university of koblenz
Mathematics / Natural Sciences

Time Series Classification by Optimal Transport Method

Masterthesis
Mathematical Modeling, Simulation and Optimization

Kishan Sheladiya
Matr.No. 222202236

13.05.2025

First Examiner:	Prof. Dr. Martin. Siebenborn Mathematical Institute Universität Koblenz
Second Examiner:	Dr. Christian. Kahle Mathematical Institute Universität Koblenz

Declaration of authorship

I herewith confirm that I alone have authored this thesis, that I did not use any resources other than those I have cited - in particular no online sources not included in the bibliography section - and that I have not previously submitted this thesis in association with any other examination procedure.

Kishan

Kishan Sheladiya
Date: October 14, 2025

Abstract

Time series classification is an important problem in many fields, ranging from medical diagnostics and sensor monitoring to energy consumption and shape analysis. This thesis investigates the effectiveness of Optimal Transport Warping (OTW) as an alignment-based distance measure for time series classification. While Dynamic Time Warping (DTW) has long been considered a standard method, it often struggles with preserving structural consistency and suffers from high computational cost. OTW, by building on the principles of optimal transport, provides a more flexible and theoretically grounded alternative that is also computationally efficient.

To evaluate the performance of OTW, seven representative datasets from the UCR Time Series Classification Archive were selected, covering domains such as devices, images, medical signals, traffic flow, and energy consumption. A GPU-accelerated implementation of OTW was developed in Python using PyTorch and CUDA, enabling efficient distance computations and repeated experimentation over multiple runs. The results show that OTW often achieves lower error rates than DTW, especially in datasets where structural patterns and global signal characteristics are important (e.g., ArrowHead, InsectEPGSmallTrain, and Chinatown). In other datasets, such as ACSF1, OliveOil, and Car, OTW performs comparably to DTW while maintaining stable results across runs.

Beyond accuracy, OTW also demonstrated significant efficiency advantages. Runtime comparisons confirmed that OTW scales linearly with sequence length, in contrast to DTW's quadratic growth. With GPU acceleration, classification tasks that took many seconds on CPU could be completed in fractions of a second, even for large datasets. Practical implementation challenges, such as GPU memory limitations, were addressed through a sample-by-sample evaluation strategy, further improving feasibility on consumer-grade hardware.

Overall, the findings highlight OTW's potential as a robust, scalable, and adaptable distance measure for real-world time series classification. This work also provides practical insights into implementing OTW efficiently and points toward future research directions, including parameter tuning, differentiable extensions, and integration with deep learning frameworks.

Contents

List of Figures v

List of Tables vi

1 Introduction 1

1.1 Motivation 1
1.2 Objectives of Study 2
1.3 Outline of the Thesis 2

2 Related Work 4

3 Theoretical Background 6

3.1 Optimal Transport 6
3.2 Kantorovich Relaxation 8
3.3 Wasserstein Distance 13
3.4 Optimal Transport Warping 17
3.4.1 Waste Cost OTW 20
3.4.2 Local Windows OTW 21
3.4.3 Loss Smooth OTW 21
3.4.4 Advantages and Applications 22

4 Classification Methods and Complexity Analysis 25

4.1 Introduction to Classification Problems 25
4.2 Multiclass Classification 26
4.3 Common Classification Algorithms 28
4.4 k-Nearest Neighbors (k-NN) 29
4.5 Optimal Transport Warping (OTW) Distance as a Metric 33
4.6 Time Complexity of Classification Algorithms 34

5 Dataset Description 36

5.1 Source: UCR Time Series Classification Archive 36
5.2 Dataset Used 36
5.2.1 Device: ACSF1 37
5.2.2 Image: ArrowHead 39
5.2.3 Spectrography: OliveOil 40
5.2.4 Sensor: Car 41

5.2.5 Medical: InsectEPGSmallTrain 42
5.2.6 Traffic: Chinatown 43
5.2.7 Power Consumption: PowerCons 44
5.3 Train/Test Split and Label Structure 45
5.4 Why this Dataset is Relevant for OTW 45

6 Numerical Experiments **47**
6.1 Implementation Details 47
6.1.1 Python Libraries Used 47
6.1.2 Code Structure 48
6.1.3 GPU Utilization 49
6.2 Experimental Setup 50
6.3 Testing Methodology 50
6.4 Comparison with DTW 51
6.4.1 Performance Table 51
6.5 Discussion and Interpretation of Results 53

7 Conclusion **55**

Bibliography **57**

List of Figures

3.1 Monge's optimal transport problem illustrated as a mapping T from μ to ν. 8
3.2 Connection between discrete and continuous optimal transport. *Left:* the discrete formulation uses a transport matrix Γ with prescribed row/column sums (mass conservation). *Right:* the continuous formulation uses a coupling measure γ that transports mass between densities f_μ and f_ν. 11
3.3 Illustration of the one-dimensional example of $W_1(\mu, \nu) = 0.5$. 16
3.4 Schematic comparison of DTW and OTW. **Left:** DTW uses a dynamic programming grid with $\mathcal{O}(n^2)$ complexity. **Right:** OTW aligns cumulative distributions with $\mathcal{O}(n)$ complexity. 22

4.1 Illustration of k-NN classification (here $k = 5$): the test point x predicts the class by majority vote among its nearest neighbors within the dashed circle. In this example, three neighbors belong to Class B (blue) and two to Class A (red), so the test point is assigned to Class B. 31

5.1 Visualization of ACSF1 dataset classes showing appliance specific consumption patterns. 38
5.2 Visualization of the three ArrowHead classes (Avonlea, Clovis, Mix) as time series representations. 39
5.3 Example of an ArrowHead shape (Clovis) with its corresponding angle based time series encoding. 40
5.4 Visualization of OliveOil dataset: spectrographic time series of four olive oil classes. 40
5.5 Visualization of Car dataset: time series representations of Sedan, Pickup, Minivan, and SUV classes. 41
5.6 InsectEPGSmallTrain dataset: EPG voltage signals for three classes Phloem Feeding, Xylem Feeding, and Non-ingestion/Probing. 42
5.7 Chinatown dataset: hourly pedestrian counts for Weekend (Class 1) and Weekday (Class 2). 43
5.8 PowerCons dataset: seasonal variation in electricity usage profiles for warm and cold seasons. 44

List of Tables

3.1 Mathematical comparison of DTW and OTW in terms of complexity, differentiability, and scalability. 23

5.1 Representative datasets selected from UCR Archive categories 37

6.1 Runtime comparison between CPU and GPU implementations of Test and Train dataset 49

6.2 Runtime comparison between DTW and OTW implementations (over 10 runs). 52

6.3 Comparison of DTW and OTW classification error rates across datasets . 52

1 Introduction

1.1 Motivation

In recent years, Optimal Transport (OT) has gained increasing attention as a powerful mathematical framework for comparing probability distributions [1, 2]. It was originally introduced by Gaspard Monge in 1781 in the context of civil engineering tasks involving the transportation of soil and materials. Later, in 1942, Leonid Kantorovich relaxed the Monge formulation by proposing a more tractable linear programming approach, making it accessible for computational applications. Since then, OT has evolved significantly and found applications in numerous domains such as computer vision, natural language processing, and statistical machine learning [3, 4, 5].

In particular, the ability of OT to measure distances between complex distributions has opened new doors in the field of time series analysis [6, 7]. Traditional methods like Dynamic Time Warping (DTW) have long been used to align sequences of varying lengths [8, 9]. However, DTW suffers from high computational complexity of quadratic order and lacks smoothness or differentiability [10, 11, 12]. In contrast, Optimal Transport (OT), when adapted to time series, offers significant advantages: it defines a true mathematical distance (satisfying non-negativity, symmetry, and triangle inequality), scales more efficiently, and is compatible with GPU-based optimization [1, 13]. These properties make OT-based approaches such as Optimal Transport Warping (OTW) more suitable for modern large-scale applications.

Among these adaptations, Optimal Transport Warping (OTW) stands out as a promising solution for time series classification [14]. Proposed by Latorre et al., OTW integrates the strengths of OT into a warp-based distance that allows better alignment between time series signals [15]. It has been empirically shown to outperform DTW and other baseline methods on benchmark datasets, especially when used with a 1-Nearest Neighbor (1-NN) classifier [6, 7].

The purpose of this thesis is to bridge the gap between the theoretical foundations of Optimal Transport (OT) and Optimal Transport Warping (OTW) and their practical application. While the mathematical underpinnings of OT and OTW are powerful, they are often presented in a dense and inaccessible manner in existing literature, assuming deep familiarity with advanced mathematical concepts. This thesis aims to provide a clear and structured explanation of these methods, focusing on their underlying logic and practical implementation. By offering reproducible experiments and a detailed background, the thesis seeks to make OT and OTW more accessible and applicable to a broader range of researchers, thereby enhancing their utility in time series analysis and other fields.

1.2 Objectives of Study

The main objective of this thesis is to provide a comprehensive and accessible explanation of Optimal Transport (OT) and its application through the Optimal Transport Warping (OTW) method, with a focus on practical implementation in time series classification. Specifically, the thesis aims to explain the historical development of Optimal Transport, from its origins in Monge's and Kantorovich's formulations to an intuitive understanding of the Wasserstein distance, transport plans, and coupling. It introduces OTW as an effective approach for comparing time series data, detailing how it builds on OT to overcome the limitations of Dynamic Time Warping (DTW). The mathematical foundations of OT and OTW are presented step by step, supported by diagrams and example based explanations to ensure clarity.

Furthermore, the thesis implements OTW in Python and evaluates its performance on benchmark datasets, including the UCR Time Series Archive with seven different representative datasets selected from diverse domains. A 1-Nearest-Neighbor (1-NN) classifier is used for classification tasks using OTW distances, with an emphasis on benchmarking OTW's performance in distance based classification. The analysis of the classification results is carried out using quantitative metrics such as precision, mean test error, and 95% confidence intervals, comparing the performance of OTW with DTW to highlight its computational and classification advantages. Finally, the thesis seeks to serve as a reusable educational resource for future researchers by providing clear guidelines for learning and implementing OT and OTW methods in practical applications.

1.3 Outline of the Thesis

This thesis is divided into seven chapters. Chapter 1 introduces the topic, explaining the motivation, objectives, and structure of the thesis. Chapter 2 reviews existing work related to Optimal Transport (OT), time series classification, and OTW, highlighting what has already been done and the contribution of this thesis. Chapter 3 presents the theoretical background, covering the mathematical foundations of OT and the development of OTW, with definitions, key concepts, and examples. Chapter 4 discusses classification methods and complexity analysis, including the 1-Nearest-Neighbor method, evaluation metrics, and the role of OTW as a distance measure. Chapter 5 describes the datasets used for testing OTW, explaining their structure, distribution, and relevance. Chapter 6 details the numerical experiments, including the implementation setup, parameters, results, and comparisons with DTW. Finally, Chapter 7 concludes the thesis by summarizing the key findings, discussing limitations, and suggesting directions for future research.

2 Related Work

Time series classification (TSC) is a foundational problem in machine learning and signal processing [16, 17, 18]. Over the past two decades, numerous algorithms have been developed to address the challenges associated with time dependent, high dimensional data. One of the most enduring methods in this space is Dynamic Time Warping (DTW) [16, 19], which allows elastic alignment between sequences of differing temporal structures. First applied in speech recognition, DTW has since been widely adopted in fields as diverse as finance, healthcare, energy, and environmental monitoring due to its simplicity and effectiveness in aligning sequences with variable temporal patterns [8, 20].

DTW operates by identifying the optimal non linear alignment between two time series, minimizing the cumulative distance between aligned points while adhering to monotonicity and continuity constraints. Despite its strengths, DTW is not without limitations [21, 22, 23]. It is known to be sensitive to noise, can lead to over warping, and does not incorporate global structural constraints. To mitigate these weaknesses, numerous extensions and alternatives have been proposed [24].

One major direction has involved integrating DTW with probabilistic and learning based models, such as Weighted DTW, Derivative DTW, and Learned DTW kernels[21, 24, 25, 26]. Another important development is Soft-DTW, introduced by Cuturi and Blondel, which smooths the DTW cost function [12] to enable gradient based optimization. While Soft-DTW has enabled its use in deep learning frameworks [17], it still retains some core limitations of classical DTW particularly in scenarios where the overall distributional structure of time series needs to be preserved.

An alternative line of research has focused on Optimal Transport (OT) theory [27, 28, 29], which provides a principled way to compare probability distributions based on the cost of transporting one distribution into another. In recent years, OT has been successfully applied in various machine learning tasks [1], including generative modeling, domain adaptation, and clustering. The appeal of OT lies in its ability to capture both local alignment and global geometric structure. Unlike DTW, OT measures cost across all possible mappings between distributions, allowing it to handle broader forms of misalignment and structural deformation.

Within the context of time series, Optimal Transport Warping (OTW) as proposed by Fabian Latorre et al. [14] combines the benefits of DTW and OT [30, 31]. OTW aligns two sequences using a transport plan derived from optimal transport theory, but restricts it to satisfy temporal alignment constraints similar to DTW. It introduces hyperparameters such as transport smoothing (β), cost scaling (m), and warping window (s), which offer flexibility in controlling alignment granularity. Latorre's study showed that OTW could outperform DTW on a range of UCR datasets, particularly in those where class distinctions are better captured through structural similarity than local temporal distortion.

The UCR Time Series Classification Archive plays a central role in evaluating such algorithms [7]. It offers over 100 benchmark datasets from domains such as gesture recognition, electrocardiogram (ECG) analysis, power consumption monitoring, image outline analysis, and spectroscopic measurements. Each dataset varies in length, number of classes, and sampling frequency, making it ideal for comprehensive algorithm testing. Prior work has used UCR to benchmark a variety of methods, from shapelets [32, 33] and bag of patterns [34] to ensemble classifiers [18, 35] and deep learning models [36, 37]. However, alignment-based distance measures especially DTW and OTW remain competitive due to their interpretability and effectiveness in small sample scenarios.

Several recent studies have compared OTW and DTW under different setups [30, 31]. For instance, OTW has shown robustness in datasets involving geometric outlines (e.g., ArrowHead) and bioelectrical signals (e.g., InsectEPG), where global patterns are more discriminative than localized fluctuations. DTW tends to retain advantages in highly repetitive or spike dominant signals, such as appliance power traces or energy peaks. These findings suggest that no single method universally dominates, but that the domain characteristics and data distribution play a critical role in choosing the right distance measure.

Finally, research in this area has also explored the computational challenges associated with alignment methods. Traditional DTW has a quadratic time complexity with respect to sequence length [16], which becomes computationally expensive for large datasets or real time applications. OTW, while more flexible, is computationally heavier due to the transport plan optimization [27, 38]. As a result, researchers have proposed GPU accelerated implementations, low Rank approximations, and hybrid models to make OT based warping more scalable [29, 28, 27].

This thesis extends these works by offering a GPU based implementation of OTW, evaluating its performance across seven representative UCR datasets chosen from different application categories. It emphasizes repeated experimentation to ensure statistical robustness and compares classification accuracy against benchmark DTW results. The findings reinforce the growing consensus that OTW is a strong alternative to DTW in scenarios that demand structural fidelity, alignment stability, and interpretability.

3 Theoretical Background

This chapter lays out the key theoretical concepts that are important for understanding the rest of the thesis. It starts with the idea of *Optimal Transport (OT)*, a mathematical framework that was first introduced in the 18th century and has since become useful in many areas like computer science, machine learning, and especially time series analysis. We begin by explaining the original OT problem as proposed by Monge, including its purpose and the difficulties in solving it. Then we move on to the *Kantorovich relaxation*, which made the OT problem easier to work with and more widely applicable. Next, we introduce the *Wasserstein distance*, a way to measure how different two distributions are, which comes directly from OT theory. Finally, we focus on *Optimal Transport Warping (OTW)*, a method designed to align time series data more effectively than traditional techniques like Dynamic Time Warping (DTW). Throughout the chapter, we explain each concept with clear definitions, visual examples, and real-world applications, giving you the background needed for the methods used later in the thesis.

3.1 Optimal Transport

Optimal Transport (OT) is a mathematical theory that focuses on finding the most efficient way to move a distribution of mass (or probability) from one configuration to another while minimizing a cost [1, 39]. At its core, the OT problem asks: given two distributions, what is the least costly way to transform one into the other? The concept combines elements from optimization, geometry, and probability, and has proven highly useful in many fields [40].

The general idea of OT can be described using a transportation metaphor: imagine a pile of sand that needs to be reshaped into a new form. The goal is to determine the most efficient way to move the grains from their original positions to new ones, where "efficiency" is defined in terms of some cost function, such as distance. Formally, given two probability measures μ and ν, defined on measurable spaces, OT seeks a map or plan that transports μ to ν with minimal total cost.

OT is important not just as a mathematical curiosity, but because it provides a powerful way to compare complex objects like images, shapes, and time series. In computer science, it's used for image recognition, shape matching, and domain adaptation in machine learning [41, 4]. In the context of time series analysis, OT offers a framework to align temporal sequences based on structural similarity rather than just point-wise differences, making it more robust to distortions and shifts.

The theory of Optimal Transport (OT) originates from the work of Gaspard Monge in 1781, who formulated the problem in the context of civil engineering: how to move soil (or other material) from one place to another with the least effort. Although motivated by practical concerns of labor and resource optimization, the mathematical formulation proved deeply complex and has since evolved into a rich area of research.

Over time, OT has found applications far beyond its original purpose. In economics, it models optimal allocation of resources. In physics, it helps describe equilibrium states. In biology, it aids in comparing genetic expressions. In machine learning, it has become an essential tool for comparing probability distributions in generative models and clustering. In computer vision, OT is used to compare images by treating intensity distributions as probability measures. In natural language processing, OT is used to align word embeddings, capturing semantic similarity more accurately than simple vector distances. In time series analysis, OT helps to align and compare temporal sequences that may differ in speed or sampling by considering the global structure of the series rather than relying on point-to-point distances. This is particularly useful in areas like speech recognition, financial trend analysis, and medical signal processing.

