POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Embedded Electronics Engineering

Master Degree Thesis

Neural network integration with domain knowledge for high-precision trajectory prediction

Supervisors
Prof. Mihai Teodor Lazarescu

Candidate
Hooman Khedersolh Sedeh

Academic Year 2024-2025

Abstract

Achieving precise and robust indoor trajectory prediction remains a significant challenge due to sensor noise, complex environments, and the need for real-time processing. This thesis addresses this challenge by systematically developing and evaluating a Neural Networks and Domain Knowledge (NNDK) framework. The core of the framework is a Temporal Convolutional Network (TCN) aimed at high-accuracy human trajectory prediction. Specifically, this work focuses on the time-series forecasting problem ((x, y)-to-(x, y)), where the model predicts a future position based on a history of known ground-truth coordinates.

The methodology emphasizes a rigorous and reproducible pipeline. Data was collected in a controlled 3m×3m indoor room using four capacitive sensors, with a high-precision ultrasound system providing ground-truth data. A modular preprocessing workflow was implemented to handle sensor artifacts and integrate domain knowledge by engineering kinematic features (velocity and acceleration). A key component of this work was a systematic hyperparameter optimization using Bayesian search to define a computationally efficient TCN architecture. This architecture was then evaluated using a robust, custom 6-fold permutation cross-validation to ensure stability.

The evaluation yielded a clear, dual finding. Within its source domain, the NNDK framework proved to be highly effective, achieving an average Root Mean Squared Error (RMSE) of approximately 4.1 cm, validated across the six cross-validation permutations. However, when the model was tested on a new dataset from a different experimental session, its performance degraded significantly to an RMSE of approximately 13.2 cm.

This result critically identifies domain shift as the primary limitation for this application. While the framework is capable of high-accuracy in-domain prediction, it is not yet robust to changes in environmental conditions or subject dynamics. The primary contribution of this thesis is therefore twofold: it provides a validated, efficient model for in-domain trajectory forecasting, and more importantly, it offers a clear, quantitative case study of the domain generalization problem. Future work should focus on overcoming this challenge by exploring techniques such as data augmentation, domain adaptation, and multi-sensor fusion.

Acknowledgements

I would like to express my deepest gratitude to those who have supported and guided me throughout the journey of completing this thesis.

First and foremost, I want to thank my parents for their unwavering love, encouragement, and support. Their belief in me has been a constant source of motivation.

I am especially grateful to my advisor, Professor Mihai Teodor Lazarescu, for his invaluable guidance, insightful feedback, and technical expertise throughout the development of this work. His mentorship has been instrumental in shaping both the methodology and direction of this research.

I would also like to thank the Politecnico di Torino for providing the resources and facilities necessary to conduct this study.

Finally, my heartfelt thanks go to my friends for their patience, encouragement, and steadfast support during every stage of this academic journey.

Contents

Abstract I

Acknowledgements II

List of Figures V

1 Introduction 1
- 1.1 Background on High-Precision Trajectory Prediction . . . 1
- 1.2 Importance of Accurate Indoor Positioning . . . 2
- 1.3 Overview of Capacitive Sensing . . . 2
- 1.4 Overview of Domain Knowledge and its Specifications . . . 3
- 1.5 Brief Introduction on TCNs . . . 4
- 1.6 Thesis Objectives and Research Questions . . . 4

2 Literature Review 6
- 2.1 Existing Trajectory Prediction Techniques and Sensor Modalities . . . 6
- 2.2 Capacitive Sensing in Trajectory Prediction . . . 7
- 2.3 Deep Learning Architectures for Trajectory Prediction . . . 8
- 2.4 Temporal Convolutional Networks (TCNs) . . . 8
  - 2.4.1 Architecture and Principles . . . 8
  - 2.4.2 Applications in Sequence Modeling and Trajectory Prediction . . . 9
- 2.5 Domain Knowledge Integration in Neural Networks . . . 10
  - 2.5.1 Existing Domain Knowledge Accumulation Techniques and their Limitations . . . 10
  - 2.5.2 Methods for Domain Knowledge Integration . . . 11
- 2.6 Addressing the Gap: The NNDK Framework . . . 11

3 Methodology 12
- 3.1 Capacitive Sensor Deployment and Data Collection . . . 12
- 3.2 NNDK Implementation . . . 13
  - 3.2.1 Preprocessing of Raw Data . . . 13
  - 3.2.2 TCN Architecture and Optimization . . . 17
- 3.3 Training and Evaluation Procedures . . . 18
  - 3.3.1 Model Compilation and Training . . . 18

3.3.2 Evaluation Framework . . . 19
3.4 Analysis and Visualization . . . 20
3.4.1 Inverse Normalization . . . 21
3.4.2 Visualization . . . 21
3.5 Generalization to a New Dataset . . . 21
3.6 Performance Metrics and Evaluation Criteria . . . 22

4 Key Findings 24
4.1 Performance on the Source Domain (CapEXP2) . . . 24
4.2 Granular Error Analysis . . . 25
4.3 Generalization Performance and Domain Shift . . . 26
4.3.1 Computational Efficiency . . . 28
4.4 Summary of Key Findings . . . 28

5 Discussion 30
5.1 Interpretation of Results . . . 30
5.2 Benchmarking: Impact of Domain Knowledge and Optimization . . . 31
5.3 Analysis of Failure Modes . . . 31
5.3.1 Difficulty with High-Acceleration Maneuvers . . . 32
5.3.2 Sensitivity to Domain Shift . . . 32

6 Practical Implications 33
6.1 For Academic Research . . . 33
6.2 For Industrial and Applied Fields . . . 33

7 Limitations and Future Directions 35
7.1 Limitations . . . 35
7.2 Future Directions . . . 36

8 Conclusion 38
8.1 Summary of Key Findings . . . 38
8.2 Contributions to the Field . . . 39
.1 Python Codes . . . 43

List of Figures

2.1 Main sensor capacitances and compensation fields 7
2.2 Dilated causal convolutional blocks of a TCN. 10

3.1 Sensor layout and corresponding spatial trajectory distribution for model input and evaluation 12
3.2 Before Removing Outliers from Dataset 14
3.3 After Removing Outliers from Dataset 14
3.4 Windowing of time series dataset `seq_len` = 15 used in the TCN model. . 16
3.5 6-fold Cross Validation 20

4.1 Training and validation loss curve for a representative fold (Fold 6). . . . 25
4.2 Euclidean error for the val of Fold 6. 25
4.3 True vs. Pred with signed error for the val of Fold 6. 26
4.4 Per-sample Euclidean Distance error for the generalization set (CapEXP1). 27
4.5 True vs. Predicted position with signed error for the generalization set (CapEXP1). 28

Chapter 1

Introduction

Nowadays, with rapidly evolving field of intelligent systems and automation, having the ability to forecast entity movement within defined environments is increasingly vital. This thesis aims to present a robust framework tailored for high-precision trajectory prediction, while relying on inherently noisy sensor inputs. Central to the project is an advanced neural network models and domain specific insights, strategically develop to address the challenges posed by sensor inaccuracies and dynamic real world conditions.

1.1 Background on High-Precision Trajectory Prediction

In smart environments and advanced technological applications, the accurate and robust prediction of trajectories from diverse sensor data has emerged as a truly critical capability. This foundational capacity is indispensable for a wide spectrum of uses, ranging from highly automated robotic systems in manufacturing and logistics, where precise path planning and collision avoidance are paramount, to advanced security monitoring systems that require real-time tracking of subjects or assets. Moreover, it plays an important role in responsive smart building management, which results in enabling adaptive energy use or optimized resource allocation based on anticipated occupancy movement. Such high-precision trajectory prediction moves beyond presence detection, aiming to understand and forecast the dynamic spatial progression of objects or individuals, which is crucial for proactive decision-making in complex systems.

Unlike outdoor environments, where GPS offers widespread and relatively accurate localization, indoor settings present a challenging landscape for positional forecasting. In most cases, these challenges stem from frequent signal interference caused by electronic devices, multi-path reflections that distort sensor readings, and consistent noise introduced by environmental variability or hardware constraints. Considering these facts, such factors complicate the pursuit of high-precision indoor localization.

Traditional time series analysis methods, while foundational for modeling sequential data, often struggle under these conditions, particularly when faced with noisy inputs and the need for generalization to unseen scenarios. These issues are intensified in real-time applications, where systems must balance accuracy with computational efficiency.

Also, Effectively predicting a trajectory requires highly accurate timing and location information. This means there is need of sophisticated analytical models that can find useful patterns even when sensor data is flawed and inconsistent.

1.2 Importance of Accurate Indoor Positioning

According to (Alam et al., 2012), one of the most critical and valuable aspects of trajectory prediction is the ability to achieve highly accurate indoor positioning which is so demanding across a range of sectors. For example, in smart homes, it makes dynamic energy management possible; HVAC and lighting can adapt instantly to people's movements. Likewise, in healthcare, it significantly improves patient safety by continuously monitoring individuals, particularly those who struggle with movement or have cognitive impairments, and facilitates rapid emergency intervention. This capability also significantly helps industrial applications. It allows for asset tracking, workflow optimization, and proactive safety monitoring by identifying potential hazards based on location. Because these scenarios require real-time data, we need prediction models that are both efficient and accurate. These examples show that predicting where things will move, instead of just knowing where they are, leads to much better system responsiveness and operational efficiency. This aligns with prior research of (Ramezani Akhmareh et al., 2016) which emphasizing the limitations of GPS indoors and the consequent need for robust indoor localization solutions.

Needless to say, while various technologies have been applied to indoor localization, each presents distinct challenges. Vision-based systems using cameras, while common, raise significant privacy concerns in private spaces like homes. Other prevalent methods using Wi-Fi or Bluetooth signals are often susceptible to signal fluctuations and multipath effects, which can limit their accuracy. (Kuki et al., 2013)

1.3 Overview of Capacitive Sensing

Our research uses capacitive sensing technology for trajectory prediction. This method works by noticing tiny shifts in an electric field when a dielectric or conductive object, like a person, comes near. As indicated in (Ramezani Akhmareh et al., 2016), unlike many other sensors, capacitive sensors are non-invasive; they don't require people to wear tags or devices. This design naturally boosts user privacy and convenience. Plus, these sensors are usually energy-efficient, which makes them great for ongoing, passive monitoring where saving power is important.

However, capacitive sensors normally have several operational complexities. Their response is non-linear respecting to distance, which means a small change in proximity close to the sensor can results in a much larger capacitance change in comparison to the same physical displacement further away. Also, according to the (Bin Tariq et al., 2021), they are inherently sensitive to various environmental factors, including fluctuations in temperature, humidity, and the presence of electromagnetic interference from nearby electronic appliances. These factors can introduce significant noise and drift into the

sensor signals, which causes substantial challenges for accurate data interpretation and trajectory inference.(Bin Tariq et al., 2021)

Regarding the research in (Tariq et al., 2020), effectively using capacitive sensors for high-precision applications demands sophisticated signal processing and smart modeling techniques. Without them, its difficult to distinguish true positional shifts from the inherent noise and contextual variations in the raw sensor data.

1.4 Overview of Domain Knowledge and its Specifications

In the context of machine learning and neural networks, domain knowledge refers to the specialized expertise, understanding, and contextual information pertinent to a specific field or environment from which data originates. According to (Ganin et al., 2016) this knowledge extends beyond the raw numerical values of a dataset, encompassing explicit constraints, underlying physical principles, characteristic behaviors, and specifications directly related to a particular application domain. Integration of this approach into neural network models is increasingly recognized as a powerful strategy to enhance their predictive capabilities, improve interpretability, and ensure that generated predictions are physically plausible and relevant in context.

As indicated in the (Ganin et al., 2016), for trajectory prediction from sensor data, domain knowledge involves detailed insights into:

- **Sensor Characteristics:** Understanding the unique response curves, noise profiles, drift tendencies, and sensitivities of specific sensors (e.g., capacitive sensors' non-linearity with distance, susceptibility to EMI).

- **Environmental Constraints:** Information about the physical environment, such as room geometries (walls, obstacles), fixed sensor installation points, and typical environmental conditions (temperature, humidity, ambient electrical fields). This guides models to operate within feasible spatial bounds.

- **Movement Dynamics:** Knowledge about the kinematics of the moving entity, including typical speeds, accelerations, turning, and common movement patterns which can be the natural flow of human walking. These enables model to enforce movement continuity and reject physically impossible trajectories.

Also regarding (Ganin et al., 2016), incorporation of such domain specific constraints and insights into neural network models, particularly during preprocessing and architectural design, allows robust pattern recognition. It guides the learning process away from spurious correlations present in noisy data and towards a physically grounded understanding, thereby critically improving the accuracy, reliability, and trustworthiness of predictions. This is a central tenet of the NNDK framework.

1.5 Brief Introduction on TCNs

To address the complexities of time-series data which had been mentioned earlier and in order to achieve high-precision trajectory prediction, this thesis employs *Temporal Convolutional Networks (TCNs)*. Needless to say, TCNs represent a significant advancement in neural network architectures which has been designed specifically for sequential modeling tasks. As indicated in (Subbicini et al., 2023), unlike traditional Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks, which process data sequentially and often face issues such as vanishing gradients over long sequences and limitations in parallel computation, TCNs offer a compelling alternative.

