Inferring Missing Trajectory Data with Temporal Convolutional Networks

Ilinca Tiriblecea

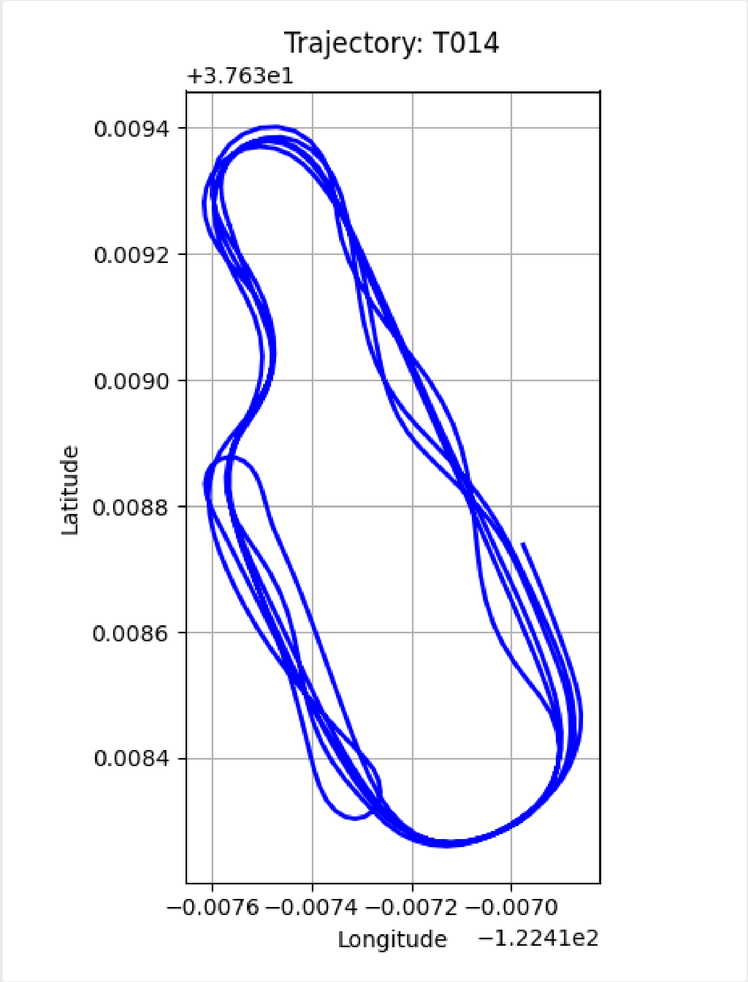
OXTS

Background

Motivations, context and prerequisites

Motivation: Why Trajectory Inference

- Many real-world trajectory tracking applications suffer from drops in data – e.g. navigation solutions are affected by GPS drop-outs
- Our goal is to reconstruct plausible trajectories from incomplete observations by ensuring continuity and over-all dynamic trends



Problem Setup

Starting from a simple setup

- Start from a simple setup, using synthetic data – easier to build dataset
- Input: sequence of 2-dimensional vectors:

$$X = (< x_1, y_1 >, < x_2, y_2 >, \dots, < x_n, y_n >)$$

- Mask (hide) roughly 20% of points, to act as data drop-outs
- Output: smooth and continuous complete trajectory

OXTS

Common approaches for ordered data inference

Why TCNs?

Method	Strength	Limitation
Linear Interpolation	Simple and fast	Ignores dynamics; produces unrealistic motion
RNN / LSTM	Learns sequential dependencies	Sequential computation → slower, vanishing gradients
Transformer	Captures long-range context	Computationally heavy; requires large datasets
Temporal Convolutional Network (TCN)	Parallel, stable, efficient temporal modeling	Fixed receptive field (but sufficient for smooth trajectories)

OXTS

Temporal Convolutional Networks (TCNs)

A brief introduction

- A TCN is a 1-dimensional convolutional network applied along the time axis
- Initially, TCNs were introduced based on the following principles:

1. Input and output have the same dimensions
2. No leakage of the future into the past

- To achieve 1. a classic TCN uses a 1-dimensional fully convolutional network architecture where each hidden layer is the same length as the input layer, and zero padding of length (kernel size − 1) is added to keep subsequent layers the same length as previous ones
- To achieve 2. TCN uses causal convolutions, convolutions where an output at time t is convolved only with elements from time t and earlier in the previous layer
- Simply put,

$$\text{TCN} = 1 - \text{D FCN} + \text{causal convolution}$$

OXTS

Temporal Convolutional Networks

Slight variation

- Main objective is trajectory inpainting, not forecasting – we do not want to enforce causality
- We will use a **variation** on the classic TCN architecture, where the present can be informed by the future as well as the past
- We did not use **causal convolutions**
- Each convolutional layer looks at a window of both **past and future** points
- Stack multiple layers which allows the model to capture long-term patterns

OXTS

Dilated Convolutional Layers

Expanding receptive field

- We want the neural network to be able to use both local and global data from out trajectory
- Dilated Convolutional Layers are a way of ensuring that without overloading the number of parameters
- A dilated convolution acts like a normal convolution but instead of applying the kernel to consecutive time steps, it skip a pre-set number of steps in-between
- Mathematically:

$$y(t) = \sum_{k=-(\mathrm{K}-1)/2}^{(K-1)/2} w_k \cdot x(t - d \cdot k),$$

where

- K = kernel size
- d = dilation factor
- w_k = kernel weight (comprising the filter)
- x = input vector

K = 3

Dilation	Points Used (for time step t)	Receptive field
1	(t−1, t, t+1)	3 time steps
2	(t−2, t, t+2)	5 time steps
4	(t−4, t, t+4)	9 time steps

Residual Connections and Normalization

- **Residual connections** add the input of a layer back to its output, which will prevent information loss and eases gradient flow
- **Layer normalization** ensures consistent feature scaling across time
- Implementing these are common practice and will make TCNs more stable

Implementation

Neural Network Architecture & Training strategy

Input structure

- Each time step is encoded as a **4-dimensional vector**:

- x – x-coordinate (possibly masked)
- y – y-coordinate (possibly masked)
- mask – 1 if missing, 0 if observed
- time – normalized time index from 0 to 1

Model Architecture Overview

Layer	Type	Channels	Dilation	Purpose
Input	—	4	—	Position + mask + time
Block 1	Conv1D + LN + ReLU + Residual	64	1	Local temporal features
Block 2	Conv1D + LN + ReLU + Residual	64	2	Medium context
Block 3	Conv1D + LN + ReLU + Residual	64	4	Longer dependencies
Block 4	Conv1D + LN + ReLU + Residual	64	8	Broader motion patterns
Block 5	Conv1D + LN + ReLU + Residual	64	16	Global temporal structure
Head	Conv1D (1×1 kernel)	2	—	Predicts ($\hat{x}$, $\hat{y}$)

OXTS

Masking Strategy

- During training, we intentionally remove segments of the trajectory (approximately 20%)
- The **mask** channel marks the missing regions
- The model will infer the missing trajectory points from the observed context before and after the gap
- For the same dataset, multiple segments will be randomly masked
- Each masking will act as a different dataset, therefore augmenting the data

Loss Function

We train the model using a **composite loss** to achieve precision, continuity and smoothness of the prediction:

Weighted MSE

$$L_{mse} = MSE_{masked} + \alpha MSE_{unmasked}$$

- Strongly penalizes errors in masked segments
- Very lightly penalizes observed areas – controlled by the parameter α

Continuity Loss

$$L_{cont} = \left|\left| y_{t_1}^{pred} - y_{t_1-1} \right|\right|^2 + \left|\left| y_{t_2}^{pred} - y_{t_2+1} \right|\right|^2$$

Where t_1, t_2 are the times corresponding to the beginning and end of the hidden segment of the trajectory.

Smoothness Loss

$$L_{smooth} = \frac{1}{T} \sum_t \left|\left| y_t^{pred} - y_{t-1}^{pred} \right|\right|^2$$

Implemented Loss Function:

$$L = L_{mse} + \lambda_1 L_{cont} + \lambda_2 L_{smooth}$$

Where λ_1, λ_2 are parameters that have to be set

OXTS

Data (Synthetic)

- For simplicity, the neural network was trained and evaluated on synthetic datasets
- Proposed type of trajectory:

2D Wiggly Trajectory with Noise

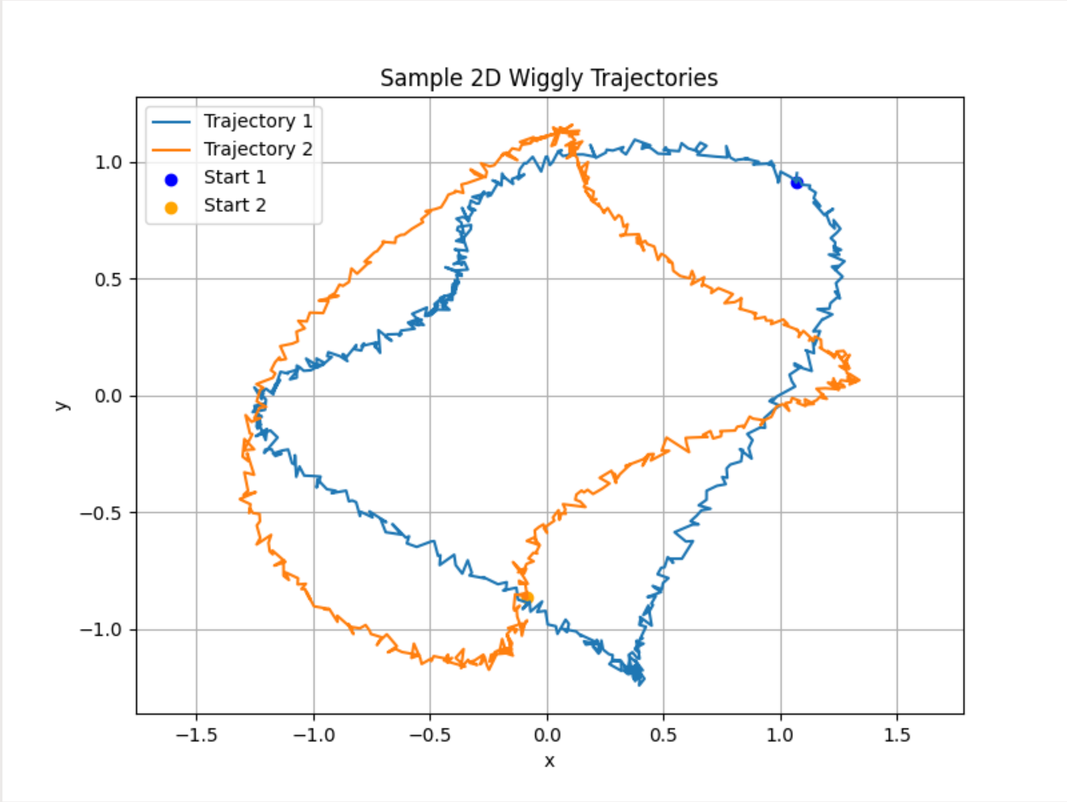
$$x(t) = \sin(2\pi t + \phi_x) + 0.3\ \sin(6\pi t + \phi_y) + \epsilon_x,$$
$$y(t) = \cos(2\pi t + \phi_y) + 0.3\ \sin(4\pi t + 0.5 + \phi_x) + \epsilon_y,$$

where:

$\phi_x, \phi_y \sim Uniform(0, 2\pi)$ are random phase shifts,

$\epsilon_x, \epsilon_y \sim \mathcal{N}(0, \sigma^2)$ are Gaussian noise terms

- Phase shifts and random noise were added for data augmentation purposes



Results & Conclusions

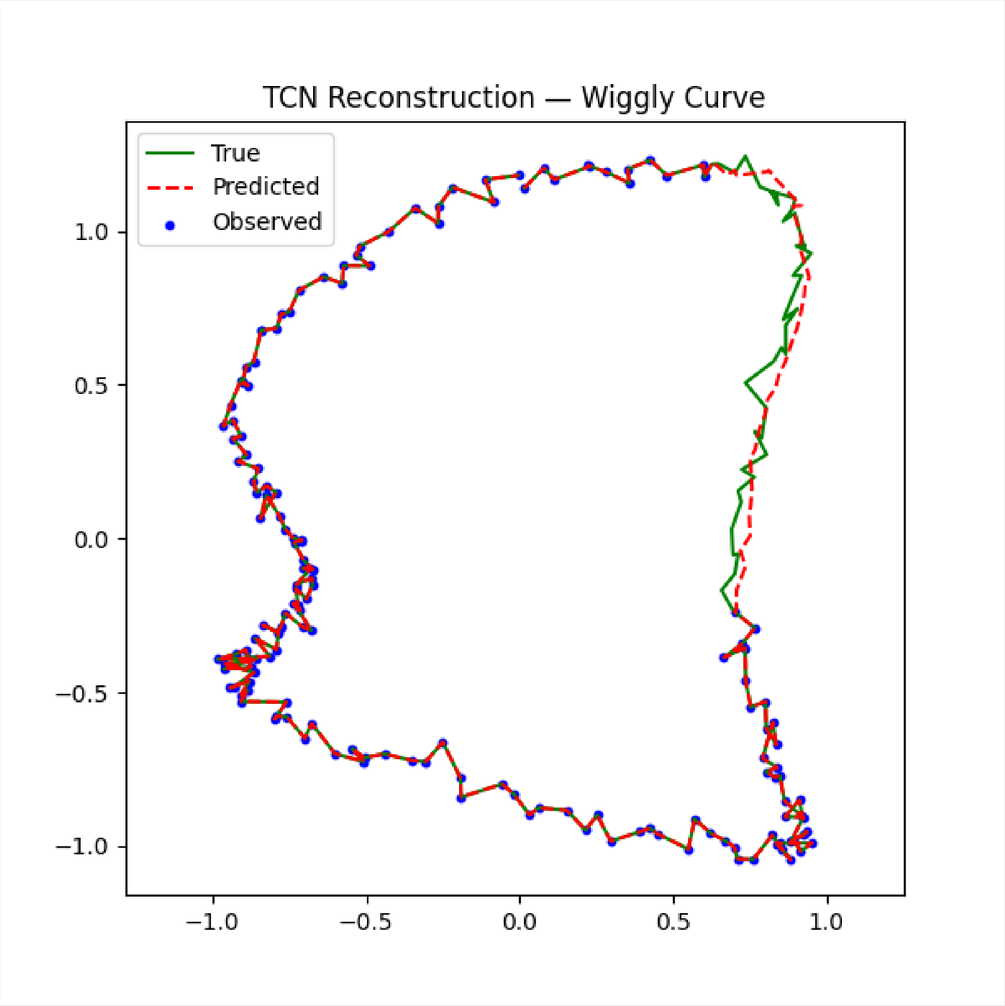
First attempt

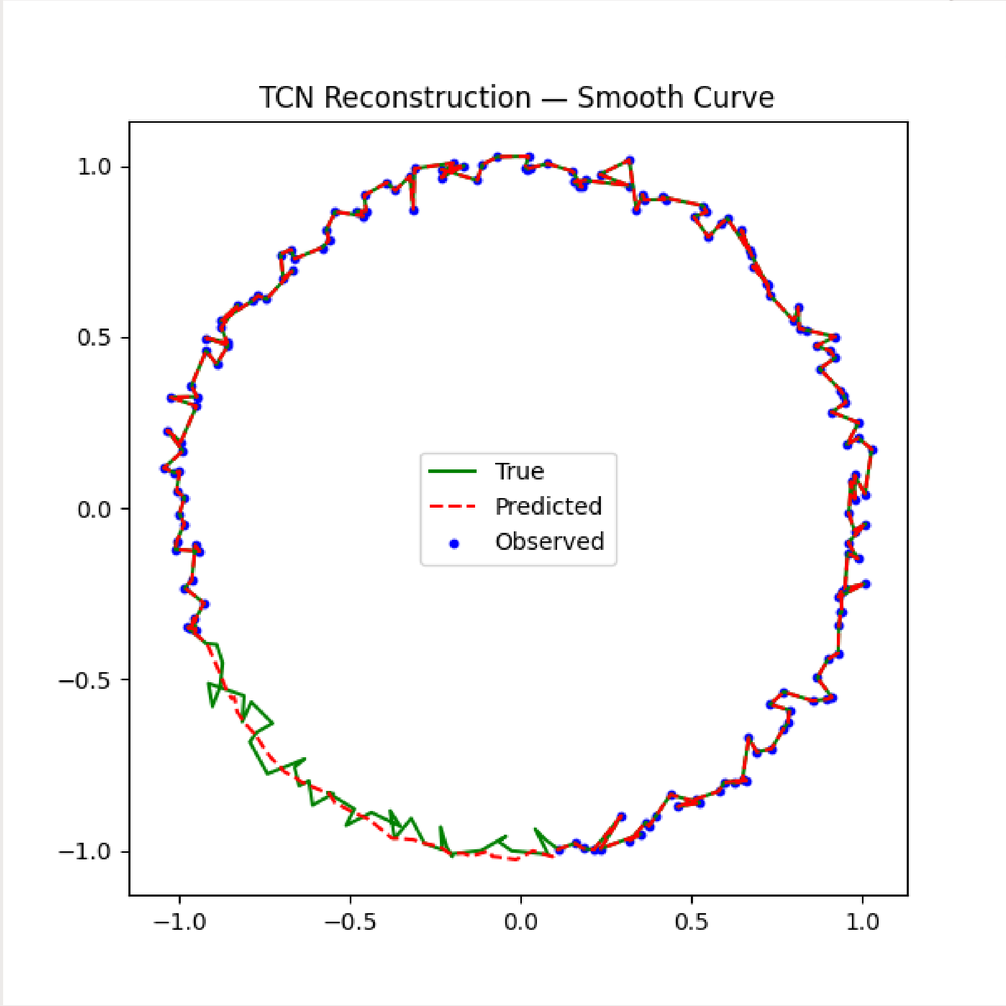
- Sanity check initial attempt was made with a synthetic dataset containing 2 trajectories
- The dataset was constructed using random different masked segments of the same 2 trajectories
- Essentially augmented dataset to have 100 entries for each generated trajectory (200 total)