The formal mathematical formulation of Monge's problem can be described as follows: let μ and ν be two probability measures defined on measurable spaces X and Y respectively. Let $c(x, y)$ be the cost function representing the cost of transporting a unit mass from point $x \in X$ to point $y \in Y$. Monge's objective is to find a transport map $T : X \to Y$ that pushes μ to ν (denoted $T_{\#}\mu = \nu$) while minimizing the total cost:

$$\min_{T} \left\{ \sum_{i} c(x_i, T(x_i)) \ : \ T_{\#}\mu = \nu \right\} \tag{3.1}$$

This is known as Monge's formulation of the OT problem. The core components include the source distribution μ, which represents the initial configuration of mass or probability, and the target distribution ν, which is the desired configuration to which the source should be transformed. The cost function $c(x, y)$ specifies the effort of moving mass from point x to point y, with the Euclidean distance $c(x, y) = \|x - y\|^p$ for $p \geq 1$ being a common choice. Finally, the transport map T is a measurable function that relocates mass from the source to the target in a way that exactly matches the distribution ν.

This schematic visualization helps illustrate how OT seeks the most efficient transport map T that minimizes the overall cost of moving mass.

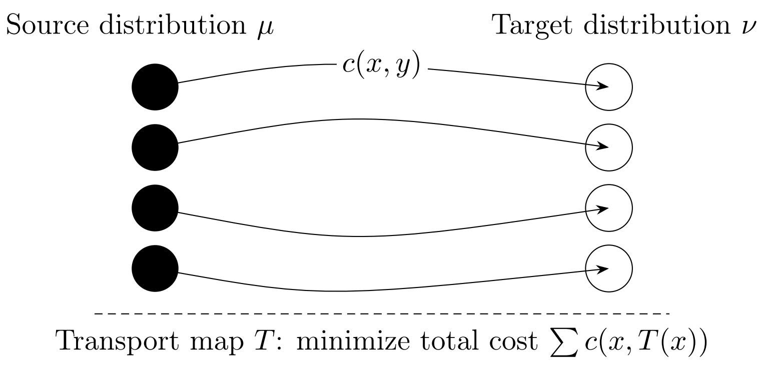
Source distribution μ
Target distribution ν
$c(x, y)$
Transport map T: minimize total cost $\sum c(x, T(x))$

Figure 3.1: Monge's optimal transport problem illustrated as a mapping T from μ to ν.

To aid visual understanding, consider the following illustrative figure (which can be generated if needed): two discrete distributions represented as piles of sand and holes. The OT task is to find the most efficient way to fill the holes using the sand piles, considering the distance or cost of each potential movement.

However, Monge's formulation poses several mathematical challenges [42, 43]. First, a transport map T may not always exist. For example, if the source and target distributions are discrete but not of the same cardinality, no such deterministic map can satisfy the condition $T_{\#}\mu = \nu$. Additionally, the problem is highly non-convex due to the functional form of the map T, making it difficult to solve using standard optimization techniques. These issues limit the practical applicability of Monge's original approach [44, 45], especially in high-dimensional and irregular data settings.

3.2 Kantorovich Relaxation

Monge's original formulation of the optimal transport (OT) problem is elegant but mathematically rigid. One of its major limitations is that it requires the existence of a transport map T that deterministically assigns each point in the source distribution to exactly one point in the target distribution. This deterministic requirement is often too restrictive in real-world problems, particularly when the source and target distributions are not perfectly aligned or are discrete and unbalanced. Moreover, Monge's formulation is non-convex, making it difficult to analyze and computationally challenging to solve, especially in high-dimensional spaces or when working with irregular distributions. These difficulties motivated researchers to seek a more flexible and solvable alternative.

The *Kantorovich relaxation*, introduced by the Russian mathematician Leonid Kantorovich in 1942, addressed these shortcomings. Instead of seeking a deterministic mapping T, Kantorovich proposed allowing *probabilistic transport plans*—that is, a plan that distributes mass from a source to multiple destinations in a fractional way. This broader formulation turned the problem into a linear programming problem, which made it more tractable both theoretically and computationally. Kantorovich's innovation earned him the Nobel Prize in Economics in 1975 due to its foundational role in resource allocation and optimization theory.

There are several fundamental differences between the Monge and Kantorovich formulations of the OT problem. First, Monge seeks a transport map $T(x)$ that assigns each $x \in X$ to a single $y \in Y$, whereas Kantorovich introduces a transport plan $\gamma(x, y)$ that allows splitting the mass at x among multiple destinations y. Second, Monge's formulation is fully deterministic, while Kantorovich's approach is more flexible, as it models transport as a joint distribution over $X \times Y$. Finally, Monge's formulation often fails to provide a solution when no bijective mapping exists between the measures, whereas Kantorovich's relaxation guarantees the existence of a solution because the admissible set of transport plans is convex.

From Monge (discrete) to Kantorovich (discrete)

Consider finite supports $X = \{x_i\}_{i=1}^m$ and $Y = \{y_j\}_{j=1}^n$, with probability vectors $\mu = (\mu_1, \ldots, \mu_m)$ and $\nu = (\nu_1, \ldots, \nu_n)$ satisfying $\mu_i \geq 0$, $\nu_j \geq 0$ and $\sum_{i=1}^m \mu_i = \sum_{j=1}^n \nu_j = 1$. Let $C \in \mathbb{R}^{m \times n}$ collect the pairwise costs $C_{ij} := c(x_i, y_j)$. Monge's discrete problem searches for a pushforward map $T : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that the mass constraints are exactly met,

$$\sum_{i:\, T(i)=j} \mu_i \; = \; \nu_j \quad \text{for all } j, \tag{3.2}$$

and minimizes the total cost

$$\mathcal{M}_c(\mu, \nu) \; := \; \inf_{T \text{ s.t. } (3.2)} \sum_{i=1}^m C_{i\,T(i)}\, \mu_i. \tag{3.3}$$

The pushforward constraint (3.2) encodes that each atom x_i ships all its mass μ_i to a single destination $y_{T(i)}$, and that destination capacities ν_j are exactly filled. This feasibility may fail if the $\{\mu_i\}$ do not partition the $\{\nu_j\}$ into exact sums (hence the potential nonexistence of T in discrete settings).

Every feasible map T induces a transport plan (a coupling) $\Gamma^T = [\gamma_{ij}^T] \in \mathbb{R}_{\geq 0}^{m \times n}$ defined by

$$\gamma_{ij}^T \; = \begin{cases} \mu_i, & j = T(i), \\ 0, & j \neq T(i), \end{cases} \qquad \Longrightarrow \qquad \sum_{j=1}^n \gamma_{ij}^T = \mu_i, \;\; \sum_{i=1}^m \gamma_{ij}^T = \nu_j, \tag{3.4}$$

where the column sums equal ν_j precisely because of (3.2). The cost of T rewrites as

$$\sum_{i=1}^{m} C_{i\,T(i)}\,\mu_i \;=\; \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}\,\gamma_{ij}^{T}. \tag{3.5}$$

Monge's problem (3.3) can therefore be seen as minimizing the linear cost $\langle C, \Gamma\rangle$ over the very restrictive subset of couplings for which each row has exactly one nonzero entry, equal to μ_i.

Kantorovich's idea is to *relax* the graph constraint and to allow any non-negative matrix Γ whose row and column sums match (μ, ν):

$$\begin{aligned} \mathcal{K}_c(\mu,\nu) \;:=\; & \min_{\Gamma\in\mathbb{R}_{\geq 0}^{m\times n}} \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}\,\gamma_{ij} \\ \text{s.t.}\quad & \sum_{j=1}^{n}\gamma_{ij} = \mu_i, \quad i = 1,\dots,m, \\ & \sum_{i=1}^{m}\gamma_{ij} = \nu_j, \quad j = 1,\dots,n. \end{aligned} \tag{3.6}$$

This is the classical transportation *linear program.* Because every map induced plan Γ^T in (3.4) is feasible for (3.6), we have the relaxation inequality

$$\mathcal{K}_c(\mu,\nu) \;\leq\; \mathcal{M}_c(\mu,\nu). \tag{3.7}$$

Equality holds whenever a feasible optimal Monge map exists; otherwise (3.6) still admits a solution and properly extends Monge's formulation by allowing mass splitting across several destinations.

It is immediate that the feasible set in (3.6) is a nonempty polytope (the *transportation polytope*): nonemptiness follows, for instance, by taking any feasible basic solution or, analytically, by viewing $\mu\otimes\nu$ as a coupling in the measure-theoretic setting. Compactness of this polytope and linearity of the objective imply that the minimum is attained. Hence the discrete Kantorovich problem is always solvable, while the discrete Monge problem may be infeasible unless (3.2) admits a solution.

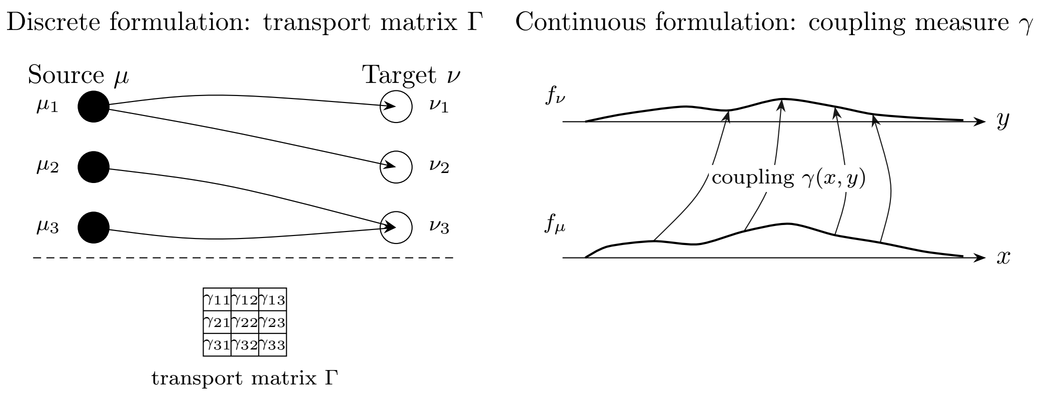
Discrete formulation: transport matrix Γ
Continuous formulation: coupling measure γ
Source μ
Target ν
μ_1
μ_2
μ_3
ν_1
ν_2
ν_3
γ_{11} γ_{12} γ_{13}
γ_{21} γ_{22} γ_{23}
γ_{31} γ_{32} γ_{33}
transport matrix Γ
f_ν
y
coupling $\gamma(x, y)$
f_μ
x

Figure 3.2: Connection between discrete and continuous optimal transport. *Left:* the discrete formulation uses a transport matrix Γ with prescribed row/column sums (mass conservation). *Right:* the continuous formulation uses a coupling measure γ that transports mass between densities f_μ and f_ν.

From the discrete LP to the continuous density formulation

The discrete model above is a special case of the measure-theoretic formulation. If μ and ν are discrete,

$$\mu = \sum_{i=1}^{m} \mu_i \, \delta_{x_i}, \qquad \nu = \sum_{j=1}^{n} \nu_j \, \delta_{y_j},$$

then any coupling γ between μ and ν is itself discrete on $X \times Y$,

$$\gamma = \sum_{i=1}^{m} \sum_{j=1}^{n} \gamma_{ij} \, \delta_{(x_i, y_j)}, \qquad \gamma_{ij} \geq 0, \qquad \sum_{j=1}^{n} \gamma_{ij} = \mu_i, \quad \sum_{i=1}^{m} \gamma_{ij} = \nu_j,$$

and the integral of the cost reduces to the discrete sum

$$\int_{X \times Y} c(x, y) \, d\gamma(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} c(x_i, y_j) \, \gamma_{ij}. \tag{3.8}$$

Conversely, for general probability measures μ on $(X, \mathcal{X})$ and ν on $(Y, \mathcal{Y})$, a *transport plan* is any probability measure γ on $(X \times Y, \mathcal{X} \otimes \mathcal{Y})$ with the prescribed marginals,

$$\gamma(A \times Y) = \mu(A), \qquad \gamma(X \times B) = \nu(B) \quad \text{for all measurable } A \subseteq X, \; B \subseteq Y. \tag{3.9}$$

The Kantorovich problem becomes

$$\inf_{\gamma \in \Pi(\mu, \nu)} \int_{X \times Y} c(x, y) \, d\gamma(x, y), \qquad \Pi(\mu, \nu) := \{\gamma \geq 0 : \text{ (3.9) holds}\}. \tag{3.10}$$

When μ and ν admit densities (with respect to Lebesgue) and γ is absolutely continuous with density $g(x,y)$, the marginal constraints (3.9) read in density form

$$\int_Y g(x,y)\,dy = f_\mu(x), \qquad \int_X g(x,y)\,dx = f_\nu(y), \tag{3.11}$$

and the cost is the double integral $\iint c(x,y)\,g(x,y)\,dx\,dy$. In the purely discrete case, (3.10) reduces to (3.6) by the identifications above; in the mixed or continuous cases, sums become integrals and the same structure persists.

To make the link between the discrete and continuous formulations more precise, one can approximate any probability measure by discrete (empirical) measures supported on finer and finer partitions of X and Y. Under mild technical assumptions namely, that the cost function c is bounded from below and lower semicontinuous the functional

$$\gamma \;\mapsto\; \int c(x,y)\,d\gamma(x,y)$$

is continuous with respect to weak-* convergence of measures. In addition, the admissible set of couplings $\Pi(\mu,\nu)$ is compact and convex. These properties ensure, by the direct method of the calculus of variations, that the infimum in (3.10) is always achieved. Furthermore, whenever an optimal Monge map exists (for instance, in the quadratic cost case under convexity and regularity conditions), the solution of the Kantorovich problem coincides with the Monge solution, and the optimal plan is supported entirely on the graph of the optimal transport map.

One of the most significant advantages of Kantorovich's formulation is *convexity* [40]. The set of admissible transport plans $\Pi(\mu,\nu)$ is convex, and the cost functional $\int c(x,y)\,d\gamma(x,y)$ is linear in γ. This makes the optimization problem a *linear program*, which is well studied and can be solved efficiently using a wide range of numerical techniques [13, 46]. Furthermore, under mild assumptions such as the cost function $c(x,y)$ being lower semicontinuous, an optimal transport plan is guaranteed to exist [2].

Compared to Monge's formulation, Kantorovich's approach is not only more flexible and solvable, but also mathematically richer. While Monge's problem fails when there is no bijective map between the source and target measures, Kantorovich's formulation handles mass splitting and unbalanced distributions naturally. It also laid the groundwork for the development of *Wasserstein distances* [47], which rely on this relaxed notion of transportation and play a crucial role in many modern applications.

Mathematically, the robustness of Kantorovich's formulation follows from the fact that the set of admissible couplings

$$\Pi(\mu,\nu) = \big\{\gamma \in \mathcal{P}(X\times Y) : \gamma(A\times Y) = \mu(A),\ \gamma(X\times B) = \nu(B)\big\}$$

is always non-empty, convex, and compact in the weak-* topology. For example, the product measure $\mu \otimes \nu$ is a valid coupling, so $\Pi(\mu, \nu) \neq \emptyset$. Moreover, the cost functional

$$\mathcal{C}(\gamma) = \int_{X \times Y} c(x, y) \, d\gamma(x, y)$$

is linear and therefore continuous with respect to γ. By the extreme value theorem, the infimum in (3.10) is attained, ensuring the existence of an optimal transport plan $\gamma^\star \in \Pi(\mu, \nu)$. This demonstrates formally why Kantorovich's problem always has a solution, even in cases where Monge's map does not exist.

To visualize this difference, consider the image generated earlier: Monge's setup requires a strict one-to-one assignment, while Kantorovich allows for a more realistic model where a portion of one sandpile might fill several holes. This makes Kantorovich's model more suited to problems where granular allocation is necessary, such as in logistics, probability theory, and signal matching.

Modern applications of Kantorovich's optimal transport are vast and growing. In machine learning, OT is used in generative adversarial networks (GANs), particularly *Wasserstein GANs* [5], where the transport cost between data distributions provides a more stable training objective. In computer vision, it helps in color and texture transfer by matching pixel distributions. In natural language processing, OT is used to compare distributions of word embeddings or topic models [48]. In economics, it models resource allocation problems more realistically than Monge's deterministic framework. In biology, it is used for aligning distributions of gene expression or cellular structures.

Overall, Kantorovich relaxation not only made the optimal transport problem computationally feasible but also extended its reach to a wide array of complex, high dimensional problems that arise in real-world data analysis and scientific modeling.

3.3 Wasserstein Distance

As the theory of Optimal Transport (OT) developed, researchers recognized the need for a meaningful way to measure the distance between probability distributions. Traditional divergence measures like Kullback Leibler (KL) divergence or Jensen Shannon (JS) divergence are commonly used in information theory, but they suffer from several limitations: they are not true metrics, may not be symmetric, and often become undefined [49] when the distributions do not share support. These shortcomings are particularly problematic in applications involving structured data like images, text, and time series, where even slight shifts in support can mislead traditional metrics.

This is where the *Wasserstein distance* comes in a concept that originates directly from the Kantorovich formulation of optimal transport [2]. The motivation behind the Wasserstein distance is to capture not just how much two distributions differ, but also how far one distribution must move to become the other, given a cost function. This notion of distance is more aligned with the underlying geometry of the space and provides better interpretability in practical problems, such as comparing shapes or measuring fairness in algorithmic decisions.

The concept of Wasserstein distance, also known as the Kantorovich–Rubinstein metric, was developed in the mid 20th century, building on the work of Leonid Kantorovich. The distance gained significant traction after formalization by mathematicians including L.V. Kantorovich (1942), and later R.L. Dobrushin. The name "Wasserstein" honors the Russian mathematician Leonid Vaseršteĭn (Wasserstein), who contributed to the development of metrics in probabilistic settings.