According to (Subbicini et al., 2023), core strength of TCNs lies in their use of dilated convolutions and residual connections. Dilated convolutions allow the network receptive field to expand exponentially with depth, enabling it to capture long range dependencies across the entire input sequence (e.g., 15 time steps) without requiring a large number of layers or parameters. Skipping of input elements allows TCNs to efficiently incorporate broad temporal context. Furthermore, regarding research of (Subbicini et al., 2023), inherent convolutional nature of TCNs facilitates parallel processing during training, making them computationally more efficient than RNNs for large datasets. Residual connections, which involve adding the input of a layer directly to its output, are crucial for training very deep TCNs. They help to mitigate the vanishing gradient problem, ensuring information flows effectively through the network which enables robust learning of temporal patterns. This makes TCNs a powerful and stable choice for tasks demanding accurate forecasting from complex sequential data such as sensor streams.

1.6 Thesis Objectives and Research Questions

This thesis addresses the aforementioned challenges by proposing and developing a *Neural Networks and Domain Knowledge (NNDK)* framework specifically implemented for high-precision trajectory prediction. The core hypothesis guiding this research is that by integrating specific domain knowledge into both the architectural design and the training methodologies of advanced neural network architectures specifically *Temporal Convolutional Networks (TCNs)*, a significant enhancement in predictive accuracy and generalization capabilities can be achieved. (Ganin et al., 2016) This remains valid even when operating with noisy sensor data, which typically poses a significant obstacle to high fidelity results.

To validate this hypothesis, this thesis addresses the following questions:

1. **RQ1:** How effectively can an optimized Temporal Convolutional Network (TCN) model complex, non-linear human trajectories from noisy capacitive sensor data?

2. **RQ2:** To what extent does the integration of kinematic domain knowledge improve accuracy and computational efficiency of the model compared with a baseline that lacks this knowledge?

3. **RQ3:** How does the performance of the proposed workflow compare with existing state-of-the-art benchmarks, and what are its generalization limits when tested on data from a different experimental session?

To answer these questions, this work pursues several primary objectives:

- **Designing and implementing a multi-stage preprocessing pipeline that integrates domain knowledge**

  This pipeline transform raw, noisy sensor data into a feature format suitable for deep learning (Ramezani Akhmareh et al., 2016). The core of this objective is moving beyond standard normalization by integrating domain knowledge about the physics of human motion. This is achieved by implementing key kinematic features from the historical trajectory data, which provide the model with crucial context about its dynamics, not just its position.

- **Implement and validate optimized *Temporal Convolutional Network (TCN)* architecture**

  TCN was chosen because of its advantages in modeling long-range temporal patterns, which makes it an strong option for this task according to the results of (Subbicini et al., 2023). This objective not only was to implement a generic TCN, but it also, ensure its final design was justified. To do so, a systematic hyperparameter optimization was performed to determine the optimal number of layers, filter sizes, and regularization, which ensured the final architecture was validated by its performance.

- **Evaluating performance of model on both known datasets (using cross validation) and on entirely unseen datasets.**

  This evaluation addresses the frameworks precision in both quantitative and qualitative aspects. Also, it demonstrate accuracy and generalization capabilities under varied conditions.

By demonstrating efficacy and robustness of the NNDK framework, this thesis ultimately aims to contribute a high accurate solution for complex trajectory forecasting. This work highlights the potential of combination advanced neural network models with explicit domain informed design principles, which demonstrate how this integration can effectively overcome real world challenges in positional prediction. The insights can eventually contribute to future developments in intelligent sensing and autonomous systems requiring reliable spatial awareness.

Chapter 2

Literature Review

Considering the research indicated in (Kong et al., 2025), among majority of dynamic environments, one of the most fundamental elements for a wide variety of applications is accurate trajectory prediction. These applications include diverse fields such as robotic navigation, augmented reality, health monitoring, and intelligent system management. This demand for precise positional forecasting has led to significant research in both sensor technologies and advanced modeling techniques.

2.1 Existing Trajectory Prediction Techniques and Sensor Modalities

Traditional approaches of trajectory prediction have explored variety of sensor technologies, each of them present unique advantages and disadvantage. To be more specific, as a result of (Panella et al., 2023), Radio Frequency Identification (RFID), which relys on transponders and readers, often faces practical constraints in terms of scalability and requires extra time for optimal performance, which limits their utility in dynamic environments. Also, Bluetooth Low Energy (BLE) and Wi-Fi, frequently interfere with challenges such as signal interference, pervasive multipath reflections within cluttered environments and inherent inconsistencies in accuracy, particularly over long distances. All these challenges results in an less accurate model using these components. While this thesis focuses on a single-modality capacitive sensing system, a common strategy for improving localization accuracy has been done in (Li et al., 2017) paper which implemented using sensor fusion. Many systems fuse data from Wi-Fi RSSI measurements with smartphone inertial sensors (PDR) using techniques like the Kalman Filter to achieve robust performance. For instance, (Li et al., 2017) report a localization error of less than 1.17m by fusing Wi-Fi and PDR data.

Moreover, according to (Zhu et al., 2020), Infrared sensors, while beneficial in tagless monitoring, are highly susceptible to environmental variations like light change or temperature gradients, which make their positional inference unreliable for robust applications.

2.2 Capacitive Sensing in Trajectory Prediction

Among various sensing modalities, capacitive sensors are compelling option for trajectory prediction because of their basic energy efficiency, privacy preserving capability and distinct behavior for passive monitoring without requiring subjects to wear active devices. This makes them highly suitable for unobtrusive applications in different settings.

However, capacitive sensors are characterized by several operational complexities which require advanced processing. According to (Bin Tariq et al., 2021), their response is non linear with respect to distance, which means a small change in proximity near the sensor can induce a much larger capacitance variation than the same displacement farther away. Also, they are sensitive to environmental factors such as temperature, humidity, and electromagnetic interference from nearby appliances. Consequently, high-precision tasks like detailed trajectory prediction demand, sophisticated signal processing and modeling techniques to extract meaningful positional changes from noisy data.

Considering the results of (Wimmer et al., 2007), Capacitive sensing operates in *active* or *passive* mode which works based on a transmitted signal and which rely on ambient fields respectively. In active sensing, a known signal is driven on a transmit electrode, couples with the body and is picked up by a receive electrode, whose signal strength reveals presence and motion. Passive sensing detects external electric fields without generating its own signal.

In addition, recent advances in load modes enable simple, sensitive and privacy-friendly indoor localization. In the load mode configuration, only one sensor plate is required while the human body acts as the second electrode, reducing hardware complexity and cost. Sensitivity scales with larger plates shows stronger signals. Figure 2.1 illustrates this layout. According to (Ramezani Akhmareh et al., 2016), researchers at Politecnico di Torino enhanced load-mode sensitivity by combining a tailored transducer with advanced data acquisition, achieving low power consumption, cost efficiency, tag-less operation and better privacy.

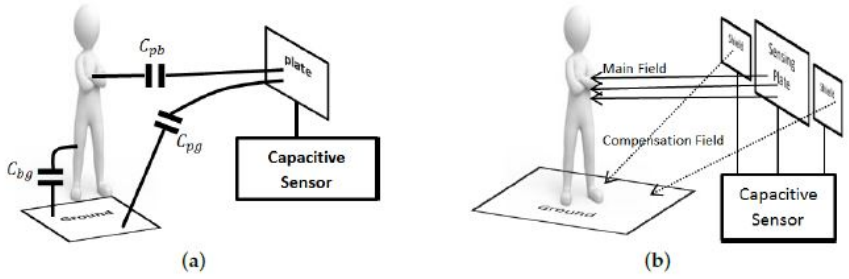


Figure 2.1: (a) Main sensor capacitances in load mode: plate-body (C_{pb}), plate-ground (C_{pg}), and body-ground (C_{bg}); (b) Use of compensation fields for short-range load-mode capacitive sensors to reduce C_{pg}.

2.3 Deep Learning Architectures for Trajectory Prediction

According to the research of (Liu et al., 2016) the field of trajectory prediction has seen widespread adoption of deep learning, with common approaches including Recurrent Neural Networks (RNNs) and their variants like LSTM, Generative Adversarial Networks (GANs), and Graph Convolutional Networks (GCNs). These methods have shown great promise in learning complex patterns from sequential data. More recent approaches have explored hybrid architectures, such as the TrajTransGCN model proposed by (Wang et al., 2023), which fuses Graph Convolutional Networks with Transformers to capture both spatial and temporal dependencies.

2.4 Temporal Convolutional Networks (TCNs)

2.4.1 Architecture and Principles

The evolution of data processing techniques for sequential time series sensor data developed over the past decade, with neural networks (NNs) playing a in depth role in trajectory prediction and positional forecasting. The core of the proposed prediction model is a Temporal Convolutional Network (TCN). As introduced by (Lea et al., 2017), TCNs serve as a powerful alternative to Recurrent Neural Networks (RNNs) for sequence modeling tasks. By using a hierarchy of causal convolutions, TCNs can capture long-range temporal patterns, which allows them for parallel computation and mitigate common issues like vanishing gradients that often affect RNNs. Other models such as LSTM and GRU also can learn temporal relationships through recurrent connections, however according to (Kong et al., 2025), they suffer several drawbacks like, high computational cost due to sequential processing, limited parallelization, vanishing or exploding gradients on long sequences and complex internal states that are difficult to interpret. This challenges decrease performance in situation which real time processing are required or resource constrained edge devices needed to get deployed.

Considering the state-of-art in (Bin Tariq et al., 2022), the advent of convolutional architectures for time series data, such as one dimensional Convolutional Neural Networks (1D CNNs), represented a step toward more efficient local pattern extraction and improved parallelism compared with RNNs. Building on this foundation, *Temporal Convolutional Networks* (TCNs) provide a significant methodological step forward. Highlighting this point, (Chen et al., 2020) has mentioned TCNs are designed to capture very long range temporal dependencies through dilated convolutions.

Fundamental aspects of TCNs design include:

- **1D Convolutional Layers:** TCNs apply convolutions across temporal sequences, meaning the network processes data sequentially while maintaining the order of the input. Unlike standard Recurrent Neural Networks (RNNs), which rely on recurrent connections, TCNs use convolutional filters to capture temporal dependencies.

- **Causal Convolutions:** In TCNs, causal convolutions ensure that the network processes data in a strictly time-ordered manner. This means that the output at a given time step is generated using only the current and previous time steps, preventing the "future" data from influencing predictions at earlier time steps.
- **Residual Connections:** TCNs incorporate residual or skip connections between layers. These connections help prevent the vanishing gradient problem and ensure that deeper layers continue to learn efficiently by allowing the network to bypass layers, improving the flow of gradients during backpropagation.
- **Dilated Convolutions:** TCNs use dilated convolutions, which allow the network to handle long-range dependencies without increasing the computational complexity. Dilations expand the receptive field of the convolutional layers, enabling the network to learn from both local and distant time steps in a sequence.

Also, as discussed in (Ramezani et al., 2020), dilations expand the receptive field exponentially with network depth, allowing model access broad temporal context without a proportional increase in layers or parameters. Because TCNs are convolutional, they can be trained in parallel over all time steps,which makes them more efficient than RNNs for large datasets. Also, according to (Chen et al., 2020), the residual connections adding each layers input directly to its output are essential for training deep TCNs: they alleviate vanishing gradient issues and promote stable learning of temporal patterns. Consequently, TCNs offer a powerful and robust solution for forecasting from complex sequential data such as sensor streams.

2.4.2 Applications in Sequence Modeling and Trajectory Prediction

Due to architectural advantages, TCNs have found increasing utility in diverse sequence modeling tasks as indicated in (Kong et al., 2025). Beyond early success in audio synthesis and natural language processing, they achieve performance competitive with or superior to recurrent networks for many time series applications while offering better parallelism and gradient stability respecting to (Ramezani et al., 2020). By stacking layers of exponentially increasing dilated convolutions (i.e., gaps between adjacent filter taps; see Figure 2.2), TCNs cover broader input windows with fewer parameters. Prior studies on capacitive and infrared based tracking report in (Subbicini et al., 2023) shows strong potential in trajectory prediction.

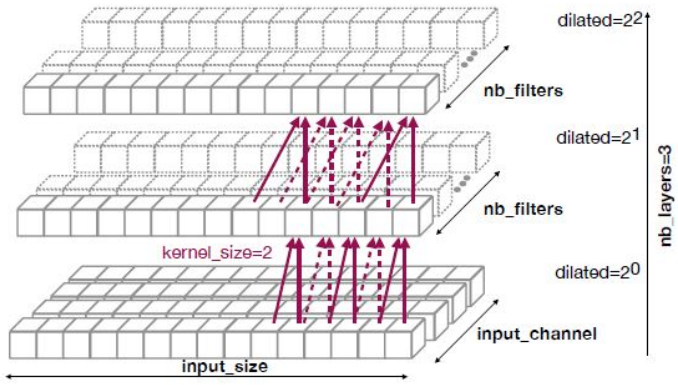


Figure 2.2: Dilated causal convolutional blocks of a TCN.

