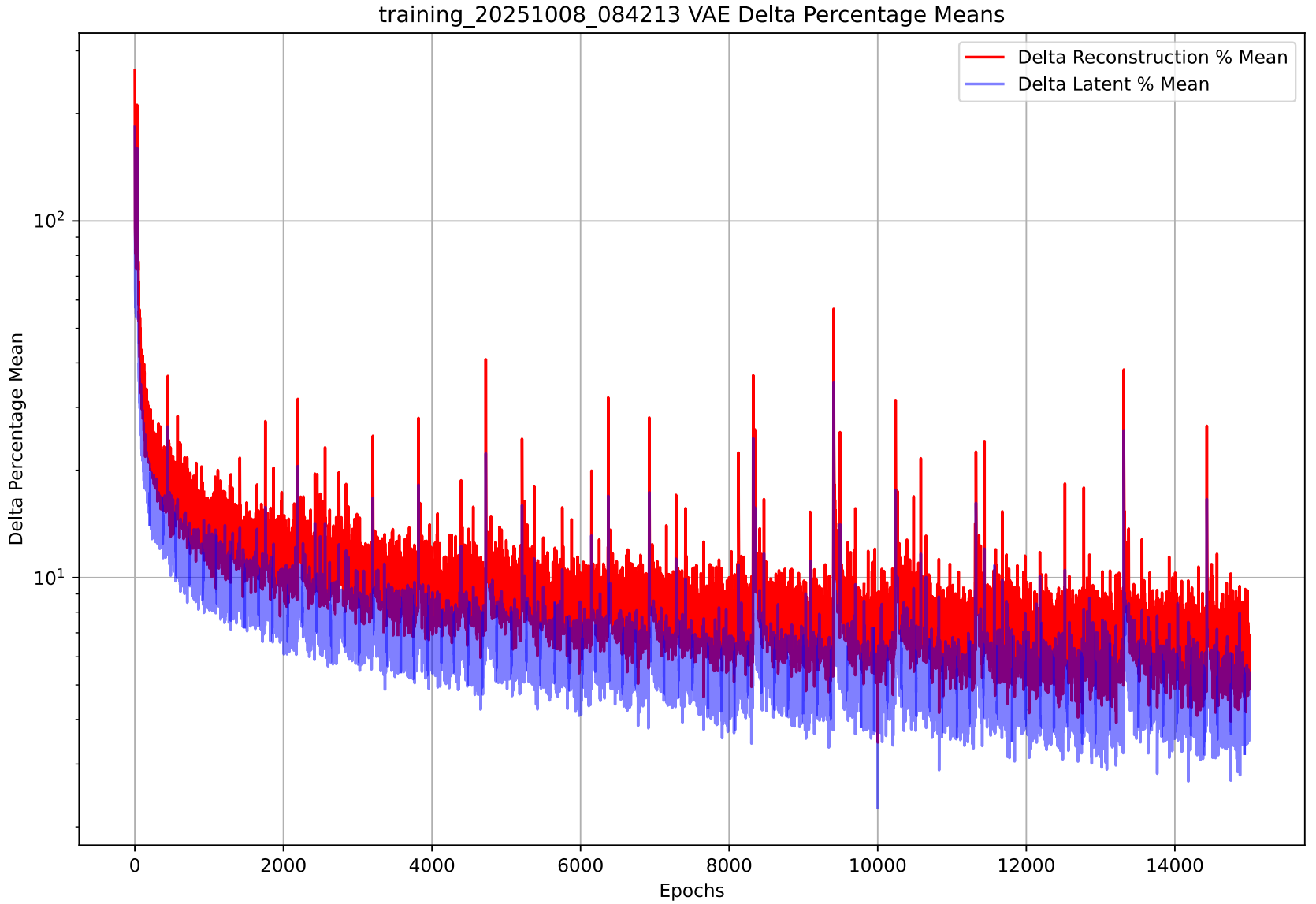
TRAINING LOSS



TRAINI

Latent S

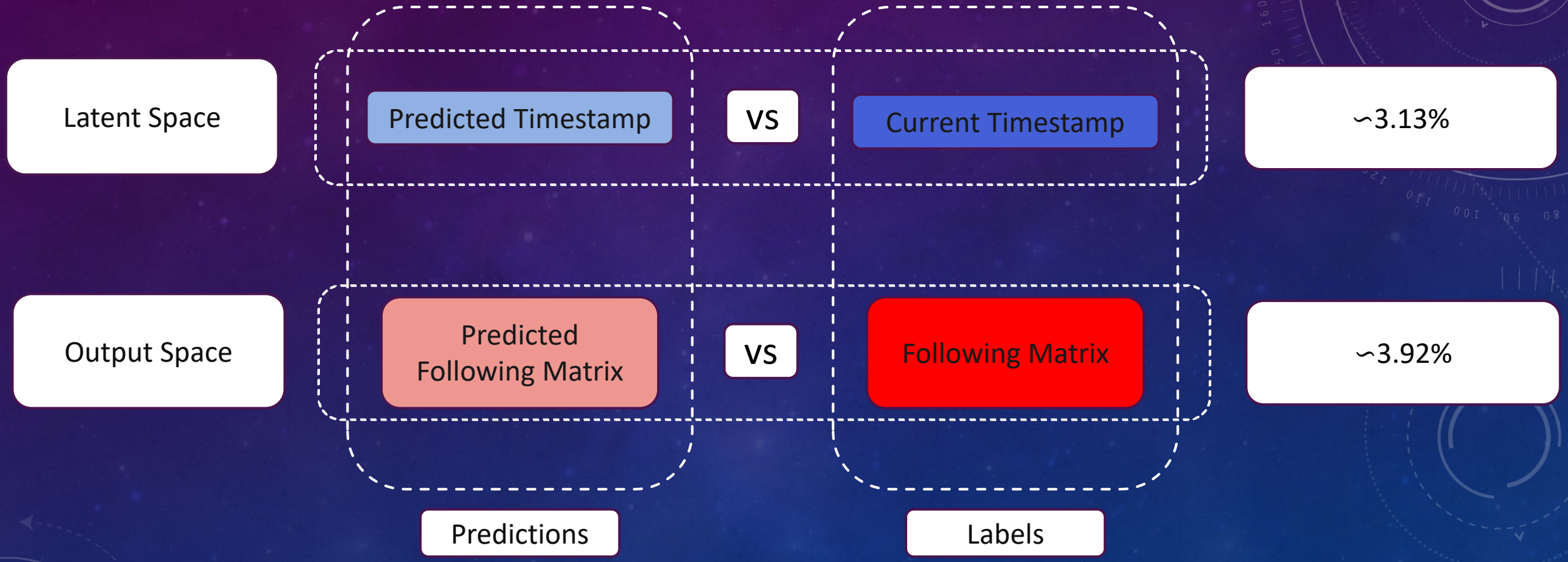