Unlike many traditional distances that only look at point wise differences in probability mass, the Wasserstein distance considers the spatial arrangement of mass. This allows it to be sensitive to the geometry of the underlying space. For example, if two probability distributions are nearly identical but one is slightly shifted, the KL divergence may report them as highly dissimilar, while the Wasserstein distance will reflect the small, smooth transformation between them. This geometric intuition is especially valuable in areas such as image comparison, time series alignment, and generative modeling.

The Wasserstein distance arises naturally from the Kantorovich formulation of optimal transport. For two probability measures μ and ν defined on a metric space (X, d), the p-Wasserstein distance for $p \geq 1$ [2, 40] is defined as:

$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Pi(\mu,\nu)} \int_{X \times Y} d(x, y)^p \, d\gamma(x, y) \right)^{1/p} \tag{3.12}$$

Here, $\Pi(\mu, \nu)$ is the set of all couplings (joint distributions) with marginals μ and ν, and $d(x, y)$ is the cost function or ground metric (often the Euclidean distance). The p-th root ensures the distance is in the same units as the ground metric. The core components of the Wasserstein distance can be summarized as follows. The probability measures μ and ν represent the two distributions being compared. The ground metric $d(x, y)$, typically chosen as a norm such as the Euclidean distance, quantifies the cost of transporting mass from point x to point y. The coupling $\gamma \in \Pi(\mu, \nu)$ specifies how mass is transported probabilistically from the source to the target. Finally, the exponent p determines the sensitivity of the distance to large displacements; for instance, $p = 1$ is less sensitive to outliers than $p = 2$.

This formulation defines a true metric on the space of probability distributions [47] with finite p-th moments. It satisfies all properties of a metric:

$$\text{Non-negativity: } W_p(\mu,\nu) \geq 0, \tag{3.13}$$

$$\text{Identity of indiscernibles: } W_p(\mu,\nu) = 0 \iff \mu = \nu, \tag{3.14}$$

$$\text{Symmetry: } W_p(\mu,\nu) = W_p(\nu,\mu), \tag{3.15}$$

$$\text{Triangle inequality: } W_p(\mu,\lambda) \leq W_p(\mu,\nu) + W_p(\nu,\lambda). \tag{3.16}$$

Special Cases

For $p = 1$, the Wasserstein distance becomes

$$W_1(\mu,\nu) = \inf_{\gamma\in\Pi(\mu,\nu)} \int_{X\times Y} d(x,y)\, d\gamma(x,y), \tag{3.17}$$

which is also known as the *Earth Mover's Distance (EMD)* because it measures the minimal work required to transform one distribution into another.

For $p = 2$, the Wasserstein distance is

$$W_2(\mu,\nu) = \left(\inf_{\gamma\in\Pi(\mu,\nu)} \int_{X\times Y} \|x-y\|^2\, d\gamma(x,y) \right)^{1/2}, \tag{3.18}$$

which is widely used in applications because of its connection to geometry and variance-like interpretations.

Dual Formulation

An important theoretical result is the Kantorovich–Rubinstein duality, which states that

$$W_1(\mu,\nu) = \sup_{\|\varphi\|_{\mathrm{Lip}}\leq 1} \left(\int_X \varphi(x)\, d\mu(x) - \int_Y \varphi(y)\, d\nu(y) \right), \tag{3.19}$$

where the supremum is taken over all 1-Lipschitz functions φ. This dual formulation connects optimal transport with functional analysis and provides alternative computational approaches.

Example in One Dimension

Consider two discrete probability distributions on $\mathbb{R}$:

$$\mu = (0.5, 0.5) \quad \text{at points } \{0,1\}, \qquad \nu = (0.0, 1.0) \quad \text{at points } \{0,1\}.$$

Here μ places equal mass at $x = 0$ and $x = 1$, while ν places all mass at $y = 1$. Using the definition of W_1, the optimal plan moves 0.5 mass from $x = 0$ to $y = 1$ at cost $|0-1| = 1$, while the other 0.5 mass at $x = 1$ remains in place at zero cost. Therefore,

$$W_1(\mu, \nu) = 0.5 \cdot 1 + 0.5 \cdot 0 = 0.5.$$

This simple example illustrates how the Wasserstein distance captures the geometry of probability distributions, unlike KL divergence which would be undefined in this case.

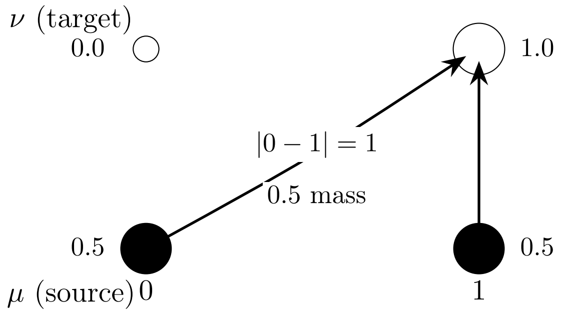

Figure 3.3: Illustration of the one-dimensional example of $W_1(\mu, \nu) = 0.5$.

Geometric Interpretation

The geometric interpretation of the Wasserstein distance is often described via the *Earth Mover's Distance (EMD)* [50, 51]. Imagine you have a pile of earth (mass) and a set of holes (locations) that need to be filled. The EMD tells you the minimum total effort required to move the earth into the holes, where "effort" is quantified as the amount of mass times the distance it travels. This analogy captures both the quantity of mass and the spatial rearrangement needed, making the Wasserstein distance especially powerful in geometric and spatial data contexts.

For example, in image retrieval, comparing histograms using EMD provides results that better match human perception. In time series, it allows alignment of data with phase shifts or warping. In machine learning, Wasserstein distances are used in Wasserstein GANs to provide a stable and meaningful loss for comparing generated and real distributions.

Note. Although the Wasserstein distance is not directly applied in the experiments of this thesis, I chose to include this section because it forms the theoretical foundation of Optimal Transport. OTW can be understood as a practical adaptation of these concepts to time series alignment, and the Wasserstein distance provides the mathematical framework for comparing probability distributions in a principled way. Including this section helped me to better understand the link between the theory of optimal transport and its application in time series, and I wanted the thesis to present a self-contained overview of these foundations even if not all of them are used explicitly in later chapters.

3.4 Optimal Transport Warping

Time series data is ubiquitous in science and industry, appearing in fields such as finance, medicine, environmental monitoring, and beyond. A critical challenge in analyzing time series is the problem of *alignment*: matching patterns or features across sequences that may be shifted, stretched, or compressed in time. Traditionally, *Dynamic Time Warping (DTW)* has been the most widely used technique for this purpose. While DTW effectively aligns non-linear temporal distortions, it has several limitations.

Formally, DTW between two sequences $a = (a_1, \dots, a_n)$ and $b = (b_1, \dots, b_m)$ is defined as

$$DTW(a, b) = \min_{\pi \in \mathcal{A}} \sum_{(i,j) \in \pi} d(a_i, b_j), \tag{3.20}$$

where π is a warping path subject to boundary, monotonicity, and continuity constraints. The presence of the min operator over admissible paths π makes the function piecewise constant with respect to the inputs: small perturbations in a or b may switch the optimal path abruptly. This leads to set-valued gradients and discontinuities, implying that DTW is non-differentiable almost everywhere and thus unsuitable for gradient-based learning methods.

Furthermore, DTW is typically computed using dynamic programming with the recurrence

$$D(i, j) = d(a_i, b_j) + \min\{D(i-1, j),\ D(i, j-1),\ D(i-1, j-1)\}, \tag{3.21}$$

[Complexity of DTW] The dynamic programming evaluation of (3.21) for DTW runs in time $\Theta(nm)$ and uses space $\Theta(nm)$. In the common case $n = m$, this is $\Theta(n^2)$ time and space.

Proof. The DTW algorithm fills a dynamic programming (DP) table $D(i, j)$ of size $n \times m$, where each entry stores the cost of aligning the first i elements of a with the first j elements of b. The table is defined by the recurrence

$$D(i, j) = d(a_i, b_j) + \min\{D(i-1, j),\ D(i, j-1),\ D(i-1, j-1)\},$$

with initialization $D(0,0) = 0$, $D(i,0) = \infty$, and $D(0,j) = \infty$ for $i, j > 0$.

Time complexity. There are $n \times m$ entries in the DP table. Computing each $D(i,j)$ requires: one distance evaluation $d(a_i, b_j)$, one addition, and two comparisons (to compute the minimum of three values).

Thus each entry requires a constant amount of work, say c operations. The total work is therefore

$$T(n,m) = c \cdot (n \cdot m) = O(nm).$$

In the special case $n = m$, this becomes

$$T(n) = O(n^2).$$

Space complexity. To recover an optimal warping path, the entire DP table must be stored. The number of cells is

$$S(n,m) = (n+1)(m+1) = O(nm),$$

which reduces to $S(n) = O(n^2)$ when $n = m$.

Hence, DTW has quadratic time and space complexity for sequences of equal length. □

To address these challenges, *Optimal Transport Warping (OTW)* was proposed by Fabian Latorre in the paper "OTW: Optimal Transport Warping for Time Series" [14]. OTW leverages the mathematical framework of optimal transport to align time series data in a way that is not only accurate and structure-aware but also computationally efficient. By modeling the alignment as a transportation problem between time-indexed distributions, OTW introduces a smooth and flexible alternative to DTW that is more suitable for modern machine learning pipelines.

There are several key differences between OTW and DTW that make OTW advantageous in many contexts:

- **Complexity**: While DTW has a time complexity of $\mathcal{O}(n^2)$ due to its dynamic programming nature, OTW is designed with linear time complexity under certain assumptions [14], making it scalable for long sequences.
- **Differentiability**: DTW is non-differentiable, which prevents it from being used in gradient-based optimization. OTW, in contrast, is formulated in a way that permits gradients, allowing seamless integration into neural networks and optimization frameworks.

- **Theoretical Foundation**: DTW is heuristic and lacks a strong probabilistic interpretation. OTW is grounded in the theory of optimal transport, providing it with a principled and flexible foundation for alignment based on distributional matching.

The formal definition of Optimal Transport Warping (OTW) frames the alignment of time series as an optimal transport problem between temporal distributions. Let $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$ be two time series with non-negative values. We treat them as discrete measures and seek a transport plan $T \in \mathbb{R}^{n \times n}$ that minimizes the cost of aligning a with b under a distance matrix D, usually defined by temporal distance $D_{i,j} = |i - j|$.

The classical OT problem for time series in this context is:

$$W_D(a,b) = \min_{T \geq 0} \left\{ \langle T, D \rangle : T\mathbf{1} = a,\ T^\top \mathbf{1} = b \right\} \tag{3.22}$$

Here, $\langle T, D \rangle$ is the total cost of transporting mass from a to b, and $\mathbf{1}$ is the vector of ones. This setup ensures that the row and column sums of the transport plan T match the input time series.

The core components of 3.22 can be described as follows. Two time series a and b are modeled as non-negative vectors that can be interpreted as empirical distributions. The alignment cost is captured by a distance matrix D, where each entry $D_{i,j} = |i-j|$ encodes the temporal misalignment penalty between index i in a and index j in b. The transport plan T is a non-negative matrix that specifies how much mass is transported from each a_i to each b_j. Finally, the formulation includes constraints that ensure mass conservation in the balanced case, or modifications to the feasible set when working with unbalanced transport problems.

In order to accommodate real-world time series which often do not have the same total mass (unbalanced) or contain negative values Latorre et al. introduced *Unbalanced Optimal Transport* and a *sink node* approach [15, 52]. This leads to an upper bound that can be computed in linear time:

$$\mathrm{OTW}_m(a,b) = m\,|A(n) - B(n)| + \sum_{i=1}^{n-1} |A(i) - B(i)| \tag{3.23}$$

where $A(i) = \sum_{j=1}^{i} a_j$ and $B(i) = \sum_{j=1}^{i} b_j$ are the cumulative sums of the time series. This version handles mass mismatch via the “sink” node and introduces a waste cost m, allowing flexible alignment even in noisy or imperfect settings.

The computational complexity of (3.23) is linear in the sequence length. This is because computing all cumulative sums $A(i)$ and $B(i)$ requires $O(n)$ operations, and evaluating the differences $|A(i) - B(i)|$ for each index $i = 1, \ldots, n$ also requires $O(n)$ operations. Therefore, the total cost of evaluating $\mathrm{OTW}_m(a, b)$ is

$$T(n) = O(n) + O(n) = O(n),$$

both in time and memory usage. This contrasts with DTW, which needs $O(n^2)$ time and space due to filling an $n \times n$ dynamic programming table.

The formulation in (3.23) offers several important advantages. It is differentiable, which enables seamless integration into neural network architectures, and it preserves linear time and space complexity, making it highly scalable to long sequences. Moreover, the method admits a closed-form and interpretable structure through cumulative distribution functions, providing both theoretical clarity and practical efficiency. As a result, this formalism allows OTW to retain the geometric and probabilistic elegance of optimal transport while remaining feasible and efficient for large-scale time series analysis and deep learning tasks.

To enhance the flexibility and applicability of OTW in various time series tasks, Latorre et al. introduced several variants and regularizations that adapt the original OT formulation to more realistic and diverse scenarios. These extensions address issues such as mismatched mass, local alignment, and the need for smooth gradients in optimization.

3.4.1 Waste Cost OTW

One limitation of standard OT is the assumption that both time series must have equal total mass. In practical settings, this is rarely the case. The *Waste Cost OTW* addresses this by introducing an extra "sink" node that can absorb or supply excess mass at a fixed cost m. Given two sequences a and b, the unbalanced OT formulation is:

$$\hat{W}_m(a, b) = \min_{T \geq 0} \left\{ \langle T, D^{(m)} \rangle : T\mathbf{1} = \hat{a},\ T^\top \mathbf{1} = \hat{b} \right\} \tag{3.24}$$

Here, $\hat{a}, \hat{b}$ are the extended vectors including the sink node, and $D^{(m)}$ is the extended cost matrix where all transport to/from the sink incurs a fixed cost m. The resulting closed-form upper bound:

$$\mathrm{OTW}_m(a, b) = m\,|A(n) - B(n)| + \sum_{i=1}^{n-1} |A(i) - B(i)| \tag{3.25}$$

where $A(i) = \sum_{j=1}^{i} a_j$ and $B(i) = \sum_{j=1}^{i} b_j$ are the cumulative sums of the time series. This formulation ensures that the alignment remains computationally efficient while handling differences in total mass linearly.

3.4.2 Local Windows OTW

While the global OTW captures broad misalignment, in many applications, *local alignment* is more relevant for instance, in speech recognition or gesture tracking where features tend to evolve locally. Inspired by the Sakoe–Chiba band constraint in DTW, *Local OTW* introduces a window parameter s that restricts transport to nearby time indices. This is done by modifying the cumulative functions:

$$A_s(i) = \sum_{j=i-s+1}^{i} a_j, \quad B_s(i) = \sum_{j=i-s+1}^{i} b_j$$

The local OTW formulation becomes:

$$\mathrm{OTW}_{m,s}(a,b) = m\,|A_s(n) - B_s(n)| + \sum_{i=1}^{n-1} |A_s(i) - B_s(i)| \tag{3.26}$$

This variant interpolates between the ℓ_1 distance (when $s = 1$) and the full unbalanced OTW (when $s = n$), allowing tailored locality in matching.

3.4.3 Loss Smooth OTW

For integration into gradient-based optimization, the non-differentiability of the absolute value function in OTW poses a problem. To overcome this, a smooth approximation is introduced using the *Huber loss* or smooth ℓ_1-loss [53]:

$$\mathcal{L}_\beta(x) = \begin{cases} \frac{x^2}{2\beta}, & \text{if } |x| < \beta \\ |x| - \frac{\beta}{2}, & \text{otherwise} \end{cases} \tag{3.27}$$

The smooth OTW distance becomes:

$$\mathrm{OTW}^{\beta}_{m,s}(a,b) = m\mathcal{L}_{\beta}(A_s(n) - B_s(n)) + \sum_{i=1}^{n-1} \mathcal{L}_{\beta}(A_s(i) - B_s(i)) \tag{3.28}$$

This formulation enables full differentiability, essential for deep learning pipelines, and maintains linear complexity using efficient cumulative sum operations.

3.4.4 Advantages and Applications

Each of these variants waste cost handling, local warping constraints, and smooth loss, expands the flexibility of OTW, enabling it to match or outperform DTW in accuracy [14] while maintaining significant computational and architectural advantages.

One of the most compelling features of OTW is its computational efficiency. As shown in Section 3.4, DTW requires filling an $n \times m$ dynamic programming table according to the recurrence in (3.21), which leads to a quadratic time and space complexity $\mathcal{O}(n^2)$ when $n = m$. In contrast, the OTW formulation in (3.23) is evaluated directly through cumulative sums of the sequences, which can be computed in a single pass, yielding linear time and space complexity $\mathcal{O}(n)$. This fundamental difference explains why OTW scales much better than DTW for long sequences. Moreover, with the incorporation of a sink node for mass imbalance [15] and optional local windows [14], the algorithm remains efficient and scalable even when extended with more realistic assumptions.

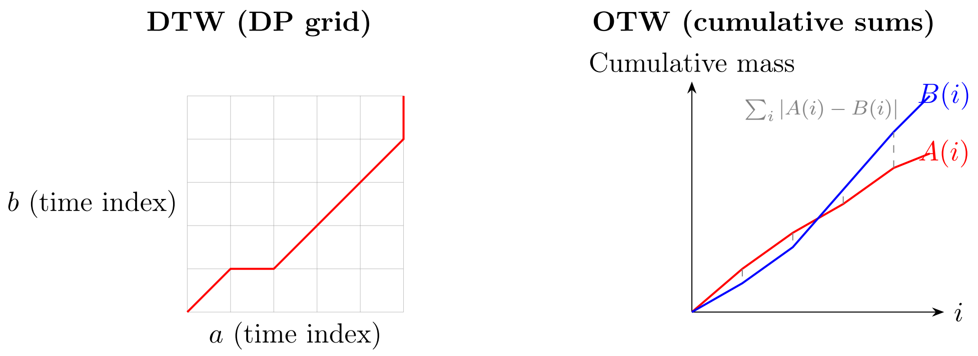

Figure 3.4: Schematic comparison of DTW and OTW. **Left:** DTW uses a dynamic programming grid with $\mathcal{O}(n^2)$ complexity. **Right:** OTW aligns cumulative distributions with $\mathcal{O}(n)$ complexity.

