UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Machine Learning-based Characterization of Single-Particle Behavior with Synthetic Experiment Videos

End of Degree Project

Bachelor's Degree in Data Science

Author: Ahsini Ouariaghli, Yusef

Tutor: Conejero Casares, José Alberto

Course 2023-2024

Resumen

Este Trabajo de Fin de Grado se centra en examinar el movimiento de partículas individuales dentro de células vivas, utilizando videos de experimentos. Este trabajo es parte de la participación en el 2nd Anomalous Diffusion (AnDi) Challenge. En ella, disponemos de videos sin procesar de varias regiones o FOVs de un experimento. Dicho experimento es un escenario biológico específico definido mediante un modelo de interacciones y un conjunto de parámetros que describen la interacción dinámica de las partículas y el entorno. El objetivo es emplear varios modelos de aprendizaje automático y algoritmos de seguimiento de video para participar en las dos tareas de la competición.

La primera tarea es la Ensemble Task, que se centra en proporcionar predicciones a nivel de conjunto reconociendo los modelos subyacentes utilizados para simular el comportamiento de las partículas en el experimento. Esto abarca cinco modelos fenomenológicos: modelo de estado único (SSM), modelo de estado múltiple (MSM), modelo de dimerización (DIM), modelo de confinamiento transitorio (TCM) y modelo de trampa enfriada (QTM), junto con la distribución de coeficientes de difusión y exponentes en diferentes condiciones experimentales.

La segunda, la Single-trajectory Task implica un análisis más detallado de las trayectorias de partículas individuales en cada FOV. Aquí, el objetivo es identificar puntos de cambio dentro de las trayectorias y caracterizar los coeficientes de difusión, exponentes y restricciones ambientales.

Palabras clave: Difusión Anómala, Seguimiento de Partículas, Visión Computacional, Aprendizaje Automático, Attention U-Nets

Abstract

This Bachelor Thesis concentrates on examining the movement of individual particles within living cells, utilizing videos of experiments. This work is linked to the participation in the 2nd Anomalous Diffusion (AnDi) Challenge. Raw videos from various regions or FOVs of an experiment are available. This experiment constitutes a specific biological scenario defined by a model of interactions and a set of parameters describing the dynamic interaction of particles and the environment. The aim is to employ Machine Learning models and video tracking algorithms to participate in the two tasks of the challenge.

The first task is the Ensemble Task, focusing on providing ensemble-level predictions by recognizing the underlying models used to simulate particle behavior in the experiment. This spans five phenomenological models: Single-state model (SSM), Multi-state model (MSM), Dimerization model (DIM), Transient-confinement model (TCM), and Quenched-trap model (QTM), alongside the distribution of diffusion coefficients and exponents across different experimental conditions.

On the other hand, the Single-trajectory Task involves a more detailed examination of individual particle trajectories in each FOV. Here, the objective is to identify changing points within trajectories and characterize the diffusion coefficients, exponents, and environmental constraints.

Key words: Anomalous diffusion, Single-Particle-tracking, Computer Vision, Machine Learning, Attention U-Nets

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

Understanding the motion of individual particles is critical for deciphering various biological and physical processes. Cell imaging, in combination with single-particle tracking in combination, has become indispensable in this regard, offering insights into the dynamics of molecules such as proteins, lipids, and other biomolecules within complex environments [1].

In the study of diffusion processes, Brownian motion is often the first example that comes to mind. In Brownian motion, the mean square displacement (MSD) of particles increases linearly with time, reflecting a simple, predictable pattern of movement. However, many systems exhibit more complex behavior known as anomalous diffusion, where this linear relationship no longer holds. Instead, the MSD varies with time according to a power law, with an exponent α that characterizes the nature of the diffusion—whether it is faster (superdiffusive) or slower (subdiffusive) than normal.

Understanding and predicting the anomalous exponent α and the generalized diffusion coefficient K are crucial for characterizing these complex systems. Unlike normal diffusion, where the relationship between MSD and time is straightforward, anomalous diffusion involves intricate dynamics, often influenced by how long a particle lingers in one spot before moving on or the interaction with the rest of the particles. This behavior must be carefully considered to distinguish between different modes of motion and identify transitions in a particle's trajectory.

1.1 Motivation

In recent years, the advancement of machine learning algorithms and the significant reduction in computational power costs have revolutionized their application across various research domains. This technological leap has made it feasible to tackle more complex problems. [2, 3].

For instance, in the field of material science, researchers are now employing machine learning to predict the properties of new materials by analyzing vast datasets of experimental results [4]. Similarly, in the medical field, these algorithms are being used to identify patterns in medical imaging that can lead to earlier and more accurate diagnoses [5].

One particularly intriguing application of these advancements is in the characterization of anomalous diffusion coefficients in particles. Traditionally, studying diffusion patterns required extensive and time-consuming experiments, but now, machine learning models can analyze particle movement data with high precision and efficiency.