Output



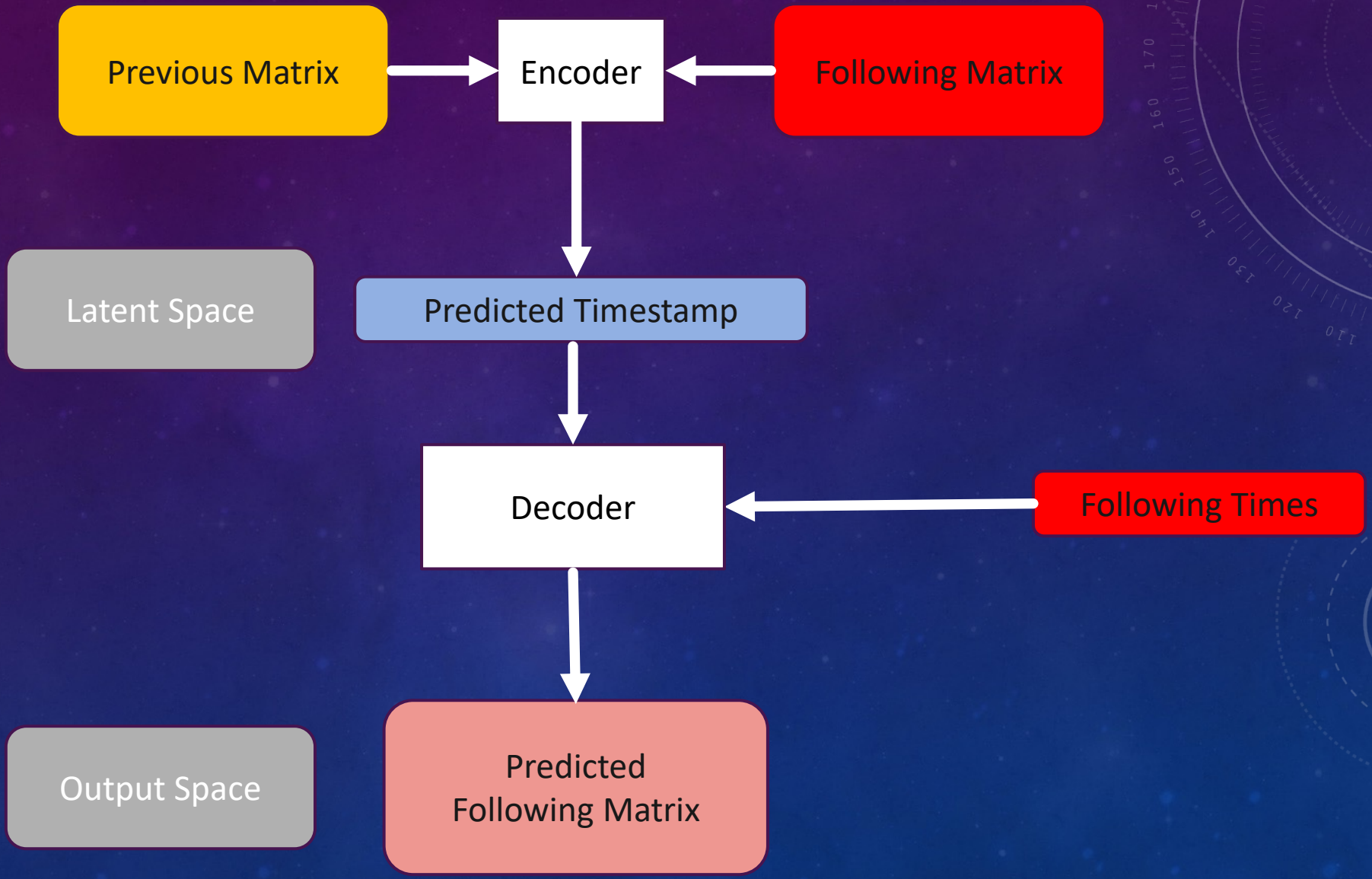
mestamps)

ing Matrices)

TRAINING LOSS