In terms of numerical computation, all key operations in OTW computing cumulative sums, applying local constraints, and evaluating the loss function can be performed using vectorized operations, making the method highly suitable for GPU acceleration and real-time applications [27, 54]. The addition of a smooth loss function (like the Huber loss)

Table 3.1: Mathematical comparison of DTW and OTW in terms of complexity, differentiability, and scalability.

Property	**DTW**	**OTW**
Time complexity	$\mathcal{O}(n^2)$ (DP table, see (3.21))	$\mathcal{O}(n)$ (cumulative sums, see (3.23))
Space complexity	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Differentiability	Non-differentiable (due to min over paths)	Differentiable (with smooth loss)
Scalability	Limited for long sequences	Efficient for large n
Integration in optimization	Hard to use in gradient-based models	Compatible with gradient-based models

[53] also ensures that the distance remains differentiable, which is essential for training models using gradient descent methods in neural networks.

Applications of OTW span a wide range of time series tasks across multiple domains. In gesture and activity recognition, OTW enables the alignment of motion sequences independent of pace or duration [55, 56]. In financial forecasting, it facilitates the comparison of price sequences or indicators measured over varying periods [57]. In medical signal analysis, such as ECG or EEG monitoring, OTW provides a robust way to match time-series signals despite irregularities or noise [58]. Finally, in audio and speech alignment, OTW supports flexible warping of phonetic sequences while preserving both local features and global structure [59]. Together, these examples highlight the versatility of OTW as a powerful tool for real-world time series analysis.

The strength of OTW lies in its ability to align sequences not only based on raw value differences but also by considering the structure and geometry of the series [60]. This makes it especially powerful in applications where DTW may overfit or fail to capture important context.

To better understand how OTW works, imagine two time series that track the same activity, such as walking speed over time for two people. One person walks steadily, while the other starts slowly and then speeds up. Although the patterns are similar, they happen at different times. If we use DTW to align these sequences, the method will force the two series to match point by point, even if one person is clearly ahead or behind. This often results in a zigzag-like matching path, which doesn't reflect the true relationship between the sequences.

Now, if we use OTW instead, it treats each time series as a distribution and finds the most natural way to move the "mass" of one sequence to match the other. This results in smoother and more meaningful alignment [14]. For example, a slow start in one series will be softly stretched to match the faster one without harsh jumps. OTW distributes the alignment cost over the whole sequence, making it better at handling differences in speed or timing.

A simple way to visualize this is by plotting both sequences on the same graph and drawing lines from matched points. With DTW, those lines often crisscross or bunch up in one area. With OTW, the lines tend to spread evenly and follow the natural flow of the data, showing a clear, smooth connection between the two patterns.

4 Classification Methods and Complexity Analysis

In this chapter, we take a closer look at classification, one of the most widely used techniques in machine learning. We begin by understanding what classification means, the different types such as binary and multiclass, and where it's used in the real world. From there, we focus on multiclass classification, exploring common strategies for handling it, along with the challenges it presents and the metrics used to evaluate its performance.

The chapter then introduces some of the most popular classification algorithms, with a special emphasis on the *k-Nearest Neighbors (k-NN)* method. Since this thesis uses *Optimal Transport Warping (OTW)* for time series analysis, we also explain how OTW can be used as a distance measure within classification tasks.

Finally, we compare how these methods perform in terms of time and space complexity, giving a practical view of their efficiency when applied to real-world data.

4.1 Introduction to Classification Problems

Classification is a type of supervised learning where the objective is to assign a label or category to input data based on previously observed patterns [61, 62]. In this setup, a model is trained using a dataset that includes input features along with their corresponding class labels. Once trained, the model can predict the label of new, unseen inputs. The core idea behind classification is to learn the relationship between features and categories so that similar inputs can be grouped under the same class. For example, given a set of images labeled as "cats" or "dogs," a classification algorithm can learn to distinguish between the two and correctly label new images.

After understanding what classification is, it's important to recognize that classification problems can vary in structure depending on the number of categories involved. The two most common types are *binary classification* and *multiclass classification.*

Binary classification refers to problems where there are only two possible outcomes. The model must decide between two classes, such as "yes" or "no," "spam" or "not spam," or “positive” vs. “negative” diagnosis [63]. This type of classification is often used in decision-making systems where a clear, two-way outcome is required.

Multiclass classification, on the other hand, involves three or more possible categories. Here, the model needs to assign each input to one class out of many. For example, recognizing handwritten digits (0 through 9) or identifying the genre of a song from multiple genres are multiclass problems [62]. These problems are more complex because the model must learn to distinguish between multiple classes that may be closely related or overlapping in their features.

Classification, whether binary or multiclass, plays a crucial role in a wide range of real-world applications. In the **medical field**, classification models assist in diagnosing diseases by analyzing patient symptoms, lab results, or imaging data. For instance, algorithms can classify whether a tumor is benign or malignant based on medical scans. In **finance**, classification helps detect fraudulent transactions by analyzing spending patterns or user behavior.

In **natural language processing**, text classification is widely used for spam detection, sentiment analysis, and topic labeling [64]. For example, email services use classifiers to separate spam from legitimate messages. In **image and speech recognition**, classification algorithms enable computers to identify objects, faces, and spoken words, which is fundamental in technologies like facial recognition and voice assistants.

These examples highlight how classification serves as a core function in intelligent systems, powering many technologies we interact with daily.

4.2 Multiclass Classification

Multiclass classification is a type of machine learning problem where each input is assigned to one class among three or more possible categories [65]. Unlike binary classification, which deals with only two outcomes, multiclass classification handles a wider range of decisions. The model learns to differentiate among multiple labels by identifying patterns in the training data that are specific to each class.

For example, consider a system that recognizes handwritten digits from 0 to 9. The model must learn to distinguish between ten different classes based on variations in shape, orientation, and writing style. Another example is a document classifier that categorizes news articles into topics like politics, sports, health, or technology. In each case, the model’s goal is to learn decision boundaries that correctly separate these categories.

Multiclass classification can be more challenging than binary classification because the model must handle more complex relationships and potential overlap between classes. This requires not only accurate prediction but also strategies that can scale well with the number of classes.

To handle multiclass classification problems using algorithms that are originally designed for binary classification, two popular strategies are commonly used: *One-vs-Rest (OvR)* and *One-vs-One (OvO)*.

In the **One-vs-Rest (OvR)** approach, the model creates a separate binary classifier for each class [66]. Each classifier is trained to distinguish between one specific class and all the others. For example, in a problem with four classes (A, B, C, D), four binary classifiers would be trained: one for A vs. not-A, one for B vs. not-B, and so on. During prediction, all classifiers are applied to the input, and the class with the highest confidence score is selected as the final output. OvR is simple and efficient, especially when the number of classes is relatively small.

In contrast, the **One-vs-One (OvO)** strategy builds a binary classifier for every possible pair of classes [67, 68]. For the same four-class example, this would result in six classifiers: A vs. B, A vs. C, A vs. D, B vs. C, B vs. D, and C vs. D. When making a prediction, each classifier votes for one of its two classes, and the class with the most votes is chosen. OvO can be more accurate in some cases, particularly when classes are closely related, but it also requires training a larger number of models, which increases computational cost.

Both strategies allow binary classifiers, such as logistic regression or support vector machines, to be effectively adapted to multiclass problems, each with trade-offs in terms of simplicity, training time, and performance.

While strategies like One-vs-Rest and One-vs-One make it possible to apply binary classifiers to multiclass problems, they also introduce several challenges. One key issue is the **imbalance of data across classes**, where some categories may have far more samples than others. This imbalance can bias the model toward majority classes, leading to poor performance on underrepresented ones [69]. Additionally, as the number of classes increases, so does the complexity of the decision boundaries, making it harder for the model to generalize well. In high-dimensional data, similar classes may overlap significantly, further complicating classification.

Another challenge is **computational efficiency**. For example, the One-vs-One strategy requires training a large number of classifiers, which becomes resource-intensive when dealing with dozens or hundreds of classes. Moreover, during prediction, aggregating the results from multiple models can slow down the decision process. Misclassification can also be harder to interpret in multiclass problems, especially when there is no clear hierarchy among the classes.

To properly evaluate multiclass models, it is important to use performance metrics that go beyond overall accuracy [70]. Accuracy, defined as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{4.1}$$

is easy to interpret but can be misleading when datasets are imbalanced. For this reason, additional metrics such as precision, recall, and the F1-score are often employed. Precision measures the proportion of true positives among all predicted positives,

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

while recall measures the proportion of true positives identified among all actual positives,

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

The F1-score combines these two into their harmonic mean,

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

In multiclass classification, these metrics can be generalized through two averaging strategies. In macro averaging, the metric is calculated independently for each class and then averaged, which treats all classes equally regardless of their size. In contrast, micro averaging aggregates the contributions of all classes before computing the metric, which effectively gives more weight to classes with larger sample sizes. Together, these approaches provide a more balanced view of model performance, ensuring that high accuracy on dominant classes does not overshadow poor results on underrepresented ones.

4.3 Common Classification Algorithms

In machine learning, a variety of classification algorithms are available, each with its own strengths, assumptions, and suitable use cases. This section briefly introduces some of the most widely used methods that form the foundation for more complex models.

Logistic Regression is one of the simplest and most interpretable classification algorithms [71]. It models the probability that a given input belongs to a certain class using a logistic (sigmoid) function. While it's mainly used for binary classification, it can be extended to multiclass problems using strategies like One-vs-Rest. Logistic regression performs well when the relationship between input features and the output class is approximately linear.

Decision Trees classify instances by learning a hierarchy of decision rules [72]. Each internal node represents a condition based on input features, and each leaf node represents a class label. Decision trees are easy to understand and visualize, making them useful for interpretability. However, they can be prone to overfitting, especially when the tree becomes deep or when the data contains noise.

Support Vector Machines (SVMs) are powerful classifiers that work well in high dimensional spaces [73]. SVMs try to find the optimal hyperplane that separates different classes by maximizing the margin between them. For non-linear problems, kernel functions (like RBF or polynomial kernels) can be used to project the data into higher dimensions. SVMs are particularly effective when the number of features is large compared to the number of samples.

k-Nearest Neighbors (k-NN) is a simple, instance-based classifier [74]. It doesn't build a model during training but instead stores the entire dataset. During prediction, it calculates the distance between the input and all training samples, then assigns the class based on the majority vote of the closest k neighbors. Because of its flexibility and non-parametric nature, k-NN is well-suited for problems where class boundaries are irregular. It is also commonly used in time series classification when combined with appropriate distance measures.

Neural Networks are inspired by the structure of the human brain and consist of layers of interconnected nodes (neurons). They are capable of learning highly complex, non-linear relationships and are widely used in applications like image and speech recognition. Although this thesis does not focus on deep learning, neural networks are worth mentioning as they represent a major direction in modern classification research.

Each of these algorithms offers different trade-offs in terms of interpretability, complexity, training time, and performance. Choosing the right method often depends on the nature of the dataset and the problem at hand.

4.4 k-Nearest Neighbors (k-NN)

The *k-Nearest Neighbors (k-NN)* algorithm is one of the most straightforward and widely used classification methods in machine learning [74]. It is a distance-based, non-parametric, and instance-based learning technique, which means it does not explicitly build a predictive model during training. Instead, it keeps the entire training dataset in memory and performs all computations at the time of making predictions.

When a new, unlabeled data point needs to be classified, the k-NN algorithm calculates the distance between this point and all the data points in the training set. Based on a chosen distance metric, the distance is usually Euclidean, but other metrics like Manhattan, Minkowski, or domain-specific distances can also be used: the algorithm identifies the k closest neighbors [75].

The k-Nearest Neighbors (k-NN) algorithm is a simple yet powerful method for both classification and regression. Given a test point $x \in \mathbb{R}^m$, the distance to each training point x_i is computed, often using the Euclidean metric

$$d(x, x_i) = \sqrt{\sum_{j=1}^{m} (x_j - x_{ij})^2} \tag{4.5}$$

The k training samples with the smallest distances are selected as the nearest neighbors $N_k(x)$. For classification, the predicted label is obtained by majority voting among these neighbors:

$$\hat{y} = \arg\max_{c \in \mathcal{C}} \sum_{i \in N_k(x)} \mathbf{1}\{y_i = c\} \tag{4.6}$$

where $\mathcal{C}$ is the set of possible classes and $\mathbf{1}\{\cdot\}$ is the indicator function. For regression tasks, the prediction is given by the average of the neighbor values:

$$\hat{y} = \frac{1}{k} \sum_{i \in N_k(x)} y_i \tag{4.7}$$

These formulations highlight that k-NN makes predictions directly from the training data without building an explicit model, which is both a strength and a limitation depending on the application.

k-NN is especially well-suited for *time series classification* because of its non-parametric nature and flexibility [6]. Time series data often includes non-linear patterns, varying lengths, and temporal shifts. Unlike traditional machine learning models that assume a fixed structure or distribution, k-NN can handle this kind of variation naturally.

Another key advantage is that k-NN does not involve any training phase. It simply stores the training data and performs all calculations during prediction. This is par-

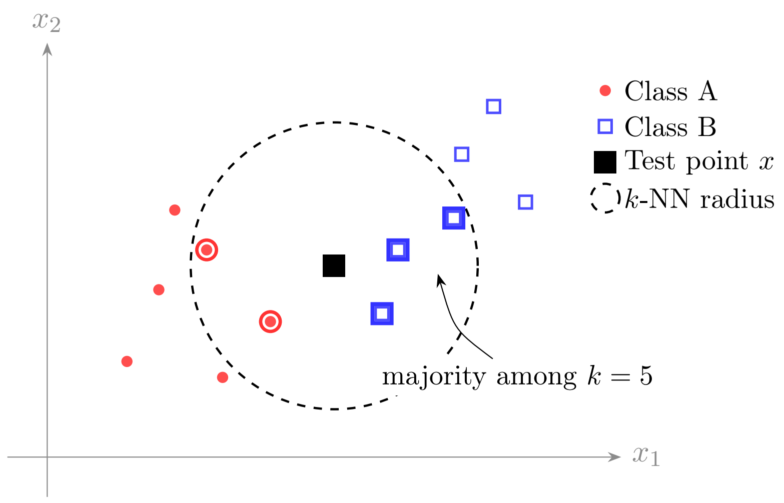

Figure 4.1: Illustration of k-NN classification (here $k = 5$): the test point x predicts the class by majority vote among its nearest neighbors within the dashed circle. In this example, three neighbors belong to Class B (blue) and two to Class A (red), so the test point is assigned to Class B.

ticularly useful when applying complex or computationally intensive distance measures, such as *Optimal Transport Warping (OTW)*. Since distances are only computed during prediction, the overhead of using sophisticated metrics like OTW is minimized.

Strengths of k-NN

The k-NN algorithm offers several notable strengths. It is conceptually simple and easy to implement, making it an accessible method for a wide range of tasks. As a non-parametric model, it makes no assumptions about the underlying data distribution and can naturally handle multiclass problems [62]. Moreover, k-NN does not involve an explicit training phase: predictions are made directly based on the stored data. This also means that no model knowledge is retained or stored for subsequent evaluations, which can be advantageous when working in dynamic environments where data distributions may shift over time.

Limitations of k-NN

Despite its advantages, k-NN also has several important limitations. Since the algorithm requires storing and comparing all training samples at prediction time, it can be computationally expensive for large datasets, both in terms of time and memory usage.

Its performance is highly sensitive to the choice of distance metric and the value of k, which may vary depending on the problem. Furthermore, k-NN can struggle in high-dimensional spaces due to the curse of dimensionality, where distances between points become less meaningful. Finally, because all training data must be retained, the method can be inefficient in scenarios where fast, repeated predictions are required[76].

Using OTW with k-NN for Time Series Classification

One of the most powerful features of k-NN is its ability to work with custom distance functions, which is especially important for time series data. Traditional distance metrics like Euclidean often fail when time series are misaligned or shifted in time, even if their shapes are similar.

This is where *Optimal Transport Warping (OTW)* becomes valuable. OTW aligns two sequences in a way that accounts for these shifts and distortions by minimizing a transport cost [14]. When k-NN uses OTW as its distance measure, it can compare time series based on their underlying structural similarity rather than their raw pointwise values.

For example, consider two signals from temperature sensors that track the same heating event, but one starts slightly later. Euclidean distance would treat them as different because the points don't align exactly. However, OTW can shift and align these sequences, capturing their similarity more effectively. When such aligned distances are used in k-NN, the classification becomes more accurate and meaningful.

The k-NN algorithm has been successfully applied across a variety of domains. In speech and audio recognition [77], it enables the alignment and classification of acoustic patterns. For human activity and gesture recognition, k-NN is often used to match motion sequences captured by sensors, even when execution speed or style varies. In the medical field, particularly in the analysis of physiological signals such as ECG or EEG, k-NN provides a simple yet effective method for detecting abnormalities and supporting diagnosis. The algorithm is also widely applied in sensor-based fault detection, where it helps to identify anomalies and ensure system reliability. These diverse applications demonstrate the versatility of k-NN for real-world time series and signal analysis tasks.

In summary, k-NN is a flexible and powerful classifier that works especially well for time series tasks when paired with advanced distance metrics like OTW. Its simplicity, lack of training phase, and compatibility with custom distance functions make it a valuable tool in scenarios where interpretability and adaptability are important.

4.5 Optimal Transport Warping (OTW) Distance as a Metric

In many machine learning tasks, especially those that involve time series, the choice of distance metric plays a crucial role in how effectively a model can compare and classify data. Traditional metrics like Euclidean distance or even Dynamic Time Warping (DTW) have limitations when it comes to capturing structural differences and temporal misalignments. This is where *Optimal Transport Warping (OTW)* comes into play as a powerful, structure-aware distance measure.