Training Setup

- Optimizer: Adam
- Batch size: 16
- Epochs: 100
- Dataset: 200 synthetic trajectories
- Mask fraction: ~20% of total trajectory length

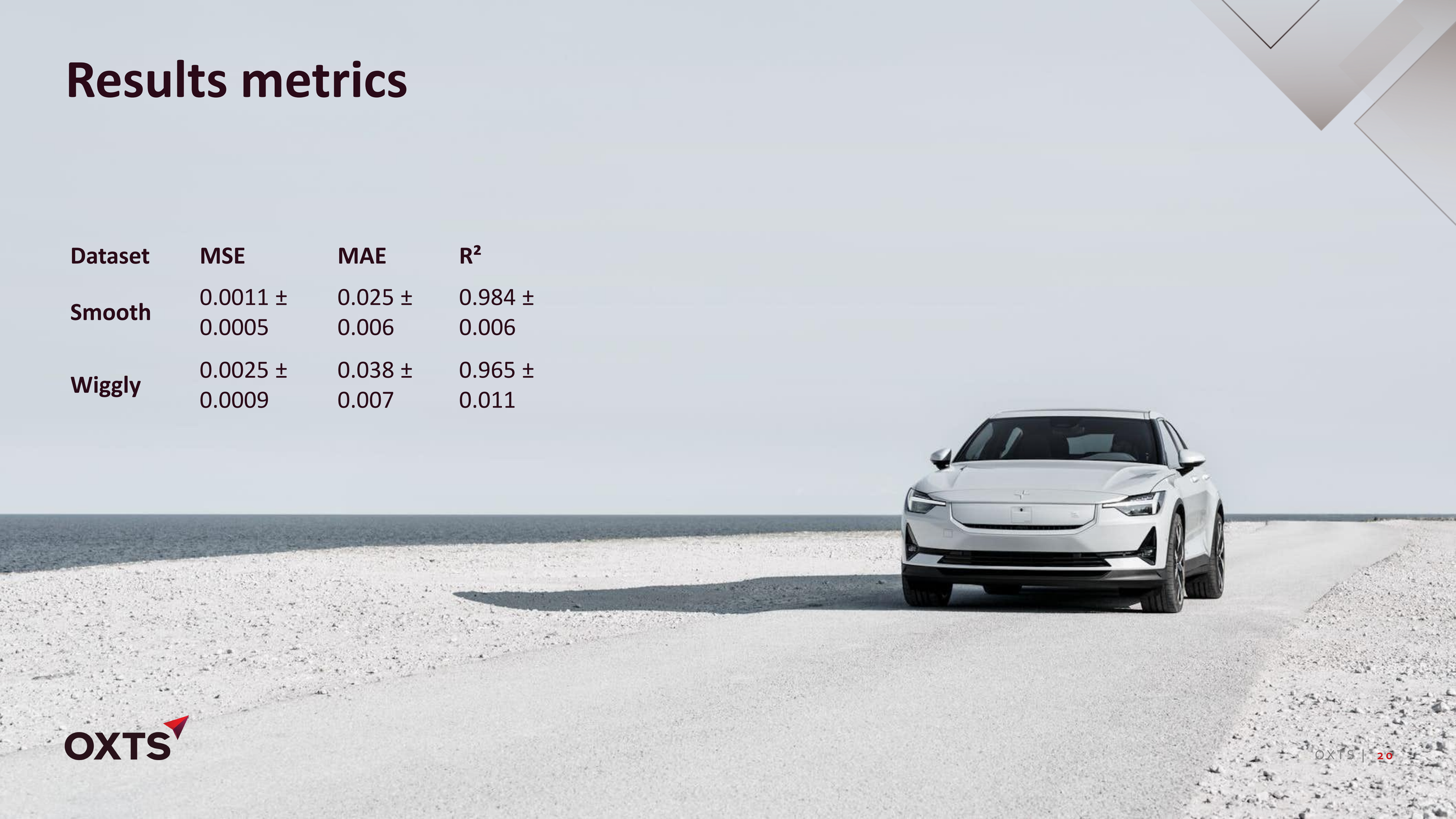
Reconstructed curve vs true





Results metrics

Dataset	MSE	MAE	R^2
Smooth	0.0011 ± 0.0005	0.025 ± 0.006	0.984 ± 0.006
Wiggly	0.0025 ± 0.0009	0.038 ± 0.007	0.965 ± 0.011