2.5 Domain Knowledge Integration in Neural Networks

Despite architectural innovation, the machine learning community is increasingly emphasizes embedding domain knowledge into data pipelines and model design which specified in (Ganin et al., 2016). Such knowledge may capture non-linear sensor specific characteristics responses, typical noise profiles or encode environmental constraints like room geometry, fixed obstacles, and common movement patterns.

2.5.1 Existing Domain Knowledge Accumulation Techniques and their Limitations

Historically, domain knowledge had been integrated through various means, ranging from manual feature engineering and rule based systems to selection of model architectures. As indicated in detail in (Zhang et al., 2022), in classical machine learning, domain experts would manually craft features such as velocity, acceleration or specific signal ratios which encode their understanding of the underlying physical processes. While these metohds were effective for most systems, this approaches may not scale well to high dimensional or highly non linear data. Rule based systems, which are another form of explicit domain knowledge, normally involve defining logical condition or physical constraint which the model output must adhere to.

Nowadays though, according to (Li et al., 2024), the emphasis has shifted on integration through data augmentation such as synthesizing new data based on domain rules or by designing network architectures which capture certain types of domain knowledge. These methods however, demonstrate a gradual application rather than an coherent approach. A key limitation of existing techniques is the lack of a unified frameworks which integrates diverse forms of domain knowledge through the entire model development.

2.5.2 Methods for Domain Knowledge Integration

The integration of domain knowledge is a key challenge in making machine learning models effective, especially when data is scarce. As surveyed by (Tsang et al., 2020), integration can take several forms, including transfromation of input data, modifying the loss function, or altering the models architecture. The approach taken in this thesis is implementing kinematic features from the raw positional data which according to (Garcez et al., 2019), falls into the category of transforming input data, where problem related to information is used to create a richer feature representation for the learning algorithm.

2.6 Addressing the Gap: The NNDK Framework

In this thesis, I am directly addressed the aforementioned gap by introducing and demonstrating a "Neural Networks and Domain Knowledge" (NNDK) framework. This NNDK framework is designed to leverage domain informed preprocessing strategies alongside a designed TCN architecture, thereby results in achieving unprecedented levels of high-precision trajectory forecasting in real world settings.

It also aim to provide a structured approach where domain insight actively guide data cleaning, feature construction, and architectural choices, which lead to more robust, interpretable, and generalizable predictive models.

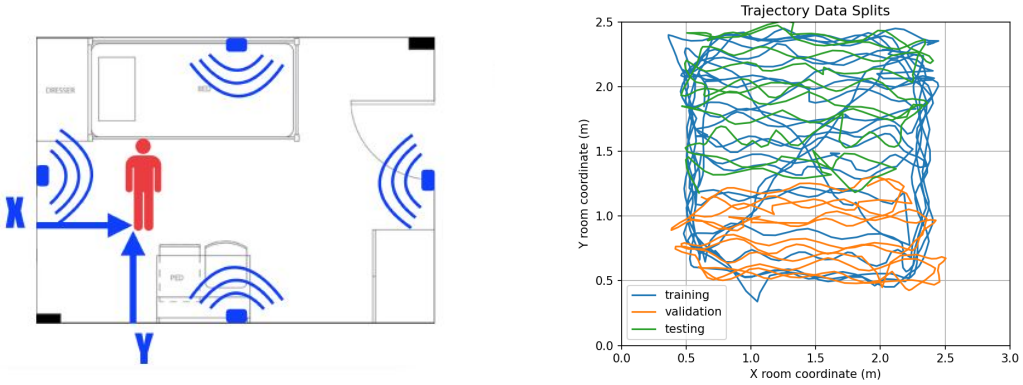
Chapter 3

Methodology

Objective of this design is the development and evaluation of a Neural Networks and Domain Knowledge (NNDK) framework, implemented for high-precision trajectory prediction. This pipeline integrates several stages, which has been mentioned in detail in the following sections.

3.1 Capacitive Sensor Deployment and Data Collection

The foundational data for this investigation was collected by (Ramezani Akhmareh et al., 2016) within a controlled indoor environment, specifically configured as an empty 3 m x 3 m room. This controlled setting enabled precise ground-truth measurement and isolation of sensor characteristics. The data set is divided into 60% for training, 20% for validation, and 20% for testing, each in time order.



(a) Combination of capacitive sensors able to estimate coordination of a person

(b) Trajectory coverage of training 60%, validation 20%, and testing 20% datasets within a 3x3 meter room

Figure 3.1: Sensor layout and corresponding spatial trajectory distribution for model input and evaluation

According to the conditions which mentioned in (Ramezani Akhmareh et al., 2016) experiment, primary sensing mechanism utilized for capturing positional data consisted of 4 custom designed capacitive sensors. These sensors were positioned within the environment to detect subtle alterations in electric fields induced by human proximity. These sensors provided raw readings, making three samples per second. Concurrently, highly accurate ground truth X,Y positional data was acquired through a reference system. All raw sensor reading from the capacitive sensors and their corresponding ground truth archived in a .csv file format, which provide a synchronized record of sensor observations and true locations over time.

3.2 NNDK Implementation

The implementation follows a modular pipeline, encompassing distinct stages for data preparation, model architecture, and systematic evaluation. A structured approach to preprocessing is foundational to the framework, ensuring that raw, noisy data is methodically transformed into a clean, feature-rich format suitable for the neural network model.

3.2.1 Preprocessing of Raw Data

A foundational component of the NNDK framework is its modular preprocessing pipeline, implemented as a '*Thin Layer*'. This architectural choice facilitate the systematic management of raw data, makes its transformation from initial acquisition into formats consumable by the neural network model. This preprocessing module is implemented to integrate domain knowledge at various operational stages, in order to enhancing the quality of data and it ultimately improve the predictive performance of the model. The detailed processing sequence is mentioned bellow:

- **Outlier and Spike Handling:** Raw positional data from sensor systems can be susceptible to noise, resulting in sudden, physically implausible jumps, or "spikes." To ensure data quality, a consistent and reproducible spike handling algorithm was applied as the first step to all datasets used in this research (i.e., the primary CapEXP2 data and the CapEXP1 generalization data).

  The core of this method is a threshold-based detection and correction procedure. A data point is identified as a spike if the positional change from the previous time step exceeds a predefined threshold.

  The threshold was set to 0.5 meters based on an analysis of plausible human movement within the physical constraints of the experiment.

  - The systems sampling rate is 3 Hz, meaning the time between consecutive measurements is approximately 0.33 seconds.
  - A positional jump of 0.5 meters in this timeframe would imply an instantaneous velocity of 1.5 m/s (5.4 km/h).

This velocity is equivalent to fast walk or a jog. It is considered physically implausible for a subject to accelerate to and from this speed between two consecutive samples (a 0.33s interval) while performing nuanced movements within a confined 3x3 meter room.

This algorithm as follows:

- For each coordinate (X and Y), absolute difference between each point and the one preceding it is calculated.
- If the difference exceed 0.5 m threshold, the point is flagged as a spike.
- Each flagged spike is then corrected by replacing its value with the value from the immediately preceding, valid time step.

This algorithmic approach is deterministic and reproducible. It serves to clean the data by removing outliers while preserving the temporal integrity of sequence, as illustrated by comparing the trajectory before (Figure 3.2) and after (Figure 3.3) the cleaning process.

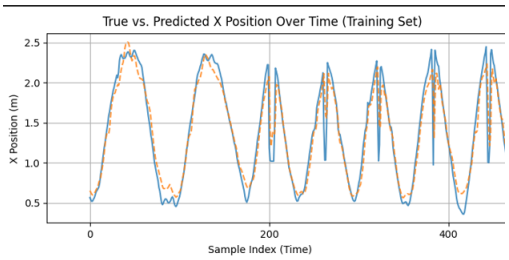


Figure 3.2: Before Removing Outliers from Dataset

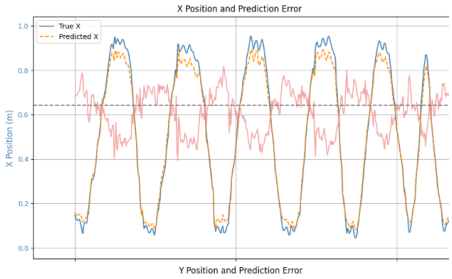


Figure 3.3: After Removing Outliers from Dataset

- **Feature Engineering and Domain Knowledge Integration:**

  To provide model with understanding the domain knowledge, the feature set boosted with kinematic information extracted from positional data. The input features include:

  - Four raw capacitive sensor readings (cap1, cap2, cap3, cap4).
  - X and Y coordinates.
  - Velocity (vx, vy): First derivative of position data, calculated as the difference between consecutive X and Y values.
  - Acceleration (ax, ay): Second derivative of the position data, calculated as the difference between consecutive velocity values.

This process make a 10-dimensional feature vector for each time step: `['cap1', 'cap2', 'cap3', 'cap4', 'X', 'Y', 'vx', 'vy', 'ax', 'ay']`. By providing velocity and acceleration, the model is given not just positional context, but also information about how that position is changing over time.

- **Normalization and feature scaling:**

  Training a neural network with input features with different scales and physical units lead to unstable training and poor convergence. To decrease this, a multi-tiered scaling strategy is employed. Also, To prevent data leakage, all scaling parameters fit on the training data and then applied across the validation and test sets.

  The features are grouped by their physical meaning and scaled as bellow:

  - **Capacitive Sensors (cap1–cap4):** These features are already scaled to a [0, 1] range.
  - **Kinematic Groups:** To preserve the physical relationships within kinematic pairs, a shared Min-Max scaling is applied to each group. Minimum and maximum values are calculated for both components of the pair in the training set.
    - **Position (X, Y):** Scaled together using a single min and max value.
    - **Velocity (vx, vy):** Scaled together using a single min and max value.
    - **Acceleration (ax, ay):** Scaled together using a single min and max value.

  The Min-Max normalization formula:

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

- **Windowing:**

  The core task of the NNDK model in this thesis is time-series forecasting. The goal is predicting next (x, y) coordinate in a sequence, given recent history of known past coordinates and their corresponding sensor readings. This framing defines the problem as predicting a future position from past positions, or (x, y) to (x, y).

  To create supervised learning examples, the processed time-series data is segmented into fixed-length, overlapping windows:

  - **Input Window:** Each input sample is a window containing the data from 15 consecutive time steps (SEQ_LEN = 15). The features in this window include the four capacitive sensor readings and the historical ground-truth trajectory data (X, Y, v_x, v_y, a_x, a_y).

– **Target:** The corresponding target for each input window is single (x, y) coordinate pair from the immediate subsequent time step (i.e., the 16^{th} position in the sequence).

In here, historical ground-truth trajectory provides the primary information for the forecast, while the sensor data offer supplementary context.

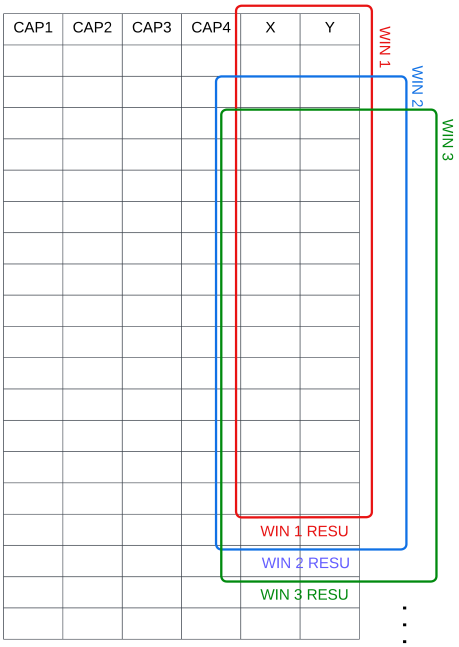


Figure 3.4: Windowing of time series dataset `seq_len` = 15 used in the TCN model.

- **Data Splitting for K-Fold Cross-Validation:**

  To ensure unbiased evaluation which respects temporal nature of the data, a fixed split is used. The dataset is partitioned into three distinct sets:

  – **Training Set (60%):** First 60% of the data used to train model and fit feature scalers.

  – **Validation Set (20%):** Next 20% of data used to monitor the models performance during training for tasks.

  – **Test Set (20%):** Final 20% of data is held out as a completely unseen set which used only once providing assessment of the trained models generalization capability.

3.2.2 TCN Architecture and Optimization

Computational core of this project is a Temporal Convolutional Network (TCN). This architecture was selected over traditional recurrent models like LSTMs because its advantages sequence modeling tasks which had been mentioned before. By employing hierarchy of dilated and causal convolutions, TCNs can capture long-range temporal dependencies.

TCN model consists of residual blocks, where each block contain dilated convolutional layers, followed by Rectified Linear Unit (ReLU) activation, spatial dropout for regularization and batch normalization. Residual connections are used across blocks to ensure stable gradient flow and effective training of a deep network.