PREDICTION: CONSTRUCTION OF THE TRAJECTORY



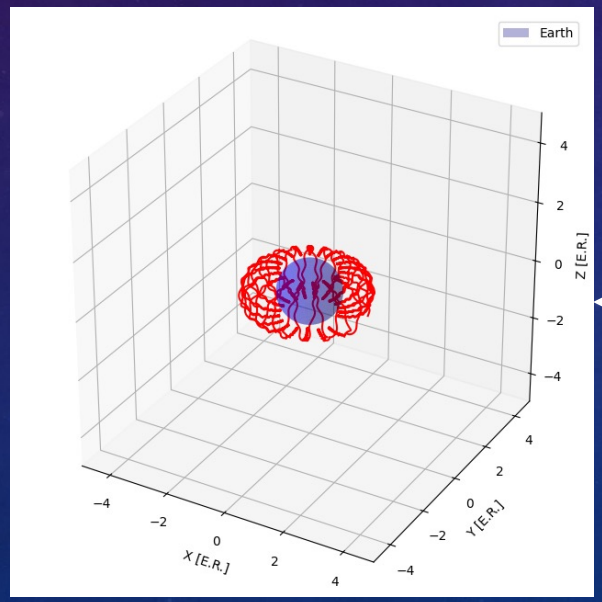
PREDICTION: CONSTRUCTION OF THE TRAJECTORY