OXTS

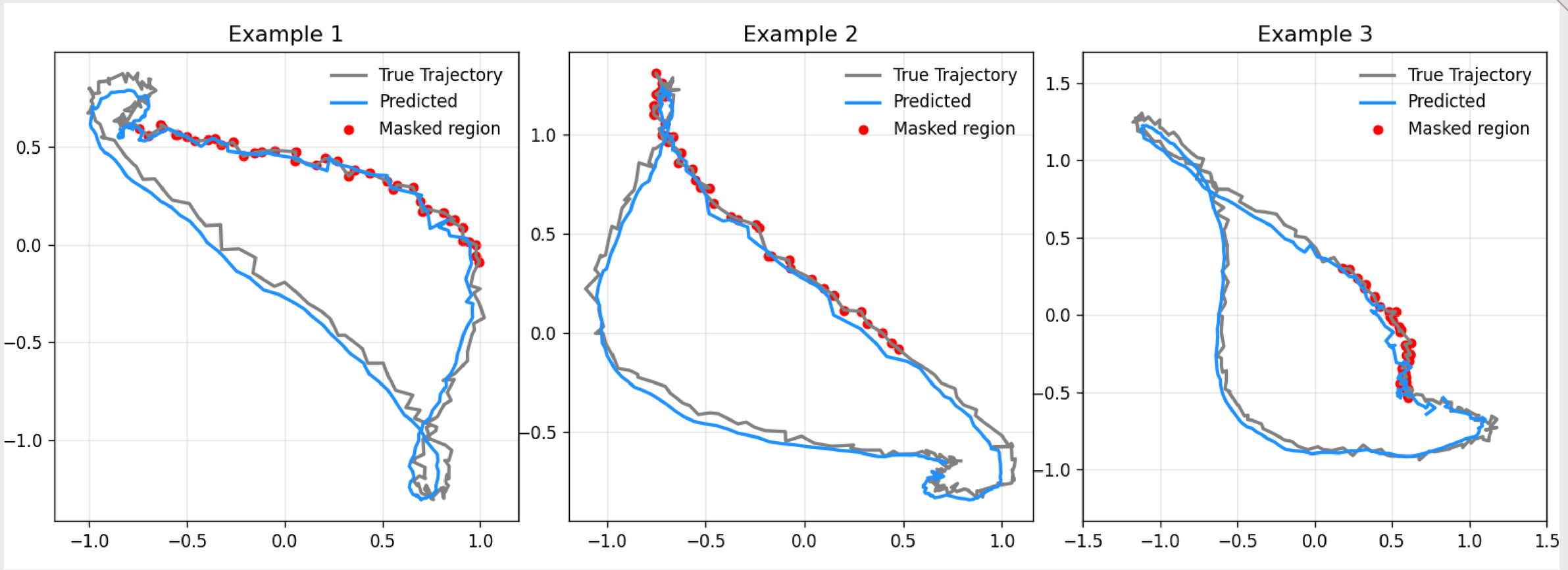
Second attempt

- For the second attempt we constructed a more comprehensive but also more challenging dataset
- Used the formulas previously described, we built up multiple different trajectories
- Masked multiple randomly selected segments for further data augmentation
- Final dataset comprised of 1000 trajectories