Although this structure provide a strong foundation, the specific architectural hyperparameters were not arbitrarily chosen. Instead, systematic hyperparameter optimization process implied determining the most effective configuration for this specific dataset and task. This process designed to move beyond default or randomly selected values and also to justify the final model design based on performance.

Hyperparameter Tuning Methodology

`KerasTuner` library was employed to conduct hyperparameter search. To provide a stable basis for evaluation during tuning, preprocessed data had been split, with the first 80% used for training models in each trial and subsequent 20% used as a fixed validation set.

Two search strategies were implemented and compared:

- **Random Search:** This method randomly samples hyperparameter combinations from predefined search space which is effective for exploring wide range of possibilities without bias.
- **Bayesian Optimization:** A more sophisticated strategy which builds model of the objective function (`val_loss`) which use the results from previous trials to make more informed decisions about which new hyperparameter combinations to try next, focusing on more promising regions of the search space.

For both strategies, search was run for 50 trials, with `EarlyStopping` callback (`patience=5`) to stop unpromising trials and improve efficiency. This Search defined as below:

- Number of TCN filters: {16, 32, 64}
- Kernel size: {3, 5, 7}
- Number of TCN stacks (hidden layers): 3–8
- Dense layer units: {64, 128, 192, 256}
- Dropout rate: {0.1, 0.2, 0.3, 0.4}
- Learning rate: {1e-2, 1e-3, 1e-4}

Tuning Results and Final Architecture

Objective of the search was to identify hyperparameter configuration which had lowest validation loss (`val_loss`). key risk with Bayesian Optimization is the potential to converge prematurely to a local minimum. To assess this risk, a Random Search was performed providing a broad, unbiased exploration of the hyperparameter space. This comparison also highlights the trade-off between computational cost and search intelligence.

While the Random Search was faster, completing its 50 trials in approximately 30 minutes, Bayesian search required roughly 66 minutes for the same number of trials. This additional time is due to the computational overhead which is required by the Bayesian method to update it internal model after each trial.

Despite the longer execution time, Bayesian method found a final validation loss of 0.0013, which outperform best result from the Random Search (0.0059). This outcome provide strong confidence that the Bayesian strategy successfully navigated search space to find a more optimal configuration, rather than getting trapped in a suboptimal region, justifying the additional computational cost.

Results of the values of optimized hyperparameter for each method has been mentioned in the tables below. Considering higher parameters in random search optimized hyperparameter and also given better performance, the set of hyperparameters discovered by Bayesian Optimization was selected for the final model. The optimal hyperparameters are detailed in Table 3.1.

Num of Filters	**Kernel Size**	**Hidden Layers**	**Learning Rate**	**Dense Units**	**Dropout**
16	3	3	0.01	128	0.1

Table 3.1: Optimal Hyperparameters Determined by Bayesian Optimization

Num of Filters	**Kernel Size**	**Hidden Layers**	**Learning Rate**	**Dense Units**	**Dropout**
16	7	8	0.01	128	0.1

Table 3.2: Optimal Hyperparameters Determined by Random Optimization

3.3 Training and Evaluation Procedures

Training and evaluation of model are designed as a systematic process to ensure robustness, also prevent overfitting, and provide assessment of the models performance on unseen data.

3.3.1 Model Compilation and Training

For each training run, the TCN model is compiled and trained using following configuration:

- **Optimizer:** Adam optimizer is used, with learning rate of 0.01 determined by hyperparameter tuning process.
- **Callbacks:** Also, for training process optimization and preventing overfitting, two callbacks employed:
  - **EarlyStopping:** This callback monitor the validation loss (`val_loss`) and stops training if no improvement seen for 15 consecutive epochs Which has been configured to restore model weights from epoch with best validation loss, ensure the final model is the most generalized version.
  - **ReduceLROnPlateau:** This callback also monitor validation loss and reduces learning rate by a factor of 0.5 if no improvement is observed for 3 consecutive epochs which allows model to make finer adjustments in the later stages of training, often leading to better convergence.

The model is trained for a maximum of 50 epochs with a batch size of 8.

3.3.2 Evaluation Framework

A custom 6-Fold Permutation Cross-Validation framework designed and had been implemented to assess the model performance and stability. This approach was chosen over standard cross-validation techniques to test the model under various temporal conditions respecting the integrity of the data sequence within each training and validation block.

Permutation-based Cross-Validation Procedure

The methodology involve partitioning the primary dataset (CapEXP2) into three conceptual segments (an early, middle, and late part) and running six distinct experiments, or "folds." In each fold, role of these segments for training (60% of data), validation (20%), and get ignored (20%) are permuted. This process, illustrated in Figure 3.5, is defined as follows:

- **Fold 1:** Train on the initial 60%, validate on the next 20%.
- **Fold 2:** Train on the initial 60%, validate on the final 20%.
- **Fold 3:** Train on the middle 60%, validate on the initial 20%.
- **Fold 4:** Train on the final 60%, validate on the initial 20%.
- **Fold 5:** Train on the middle 60%, validate on the final 20%.
- **Fold 6:** Train on the final 60%, validate on the middle 20%.

This structured permutation tests models ability to not only forecast future events from past data such as Fold 1 but also to "backcast" past events from future data such as Fold 4 which providing a comprehensive assessment of pattern recognition capabilities.

For each of these 6 folds, model is trained, and its performance is recorded on the corresponding validation set. Primary outcome of this process is mean and standard deviation of the performance metrics (MSE, MAE, RMSE) across all 6 validation results, which indicates the models average performance and stability. The final figure of 6-fold Permutation Cross-Validation had been illustrated in figure below:

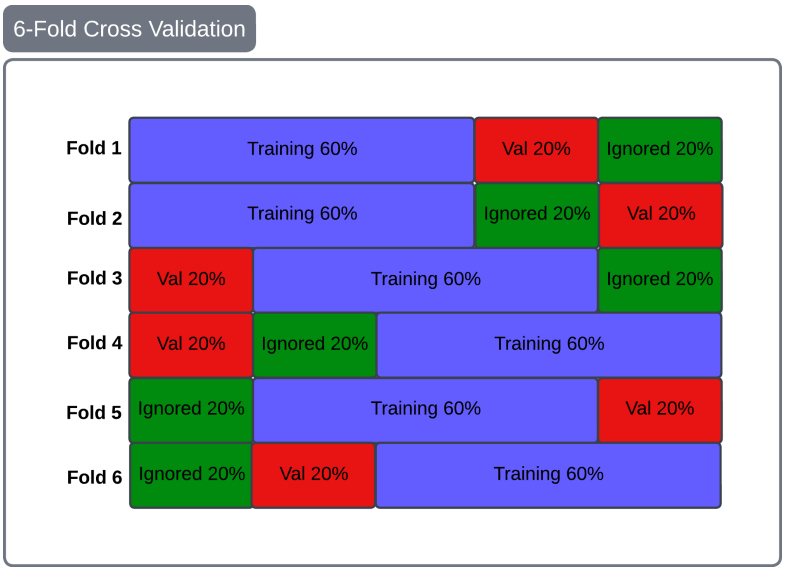


Figure 3.5: 6-fold Cross Validation

Generalization Testing

To assess performance on new domain, each of 6 models trained during the permutation cross-validation are evaluated on the separate CapEXP1 generalization dataset.

The final generalization performance is reported as the mean and standard deviation of the results from all 6 models. This approach provides a more reliable and unbiased estimate of how the model is expected to perform on truly new data rather than evaluating just a single model.

3.4 Analysis and Visualization

Following model training and prediction, a post-processing stage is essential to transform the models raw, scaled outputs into a interpretable format and to conduct perrformance analysis. TCN model is trained on targets which had been normalized; therefore, its predictions are also in a normalized scale. The post-processing procedure involve inverse normalization, metric calculation and generation of detailed visualizations.

3.4.1 Inverse Normalization

First step is to convert the model scaled predictions back into their original physical units (meters). This is achieved by applying the inverse transformation of the MinMaxScaler that was previously fitted only on the raw target data (X, Y columns) from training set. This ensures that all subsequent error calculations and visualizations are in a directly interpretable, real-world scale.

3.4.2 Visualization

A suite of visualizations is generated to provide comprehensive and qualitative understanding of models behavior on the training, validation, and test sets.

- **Training vs. Validation Loss:** A plot of MSE loss for training and validation sets over epochs. This is crucial for diagnosing learning progress and identifying potential overfitting.
- **Euclidean Distance Over Time:** This plot shows the per-sample spatial error across the entire dataset sequence. It is used to identify specific moments or segments of trajectorys where the model exhibits higher errors like during sharp turns or sudden stops.
- **Combined Trajectory and Error Analysis:** For each coordinate (X and Y), a detailed time-series plot has been generated. This visualization overlays the true and predicted positions over time. On y-axis, the signed error (Predicted - True) is also plotted, helping to identify any systematic biases in the predictions (i.e., whether the model consistently overestimate or underestimate the position).

3.5 Generalization to a New Dataset

A critical test of any predictive model is it ability to generalize to data which is entirely new and collected under different conditions. To assess this, trained models are evaluated on a completely unseen dataset, `preprocessed-CapEXP1.csv`, which originate from a different experimental session than primary dataset (`preprocessed-CapEXP2.csv`) used for training and cross-validation.

This process evaluates the models capacity to transfer its learned patterns to a new context.

The evaluation on the generalization dataset follows a strict protocol to ensure results are unbiased and methodologically sound:

- **Independent Preprocessing:** New dataset undergoes exact same preprocessing pipeline, including spike handling and kinematic feature engineering. Critically, for feature scaling, pipeline uses the scalers and parameters which had been fitted on the original CapEXP2 training data. This is essential for preventing any data leakage from new dataset into the model and ensures data is transformed in a manner consistent with training procedure.

- **Comprehensive Model Evaluation:** Rather than selecting a single "best" model, all 6 models generated during the 6-fold cross-validation are used for evaluation. Each of the 6 models makes predictions on fully preprocessed generalization dataset.

- **Aggregate Performance Reporting:** The performance metrics (MSE, MAE, and RMSE) are calculated for each of the 6 models. Final reported generalization performance is mean and standard deviation of these metrics across all 6 models.

This approach provide much more robust and honest measure of the models real-world potential. By averaging the results, we get a reliable estimate of expected performance, while standard deviation indicates stability and consistency of model when faced with completely new data.

3.6 Performance Metrics and Evaluation Criteria

The models performance is assessed using a combination of quantitative metrics and qualitative visual analysis. This dual approach provides comprehensive understanding of models accuracy, robustness, and specific failure modes.

Quantitative Metrics

The following standard regression metrics are used to quantify the models predictive accuracy. For aggregate metrics, values are computed across an entire dataset like validation fold or the test set.

Mean Squared Error (MSE): As primary loss function for training, MSE measures the average of the squares of errors. It is particularly sensitive to large errors.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(Y_{\text{pred},i} - Y_{\text{true},i})^2 \tag{3.1}$$

Mean Absolute Error (MAE): MAE measures the average absolute difference between predicted and true values, provide a error measure in the same units as target that is easy to interpret.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|Y_{\text{pred},i} - Y_{\text{true},i}| \tag{3.2}$$

Root Mean Squared Error (RMSE): As square root of MSE, RMSE is also in the same units as the target (meters), which make it more interpretable than MSE while still penalizing large errors more heavily than MAE.

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{3.3}$$

Euclidean Distance: To assess error on a per-sample basis, the Euclidean distance is calculated. This metric represent the direct spatial distance between predicted and true coordinate for each individual time step.

$$\text{Error}_{\text{Euclidean}} = \sqrt{(X_{\text{pred}} - X_{\text{true}})^2 + (Y_{\text{pred}} - Y_{\text{true}})^2} \tag{3.4}$$

Qualitative Visual Analysis

In addition to metrics, several visualizations has been generated to gain qualitative insights into the models behavior:

- **Training & Validation Loss Plot:** Which shows MSE loss over epochs for both training and validation sets, which is essential for diagnosing learning progress and overfitting.
- **Euclidean Distance vs. Time:** Plots the per-sample spatial error over the course of a trajectory which helps identifying specific events or types of movement like sharp turns where the models error is highest.
- **Combined Trajectory and Signed Error Plots:** This detailed visualization shows the true and predicted paths for each coordinate (X and Y) over time. A secondary y-axis displays the signed error (Predicted - True), which reveals any systematic biases like over or under shooting.

This combination of quantitative and qualitative evaluations provides robust framework for understanding the models performance and it limitations.

Chapter 4

Key Findings

This chapter presents key outcom from evaluation of the proposed Neural Networks and Domain Knowledge (NNDK) framework. The findings are detailed in two parts. First, framework high accuracy and stability on it source domain (CapEXP2) are analyzed which has been supported by results from a 6-fold permutation cross-validation. This section highlights the effectiveness of the TCN model when enriched with domain knowledge in the form of kinematic features. Second, framework performance assessed on a new, unseen domain (CapEXP1) to identify it generalization capabilities and limitations.

4.1 Performance on the Source Domain (CapEXP2)

The NNDK framework performance on the source dataset was assessed using custom 6-fold permutation cross-validation. Results show framework achieves high degree of accuracy and is stable across various training and validation configurations.