OTW is designed to align time series in a way that reflects not just pointwise similarity but also the underlying shape and distribution of the data over time. Unlike DTW, which focuses solely on aligning individual time steps (and does so with quadratic complexity), OTW approaches alignment from the perspective of optimal transport theory. It treats time series as distributions and calculates the most efficient way to "move" one sequence onto another while minimizing a cost function. This makes OTW particularly effective at capturing temporal shifts, varying speeds, or non-linear changes between sequences.

The main reason OTW is used in place of standard metrics is its ability to combine accuracy, robustness, and computational efficiency. As shown earlier in Section 3.4, DTW requires filling an $n \times m$ dynamic programming table, resulting in quadratic time complexity $\mathcal{O}(n^2)$. In contrast, the cumulative formulation of OTW in (3.23) can be computed in a single pass, yielding linear complexity $\mathcal{O}(n)$. Moreover, unlike DTW, OTW is differentiable when combined with smooth loss functions (see Section 3.4.3), which makes it compatible with gradient-based optimization and modern deep learning pipelines. These properties make OTW not only efficient but also well suited to large-scale machine learning tasks.

In the context of classification, OTW can be integrated with k-NN or other distance-based methods to significantly improve performance on time series tasks. By using OTW as the distance metric, the classifier becomes more sensitive to structural similarities, even when two sequences are not perfectly aligned in time. This leads to more accurate predictions in practical domains such as gesture recognition, ECG signal analysis, audio and speech processing, and human activity monitoring, many of which were discussed in Sections 3.4 and 4.4. In summary, OTW serves as a problem-aware distance measure that enhances classifier performance by aligning sequences according to their global structure rather than individual time steps. Its balance of accuracy and efficiency makes it a compelling alternative to conventional metrics in time series classification.

4.6 Time Complexity of Classification Algorithms

Computational complexity refers to the amount of time and memory resources required by an algorithm to complete a task as a function of the input size. It provides a theoretical framework to evaluate and compare algorithms based on their efficiency. *Time complexity* describes how the number of operations grows with input size, while *space complexity* measures how much memory is needed.

In machine learning, understanding the computational complexity of a model is essential especially when working with large datasets or real-time systems [71]. An algorithm with high accuracy but poor efficiency might be impractical for deployment, particularly in time-sensitive or resource-constrained environments. By analyzing complexity, we can choose models and methods that strike a balance between predictive power and scalability.

In algorithm analysis, different types of complexity give us insight into how computation scales as data size increases. Two common classes are linear complexity, denoted as $\mathcal{O}(n)$, and quadratic complexity, denoted as $\mathcal{O}(n^2)$.

An algorithm with **linear complexity** ($\mathcal{O}(n)$) scales proportionally with the size of the input. This means that if the input size doubles, the time or memory required also roughly doubles. Linear-time algorithms are generally efficient and suitable for large-scale data processing. For example, a single pass through a data set, such as computing the sum of all elements, has linear complexity.

In contrast, **quadratic complexity** ($\mathcal{O}(n^2)$) involves performing operations on every pair of elements, which causes the required time to grow much faster. If the input size doubles, the computation time increases by a factor of four. A classic example is comparing every item in a dataset with every other item, such as in brute-force distance calculations between all pairs in a classification task [63].

Understanding this difference is particularly important in tasks like time series classification, where sequences can be long. Algorithms that rely on pairwise comparisons (like DTW) may quickly become computationally expensive, while linear-time alternatives (like OTW) can offer a more practical solution for real-time or large-scale applications.

The **k-Nearest Neighbors (k-NN)** algorithm, while simple and intuitive, can vary significantly in its computational complexity depending on how it is implemented [74]. In the brute-force approach, where the distance between the query point and every training sample is computed individually, the time complexity is $\mathcal{O}(n)$ for a single prediction, where n is the number of training samples. However, when we need to perform this operation for multiple test samples, the total complexity can grow to $\mathcal{O}(n^2)$, especially in scenarios like leave-one-out validation or when classifying all points in a large dataset.

However, in high-dimensional data (such as time series), these tree-based optimizations often break down due to the “curse of dimensionality,” and brute-force search becomes the default [76]. In such cases, the algorithm becomes computationally more expensive and may require additional strategies, such as dimensionality reduction or efficient distance measures, to stay practical.

When using **Optimal Transport Warping (OTW)** as a distance metric, understanding its computational cost is essential. One of the advantages of OTW over traditional alignment methods like *Dynamic Time Warping (DTW)* is its **linear time complexity**, typically denoted as $\mathcal{O}(n)$, where n is the length of the time series. Unlike DTW, which performs pairwise comparisons and has a quadratic complexity of $\mathcal{O}(n^2)$ [78], OTW leverages optimal transport theory and efficient relaxation techniques to align sequences with significantly less computational overhead [14].

This efficiency makes OTW particularly suitable for large-scale or real-time applications, where quick distance evaluations are necessary. Moreover, OTW is **differentiable**, enabling its integration into gradient-based learning frameworks, something not feasible with DTW. This further extends its usability in modern machine learning pipelines.

However, while OTW offers notable gains in efficiency, it’s important to consider the trade-offs between accuracy and computational cost. More complex algorithms or metrics, such as traditional DTW or kernelized methods, can sometimes capture subtle alignments better, especially in highly irregular or noisy data. Yet, these gains often come at the cost of slower processing and increased memory usage.

OTW provides a strong middle ground by delivering a balance of interpretability, accuracy, and computational efficiency. For many practical scenarios, especially those involving time-sensitive predictions or large datasets, the slight loss in alignment granularity (compared to more exhaustive methods) is outweighed by OTW’s ability to deliver fast and robust performance.

5 Dataset Description

5.1 Source: UCR Time Series Classification Archive

The dataset used in this thesis is obtained from the UCR Time Series Classification Archive [7]. This archive is one of the most widely recognized and frequently used collections of datasets specifically designed for time-series classification research.

Initially created by researchers Keogh and Folias in 2002, the purpose of the UCR archive was to provide standardized datasets to support reproducible and fair comparison of different time-series classification methods. Over the years, the archive has significantly grown and evolved. In its earliest form, it offered 16 datasets, and by 2018, it expanded dramatically to include 128 different datasets for univariate time series and an additional 30 datasets for multivariate cases.

The datasets within the archive cover a broad range of fields, including medical diagnosis, sensor data, power consumption monitoring, traffic analysis, and image analysis, among others. Each dataset is formatted consistently, containing clearly separated training and testing data along with predefined class labels. This structure makes it easy for researchers to compare results and validate their algorithms consistently.

The widespread usage of the UCR archive datasets is evident through the numerous academic papers published using these datasets to test and benchmark new algorithms. The archive is actively maintained and supported through various research grants, highlighting its importance and credibility in the scientific community.

5.2 Dataset Used

The UCR Archive contains a total of 92 datasets without missing values. It is impractical and overly lengthy to detail all these datasets. Therefore, this study selected one representative data set from each of the seven main categories: Device, Image, Spectrography, Sensor, Medical, Traffic, and Power Consumption to illustrate the nature and characteristics of the data clearly.

Table 5.1: Representative datasets selected from UCR Archive categories

Category	**Dataset**	**No. of Classes**	**Train Size**	**Test Size**	**Length**
Device	ACSF1	10	100	100	1460
Image	ArrowHead	3	36	175	251
Spectrography	OliveOil	4	30	30	570
Sensor	Car	4	60	60	577
Medical	InsectEPGSmallTrain	3	17	249	601
Traffic	Chinatown	2	20	343	24
Power Consumption	PowerCons	2	180	180	144

5.2.1 Device: ACSF1

The ACSF1 dataset comes from the Appliance Consumption Signature Database (ACS-F1) [79], which was developed to capture and distinguish the power usage patterns of various household appliances. This dataset includes ten classes, each representing a specific type of device such as mobile phone chargers, coffee machines, computer workstations, fridges and freezers, Hi-Fi systems, compact fluorescent lamps (CFLs), laptops, microwave ovens, printers, and televisions. The data reflects the behavior of these appliances in real-world settings and is valuable for developing classification systems in smart home environments.

Each time series in this dataset is 1460 data points long. These points represent power consumption measurements collected over time, though the exact sampling interval is not specified. The values likely represent standardized current or power readings recorded at short, regular intervals throughout a complete operation cycle or a defined duration. The visualizations (shown below) provide a clear understanding of how each class behaves: some appliances, like lamps and Hi-Fi systems, show consistent energy usage, while others, such as microwaves and fridges, display intermittent but strong spikes. These differences in signal characteristics across devices make the ACSF1 dataset particularly suitable for tasks like load disaggregation and appliance recognition.

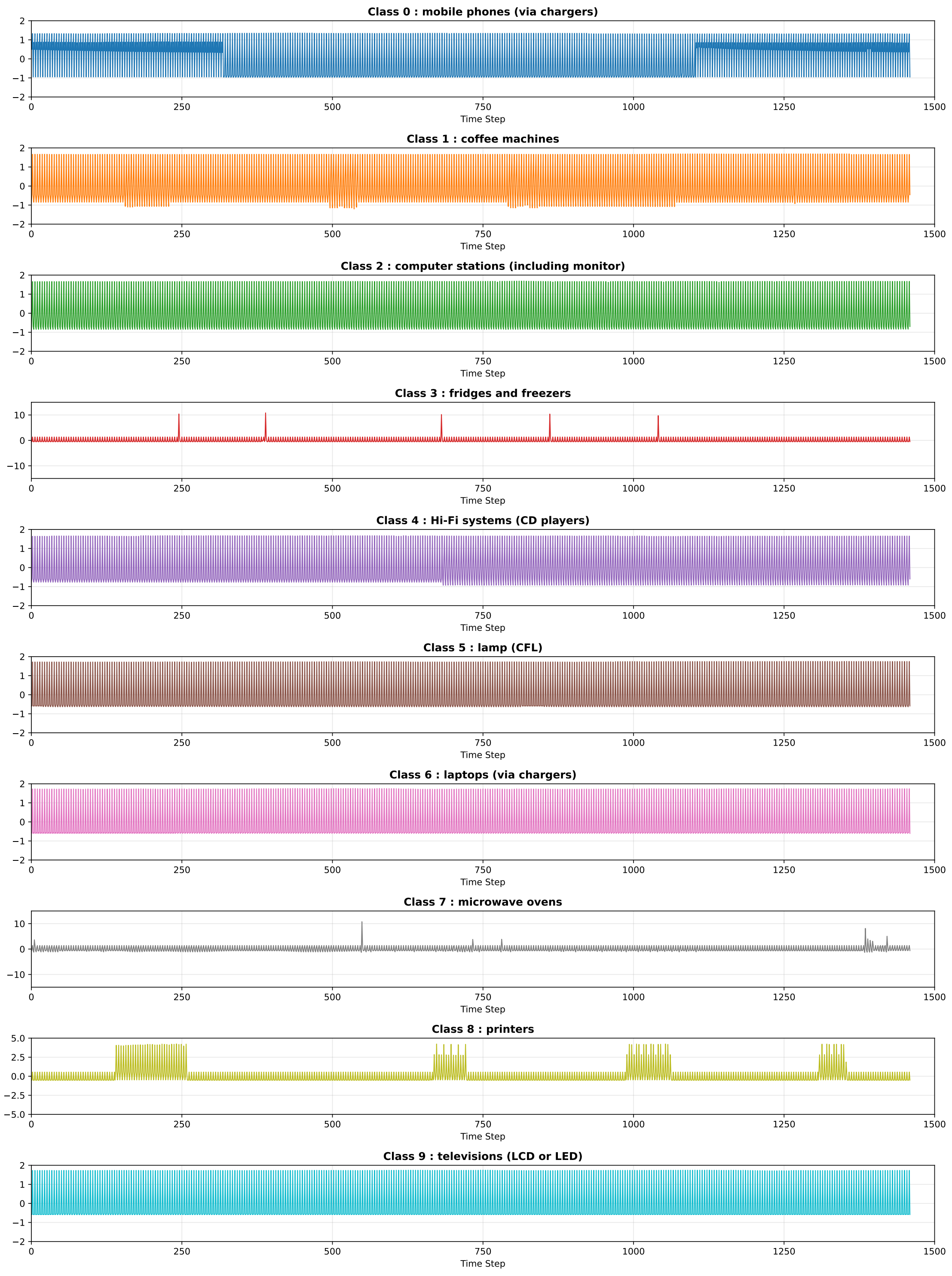

Figure 5.1: Visualization of ACSF1 dataset classes showing appliance specific consumption patterns.

5.2.2 Image: ArrowHead

The ArrowHead dataset is created from images of projectile points, commonly known as arrowheads [32]. These images are converted into time series by tracing the shape's outline and calculating the angles between consecutive points along the edge. This angle-based transformation results in a sequence of values that capture the curvature of the arrowhead's silhouette. Each time series consists of 251 data points, with the values encoding the shape rather than time. This approach ensures that important geometric features of the shape, such as bends and notches, are retained in the sequence.

The dataset includes three classes: *Avonlea*, *Clovis*, and *Mix*. These classes reflect archaeological distinctions in arrowhead shapes, particularly focusing on t he presence, position, and structure of notches. For example, Clovis points are known for their central flutes, while Avonlea types have smoother profiles. The Mix class combines features from both. As shown in the graphs below, each class demonstrates a unique curvature pattern that corresponds to its shape features. This dataset is useful for studying classification tasks that involve shape-based features, and it has applications in archaeological analysis and cultural identification.

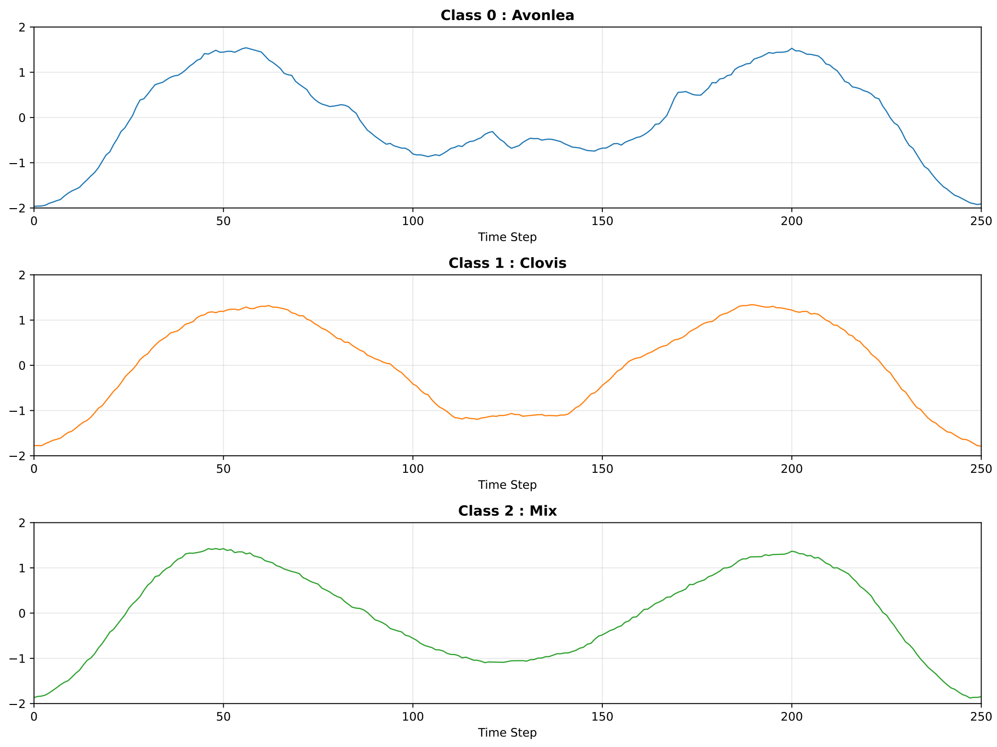

Figure 5.2: Visualization of the three ArrowHead classes (Avonlea, Clovis, Mix) as time series representations.

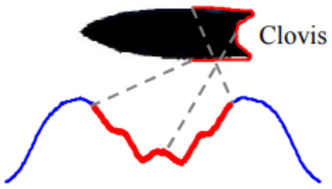

Figure 5.3: Example of an ArrowHead shape (Clovis) with its corresponding angle based time series encoding.

5.2.3 Spectrography: OliveOil

The OliveOil dataset originates from the field of chemometrics and is based on spectroscopic analysis of extra virgin olive oils. Specifically, each time series is derived from a Fourier Transform Infrared (FTIR) spectrograph [80], where the input spectrum reflects the absorption intensity of infrared light at different frequencies. Each series contains 570 values, corresponding to absorbance at evenly spaced wavenumber intervals across the infrared range. These signals are sensitive to the chemical makeup of the samples, making the data suitable for identifying different oils based on their origin or composition.

This dataset includes four classes, each representing olive oil sourced from a different country. As shown in the graph below, the time series for all classes share a similar overall shape, but subtle variations in amplitude and peak positions reflect underlying chemical differences. The signals include major peaks at consistent positions, which likely correspond to specific molecular vibrations. These subtle differences make the OliveOil dataset a good example of a low-variance, high-complexity classification problem and an important benchmark in food quality assessment research.

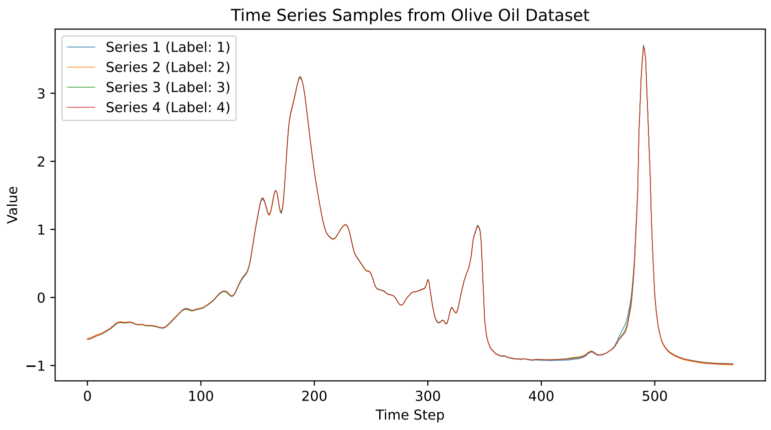

Figure 5.4: Visualization of OliveOil dataset: spectrographic time series of four olive oil classes.