Measeured particle
x_0
y_0
z_0
v_0^x
v_0^y
v_0^z
a_0^x
a_0^y
a_0^z

x_i	y_i	z_i	v_i^x	v_i^y	v_i^z

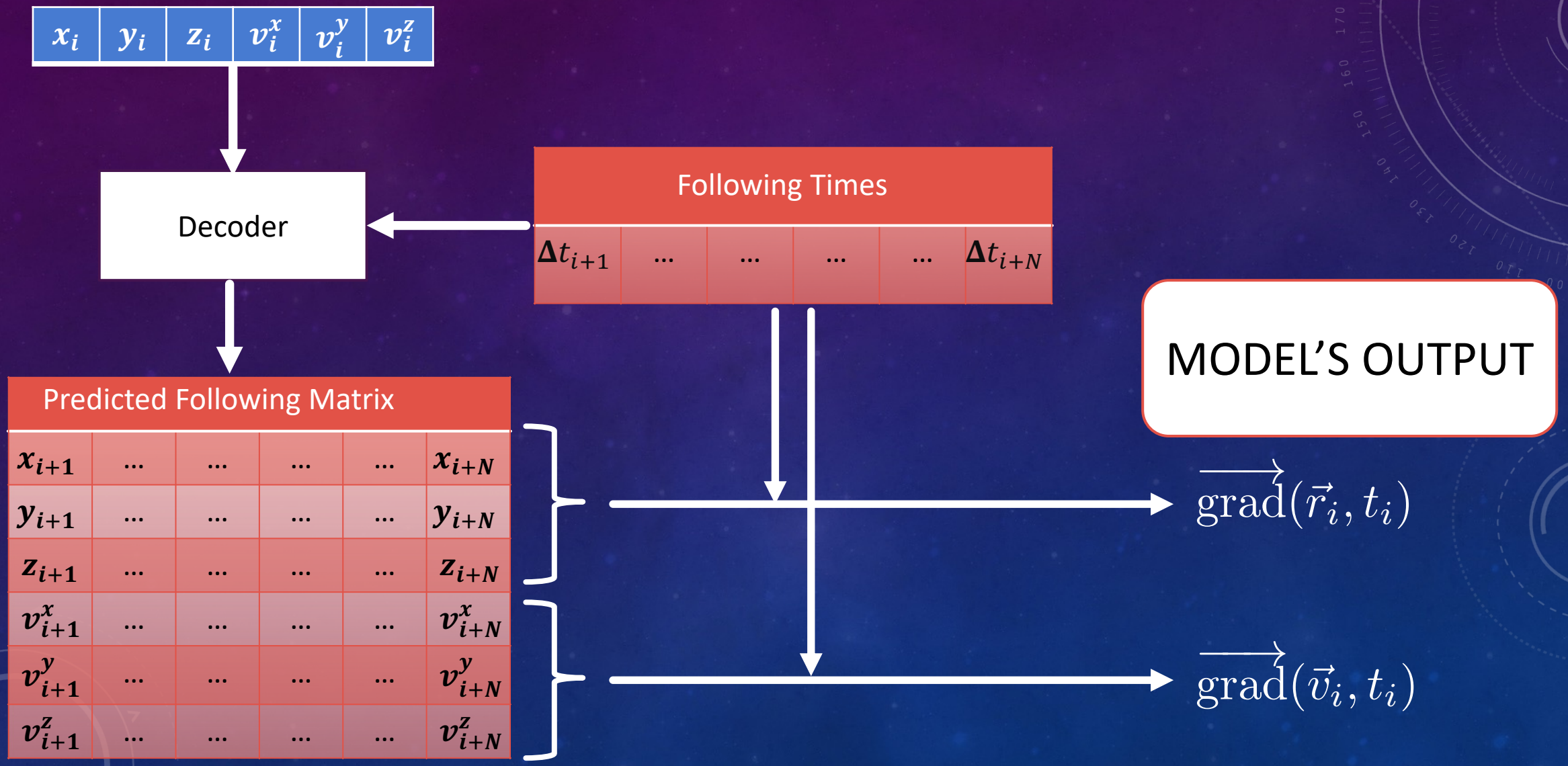
Following Times					
Δt_{i+1}	...	...	...	...	Δt_{i+N}