Aggregate performance is summarized in Table 4.1. Model achieved an average Root Mean Squared Error (RMSE) of 4.14 cm $\pm$ 0.75 cm across six validation permutations. Low standard deviation confirms that this level of performance is consistent and reproducible.

Table 4.1: Aggregate performance metrics on CapEXP2 using the 6-Fold Permutation CV

Metric	**Validation Average (Mean $\pm$ STD)**
MSE (m^2)	0.0018 $\pm$ 0.0006
RMSE (m)	0.0414 $\pm$ 0.0075
MAE (m)	0.0332 $\pm$ 0.0060

To provide insight into training dynamics, visualizations were generated for each of the six folds. Plots from Fold 6 are presented below as a representative example. Training and validation loss curve for this fold (Figure 4.1) show validation loss generally

tracks training loss downwards, which confirm model learns effectively without severe overfitting.

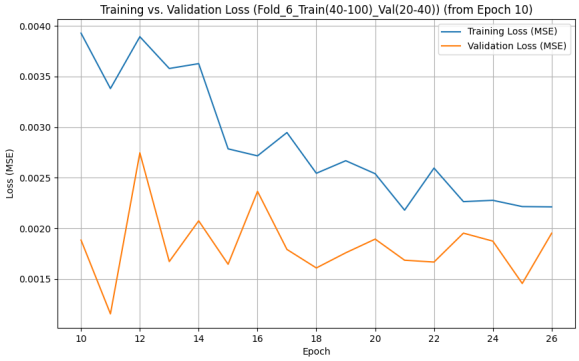


Figure 4.1: Training and validation loss curve for a representative fold (Fold 6).

4.2 Granular Error Analysis

While aggregate metrics in Table 4.1 are strong, analyzing per-sample error provide a deeper understanding of models behavior. For each fold, Euclidean distance and time-series trajectory errors were plotted. Following analysis of Fold 6 illustrates typical error characteristics observed across all folds.

The Euclidean distance plot (Figure 4.2) shows while prediction error is often low (mostly below 18 cm), there are distinct moments where error spikes, reaching high as 30 cm.

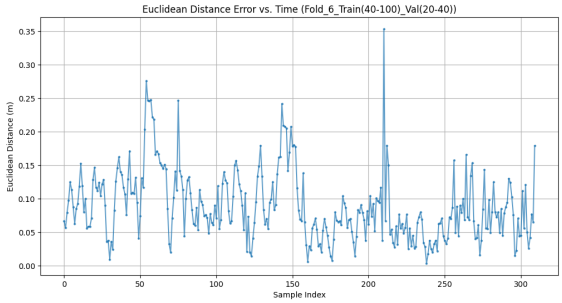


Figure 4.2: Euclidean error for the val of Fold 6.

To understand cause of these spikes, we can examine the detailed trajectory plot in Figure 4.3. This plot shows true and predicted paths for X and Y coordinates, along signed error. A clear pattern demonstraret:

- The largest errors (the spikes in the red error line) consistently occur at the peaks and troughs of the trajectory.

- These are the precise moments when the subject changes direction, corresponding to high-acceleration maneuvers.

This visual evidence confirm a key finding: the model is highly proficient at tracking smooth motion but is most challenged by abrupt changes in direction.

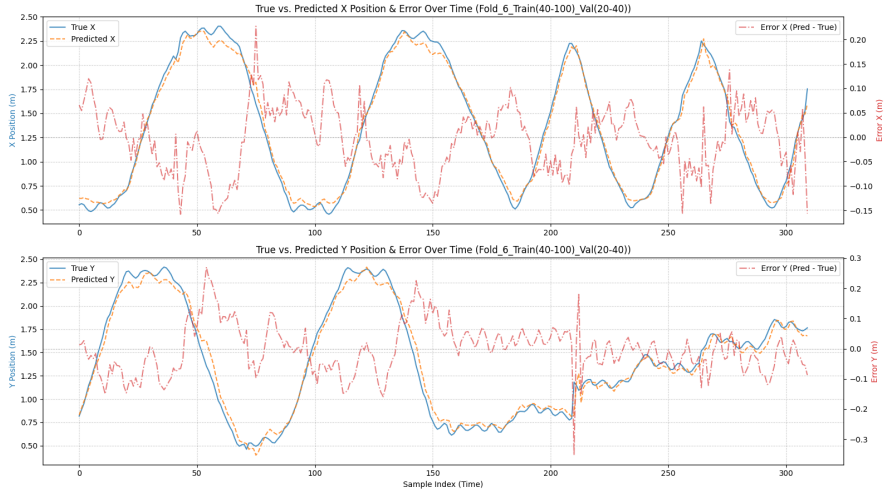


Figure 4.3: True vs. Pred with signed error for the val of Fold 6.

4.3 Generalization Performance and Domain Shift

A more challenging test was conducted to assess the models ability to generalize to new datasets (CapEXP1) recorded in different experimental session. This test reveals models limitations when faced with a domain shift, a critical challenge in real-world applications.

Generalization Set Performance: On the CapEXP1 dataset, the average model performance decreased significantly to RMSE of 13.22 cm $\pm$ 3.77 cm.

This threefold increase in error (from $\sim$4.1 cm on the source domain) indicates that while the model learned the specific dynamics of the CapEXP2 dataset very effectively, those learned patterns didnt fully transfer to the new domain. The larger standard deviation also shows that the models predictions were less consistent on this new data. This highlight domain generalization as the primary challenge for this approach. The result of the generalization part are been given in Table 4.2.

Table 4.2: NNDK Generalization Performance on CapEXP1 (Average of 6 Models)

Metric	**Result (Mean ± STD)**
MSE (m^2)	0.0187 ± 0.00995
RMSE (m)	0.1322 ± 0.03771
MAE (m)	0.1035 ± 0.0292

Qualitative Analysis of Generalization Errors

The quantitative performance degradation has been clearly reflected in the qualitative analysis. The Euclidean distance plot for the generalization set (Figure 4.4) shows much higher baseline error compared to the source domain plots, with frequent severe error spikes often exceeding 30 cm and even 40 cm.

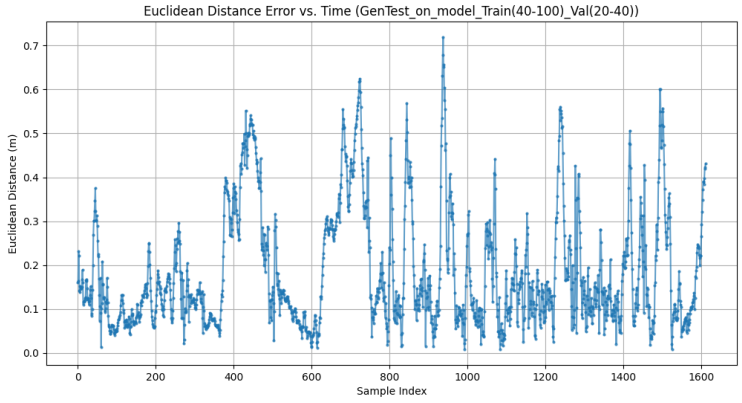


Figure 4.4: Per-sample Euclidean Distance error for the generalization set (CapEXP1).

The trajectory plot in Figure 4.5 provide a clear explanation for this. The movement patterns in the CapEXP1 dataset are fundamentally different from smoother, more repetitive paths in CapEXP2. They include long periods of inactivity followed by sudden movements and less regular oscillations.

The plot shows model struggling to track these complex dynamics. Prediction errors are no longer confine to sharp turns; they are distributed throughout the trajectory, indicating that the models learned patterns from the source domain are a poor fit for this new data distribution. This visual evidence strongly supports the conclusion of domain shift being the primary cause of the performance drop.

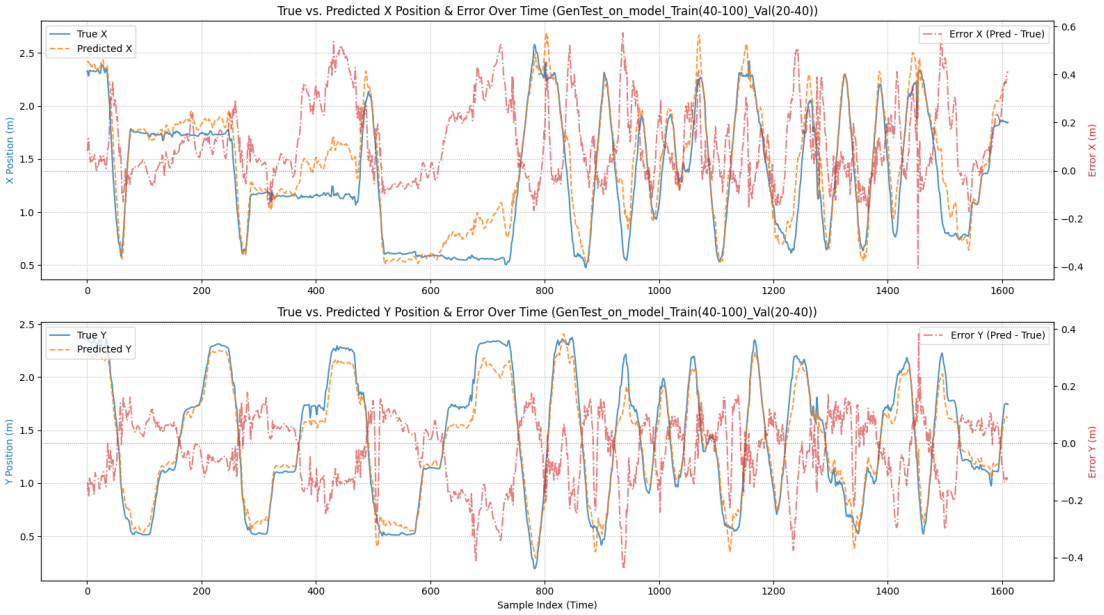


Figure 4.5: True vs. Predicted position with signed error for the generalization set (CapEXP1).

4.3.1 Computational Efficiency

The NNDK framework, which utilizes a TCN as its core, is wellsuited for efficient computation. This is due to several architectural advantages of Temporal Convolutional Networks. Unlike recurrent models like LSTMs, which process data sequentially, the convolutional nature of TCNs allows the parallel processing of entire input sequences, leading to faster training and inference (Tariq et al., 2020).

A key finding from systematic hyperparameter optimization process was identification of a final model architecture that is not only accurate but also remarkably lightweight and efficient. The final, optimized model contains only 7,410 total parameters, of which 7,218 are trainable.

The significance of these finding is that framework achieves its high in-domain accuracy without relying on a large, computationally expensive model. This small memory footprint (less than 30 KB) and lower computational requirement make approach a strong candidate for real-time applications and for future deployment on resource-constrained hardware, such as embedded systems.

4.4 Summary of Key Findings

The systematic evaluation of the NNDK framework produced several critical findings that clarify both its capabilities and its limitations for trajectory prediction:

- **High In-Domain Accuracy and Stability:** The framework demonstrated high-fidelity tracking on its source domain (CapEXP2), achieving a low average RMSE of ∼4.1 cm across the 6-fold cross-validation. Stability of this result confirmed by low standard deviation of validation scores.
- **Domain Shift is the Primary Limitation:** Framework performance did not generalize well to a new dataset (CapEXP1). The error rate increased significantly to an RMSE of ∼13.2 cm, clearly identifying domain shift as the most significant challenge for this approach.
- **Prediction Errors are Linked to Maneuvers:** The qualitative analysis of trajectory plots revealed that largest prediction errors consistently occur during high-acceleration maneuvers, such as when subject rapidly changes direction.
- **Computationally Efficient Architecture:** Systematic hyperparameter optimization result in a final model that is not only accurate but also highly efficient, containing just 7,218 trainable parameters.

Chapter 5

Discussion

This chapter provides interpretation of the key findings presented in the previous chapter. It analyzes the significance of the results, discusses the strengths and limitations of the proposed NNDK framework and contextualizes work within field of trajectory prediction.

5.1 Interpretation of Results

Evaluation of NNDK framework reveals a distinct and highly informative dual outcome. On one hand, framework demonstrates that a systematically optimized TCN, enriched with kinematic domain knowledge, can achieve high accuracy and stability for trajectory prediction within its source domain. On the other hand, drop in performance on a new dataset highlights critical challenge of domain shift.

The high in-domain accuracy ($\sim$4.1 cm RMSE) signifies that TCN architecture is indeed capable of learning complex, non-linear relationships between capacitive sensor data, movement dynamics, and physical location. Exceptional stability of this result, confirmed by low standard deviation in cross-validation folds, underscores robustness of the methodology, including hyperparameter tuning and training procedures. This suggests that for a well-defined and consistent operational environment, framework is reliable.

However, threefold increase in error on the generalization dataset is most critical finding. It strongly indicates that model did not purely learn general physics of human motion. Instead, it also "overlearned" specific statistical properties of CapEXP2 dataset, including unique noise profile of sensors, specific cadence and style of subjects movements and other environmental artifacts from that session. This is a classic example of domain shift, an common problem where machine learning models fail to generalize when deployed in environment that differs, even slightly, from the one they were trained in.