5.2.4 Sensor: Car

The Car dataset is constructed from silhouette outlines of different vehicle types, including Sedan, Pickup, Minivan, and SUV, extracted from real video footage. Each silhouette is transformed into a univariate time series by computing the turn angle at equidistant points along the vehicle contour [81]. This approach captures the geometric curvature of the vehicle's shape in 128 evenly spaced segments. The resulting time series, each of length 577, reflect the structural profile of vehicles in a way that is invariant to scale, rotation, and translation.

The dataset consists of four distinct classes: Sedan, Pickup, Minivan, and SUV. As shown in the graph below, each vehicle class exhibits its own unique signal characteristics. For instance, the curves for the Sedan and Minivan are smoother, potentially indicating more consistent driving dynamics, whereas the Pickup and SUV may introduce more abrupt changes in the signal due to their weight or suspension behavior. These variations provide meaningful features for classification tasks and can be used to develop algorithms for vehicle identification or driving behavior analysis based on sensor data.

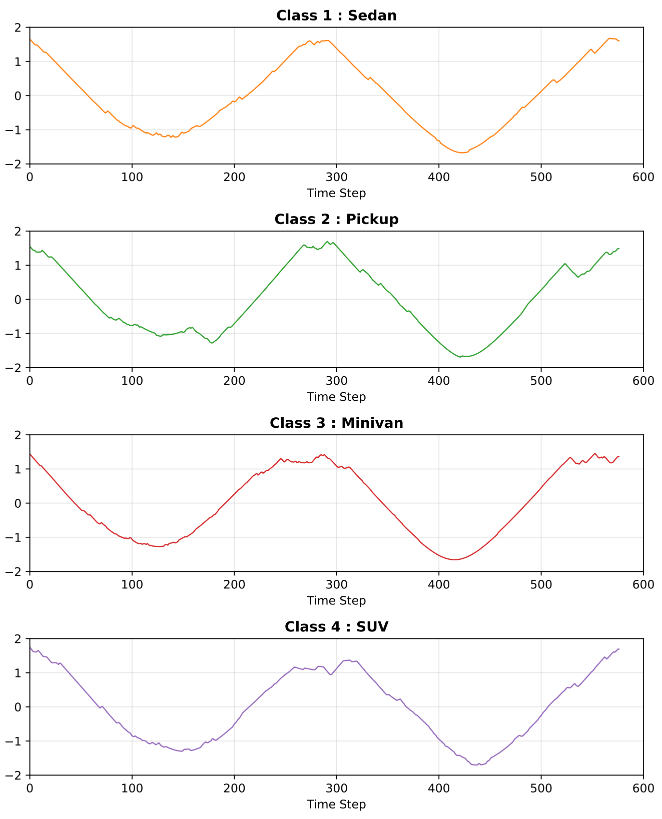

Figure 5.5: Visualization of Car dataset: time series representations of Sedan, Pickup, Minivan, and SUV classes.

5.2.5 Medical: InsectEPGSmallTrain

The InsectEPGSmallTrain dataset represents electrical signals recorded using Electrical Penetration Graph (EPG) technology, which monitors insect feeding behaviors [82] by detecting voltage changes as insects interact with plant tissues. Each time series in the dataset has a length of 601 and is collected at regular time intervals while an insect feeds or probes a plant surface. The voltage values vary based on internal insect activities and the type of tissue being accessed, making these signals an indirect but powerful indicator of behavioral patterns.

The dataset includes three classes: Phloem Feeding, Xylem Feeding, and Non-ingestion/ Probing. These are reflected in the graphs shown below, where each class exhibits distinct signal patterns. For instance, Phloem Feeding shows fluctuating, high-amplitude bursts; Xylem Feeding is characterized by a sudden voltage drop followed by gradual recovery; and Non-ingestion events feature chaotic, low-voltage patterns. The fine granularity and temporal resolution of these recordings provide valuable data for entomological studies and automated behavior classification models.

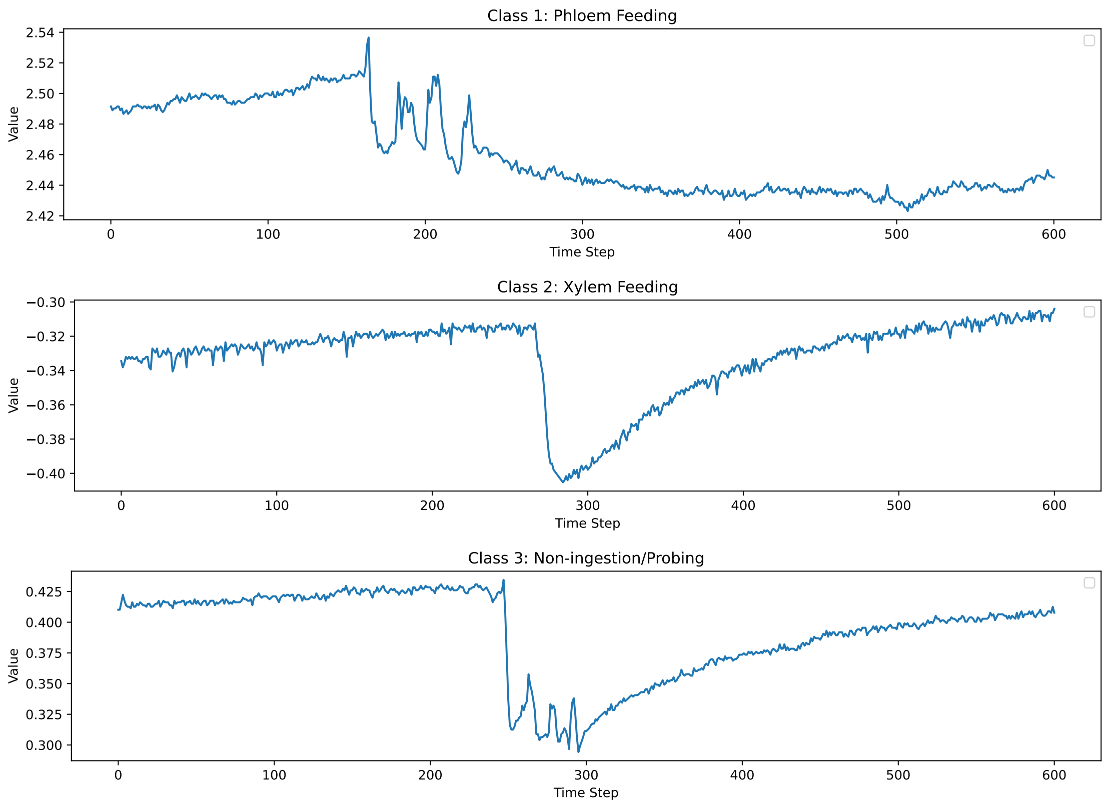

Figure 5.6: InsectEPGSmallTrain dataset: EPG voltage signals for three classes Phloem Feeding, Xylem Feeding, and Non-ingestion/Probing.

5.2.6 Traffic: Chinatown

The Chinatown dataset is derived from an automated pedestrian counting system implemented by the City of Melbourne, Australia [83]. This specific dataset records hourly pedestrian traffic counts on Swanston Street (North) in Chinatown, sampled over the full calendar year of 2017. Each time series consists of 24 values each corresponding to one hour of a day capturing the temporal distribution of pedestrian movement. The dataset is divided into two classes: weekdays and weekends. These classes help reflect differences in human activity patterns based on the day type.

As seen in the graph below, pedestrian activity during weekdays shows a sharp rise between 8:00 AM and 6:00 PM, likely corresponding to commuting hours and working hours. In contrast, weekend patterns exhibit more gradual and sustained foot traffic throughout the day, with peaks during midday and evening leisure periods. These patterns make the Chinatown dataset ideal for studying temporal behavior in urban spaces, especially for applications such as smart city planning, anomaly detection, or public infrastructure optimization.

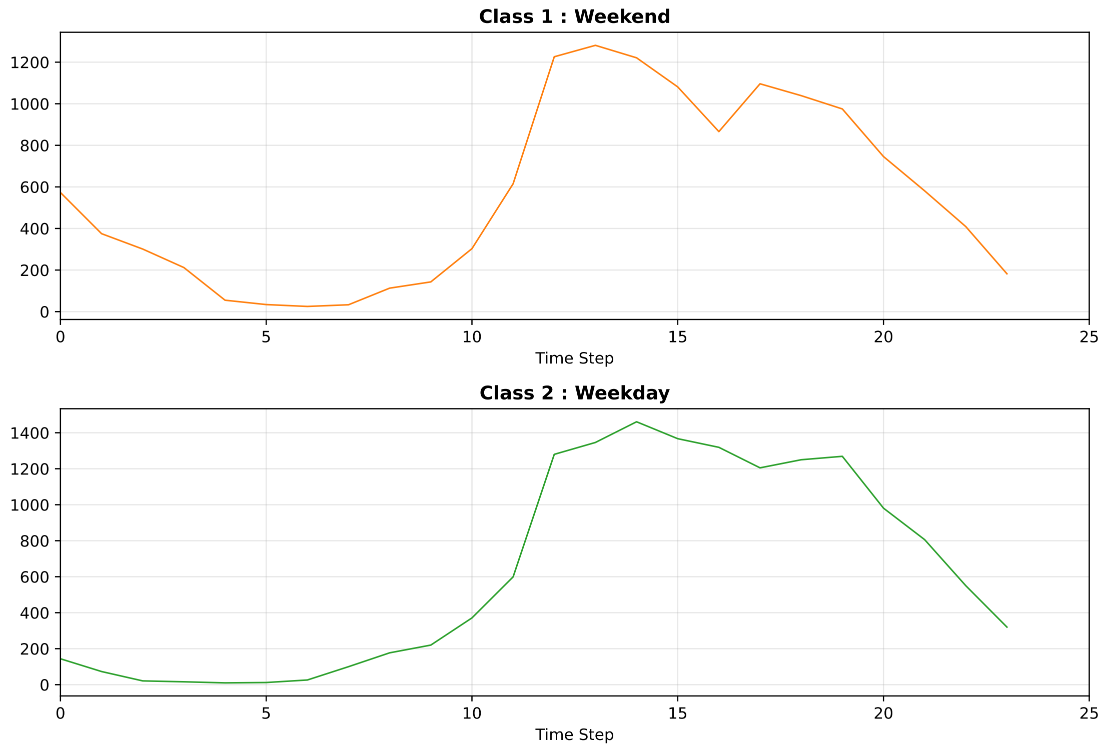

Figure 5.7: Chinatown dataset: hourly pedestrian counts for Weekend (Class 1) and Weekday (Class 2).

5.2.7 Power Consumption: PowerCons

The PowerCons dataset comprises residential electric power consumption data sampled every 10 minutes over a 24-hour period, resulting in 144 measurements per time series. This high-resolution dataset is split into two classes based on seasonal conditions: warm season (April to September) and cold season (October to March). The data was sourced from EDF R&D in Clamart, France [84], and includes both training and testing sets of equal size (180 samples each). The goal is to distinguish seasonal usage patterns in electricity demand, which are influenced by weather, heating, and cooling behaviors.

The visualizations highlight how energy usage profiles vary between seasons. In the warm season, there are multiple sharp peaks in consumption, likely due to the use of air conditioning or other cooling appliances during the hottest parts of the day. In contrast, the cold season shows a more gradual buildup of power usage, with peaks occurring later in the day, reflecting heating system usage during colder evenings. These distinctive usage profiles make the dataset ideal for energy forecasting, season-aware consumption modeling, and optimizing grid resource allocation.

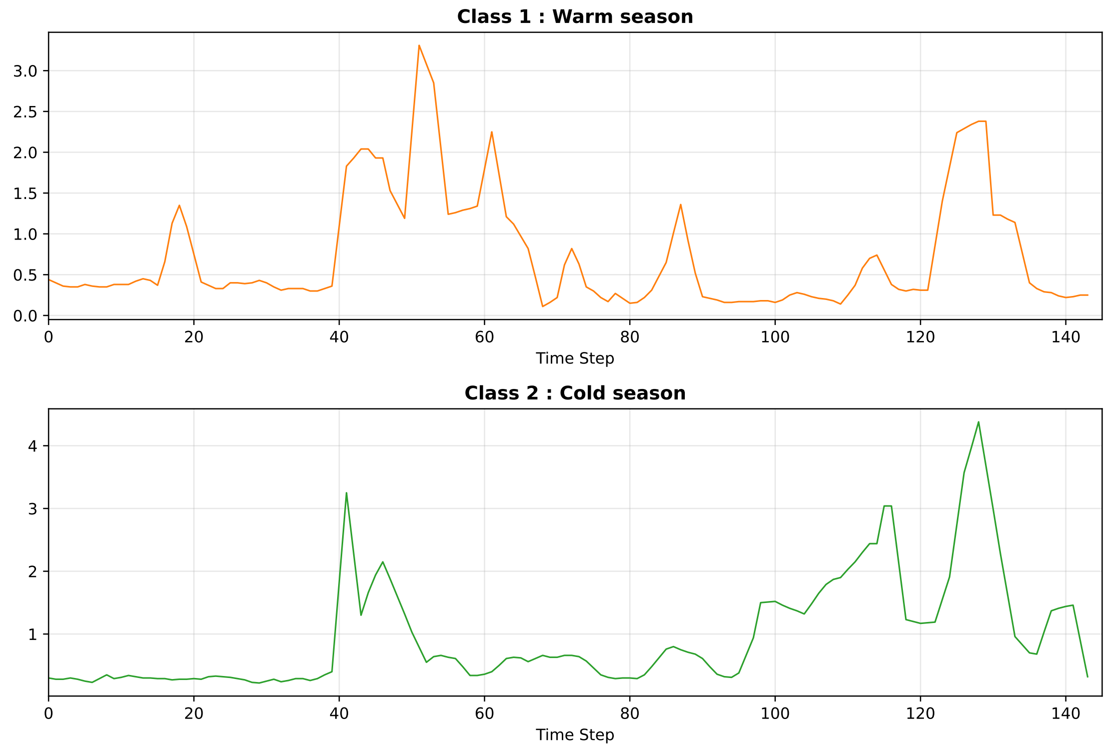

Figure 5.8: PowerCons dataset: seasonal variation in electricity usage profiles for warm and cold seasons.

5.3 Train/Test Split and Label Structure

In addition to the official train/test splits provided by the UCR Archive, this study further divides the training set using an 80/20 ratio to create a validation set. This internal validation split is used exclusively for hyperparameter tuning and model selection purposes. It ensures that model performance is not overfitted to the training data and that final evaluation remains unbiased on the untouched test set [7].

The UCR datasets used in this study come pre-divided into training and testing sets, as part of the standardized benchmarking format. Each dataset includes a fixed split that is consistent across research works to ensure reproducibility and fair performance comparisons. The split sizes vary by dataset but typically reflect a reasonable balance to support both training generalization and evaluation.

All datasets use a class label system where each time series is tagged with an integer representing its class. Labels are one-based (i.e., they start from 1), and each class corresponds to a meaningful category in the dataset's context, such as appliance type in ACSF1, vehicle shape in Car, or season in PowerCons. The class distributions are mostly balanced within individual datasets, although a few datasets (like InsectEPGSmallTrain) exhibit skewed ratios between training and testing sizes due to the small sample availability.

Additionally, the time series are aligned and normalized in most datasets to remove variations due to scale or offset. The use of consistent time-series lengths across samples in a dataset helps ensure that models learn structural patterns rather than superficial distortions. This standardized format also allows easy comparison across various classification algorithms and techniques.

The visualizations highlight how energy usage profiles vary between seasons. In the warm season, there are multiple sharp peaks in consumption, likely due to the use of air conditioning or other cooling appliances during the hottest parts of the day. In contrast, the cold season shows a more gradual buildup of power usage, with peaks occurring later in the day, reflecting heating system usage during colder evenings. These distinctive usage profiles make the dataset ideal for energy forecasting, season-aware consumption modeling, and optimizing grid resource allocation.

5.4 Why this Dataset is Relevant for OTW

Optimal Transport Warping (OTW) is designed to compare time series data by identifying the most cost-effective alignment between two sequences. Unlike traditional distance metrics such as Euclidean or DTW, OTW can account for non-linear distortions and local differences, making it highly suitable for real-world, irregular time series data. The goal of this study is to evaluate OTW's performance across a diverse set of domains, and the chosen datasets from the UCR Archive provide an ideal testbed for this purpose [7].

Each of the seven selected datasets introduces unique challenges that are well-aligned with OTW's strengths. For example, ACSF1 and PowerCons involve consumption patterns with irregular spikes; InsectEPGSmallTrain and OliveOil capture complex signal fluctuations; and ArrowHead and Car emphasize shape-based temporal variation. The diversity in sequence length, domain context, and class complexity allows for a robust assessment of OTW's ability to align and discriminate between sequences under a wide range of conditions. This combination ensures that the evaluation reflects both OTW's theoretical robustness and its practical utility.

The UCR datasets used in this study come pre-divided into training and testing sets, as part of the standardized benchmarking format. Each dataset includes a fixed split that is consistent across research works to ensure reproducibility and fair performance comparisons. The split sizes vary by dataset but typically reflect a reasonable balance to support both training generalization and evaluation.

All datasets use a class label system where each time series is tagged with an integer representing its class. Labels are one-based (i.e., they start from 1), and each class corresponds to a meaningful category in the dataset's context, such as appliance type in ACSF1, vehicle shape in Car, or season in PowerCons. The class distributions are mostly balanced within individual datasets, although a few datasets (like InsectEPGSmallTrain) exhibit skewed ratios between training and testing sizes due to the small sample availability.