Decoder

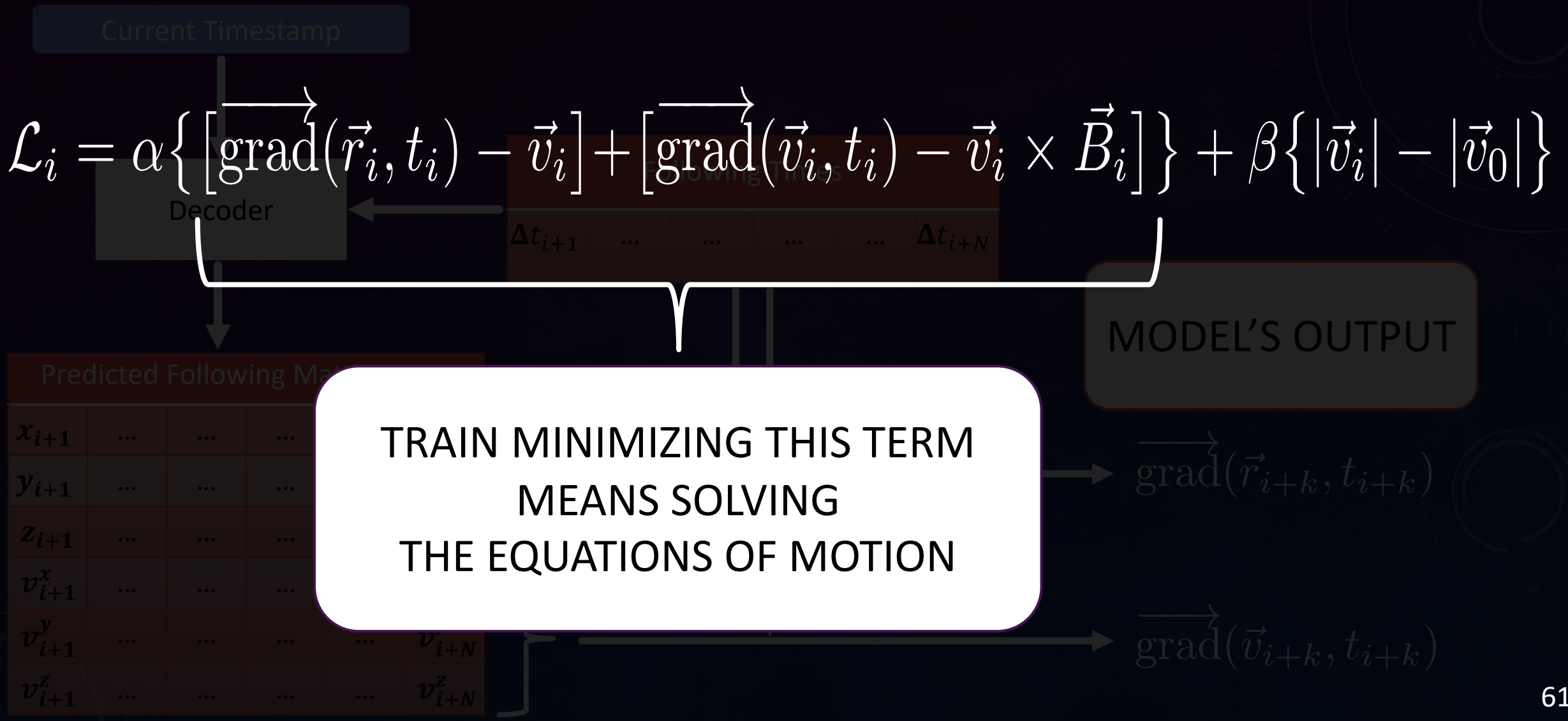


Predicted Following Matrix					
x_{i+1}	...	...	...	...	x_{i+N}
y_{i+1}	...	...	...	...	y_{i+N}
z_{i+1}	...	...	...	...	z_{i+N}
v_{i+1}^x	...	...	...	...	v_{i+N}^x
v_{i+1}^y	...	...	...	...	v_{i+N}^y
v_{i+1}^z	...	...	...	...	v_{i+N}^z