The granular error analysis supports this interpretation. The models primary weakness-predicting high-acceleration maneuvers-was present even in the source domain data. This weakness was significantly amplified on the generalization dataset, which had been contained different and more complex movement patterns. This show the model learned

patterns were brittle and not fully transferable, which leading to a breakdown in predictive accuracy.

5.2 Benchmarking: Impact of Domain Knowledge and Optimization

To quantify specific contributions of methodological improvements made in this work an direct benchmark was performed against baseline version of the model. This baseline represents initial approach, which used a TCN with un-optimized architecture and was trained without the kinematic domain knowledge features.

The results of this comparison are summarized in Table 5.1.

Table 5.1: Performance and Efficiency Benchmark.

Metric	**Baseline Model**	**Final NNDK Model**
Avg. Validation RMSE	6.38 cm	4.14 cm
Avg. Validation MAE	4.65 cm	3.32 cm
Trainable Parameters	∼397,000	∼7,218

The comparison reveals two clear and significant improvements:

Improved Predictive Accuracy: The final NNDK model achieved ∼30% reduction in RMSE compare to baseline. This gain in accuracy is primarily attributed to integration of domain knowledge. By engineering velocity and acceleration features, model was has been provided with explicit information about the objects movement dynamics. This allow it to learn more robust patterns and better predict future positions, especially during high-acceleration maneuvers that were identified as a key source of error.

Drastic Increase in Computational Efficiency: The most improvement is the reduction in model size by over 98% (from ∼397,000 to ∼7,218 trainable parameters). This is a direct result of systematic hyperparameter optimization. The Bayesian search successfully has identified a much more compact and efficient TCN architecture which not only outperformed larger baseline model but did so with a fraction of the computational resources.

In conclusion, this internal benchmark provided strong evidence for the efficacy of the NNDK methodology. The combination of domain-informed feature engineering and rigorous model optimization was critical also in developing a final model that is simultaneously more accurate and more efficient.

5.3 Analysis of Failure Modes

A crucial aspect of this research is understanding not just when the model succeeds, but more importantly, when and why it fail. The evaluation revealed two primary failure

modes: a difficulty in modeling high-acceleration maneuvers and sensitivity to domain shift.

5.3.1 Difficulty with High-Acceleration Maneuvers

The granular error analysis in the previous chapter consistently showed that largest prediction errors occur during moments of rapid change in direction or speed. While the model excels at tracking predictable and smooth motions it accuracy decrease significantly during these maneuvers. This failure can be attributed to several factors:

- **Limited Predictive Horizon:** The model uses a fixed-size input window of 15 time steps to make a prediction. A sudden, sharp turn may not be sufficiently foreshadowed within this size of the window, which gives the model inadequate information to anticipate the abrupt change in trajectory.
- **Data Imbalance:** In most trajectories, periods of smooth motion are far more common than sharp maneuvers. The training data is therefore inherently imbalanced, with fewer examples of these high-acceleration events. As a result, model is biased towards learning the patterns of simpler movements and has not developed a sufficiently robust understanding of more complex dynamics.

5.3.2 Sensitivity to Domain Shift

The most significant failure mode is the models inability to generalize to the CapEXP1 dataset. This sensitivity to domain shift indicates that model has learned patterns which are too specific to the source domain (CapEXP2) and not robust enough for new environments. This can be explained by:

- **Overfitting to Source Specific Artifacts:** The model likely learned not just the general physics of motion but also correlations which are unique to the training data. These could include the specific noise signature of the sensors during that session, the unique gait and turning style of the subject, or other environmental factors. When these conditions change in the new domain, the models performance break down.
- **Lack of Diverse Training Data:** The model was trained on data from a single, relatively controlled environment and experimental session. It was not exposed to wide variety of movement styles, sensor conditions or room layouts. This lack of diversity in training data is a primary cause of its brittleness when faced with a new domain.

Understanding these failure modes is essential, as they reveal that models errors are not random but are systematically linked to these specific, identifiable challenges. This provides a clear roadmap for targeted improvements in future work.

Chapter 6

Practical Implications

The findings of this research, which detail both the high in-domain accuracy of the NNDK framework and its significant challenges with domain generalization, have practical implications for both academic research and potential industrial applications. The results serve as proof of concept for the potential of this technology while also highlighting the critical challenges that must be overcome for real world deployment.

6.1 For Academic Research

This thesis provides several key contributions to academic community:

- **A Methodological Blueprint:** It offers detailed, end-to-end case study on applying a TCN based model for trajectory prediction. This includes a robust pipeline for data preprocessing, kinematic feature engineering, systematic hyperparameter optimization, and a rigorous cross-validation evaluation framework.
- **Validation of TCNs for Trajectory Tasks:** Research validates that a well-tuned TCN is a powerful and computationally efficient architecture for modeling complex, non-linear trajectories from sensor data, achieving high accuracy and stability within a specific domain.
- **A Clear Case Study of Domain Shift:** By quantitatively demonstrating a threefold increase in error when the model is applied to a new dataset, this work provide a clear and valuable case study on the problem of domain shift in sensor-based human tracking. It offers a concrete benchmark for future research aimed at developing more generalizable and robust models.

6.2 For Industrial and Applied Fields

While current model is not ready for direct deployment due to generalization issues, this research highlights potential of technology and clarifies primary obstacle for its industrial application.

High in-domain accuracy points to potential in areas such as:

- **Robotics and Autonomous Systems:** For functions like human robot collaboration and collision avoidance in controlled environments like an specific assembly line.
- **Smart Environments:** For context-aware systems like adaptive lighting or HVAC, within a single, stable installation where system can be calibrated.

However critical implication for all industrial applications is necessity of solving domain generalization problem. For this technology to be commercially viable, it must be robust to changes of environment and subject. A system trained in one room must work reliably in another, and it must perform accurately regardless of which individual is being tracked.

Finally, high computational efficiency of final model with only ∼7,200 trainable parameters, is significant finding. It implies such high-fidelity tracking does not necessarily require large, power-hungry models, which opens the possibility for future deployment on low-cost, low-power edge devices.

Chapter 7

Limitations and Future Directions

While the NNDK framework has demonstrated high accuracy within its source domain, research has also highlights several key limitations. Acknowledging these is crucial for contextualizing the results and establishing a clear foundation for future research.

7.1 Limitations

The primary limitations of this study are:

- **Scope of the Prediction Task:** The most significant limitation is the fundamental scope of the problem solved. The implemented framework addresses a time-series forecasting problem ((x, y) to (x, y)), where it predicts the next position based on the history of known ground-truth coordinates. This is distinct from more complex position inference problem (CAP_n to (x, y)), which would involve determining location solely from raw sensor readings. This scope limits the models direct use in real-world scenarios where a stream of ground-truth data is not available.

- **Sensitivity to Domain Shift:** As a direct consequence of limited data, models performance degraded significantly when applied to a new dataset. This shows the model is not yet robust to changes in environment or subject movement, which is a major barrier to practical deployment.

- **Controlled and Limited Training Data:** Model was trained on data from a single, empty 3m×3m room with only one subject. This controlled environment does not capture complexities of real-world scenarios, which include furniture, obstacles, and multiple people.

- **Single-Step Prediction Horizon:** Current model is designed only to predict immediate next step in trajectory. This limits its utility for applications which require longer-term planning or proactive decision-making.

- **Reliance on a Single Sensor Modality:** Framework relies exclusively on capacitive sensors, making system vulnerable to their specific failure modes and limiting the potential for enhanced robustness through sensor fusion.

7.2 Future Directions

Future work should focus directly on addressing these limitations to move towards a more robust and practical solution.

Transitioning to Direct Position Inference (Top Priority)

Most critical future work is to extend framework to solve full CAP_n to (x, y) position inference problem. This would involve:

- Reworking the model inputs to rely primarily on sequence of capacitive sensor readings, removing dependency on historical ground-truth coordinates.
- Revisiting model architecture to handle much more complex task of learning the underlying physics between capacitance and spatial location.

This represents most significant step toward creating a truly standalone indoor positioning system.

Improving Generalization of the Current Model

- **Data Diversity and Augmentation:** To combat domain shift in the existing forecasting model, future work should collect data from a wide variety of environments, with different subjects and more complex movement patterns.
- **Domain Adaptation Techniques:** Advanced methods like domain-adversarial training should be explored to encourage model to learn features that are invariant across different domains.

Multi-Step Trajectory Forecasting

Model architecture should be extended to predict a sequence of future steps rather than just one. This would likely involve:

- Exploring recursive prediction strategies.
- Developing mechanisms to mitigate compounding errors that can occur over longer horizons.

Enhancing Real-World Robustness

- **Complex Environments & Multi-Subject Tracking:** The framework should be tested in larger, cluttered environments and expanded to handle tracking multiple subjects.
- **Sensor Fusion:** To improve reliability, capacitive sensor data should be fused with complementary modalities, such as Inertial Measurement Units (IMUs).

Improving Model Trustworthiness

- **Uncertainty Quantification:** Future models should be developed to output not just a coordinate prediction but also a confidence score, which is critical for safety-related applications.
- **Model Explainability:** Techniques like SHAP (SHapley Additive exPlanations) should be used to better understand how model makes its decisions, increasing trust and aiding in debugging.

Chapter 8

Conclusion

This thesis addresse challenge of high-precision trajectory prediction from noisy capacitive sensor data by developing and systematically evaluating a Neural Networks and Domain Knowledge (NNDK) framework. The work successfully demonstrate proposed approach, which integrates kinematic features into a Temporal Convolutional Network (TCN), can achieve high predictive accuracy and stability within its source domain. However, it also quantifies the significant challenge of domain shift, revealing a pronounced drop in performance when model is applied to data from a new, unseen experimental session.

8.1 Summary of Key Findings

The core findings of this research provide a balanced view of the frameworks capabilities:

- **High In-Domain Performance:** The systematically optimized NNDK framework is capable of high-fidelity prediction on its source dataset (CapEXP2), whih achieves a low average RMSE of ∼4.1 cm across the 6-fold permutation cross-validation, with low standard deviation confirming stability of this result.
- **Domain Shift is the Key Limitation:** The models performance did not generalize well to a new dataset (CapEXP1). The error rate increased threefold to an RMSE of ∼13.2 cm, clearly identifying domain shift as most significant challenge for this approach.
- **Errors are Systematic:** Qualitative analysis of trajectory plots revealed largest prediction errors are not random but are systematically linked to high-acceleration maneuvers, such as when subject rapidly changes direction.
- **Computationally Efficient Architecture:** The systematic hyperparameter optimization resulted in a final model that is not only accurate but also highly efficient, containing just 7,218 trainable parameters.

8.2 Contributions to the Field

This thesis makes several contributions to the field of trajectory prediction:

- **A Methodological Blueprint:** It provides a complete, end-to-end methodology for applying and evaluating a TCN for this task. This includes a pipeline for data cleaning, domain-informed feature engineering, systematic hyperparameter optimization, and a custom permutation-based cross-validation scheme.
- **An Empirical Case Study:** The research serves as a detailed case study that both validates effectiveness of TCNs for high-accuracy in-domain prediction and provides a clear, quantitative benchmark of domain shift problem, which is a valuable contribution for researchers working on model generalization.
- **An Efficient, High-Performing In-Domain Model:** The work produced a lightweight and accurate predictive model which is well suited for its source domain, demonstrating the potential of the approach under controlled conditions.

In conclusion, while the NNDK framework shows significant promise, this thesis establish the path toward building truly robust and reliable real-world trajectory prediction systems must prioritize solving the fundamental challenge of domain generalization.

Bibliography

Muhammad Raisul Alam, M. B. I. Reaz, and M. A. Mohd Ali. SPEED: An Inhabitant Activity Prediction Algorithm for Smart Homes. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 42(4):985–990, 2012. ISSN 1083-4427. doi: 10.1109/TSMCA.2011.2173568.

Osama Bin Tariq, Mihai Teodor Lazarescu, and Luciano Lavagno. Drift rejection differential frontend for single-plate capacitive sensors. *IEEE Transactions on Instrumentation and Measurement*, 70:1–9, 2021. doi: 10.1109/TIM.2021.3052276.

Osama Bin Tariq, Mihai Teodor Lazarescu, and Luciano Lavagno. Neural networks for indoor person tracking with infrared sensors. *IEEE Sensors Journal*, 22(2):1198–1208, 2022. doi: 10.1109/JSEN.2021.3119212.

Long Chen, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. Robust fall detection using a wearable capacitive sensing device based on lstm recurrent neural networks. *Scientific Reports*, 10(1):8672, 2020. doi: 10.1038/s41598-020-65070-5.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, FranÃ§ois Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016. URL http://jmlr.org/papers/v17/15-239.html.

Artur S. D'Avila Garcez, Tarek R. Besold, Luc de Raedt, Peter FÃ¶ldiak, Pascal Hitzler, Thomas Icard, Kai-Uwe KÃ¼hnberger, Luis C. Lamb, Risto Miikkulainen, and David L. Silver. Injecting domain knowledge in neural networks. *Neural Networks*, 120:1–6, 2019. doi: 10.1016/j.neunet.2019.06.014.