Additionally, the time series are aligned and normalized in most datasets to remove variations due to scale or offset. The use of consistent time-series lengths across samples in a dataset helps ensure that models learn structural patterns rather than superficial distortions. This standardized format also allows easy comparison across various classification algorithms and techniques.

The visualizations highlight how energy usage profiles vary between seasons. In the warm season, there are multiple sharp peaks in consumption, likely due to the use of air conditioning or other cooling appliances during the hottest parts of the day. In contrast, the cold season shows a more gradual buildup of power usage, with peaks occurring later in the day, reflecting heating system usage during colder evenings. These distinctive usage profiles make the dataset ideal for energy forecasting, season-aware consumption modeling, and optimizing grid resource allocation.

6 Numerical Experiments

This chapter focuses on evaluating the performance of Optimal Transport Warping (OTW) across a diverse set of time series datasets selected from the UCR Archive [7]. The main objective is to compare OTW with the widely used Dynamic Time Warping (DTW) method, using classification accuracy as the key metric. By selecting one representative dataset from each of seven different categories including domains like power consumption, traffic flow, sensor readings, and medical signals the experiments aim to demonstrate how well OTW performs under varying data characteristics. The results are based on multiple runs to ensure reliability and to capture performance variability.

6.1 Implementation Details

This section describes the experimental setup and implementation used to evaluate Optimal Transport Warping (OTW) across selected datasets from the UCR Time Series Classification Archive. All experiments were implemented in Python using Visual Studio Code as the development environment. The workflow was structured around modular code files and executed using Python scripts and Jupyter-style functionality embedded in VS Code. The primary goal of these experiments was to evaluate OTW's classification accuracy across diverse datasets and compare it with established benchmarks.

Each dataset's train and test split was taken directly from the official UCR repository, while DTW results were not computed manually; instead, they were obtained from the published benchmark results available on the UCR Time Series website [7]. OTW, in contrast, was implemented and run on each dataset, with the classification performance evaluated over 10 separate runs. Within each training set, an 80/20 validation split was applied to tune parameters and monitor training performance consistency.

6.1.1 Python Libraries Used

The following Python libraries were used throughout the implementation:

- NumPy and Pandas for handling arrays and data structures

- Matplotlib for plotting and visualizing results
- PyTorch for tensor operations and GPU acceleration using CUDA
- Custom distance computation code for OTW, designed from scratch for this project

6.1.2 Code Structure

The codebase is structured to support:

1. Loading UCR datasets from `.tsv` files
2. Preprocessing the data (normalization, class extraction)
3. Splitting the training data into training and validation sets
4. Computing OTW distance matrices
5. Performing classification with 1-Nearest Neighbor (1-NN)
6. Logging accuracy, mean error, and confidence intervals over 10 repeated runs

This structure allows consistent evaluation across multiple datasets while making it easy to extend or adapt individual parts of the code for additional experiments.

Algorithm 1 OTW-Based Time Series Classification Framework

```
Require: Train set $X_{train} \in \mathbb{R}^{n \times t}$, Test set $X_{test} \in \mathbb{R}^{m \times t}$, Fixed parameters $m, s, \beta$,
    Number of runs $R = 10$
Ensure: Mean classification accuracy ± CI95
 1: Initialize $accuracy\_list \leftarrow [\,]$
 2: for $run = 1$ to $R$ do
 3:     for each test sample $x \in X_{test}$ do
 4:         for each train sample $y \in X_{train}$ do
 5:             Compute $OTW_{\beta}^{m,s}(x, y)$
 6:         end for
 7:         Predict class of $x$ using 1-NN
 8:     end for
 9:     Calculate accuracy of predictions
10:     Append result to $accuracy\_list$
11: end for
12: Compute mean accuracy and CI95
13: return $accuracy\_list$ ± CI95
```

6.1.3 GPU Utilization

GPU resources were actively utilized during the experiments to improve computational efficiency. The implementation leverages PyTorch's CUDA support to transfer both training and testing data to the GPU, enabling OTW distance computations and tensor operations to run in parallel. This was particularly beneficial during repeated experiment loops, where large batches of pairwise distance calculations were required.

All experiments were conducted on a laptop equipped with a 13th Gen Intel(R) Core(TM) i7-13650HX CPU and 16 GB RAM, alongside an NVIDIA GeForce RTX 4050 Laptop GPU with 6 GB VRAM running CUDA 12.2. GPU acceleration was enabled through PyTorch's CUDA backend, which automatically detected the availability of CUDA and switched between CPU and GPU execution when necessary. This ensured portability of the implementation while providing significant speedups whenever GPU resources were available.

To illustrate the benefits of GPU acceleration, Table 6.1 reports the runtimes for both CPU and GPU implementations across representative datasets. Even though most of the datasets used in this study are relatively small, the results clearly show that GPU execution provides a substantial speedup. For example, in the ACSF1 dataset the runtime dropped from 14.2 seconds on CPU to only 0.1 seconds on GPU, and similar reductions were observed for PowerCons and Car. The improvement is even more striking for larger datasets such as FaceAll, where GPU execution reduced the runtime from over 180 seconds to under 2 seconds. Overall, this shows that the PyTorch + CUDA version of OTW runs much faster than the CPU version built on NumPy and Pandas, and this holds true for both small and large datasets.

Table 6.1: Runtime comparison between CPU and GPU implementations of Test and Train dataset

Dataset	**CPU Runtime (s)**	**GPU Runtime (s)**
ACSF1	14.2	0.1
ArrowHead	2.3	0.1
OliveOil	0.6	0.1
Car	2.5	0.1
InsectEPGSmallTrain	2.9	0.2
Chinatown	1.2	0.4
PowerCons	7.4	0.1
FaceAll	183.7	1.8

6.2 Experimental Setup

The experiments in this study were conducted using the official train/test splits provided by the UCR Archive. Within each training set, an internal 80/20 validation split was applied to ensure that the model evaluation was consistent and robust. Each experiment was repeated ten times per dataset to capture variability and assess average classification performance.

All OTW experiments used fixed values for the internal parameters m, s, and β. These parameters were not individually tuned for each dataset. Instead, they were kept constant across all runs to maintain consistency and allow for fair comparisons. This approach was chosen to focus the evaluation on the general effectiveness of OTW rather than optimizing performance through a parameter search. The values were selected based on previous studies and preliminary tests [14], ensuring that they provided stable performance across diverse datasets without overfitting to any specific case.

6.3 Testing Methodology

To ensure the reliability of the results, each experiment was repeated ten times per dataset, using the official UCR train/test split. In each run, the training set was further divided using an 80/20 ratio to create a validation subset. Unlike random shuffling, this study employed a deterministic GPU-based splitting method to ensure consistency across runs while still varying the selected validation samples. This was implemented directly in PyTorch using CUDA through a custom function `deterministic_train_test_split_gpu()`, shown in Algorithm 2. The validation data were only used for monitoring and parameter stability, while the final evaluation was always conducted on the untouched test set. Repeating this deterministic procedure across multiple runs provides a robust estimate of performance while avoiding excessive randomness in data partitioning.

Algorithm 2 `deterministic_train_test_split_gpu()` – Deterministic Train and Validation Split Using Torch on GPU

```
Require: Input tensors data, labels (on GPU); parameters step, run, val_ratio
Ensure: data_train, data_val, labels_train, labels_val
 1: Get total number of samples n from data.
 2: Compute validation block size based on val_ratio.
 3: Determine starting index using run and step.
 4: Generate validation indices on GPU using modulo indexing.
 5: Create a boolean mask for training data and exclude validation indices.
 6: Extract training and validation subsets for both data and labels.
 7: Return all four tensors for deterministic splitting.
```

During the evaluation, one practical issue was encountered with GPU memory usage. An initial attempt to compute the full pairwise OTW distance matrix between all test and training samples at once quickly exhausted the available GPU memory (6 GB VRAM on an NVIDIA RTX 4050). This happened because storing the intermediate results required more memory than the device could provide. To address this, the evaluation was restructured to process one test sample at a time, compute its distances to the training set, and store only the predicted label. While this slightly increased runtime, it drastically reduced memory consumption and produced identical classification results. This trade-off highlights that OTW implementations on GPUs may require careful memory management, especially on consumer-grade hardware.

The classification accuracy for each run was recorded, and results were reported in the form of mean ± 95% confidence interval (CI95). This format highlights not only the central tendency of the model's performance but also its consistency across multiple runs. The confidence interval was calculated using the standard formula for CI95 under a normal distribution assumption, given by [85]:

$$CI_{95} = \mu \pm 1.96 \times \frac{\sigma}{\sqrt{n}} \tag{6.1}$$

where μ is the mean accuracy, σ is the standard deviation, and n is the number of runs (10 in this case). This approach ensures a statistically sound and fair comparison between OTW and DTW across all datasets.

6.4 Comparison with DTW

Beyond classification accuracy, runtime efficiency is a critical factor in practical deployment. DTW requires filling an $n \times m$ dynamic programming table, yielding quadratic complexity $\mathcal{O}(n^2)$ (see Section 3.4), whereas OTW requires only cumulative sums, resulting in linear complexity $\mathcal{O}(n)$ (see Section 3.23).

Table 6.2 provides empirical runtime measurements for both methods across representative datasets. The results confirm the theoretical analysis: DTW runtimes grow rapidly with sequence length, while OTW remains efficient and scalable.

6.4.1 Performance Table

The table below summarizes classification error rates for both DTW and OTW. It includes:

Table 6.2: Runtime comparison between DTW and OTW implementations (over 10 runs).

Dataset	OTW Runtime (s)	DTW Runtime (s)
ACSF1	0.1	19,726.5
ArrowHead	0.1	268.5
OliveOil	0.1	253.3
Car	0.1	783.5
InsectEPGSmallTrain	0.2	1231.2
Chinatown	0.4	3,5
PowerCons	0.1	447.9
FaceAll	1.8	13,376.3

- The DTW error rate from UCR [7].
- OTW error rate reported by Fabian Latorre's original study [14].
- OTW error rate obtained through our implementation and experiments.
- Train/test performance comparison from our experiment to observe consistency across splits.

Table 6.3: Comparison of DTW and OTW classification error rates across datasets

Dataset	Type	DTW Error	OTW Error [14]	OTW Error (Ours)	Train/Test Consistency
ACSF1	Device	0.38	0.34 ± 0.06	0.35 ± 0.04	**0.35**
ArrowHead	Image	0.20	0.23 ± 0.03	0.12 ± 0.03	0.21
OliveOil	Spectro	0.13	0.14 ± 0.03	0.00 ± 0.00	**0.13**
Car	Sensor	0.23	0.27 ± 0.06	0.13 ± 0.03	0.27
InsectEPGSmallTrain	Medical	0.31	0.00 ± 0.00	0.00 ± 0.00	**0.00**
Chinatown	Traffic	0.05	0.07 ± 0.03	0.00 ± 0.00	0.06
PowerCons	Power	0.08	0.04 ± 0.01	0.04 ± 0.01	**0.03**

This comparison reveals that OTW performs competitively across most datasets, outperforming DTW in some cases (e.g., ArrowHead, InsectEPGSmallTrain, and Chinatown) while underperforming in others (e.g., ACSF1). Notably, in datasets with high structural complexity or strong geometric features, OTW's flexible alignment often proves advantageous.

6.5 Discussion and Interpretation of Results

The results in Table 6.3 provide several important insights into how OTW compares with DTW across different datasets. Overall, these experiments suggest that OTW can be a strong alternative to DTW, often achieving lower classification error rates while also being much more efficient to compute.

One thing I observed is that OTW consistently performs better in datasets where the structural or geometric properties of the signals are most important. For example, in the ArrowHead dataset, OTW achieved an error rate of 0.12 ± 0.03 compared to DTW's 0.20, which shows that OTW is particularly effective in shape-based classification tasks. The advantage is even clearer in the Chinatown dataset, where OTW reduced the error to 0.00 ± 0.00 while DTW reported 0.05. In the InsectEPGSmallTrain dataset, OTW even reached perfect classification across all runs (0.00 ± 0.00), compared to DTW's 0.31. These results made it clear to me that OTW is especially good at capturing global patterns and smoother variations in time series data.

At the same time, there are cases where DTW still holds up quite well. In the ACSF1 dataset, for example, DTW reported an error of 0.38, while OTW was very close at 0.35 ± 0.04. Here the benefit of OTW was not as strong, and I think this is because ACSF1 contains sharp spikes and irregular transitions where DTW's more local warping can sometimes help. A similar trend appeared in OliveOil and Car: in OliveOil, OTW achieved 0.00 ± 0.00 compared to DTW's 0.13, while in Car, OTW reached 0.13 ± 0.03 compared to DTW's 0.23. These cases suggest that in datasets with lower variance or sudden high-amplitude changes, DTW can still be competitive, although OTW usually remains stable.

Another point that stood out to me is the consistency between training and test errors. As shown in Table 6.3, the results across splits were generally well aligned, which suggests that the OTW implementation did not overfit, even when tested repeatedly with deterministic validation splits. This was especially clear in the smaller datasets, such as OliveOil and InsectEPGSmallTrain, where the results barely varied across runs.

Beyond accuracy, runtime efficiency is just as important. The comparisons in Table 6.2 confirmed the theoretical advantage of OTW: while DTW runtimes quickly became very large for longer sequences (for example, several hundred seconds for Car and PowerCons), OTW maintained runtimes that were nearly constant across datasets. When I applied GPU acceleration, the speedups became even more obvious. Table 6.1 shows that tasks which took many seconds on CPU dropped to fractions of a second on GPU. The FaceAll dataset in particular went from over 180 seconds on CPU to under 2 seconds on GPU. What I found interesting here is that GPU acceleration helped not only for large datasets but also for smaller ones, making the PyTorch + CUDA version consistently faster than the NumPy and Pandas implementation on CPU.

In summary, my experiments show that OTW is both fast and reliable, and it clearly has advantages in datasets where structure and smooth alignment matter. DTW is still relevant in some cases, especially with spike-heavy signals, but overall OTW feels like a more flexible and scalable method. From this, I would argue that OTW is a promising candidate for broader use in time series classification, both in research and in practical applications.

7 Conclusion

This thesis explored the application and performance of Optimal Transport Warping (OTW) for time series classification across a diverse set of domains using benchmark datasets from the UCR Time Series Archive. The main question I wanted to answer was whether OTW could serve as a competitive alternative to Dynamic Time Warping (DTW), especially in handling real-world, complex, and variably structured time series.

To address this, I selected seven representative datasets spanning device consumption, image-based shape analysis, spectroscopic data, sensor signals, medical monitoring, urban traffic flow, and energy consumption. Each dataset came with its own challenges, such as long sequence lengths, irregular spikes, class imbalances, or subtle structural differences. Using a consistent experimental framework and a GPU-based implementation in PyTorch, I evaluated the classification accuracy of OTW over multiple runs and compared it against the established DTW benchmarks.

From my experiments, several clear patterns emerged. OTW proved especially effective in datasets where global structure and smooth alignment matter most. For example, ArrowHead and InsectEPGSmallTrain showed large improvements with OTW, and in the Chinatown dataset OTW even reached zero classification error. At the same time, in spike-heavy or low-variance datasets such as ACSF1, OliveOil, and Car, OTW and DTW performed more similarly, suggesting that DTW's more local warping can still be helpful in certain cases. Overall, OTW matched or outperformed DTW in most domains while remaining consistent across repeated runs, which gave me confidence in its robustness.

Another important conclusion is that OTW is not just accurate but also highly efficient. As shown in the runtime comparisons, OTW scales linearly with sequence length, unlike DTW which grows quadratically and quickly becomes impractical for long series. With GPU acceleration through PyTorch and CUDA, the speedups were even more striking: runtimes that took many seconds on CPU dropped to fractions of a second on GPU, and even large datasets such as FaceAll could be processed in under two seconds. This confirmed for me that OTW is not only theoretically scalable but also practically feasible on consumer-grade hardware.

From a methodological perspective, this thesis also contributed a GPU-optimized training and validation pipeline, along with a deterministic splitting strategy to ensure fair and reproducible results. I also had to address practical challenges such as GPU memory limits, which I solved by restructuring the evaluation to process one test sample at a time. Although this slightly increased runtime, it reduced memory consumption dramatically and showed me how important implementation details can be in real-world experiments.

Looking forward, I see several promising directions for future work. Automated parameter tuning could help adapt OTW more flexibly to different datasets, while extensions such as dynamic OTW or smooth differentiable variants could improve both accuracy and usability in machine learning pipelines. Integrating OTW into neural architectures for end-to-end time series classification also seems like a natural next step. Finally, expanding the experimental scope to larger and more diverse datasets, including real-time data sources, would further validate the scalability and adaptability of OTW.

In conclusion, my experiments showed that OTW is a fast, reliable, and competitive method for time series classification. While DTW still has value in certain domains, OTW offers a more flexible and scalable approach, and I believe it has strong potential for future applications in both research and practice.

Bibliography

[1] Gabriel Peyré and Marco Cuturi. "Computational optimal transport". In: *Foundations and Trends in Machine Learning*, 11:355–607, (2019). `doi:10.1561/2200000073`.

[2] Cédric Villani. *"Optimal Transport: Old and New"*, volume 338 of *Grundlehren der mathematischen Wissenschaften.* Springer, Berlin, Heidelberg, (2009). `doi:10.1007/978-3-540-71050-9`.

[3] Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, and Tomaso Poggio. "Learning with a Wasserstein Loss". In *Advances in Neural Information Processing Systems (NeurIPS)*, (2015). URL: `https://dl.acm.org/doi/10.5555/2969442.2969469`.

[4] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. "Optimal Transport for Domain Adaptation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2017). `doi:10.1109/TPAMI.2016.2615921`.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In *Proceedings of the 34th International Conference on Machine Learning (ICML).* PMLR, (2017). URL: `https://proceedings.mlr.press/v70/arjovsky17a.html`.