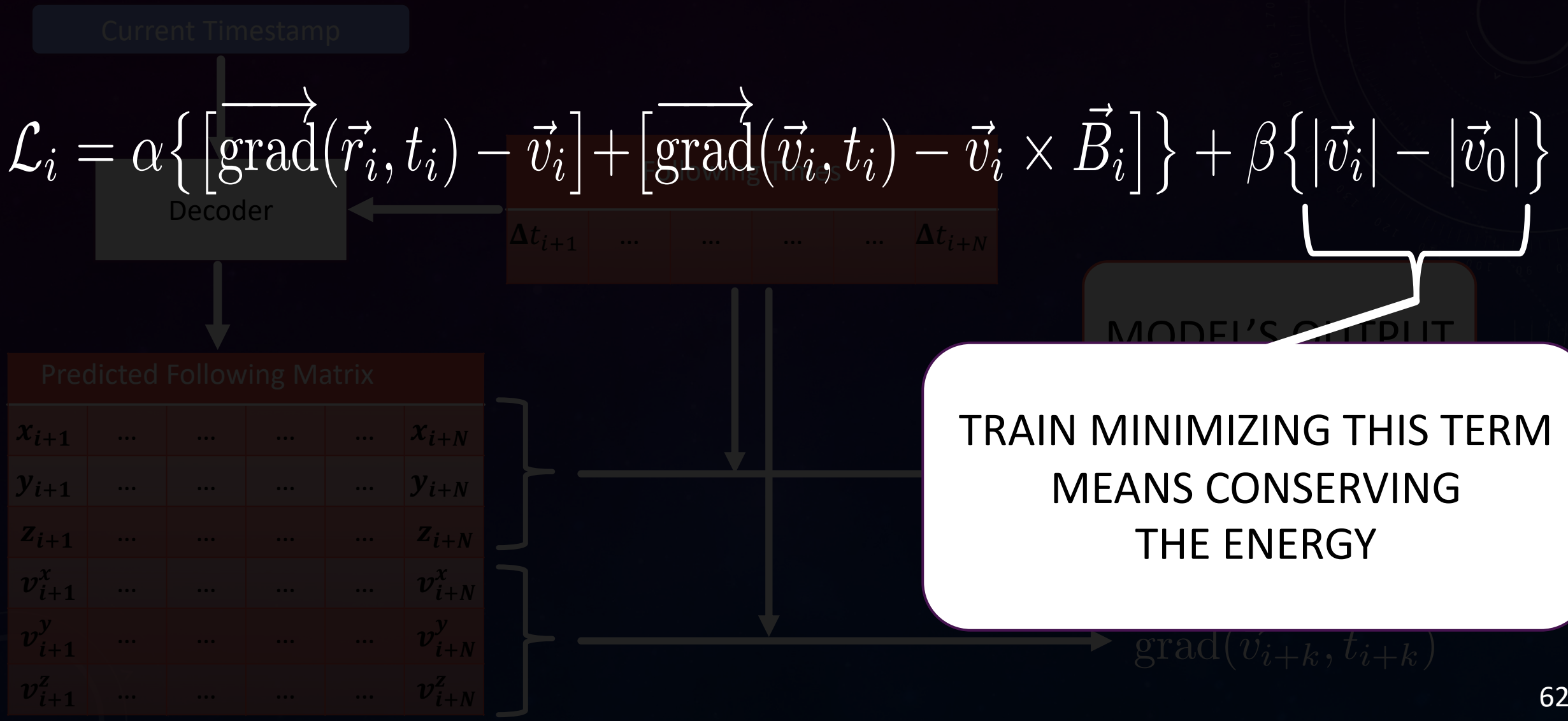
RE-TRAINING: PINN



RE-TRAINING: PINN



RE-TRAINING: PINN



$$\mathcal{L}_i = \alpha\{[\overrightarrow{\text{grad}}(\vec{r}_i, t_i) - \vec{v}_i] + [\overrightarrow{\text{grad}}(\vec{v}_i, t_i) - \vec{v}_i \times \vec{B}_i]\} + \beta\{|\vec{v}_i| - |\vec{v}_0|\}$$

TRAIN MINIMIZING THIS TERM
MEANS CONSERVING
THE ENERGY