Xiangjie Kong, Zhenghao Chen, Weiyao Liu, Kaili Ning, Lechao Zhang, Syauqie Muhammad Marier, Yichen Liu, Yuhao Chen, and Feng Xia. Deep learning for time series forecasting: a survey. *International Journal of Machine Learning and Cybernetics*, 2025. doi: 10.1007/s13042-025-02560-w. Online First.

Masato Kuki, Hiroshi Nakajima, Naoki Tsuchiya, and Yutaka Hata. Multi-human locating in real environment by thermal sensor. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4623–4628. IEEE, 2013. doi: 10.1109/SMC.2013.787.

Colin Lea, Michael D. Flynn, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 156–165, 2017.

Benyang Li, Zhiqiang Li, Xinying Li, and Wenjun Wu. A sensor fusion framework for indoor localization using smartphone sensors and wi-fi rssi measurements. *Sensors*, 17 (6):1325, 2017. doi: 10.3390/s17061325.

Changlong Li, Jianhong Wang, Min Zhang, Xinqi Liang, Xiaoguang Sun, and Hengji Yang. Self-organizing neural networks integrating domain knowledge and reinforcement learning. *Neurocomputing*, 550:127382, 2024. doi: 10.1016/j.neucom.2023.127382.

Hailin Liu, Qixin Zhang, Haining Zhang, and Jizhong Zhao. A sensor fusion method for wi-fi-based indoor positioning. *Sensors*, 16(9):1424, 2016. doi: 10.3390/s16091424.

Matteo Panella, Andrea Monteriù, Emanuele Frontoni, and Primo Zingaretti. A comparative evaluation of the interpretability of neural networks applied to human activity recognition. *Sensors*, 23(20):8598, 2023. doi: 10.3390/s23198598.

Alireza Ramezani, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. Automatic segmentation and activity classification using accelerometer data and deep learning. *Applied Sciences*, 10(7):2322, 2020. doi: 10.3390/app10072322.

Alireza Ramezani Akhmareh, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. A tagless indoor localization system based on capacitive sensing technology. *Sensors*, 16(9):1448, 2016. doi: 10.3390/s16091448.

Giorgia Subbicini, Luciano Lavagno, and Mihai T Lazarescu. Enhanced exploration of neural network models for indoor human monitoring. In *2023 9th International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 109–114. IEEE, 2023. doi: 10.1109/IWASI57999.2023.10182625.

Osama Bin Tariq, Mihai Teodor Lazarescu, and Luciano Lavagno. Neural networks for indoor human activity reconstructions. *IEEE Sensors Journal*, 20(22):13571–13584, 2020. doi: 10.1109/JSEN.2020.3006834.

Ivor W Tsang, Sinno Jialin Pan, Yew-Soon Liu, and Xingquan Zhu. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3409–3423, 2020. doi: 10.1109/TNNLS.2019.2950644.

Xiaowei Wang, Haoran Luo, Zhifang He, Biao Jiang, and Qian Zhao. Enhancing trajectory prediction by fusing transformer and graph neural networks. *Information Sciences*, 638:119103, 2023. doi: 10.1016/j.ins.2023.119103.

Raphael Wimmer, Matthias Kranz, Sebastian Boring, and Albrecht Schmidt. A capacitive sensing toolkit for pervasive activity detection and recognition. In *Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 171–180, White Plains, NY, USA, 2007. doi: 10.1109/PERCOM.2007.1.

Yucheng Zhang, Zhe Feng, Yang Li, Xin Zhao, and Weijie Li. Domain knowledge-driven relation extraction method based on graph neural network. *Neural Computing and Applications*, 2022. doi: 10.1007/s00521-022-07384-0.

Shaojie Zhu, Lei Zhang, Bailong Liu, Shumin Cui, Changxing Shao, and Yun Li. Mode normalization enhanced recurrent model for multi-modal semantic trajectory prediction. *IEICE Transactions on Information and Systems*, E103.D(1):174–176, 2020. doi: 10.1587/transinf.2019EDL8130.

Appendix

.1 Python Codes

```
# run_pipeline.py

import os
import matplotlib.pyplot as plt
import numpy as np
import joblib
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import pandas as pd
import sys
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
from sklearn.model_selection import KFold
from preprocessing import TrajectoryPreprocessor
from nndk_core import train_model, load_trained_model

# --- GLOBAL CONFIGURATION ---
CSV_PATH      = r"../../datasets/preprocessed-CapEXP2.csv"
GENERALIZATION_CSV_PATH = r"../../datasets/preprocessed-CapEXP1.csv"
SEQ_LEN       = 15
TARGET_COLS   = ['X', 'Y']
EPOCHS        = 50
BATCH_SIZE    = 8
# --- Hyperparameters from tuning ---
LR            = 0.01
NUM_FILTERS   = 16
HIDDEN_LAYERS = 3
K_SIZE        = 3
DENSE_UNITS   = 128
DROPOUT_RATE  = 0.1
# -----------------------------------
RESULTS_DIR = 'results_final'
os.makedirs(RESULTS_DIR, exist_ok=True)
FEATURE_SCALER_DIR = RESULTS_DIR
TARGET_SCALER_PATH = os.path.join(RESULTS_DIR, "target_scaler.pkl")

# --- PLOTTING FUNCTION ---
def generate_and_save_plots(history, X_data, Y_true_raw, model,
    split_name, plot_output_dir, scaler_y):
```

```
    os.makedirs(plot_output_dir, exist_ok=True)
    print(f"Generating plots for {split_name} in {plot_output_dir}...")
    Y_pred_scaled = model.predict(X_data)
    Y_pred_orig = scaler_y.inverse_transform(Y_pred_scaled)
    if history:
        plt.figure(figsize=(10, 6))

        train_loss = history.history['loss']
        val_loss = history.history['val_loss']

        plot_start_epoch = 10
        if len(train_loss) >= plot_start_epoch:
            epochs_range = range(plot_start_epoch, len(train_loss) + 1)
            plt.plot(epochs_range, train_loss[plot_start_epoch-1:], label
                ='Training Loss (MSE)')
            plt.plot(epochs_range, val_loss[plot_start_epoch-1:], label='
                Validation Loss (MSE)')
            plt.title(f'Training vs. Validation Loss ({split_name}) (from
                 Epoch {plot_start_epoch})')
        else:
            epochs_range = range(1, len(train_loss) + 1)
            plt.plot(epochs_range, train_loss, label='Training Loss (MSE)
                ')
            plt.plot(epochs_range, val_loss, label='Validation Loss (MSE)
                ')
            plt.title(f'Training vs. Validation Loss ({split_name})')

        plt.xlabel('Epoch')
        plt.ylabel('Loss (MSE)')
        plt.legend()
        plt.grid(True)
        plt.savefig(os.path.join(plot_output_dir, f'Loss_vs_Epoch_{
            split_name}.png'))
        plt.close()

    error_x = Y_pred_orig[:, 0] - Y_true_raw[:, 0]
    error_y = Y_pred_orig[:, 1] - Y_true_raw[:, 1]
    euclidean_dist = np.sqrt(error_x**2 + error_y**2)

    plt.figure(figsize=(12, 6))
    plt.plot(euclidean_dist, marker='o', linestyle='-', markersize=2,
        alpha=0.7)
    plt.title(f'Euclidean Distance Error vs. Time ({split_name})')
    plt.xlabel('Sample Index'); plt.ylabel('Euclidean Distance (m)'); plt
        .grid(True)
    plt.savefig(os.path.join(plot_output_dir, f'Euclidean_Distance_{
        split_name}.png'))
    plt.close()

    # Plot 2: True & Predicted Trajectory with Signed Error
    error_x_signed = Y_pred_orig[:, 0] - Y_true_raw[:, 0]
    error_y_signed = Y_pred_orig[:, 1] - Y_true_raw[:, 1]
    plt.figure(figsize=(16, 9))
```

```
    # Subplot for X
    ax1 = plt.subplot(2, 1, 1)
    ax1.plot(np.arange(len(Y_true_raw)), Y_true_raw[:, 0], label='True X'
        , alpha=0.8, color='tab:blue')
    ax1.plot(np.arange(len(Y_true_raw)), Y_pred_orig[:, 0], label='
        Predicted X', linestyle='--', alpha=0.8, color='tab:orange')
    ax1.set_ylabel('X Position (m)', color='tab:blue')
    ax1.legend(loc='upper left')
    ax1.grid(True, linestyle=':')
    ax1_twin = ax1.twinx()
    ax1_twin.plot(np.arange(len(Y_true_raw)), error_x_signed, label='
        Error X (Pred - True)', alpha=0.6, color='tab:red', linestyle='-.'
        )
    ax1_twin.set_ylabel('Error X (m)', color='tab:red')
    ax1_twin.axhline(0, color='gray', linestyle=':', linewidth=0.8)
    ax1_twin.legend(loc='upper right')
    ax1.set_title(f'True vs. Predicted X Position & Error Over Time ({
        split_name})')

    # Subplot for Y
    ax2 = plt.subplot(2, 1, 2, sharex=ax1)
    ax2.plot(np.arange(len(Y_true_raw)), Y_true_raw[:, 1], label='True Y'
        , alpha=0.8, color='tab:blue')
    ax2.plot(np.arange(len(Y_true_raw)), Y_pred_orig[:, 1], label='
        Predicted Y', linestyle='--', alpha=0.8, color='tab:orange')
    ax2.set_xlabel('Sample Index (Time)')
    ax2.set_ylabel('Y Position (m)', color='tab:blue')
    ax2.legend(loc='upper left')
    ax2.grid(True, linestyle=':')
    ax2_twin = ax2.twinx()
    ax2_twin.plot(np.arange(len(Y_true_raw)), error_y_signed, label='
        Error Y (Pred - True)', alpha=0.6, color='tab:red', linestyle='-.'
        )
    ax2_twin.set_ylabel('Error Y (m)', color='tab:red')
    ax2_twin.axhline(0, color='gray', linestyle=':', linewidth=0.8)
    ax2_twin.legend(loc='upper right')
    ax2.set_title(f'True vs. Predicted Y Position & Error Over Time ({
        split_name})')

    plt.tight_layout()
    plt.savefig(os.path.join(plot_output_dir, f'
        Combined_XY_Error_vs_Time_{split_name}.png'))
    plt.close()

# --- GENERALIZATION TEST FUNCTION ---
def run_generalization_test(new_csv_path: str, trained_model_path: str,
    target_scaler_path: str, feature_scaler_dir: str, experiment_name: str
    ):
    print(f"\n--- Starting Generalization Test on: {os.path.basename(
        new_csv_path)} ---")
    preprocessor = TrajectoryPreprocessor(seq_len=SEQ_LEN, target_cols=
        TARGET_COLS)
```

```
    df_gen = pd.read_csv(new_csv_path)
    df_gen = preprocessor._handle_spikes(df_gen)
    df_gen = preprocessor._add_kinematic_features(df_gen)

    features_normalized_gen = preprocessor.load_scalers_and_transform(
        df_gen, feature_scaler_dir)
    targets_raw_gen = df_gen[TARGET_COLS].to_numpy(dtype=np.float32)
    X_gen, Y_gen_raw = preprocessor._make_windows(features_normalized_gen
        , targets_raw_gen)

    model_for_test = load_trained_model(trained_model_path)
    scaler_y_loaded = joblib.load(target_scaler_path)

    Y_gen_pred_scaled = model_for_test.predict(X_gen)
    Y_gen_pred_orig = scaler_y_loaded.inverse_transform(Y_gen_pred_scaled
        )

    gen_mse = mean_squared_error(Y_gen_raw, Y_gen_pred_orig)
    gen_mae = mean_absolute_error(Y_gen_raw, Y_gen_pred_orig)
    gen_rmse = np.sqrt(gen_mse)
    plot_output_dir = os.path.join(RESULTS_DIR, "plots", "
        generalization_tests")
    generate_and_save_plots(None, X_gen, Y_gen_raw, model_for_test, f"
        GenTest_on_model_{experiment_name}", plot_output_dir,
        scaler_y_loaded)
    return {'MSE': gen_mse, 'RMSE': gen_rmse, 'MAE': gen_mae}


if __name__ == '__main__':
    # 1. DATA LOADING AND PREPARATION
    print("--- Loading and Preparing Full Dataset ---")
    preprocessor = TrajectoryPreprocessor(seq_len=SEQ_LEN, target_cols=
        TARGET_COLS)
    df_full = pd.read_csv(CSV_PATH)
    df_full_processed = preprocessor._handle_spikes(df_full)
    df_full_processed = preprocessor._add_kinematic_features(
        df_full_processed)
    features_normalized_full = preprocessor.
        fit_and_transform_training_data(df_full_processed,
        FEATURE_SCALER_DIR)

    scaler_y = MinMaxScaler()
    targets_raw_full = df_full_processed[TARGET_COLS].to_numpy(dtype=np.
        float32)
    targets_scaled_full = scaler_y.fit_transform(targets_raw_full)
    joblib.dump(scaler_y, TARGET_SCALER_PATH)

    X_full, Y_full_scaled = preprocessor._make_windows(
        features_normalized_full, targets_scaled_full)
    _, Y_full_raw = preprocessor._make_windows(features_normalized_full,
        targets_raw_full)

 # 2. SETUP FOR CUSTOM 6-FOLD PERMUTATION CROSS VALIDAITOM
```

```
n_samples = len(X_full)
cv_results = []
gen_results = []
fold_definitions = {
    1: {'name': 'Train(0-60)_Val(60-80)',   'train': (0.0, 0.6), 'val
        ': (0.6, 0.8)},
    2: {'name': 'Train(0-60)_Val(80-100)',  'train': (0.0, 0.6), 'val
        ': (0.8, 1.0)},
    3: {'name': 'Val(0-20)_Train(20-80)',   'train': (0.2, 0.8), 'val
        ': (0.0, 0.2)},
    4: {'name': 'Val(0-20)_Train(40-100)',  'train': (0.4, 1.0), 'val
        ': (0.0, 0.2)},
    5: {'name': 'Train(20-80)_Val(80-100)', 'train': (0.2, 0.8), 'val
        ': (0.8, 1.0)},
    6: {'name': 'Train(40-100)_Val(20-40)', 'train': (0.4, 1.0), 'val
        ': (0.2, 0.4)},
}

print(f"\n--- Starting Custom 6-Fold Permutation Cross-Validation ---
    ")
for fold_num, fold_info in fold_definitions.items():
    print(f"\n--- Processing Fold {fold_num}/6: {fold_info['name']}
        ---")

    tr_start_idx = int(fold_info['train'][0] * n_samples)
    tr_end_idx = int(fold_info['train'][1] * n_samples)
    val_start_idx = int(fold_info['val'][0] * n_samples)
    val_end_idx = int(fold_info['val'][1] * n_samples)
    X_train_fold, Y_train_fold = X_full[tr_start_idx:tr_end_idx],
        Y_full_scaled[tr_start_idx:tr_end_idx]
    X_val_fold, Y_val_fold = X_full[val_start_idx:val_end_idx],
        Y_full_scaled[val_start_idx:val_end_idx]
    Y_val_raw_fold = Y_full_raw[val_start_idx:val_end_idx]

    model_path = os.path.join(RESULTS_DIR, f"model_fold_{fold_num}.
        keras")
    history, val_metrics_list = train_model(
        X_train_fold, Y_train_fold, [],
        X_val_fold, Y_val_fold, [],
        X_val_fold, Y_val_fold, [],
        save_path=model_path,
        epochs=EPOCHS, batch_size=BATCH_SIZE, learning_rate=LR,
        hidden=HIDDEN_LAYERS, num_filters=NUM_FILTERS,
        k_size=K_SIZE, dense_units=DENSE_UNITS,
        dropout_rate=DROPOUT_RATE
    )
    cv_results.append({
        'val_MSE': val_metrics_list[0], 'val_MAE': val_metrics_list
            [1], 'val_RMSE': val_metrics_list[2]
    })

    # Generate plots for this fold's validation set
    model = load_trained_model(model_path)
```

```
        plot_output_dir = os.path.join(RESULTS_DIR, "plots", f"fold_{
            fold_num}")
        generate_and_save_plots(history, X_val_fold, Y_val_raw_fold,
            model, f"Fold_{fold_num}_{fold_info['name']}", plot_output_dir
            , scaler_y)
        if os.path.exists(GENERALIZATION_CSV_PATH):
            gen_metrics = run_generalization_test(
                new_csv_path=GENERALIZATION_CSV_PATH, trained_model_path=
                    model_path,
                target_scaler_path=TARGET_SCALER_PATH, feature_scaler_dir
                    =FEATURE_SCALER_DIR,
                experiment_name=fold_info['name']
            )
            gen_results.append(gen_metrics)

    # 4. CREATE AND PRINT THE FINAL SUMMARY TABLES
    print("\n\n--- Cross-Validation Performance Summary ---")
    cv_df = pd.DataFrame(cv_results)
    cv_summary = pd.DataFrame({
        "Mean": cv_df.mean(),
        "STD": cv_df.std()
    }).T
    cv_summary.index.name = "Statistic"
    print("The table below shows the average performance across the 6
        validation permutations.")
    print(cv_summary.to_string())
    cv_summary.to_csv(os.path.join(RESULTS_DIR, "permutation_cv_summary.
        csv"))

    if gen_results:
        print("\n\n--- Generalization Performance Summary ---")
        gen_df = pd.DataFrame(gen_results)
        gen_summary = pd.DataFrame({
            "Mean": gen_df.mean(),
            "STD": gen_df.std()
        }).T
        gen_summary.index.name = "Statistic"
        print("The table below shows the average generalization
            performance across all 6 models.")
        print(gen_summary.to_string())
        gen_summary.to_csv(os.path.join(RESULTS_DIR, "
            generalization_summary.csv"))
```