[6] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery*, (2017). `doi:10.1007/s10618-016-0483-9`.

[7] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. "The UCR Time Series Archive". In: *IEEE/CAA Journal of Automatica Sinica*, (2019). `doi:10.1109/JAS.2019.1911747`.

[8] Hiroaki Sakoe and Seibi Chiba. "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing*, (1978). `doi:10.1109/TASSP.1978.1163055`.

[9] Chotirat Ann Ratanamahatana and Eamonn Keogh. “Three Myths about Dynamic Time Warping Data Mining”. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*. SIAM, (2005). doi:10.1137/1.9781611972757.50.

[10] Stan Salvador and Philip Chan. “Toward accurate dynamic time warping in linear time and space”. In: *Intelligent Data Analysis*, (2007). doi:10.3233/IDA-2007-11508.

[11] Renjie Wu and Eamonn J. Keogh. “FastDTW is approximate and generally slower than the algorithm it approximates”. *IEEE*, (2020). doi:10.1109/TKDE.2020.3033752.

[12] Marco Cuturi and Mathieu Blondel. “Soft-DTW: A Differentiable Loss Function for Time-Series”. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. PMLR, (2017). URL: https://proceedings.mlr.press/v70/cuturi17a.html.

[13] Marco Cuturi. “Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances”. In *Advances in Neural Information Processing Systems*, volume 26, 2013. URL: https://dl.acm.org/doi/10.5555/2999792.2999868.

[14] Fabian Latorre, Chenghao Liu, Doyen Sahoo, and Steven C. H. Hoi. “OTW: Optimal Transport Warping for Time Series”. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023. doi:10.1109/ICASSP49357.2023.10095915.

[15] Lénaïc Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard. “Unbalanced optimal transport: Dynamic and Kantorovich formulations”. In: *Journal of Functional Analysis*, (2018). doi:10.1016/j.jfa.2018.03.008.

[16] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. “Experimental Comparison of Representation Methods and Distance Measures for Time Series Data”. In: *Data Mining and Knowledge Discovery*, (2013). doi:10.1007/s10618-012-0250-5.

[17] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lassaad Idoumghar, and Pierre-Alain Muller. “Deep Learning for Time Series Classification: A Review”. In: *Data Mining and Knowledge Discovery*, (2019). doi:10.1007/s10618-019-00619-1.

[18] Jason Lines, Sarah Taylor, and Anthony Bagnall. “Time Series Classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles”. In: *ACM Transactions on Knowledge Discovery from Data*, (2018). doi:10.1145/3182382.

[19] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures". In: *Proceedings of the VLDB Endowment*, (2008). `doi:10.14778/1454159.1454226`.

[20] Pavel Senin. "Dynamic Time Warping Algorithm Review". Technical report, University of Hawai'i at Mānoa, Department of Information and Computer Sciences, Honolulu, HI, USA, (2008). URL: `https://csdl.ics.hawaii.edu/techreports/2008/08-04/08-04.pdf`.

[21] Yong Keun Jeong, Olufemi A. Omitaomu, et al. "Weighted Dynamic Time Warping for Time Series Classification". In: *Pattern Recognition*, (2011). `doi:10.1016/j.patcog.2010.09.022`.

[22] Lei Chen and Raymond T. Ng. "On the Marriage of L_p-Norms and Edit Distance". In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, Toronto, Canada, (2004). URL: `https://dl.acm.org/doi/abs/10.5555/1316689.1316758`.

[23] Pierre-François Marteau. "Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2009). `doi:10.1109/TPAMI.2008.76`.

[24] Eamonn J. Keogh and Michael J. Pazzani. "Derivative Dynamic Time Warping". In *Proceedings of the 2001 SIAM International Conference on Data Mining (SDM)*, pages 1–11. SIAM, (2001). `doi:10.1137/1.9781611972719.1`.

[25] Marco Cuturi, Jean-Philippe Vert, Oystein Birkenes, and Tomoko Matsui. "A Kernel for Time Series Based on Global Alignments". In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages II–413–II–416, 2007. `doi:10.1109/ICASSP.2007.366260`.

[26] Marco Cuturi. "Fast Global Alignment Kernels". In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, (2011). URL: `https://icml.cc/2011/papers/489_icmlpaper.pdf`.

[27] Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouvé, and Gabriel Peyré. "Interpolating Between Optimal Transport and MMD Using Sinkhorn Divergences". In *Proceedings of AISTATS 2019*, (2019). URL: `https://proceedings.mlr.press/v89/feydy19a.html`.

[28] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, et al. "POT: Python Optimal Transport". In: *Journal of Machine Learning Research*, (2021). URL: `https://dl.acm.org/doi/10.5555/3546258.3546336`.

[29] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. "Kernel Operations on the GPU, with Autodiff, without Memory Overflows". In: *Journal of Machine Learning Research*, 22(74):1–6, 2021. URL: `https://dl.acm.org/doi/10.5555/3546258.3546332`.

[30] Hicham Janati, Marco Cuturi, and Alexandre Gramfort. "Spatio-Temporal Alignments: Optimal Transport through Space and Time". In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, (2020). URL: `https://proceedings.mlr.press/v108/janati20a.html`.

[31] Samuel Cohen, Giulia Luise, Alexander Terenin, Brandon Amos, and Marc Peter Deisenroth. "Aligning Time Series on Incomparable Spaces". In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 130, pages 2089–2097. PMLR, 2021. URL: `https://proceedings.mlr.press/v130/cohen21a.html`.

[32] Lexiang Ye and Eamonn Keogh. "Time Series Shapelets: A New Primitive for Data Mining". In: *Data Mining and Knowledge Discovery*, (2011). `doi:10.1145/1557019.1557122`.

[33] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. "Classification of Time Series by Shapelet Transformation". In: *Data Mining and Knowledge Discovery*, (2013). `doi:10.1007/s10618-013-0322-1`.

[34] Patrick Schäfer. "The BOSS is Concerned with Time Series Classification in the Presence of Noise". In: *Data Mining and Knowledge Discovery*, (2014). `doi:10.1007/s10618-014-0377-7`.

[35] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar A. Zaidi, Bart Goethals, François Petitjean, and Geoffrey I. Webb. "Proximity Forest: An Effective and Scalable Distance-Based Classifier for Time Series". In: *Data Mining and Knowledge Discovery*, (2019). `doi:10.1007/s10618-019-00617-3`.

[36] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lassaad Idoumghar, Pierre-Alain Muller, and François Petitjean. "InceptionTime: Finding AlexNet for Time Series Classification". In: *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020. `doi:10.1007/s10618-020-00710-y`.

[37] Angus Dempster, François Petitjean, and Geoffrey I. Webb. "ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels". In: *Data Mining and Knowledge Discovery*, (2020). `doi:10.1007/s10618-020-00701-z`.

[38] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. "Near-Linear Time Approximation Algorithms for Optimal Transport via Sinkhorn Iterations". In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL: `https://dl.acm.org/doi/10.5555/3294771.3294958`.

[39] Filippo Santambrogio. *"Optimal Transport for Applied Mathematicians"*. Birkhäuser, Cham, (2015). `doi:10.1007/978-3-319-20828-2`.

[40] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *"Gradient Flows in Metric Spaces and in the Space of Probability Measures"*. Birkhäuser, Basel, 2 edition, (2008). URL: `https://www2.stat.duke.edu/~sayan/ambrosio.pdf`.

[41] Justin Solomon, Fernando de Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. "Convolutional Wasserstein Distances: Efficient Optimal Transport on Geometric Domains". In: *ACM Transactions on Graphics*, 34(4):66:1–66:11, (2015). `doi:10.1145/2766963`.

[42] Leonid V. Kantorovich. "On the Translocation of Masses". In: *Journal of Mathematical Sciences*, 133(4):1381–1382, (2006). `doi:10.1007/s10958-006-0049-2`.

[43] Robert J. McCann. "A Convexity Principle for Interacting Gases". In: *Advances in Mathematics*, 128(1):153–179, (1997). `doi:10.1006/aima.1997.1634`.

[44] Svetlozar T. Rachev and Ludger Rüschendorf. *"Mass Transportation Problems: Volume I: Theory"*. Springer, New York, (1998). `doi:10.1007/b98893`.

[45] Yann Brenier. "Polar Factorization and Monotone Rearrangement of Vector-Valued Functions". In: *Communications on Pure and Applied Mathematics*, 44(4):375–417, (1991). `doi:10.1002/cpa.3160440402`.

[46] Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. "Iterative Bregman Projections for Regularized Transportation Problems". In: *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, (2015). `doi:10.1137/141000439`.

[47] Clark R. Givens and Rae Michael Shortt. "A Class of Wasserstein Metrics for Probability Distributions". In: *Michigan Mathematical Journal*, 31(2):231–240, (1984). `doi:10.1307/mmj/1029003026`.

[48] Matt J. Kusner, Yu Sun, Nicholas Kolkin, and Kilian Q. Weinberger. "From Word Embeddings to Document Distances". In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 957–966, (2015). URL: `https://proceedings.mlr.press/v37/kusnerb15.html`.

[49] Alison L. Gibbs and Francis E. Su. "On Choosing and Bounding Probability Metrics". In: *International Statistical Review*, 70(3):419–435, (2002). `doi:10.1111/j.1751-5823.2002.tb00178.x`.

[50] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. "The Earth Mover's Distance as a Metric for Image Retrieval". In: *International Journal of Computer Vision*, 40(2):99–121, (2000). `doi:10.1023/A:1026543900054`.

[51] Ofir Pele and Michael Werman. "Fast and Robust Earth Mover's Distances". In *2009 IEEE 12th International Conference on Computer Vision (ICCV)*, pages 460–467, (2009). `doi:10.1109/ICCV.2009.5459199`.

[52] Matthias Liero, Alexander Mielke, and Giuseppe Savaré. "Optimal Entropy-Transport Problems and a New Hellinger-Kantorovich Distance Between Positive Measures". In: *Inventiones Mathematicae*, 211(3):969–1117, (2018). `doi:10.1007/s00222-017-0759-8`.

[53] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics*, 35(1):73–101, (1964). `doi:10.1214/aoms/1177703732`.

[54] Marco Cuturi, Laetitia Meng-Papaxanthos, Yingtao Tian, Charlotte Bunne, Geoff Davis, and Olivier Teboul. "Optimal Transport Tools (OTT): A JAX Toolbox for All Things Wasserstein". *ArXiv*, abs/2201.12324, 2022. URL: `https://api.semanticscholar.org/CorpusID:246411679`.

[55] Xinyu Li, Xiaoyuan Zhang, Xin Chen, Xun Chen, and Aiguo Liu. "Cross-user Gesture Recognition from sEMG Signals Using an Optimal Transport Assisted Student–Teacher Framework". In: *Computers in Biology and Medicine*, 165:107327, (2023). `doi:10.1016/j.compbiomed.2023.107327`.

[56] Bin Xue, Liang Wu, Ke Wang, Xiaoyuan Zhang, Jun Cheng, and Xin Chen. "Multiuser Gesture Recognition Using sEMG Signals via Canonical Correlation Analysis and Optimal Transport (CCA-OT)". In: *Computers in Biology and Medicine*, 134:104560, (2021). `doi:10.1016/j.compbiomed.2020.104188`.

[57] Shujian Liao, Hao Ni, Łukasz Szpruch, Magnus Wiese, Marc Sabate-Vidales, and Baoren Xiao. "Conditional Sig-Wasserstein GANs for Time Series Generation". In *Proceedings of the 1st ACM International Conference on AI in Finance (ICAIF 2020)*, 2020. `doi:10.1145/3490354.3494393`.

[58] Jiacheng Zhu, Jielin Qiu, Zhuolin Yang, Douglas Weber, Michael A. Rosenberg, Emerson Liu, Bo Li, and Ding Zhao. "GeoECG: Data Augmentation via Wasserstein Geodesic Perturbation for Robust Electrocardiogram Prediction". In *Proceedings of the 5th Conference on Health, Inference, and Learning (CHIL 2022)*, volume

182, pages 376–390. PMLR, 2022. URL: `https://proceedings.mlr.press/v182/zhu22a.html`.

[59] Mitsuhiko Horie and Hiroyuki Kasai. “Auto-weighted Sequential Wasserstein Distance and Application to Sequence Matching”. In *EUSIPCO 2022*, pages 1472–1476, (2022). `doi:10.23919/EUSIPCO55093.2022.9909780`.

[60] Natalia Kravtsova, Reginald L. McGee II, and Adriana T. Dawes. “Scalable Gromov–Wasserstein Based Comparison of Biological Time Series”. In: *Bulletin of Mathematical Biology*, 85(8):77, (2023). `doi:10.1007/s11538-023-01175-y`.

[61] Tom M. Mitchell. *“Machine Learning”*. McGraw-Hill, New York, (1997). URL: `https://www.cs.cmu.edu/~tom/mlbook.html`.

[62] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *“The Elements of Statistical Learning: Data Mining, Inference, and Prediction”*. Springer, 2 edition, (2009). `doi:10.1007/978-0-387-84858-7`.

[63] Jiawei Han, Micheline Kamber, and Jian Pei. *“Data Mining: Concepts and Techniques”*. Morgan Kaufmann, 3rd edition, 2012. `doi:10.1016/C2009-0-61819-5`.

[64] Fabrizio Sebastiani. “machine learning in automated text categorization”. *ACM Computing Surveys*, 34(1):1–47, 2002. `doi:10.1145/505282.505283`.

[65] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. “Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers”. In: *Journal of Machine Learning Research*, 1:113–141, (2000). URL: `https://dl.acm.org/doi/abs/10.1162/15324430152733133`.

[66] Ryan Rifkin and Aldebaro Klautau. “In Defense of One-Vs-All Classification”. In: *Journal of Machine Learning Research*, 5:101–141, (2004). URL: `https://www.jmlr.org/papers/v5/rifkin04a.html`.

[67] Chih-Wei Hsu and Chih-Jen Lin. “A Comparison of Methods for Multiclass Support Vector Machines”. In: *IEEE Transactions on Neural Networks*, 13(2):415–425, (2002). `doi:10.1109/72.991427`.

[68] Trevor Hastie and Robert Tibshirani. “Classification by Pairwise Coupling”. In: *The Annals of Statistics*, 26(2):451–471, (1998). `doi:10.1214/aos/1028144844`.

[69] Haibo He and Edwardo A. Garcia. “Learning from Imbalanced Data”. In: *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, (2009). `doi:10.1109/TKDE.2008.239`.

[70] Marina Sokolova and Guy Lapalme. "A Systematic Analysis of Performance Measures for Classification Tasks". In: *Information Processing & Management*, 45(4):427–437, (2009). doi:10.1016/j.ipm.2009.03.002.

[71] Christopher M. Bishop. *"Pattern Recognition and Machine Learning"*. Springer, New York, (2006). URL: https://link.springer.com/book/9781493938438.

[72] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *"Classification and Regression Trees"*. Wadsworth, Belmont, CA, (1984). doi: 10.1201/9781315139470.

[73] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning*, 20(3):273–297, (1995). doi:10.1007/BF00994018.

[74] Thomas M. Cover and Peter E. Hart. "Nearest Neighbor Pattern Classification". In: *IEEE Transactions on Information Theory*, 13(1):21–27, (1967). doi:10.1109/TIT.1967.1053964.

[75] Naomi S. Altman. "an introduction to kernel and nearest-neighbor nonparametric regression". In:*The American Statistician*, 46(3):175–185, 1992. doi:10.1080/00031305.1992.10475879.

[76] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. "When Is Nearest Neighbor Meaningful?". In *Proceedings of the 7th International Conference on Database Theory (ICDT)*, (1997). URL: https://www.researchgate.net/publication/2845566.

[77] Lawrence R. Rabiner. "a tutorial on hidden markov models and selected applications in speech recognition". In: *Proceedings of the IEEE*, 77(2):257–286, 1989. doi: 10.1109/5.18626.

[78] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. "Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping". In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 262–270, (2012). doi:10.1145/2339530.2339576.

[79] Christophe Gisler, Antonio Ridi, and Jean Hennebert. "Appliance Consumption Signature Database and Recognition Test Protocols". In *2013 8th International Workshop on Systems, Signal Processing and their Applications (WoSSPA)*, pages 336–341, (2013). doi:10.1109/WoSSPA.2013.6602387.

[80] Henri S. Tapp, Marianne Defernez, and E. Katherine Kemsley. "FTIR Spectroscopy and Multivariate Analysis Can Distinguish the Geographic Origin of Extra Virgin

Olive Oils". In: *Journal of Agricultural and Food Chemistry*, 51(21):6110–6115, (2003). doi:10.1021/jf030232s.

[81] Ninad Thakoor and Jean Gao. "Shape Classifier Based on Generalized Probabilistic Descent Method with Hidden Markov Descriptor". In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV)*, pages 495–502, (2005). doi:10.1109/ICCV.2005.220.

[82] Derek S. Willett, Jason George, Nathan S. Willett, Lukasz L. Stelinski, and Stephen L. Lapointe. "Machine Learning for Characterization of Insect Vector Feeding". In: *PLOS Computational Biology*, 12(11):e1005158, (2016). doi:10.1371/journal.pcbi.1005158.

[83] City of Melbourne. "Pedestrian Counting System — Monthly Counts per Hour (Chinatown / Swanston St North)". Open Data Portal, (2017). Accessed 2025-09-09. URL: https://www.data.gov.au/data/dataset/melbourne-pedestrian-counting-system-monthly-counts-per-hour.

[84] UEA/UCR Time Series Classification Repository. "PowerCons Dataset Description". UCR/UEA Time Series Classification Repository, (2018). Accessed 2025-09-09. URL: https://www.timeseriesclassification.com/description.php?Dataset=PowerCons.

[85] Larry Wasserman. *"All of Statistics: A Concise Course in Statistical Inference"*. Springer, New York, (2004). doi:10.1007/978-0-387-21736-9.