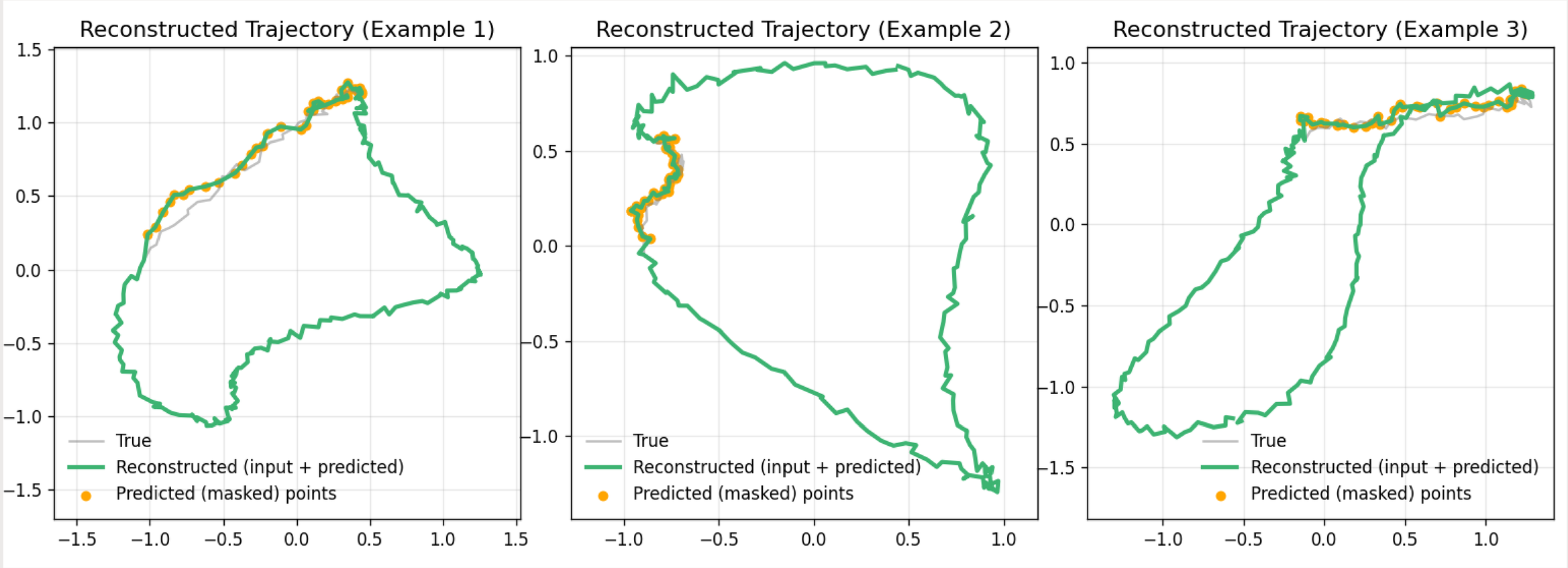
Training Setup

- Optimizer: Adam
- Batch size: 1
- Epochs: 50
- Dataset: 1000 synthetic trajectories
- Mask fraction: ~20% of total trajectory length

Curve comparison true vs predicted



Reconstructed curve



Metrics

Metric	Mean ± Std
MSE	0.003863 ± 0.003155
MAE	0.048896 ± 0.013661
R^2	0.783136 ± 0.726463

OXTS

Conclusion & Further work

Both the visual and quantitative results suggest that this is a viable strategy for trajectory inpainting

More work would be needed to make results more reliable, if this is to be used in a real-life application

Further work could involve getting the work flow working with real trajectory data

Thank you!

References

- Bai, S., Kolter, J. Z., & Koltun, V. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.* arXiv:1803.01271 [cs.LG]. https://arxiv.org/abs/1803.01271
- Lea, C., Vidal, R., Reiter, A., & Hager, G. D. (2016). *Temporal Convolutional Networks: A Unified Approach to Action Segmentation.* arXiv:1608.08242 [cs.CV]. https://arxiv.org/abs/1608.08242
- GitHub - locuslab/TCN: Sequence modeling benchmarks and temporal convolutional networks
- paul-krug/pytorch-tcn: (Realtime) Temporal Convolutions in PyTorch

OXTS