Listing 1: Run Pipeline

```python
# preprocessing.py

import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import joblib
import os

class TrajectoryPreprocessor:
    def __init__(self, seq_len: int, target_cols: list):
        """Initializes the preprocessor."""
        self.seq_len = seq_len
        self.target_cols = target_cols
        self.pos_cols = ['X', 'Y']
        self.vel_cols = ['vx', 'vy']
        self.accel_cols = ['ax', 'ay']
        self.other_cols = ['cap1', 'cap2', 'cap3', 'cap4']
        self.feature_cols = self.other_cols + self.pos_cols + self.
            vel_cols + self.accel_cols
        self.scaler_other = MinMaxScaler()
        self.kinematic_params = {}

    def _add_kinematic_features(self, df: pd.DataFrame) -> pd.DataFrame:
        """Calculates velocity and acceleration from X, Y columns."""
        df['vx'] = df['X'].diff().fillna(0)
        df['vy'] = df['Y'].diff().fillna(0)
        df['ax'] = df['vx'].diff().fillna(0)
        df['ay'] = df['vy'].diff().fillna(0)
        return df

    def _handle_spikes(self, df: pd.DataFrame, threshold: float = 0.5) ->
         pd.DataFrame:
        """Identifies and corrects sharp spikes in positional data."""
        print(f"Applying threshold-based spike handling with threshold: {
            threshold}m")
        for col in ['X', 'Y']:
            diffs = df[col].diff().abs()
            spike_indices = diffs[diffs > threshold].index
            for idx in spike_indices:
                if idx > 0:
                    df.loc[idx, col] = df.loc[idx - 1, col]
        return df

    def fit_and_transform_training_data(self, df: pd.DataFrame,
        scaler_dir: str) -> np.ndarray:
        """
        Fits all scalers on the training data, saves them, and transforms
             the data.
        """
        print("Fitting shared Min-Max scalers on training data...")
        os.makedirs(scaler_dir, exist_ok=True)
        self.scaler_other.fit(df[self.other_cols])
```

```
        joblib.dump(self.scaler_other, os.path.join(scaler_dir, "
            other_features_scaler.pkl"))
        for key, cols in [('pos', self.pos_cols), ('vel', self.vel_cols),
             ('accel', self.accel_cols)]:
            group_data = df[cols].values
            self.kinematic_params[key] = {
                'min': group_data.min(),
                'max': group_data.max()
            }

        joblib.dump(self.kinematic_params, os.path.join(scaler_dir, "
            kinematic_params.pkl"))
        print(f"Scalers fitted and saved to {scaler_dir}")

        return self.transform_features(df)

    def load_scalers_and_transform(self, df: pd.DataFrame, scaler_dir:
        str) -> np.ndarray:
        """
        Loads pre-fitted scalers from disk and uses them to transform new
             data.
        """
        print(f"Loading scalers from {scaler_dir} and transforming data
            ...")

        self.scaler_other = joblib.load(os.path.join(scaler_dir, "
            other_features_scaler.pkl"))
        self.kinematic_params = joblib.load(os.path.join(scaler_dir, "
            kinematic_params.pkl"))

        return self.transform_features(df)

    def transform_features(self, df: pd.DataFrame) -> np.ndarray:
        """Helper function to apply all transformations."""
        scaled_other = self.scaler_other.transform(df[self.other_cols])

        scaled_pos = (df[self.pos_cols] - self.kinematic_params['pos']['
            min']) / \
                     (self.kinematic_params['pos']['max'] - self.
                         kinematic_params['pos']['min'])

        scaled_vel = (df[self.vel_cols] - self.kinematic_params['vel']['
            min']) / \
                     (self.kinematic_params['vel']['max'] - self.
                         kinematic_params['vel']['min'])

        scaled_accel = (df[self.accel_cols] - self.kinematic_params['
            accel']['min']) / \
                       (self.kinematic_params['accel']['max'] - self.
                           kinematic_params['accel']['min'])

        return np.concatenate([scaled_other, scaled_pos, scaled_vel,
            scaled_accel], axis=1)
```

```
    def _make_windows(self, feats: np.ndarray, targs: np.ndarray):
        """Creates overlapping windows from the time-series data."""
        N = len(feats)
        X, Y = [], []
        for i in range(N - self.seq_len):
            X.append(feats[i:i + self.seq_len])
            Y.append(targs[i + self.seq_len])
        return np.array(X), np.array(Y)

    def _make_target_windows(self, targs: np.ndarray):
        """Creates historical target windows."""
        N = len(targs)
        seqs = []
        for i in range(N - self.seq_len + 1):
            seqs.append(targs[i:i + self.seq_len])
        return np.array(seqs)
```

Listing 2: Pre Processing

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.models import load_model
from tcn_simple import TCN_model
from tcn_model  import model_TCN_simple
from utils import compute_test

def train_model(
    X_train, Y_train, Y_train_seq,
    X_val,   Y_val,   Y_val_seq,
    X_test,  Y_test,  Y_test_seq,
    save_path: str = "NNdk_TCN_model.keras",
    epochs: int = 50,
    batch_size: int = 32,
    learning_rate: float = 1e-3,
    hidden: int = 6,
    num_filters: int = 64,
    k_size: int = 5,
    dense_units: int = 128,
    dropout_rate: float = 0.2
):
    """
    Train a TCN model (NNdk) to predict the next (X,Y) from a history of
        past positions.

    Input:
      - X_train, X_val, X_test: shape (n_windows, seq_len, 2) past (X,Y)
      - Y_train, Y_val, Y_test: shape (n_windows, 2) next-step targets
      - Y_*_seq: shape (n_windows, seq_len, 2) full history windows

    Returns:
      - history: Keras History object
      - test_metrics: [test_loss (MSE), test_mae]
    """
    seq_len = X_train.shape[1]
    feature_dim = X_train.shape[2]
    target_dim = Y_train.shape[1]

    print(f"Building TCN with input ({seq_len},{feature_dim}) -> output {
        target_dim}")
    model = model_TCN_simple(
        seq_len=seq_len,
        feature_dim=feature_dim,
        hidden=hidden,
        num_filters=num_filters,
        k_size=k_size,
        dense_units=dense_units,
        output_dim=target_dim,
        dropout_rate=dropout_rate
```

```
    )
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mse', metrics=['mae', tf.
        keras.metrics.RootMeanSquaredError(name='rmse')])
    model.summary()
    es = EarlyStopping(monitor='val_loss', patience=15,
        restore_best_weights=True, mode='min')
    rlp = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
        mode='min')

    print(f"Starting training: epochs={epochs}, batch_size={batch_size}")
    history = model.fit(
        X_train, Y_train,
        validation_data=(X_val, Y_val),
        epochs=epochs,
        batch_size=batch_size,
        callbacks=[es, rlp]
    )
    print("Evaluating on test data...")
    test_metrics = model.evaluate(X_test, Y_test, verbose=0)

    print(f"Test MSE: {test_metrics[0]:.4f}, Test MAE: {test_metrics
        [1]:.4f}, Test RMSE: {test_metrics[2]:.4f}")
    if save_path:
        save_dir = os.path.dirname(save_path)
        os.makedirs(save_dir, exist_ok=True)
        print(f"Saving model to {save_path}...")
        model.save(save_path)
    compute_test(
            model,
            "NNdk_TCN",
            X_train=X_train, Y_train=Y_train,
            X_val=X_val,     Y_val=Y_val,
            X_test=X_test,   Y_test=Y_test,
            path=save_dir,
            trnable_params=model.count_params(),
            nb_filters=num_filters,
            kernel_size=k_size,
            nb_stacks=hidden,
            dense=dense_units,
            hidden=hidden
    )
    return history, test_metrics

def load_trained_model(model_path: str = "NNdk_TCN_model.keras"):
    print(f"Loading model from {model_path}...")
    from tensorflow.keras.models import load_model
    from tensorflow.keras.utils import custom_object_scope
    from tcn_simple import TCN_model
    with custom_object_scope({
        'TCN_model': TCN_model,
    }):
        model = load_model(model_path, compile=False)
```

```
    model.compile(loss='mse', metrics=['mae', tf.keras.metrics.
        RootMeanSquaredError(name='rmse')])

    print("Model loaded.")
    return model
```

Listing 3: NNDK Core