An example of this is the AnDi Challenge 2020, which centered around this task. As outlined by [6] (hot paper according to Web of Science), the challenge involved characterizing particles in 1D, 2D, or 3D and included various tasks, such as inferring the anomalous diffusion exponent α, classifying diffusion models, and segmenting trajectories. Participants were also given a python library to generate synthetic experiments through simulation, and validate their methods using real-world particle tracking datasets.

The main **motivation** for this **End-of-Degree project** arises from the launch of the **2nd Andi Challenge** in December 2023 [1]. This challenge introduced several updates, including new phenomenological models, additional tracks, and tasks. It also expanded the competition to require the inference of not only the anomalous exponent α but also the generalized diffusion coefficient K. Of particular relevance to this project is the introduction of the **video track**, where instead of working with raw particle trajectories, the input of the models will be videos of the particles in motion, providing the framework for the development of this work.

1.2 Objectives

The objective is to **create a functional model** for the **video track** of the **2nd Andi Challenge**, combining machine learning techniques acquired during the Bachelor's program with video tracking and change-point detection algorithms. The final model should aim to achieve competitive results in the challenge's validation stage, with the goal of securing a **Top-5 position in any of the video tasks**. Meeting this target leads to the model being included in the competition's article in principle accepted at **Nature Communications** and co-authorship on it.

1.3 Structure of the memory

The memory is structured in the following parts: The structure of this memory is organized into the following chapters:

- **Chapter 2** introduces the concept of anomalous diffusion in the field of physics. It delves into the theory behind Fractional Brownian Motion (FBM) and provides a comprehensive overview of different models of motion, highlighting their relevance to anomalous diffusion phenomena.
- **Chapter 3** is dedicated to the description of the 2nd ANDI Challenge. This chapter outlines the nature and objectives of the challenge, the dataset provided, and the evaluation framework employed and the established metrics.

- **Chapter 4** presents a detailed review of the state-of-the-art techniques and methodologies relevant to the topic. It surveys recent developments, key research papers, and advances in the modeling and analysis of anomalous diffusion.
- **Chapter 5** discusses the methodology developed and applied in this work. It includes an in-depth explanation of the techniques utilized, the rationale behind their selection, and the step-by-step process of implementation, providing insights into the challenges and solutions encountered during the process.
- **Chapter 6** focuses on the validation of the proposed methodology. This chapter explains the validation process, presenting the criteria and metrics used to assess the performance of the methodology, along with a discussion of the results obtained from this validation.
- **Chapter 7** presents and analyzes the results obtained from participating in the 2nd ANDI Challenge.
- **Chapter 8** concludes the memory by summarizing the key findings and results of the work. Additionally, it reflects on the relationship between the research conducted and of studies pursued.

CHAPTER 2
Anomalous Diffusion

In physics, diffusion refers to the process by which particles spread out over time, moving from regions of higher concentration to regions of lower concentration. The standard model of diffusion, known as Brownian motion, describes this spread as a random walk. In this model, the mean squared displacement (MSD) of particles grows linearly with time. Mathematically, this relationship is expressed as:

$$\langle \Delta r^2(t) \rangle = 2dDt$$

where:

- $r(t)$ indicates the displacement at time t respect of the origin of the particle. $\langle \Delta r^2(t) \rangle$ is the mean squared displacement after a time t.
- D is the diffusion coefficient, which measures how fast particles are diffusing.
- d is the spatial dimension (e.g., $d = 1$ for one-dimensional motion, $d = 2$ for two-dimensional motion).
- t t is the time elapsed.

In normal diffusion, the MSD grows linearly with time, indicating that particles move in a random and uncorrelated manner. This type of diffusion is typically observed in simple liquids and gases, where there are no significant obstacles or interactions affecting the particles' motion and it is called Brownian motion, as it was first observed by R. Brown in 1827 looking at how pollen grains move under a microscope. Almost one century later, Einstein modeled that these particles were moved by water molecules [7].

However, in many complex systems, such as in crowded environments, porous media, or biological systems, the diffusion process deviates from this standard behavior, leading to what is known as *anomalous diffusion*. In anomalous diffusion, the MSD does not grow linearly with time but instead follows a power-law dependence:

$$\langle \Delta r^2(t) \rangle = 2dK_\alpha t^\alpha$$

where:

- K_α is the generalized diffusion coefficient, which encapsulates the effects of the medium on diffusion.
- α is the anomalous diffusion exponent, which characterizes the deviation from normal diffusion and are numbers belonging to $]0, 2]$ and different from 1.

The value of α determines the type of diffusion:

- **Normal diffusion** occurs when $\alpha = 1$, which is the standard Brownian motion case with linear growth of MSD.
- **Subdiffusion** occurs when $\alpha < 1$, where the MSD grows more slowly than linear time, often due to obstacles, trapping, or interactions that hinder particle movement.
- **Superdiffusion** occurs when $2 > \alpha > 1$, where the MSD grows faster than linear time, which may be associated with active transport mechanisms or long-range correlations in the system. **Direct diffusion** occurs when $\alpha = 2$.

A prominent model for describing anomalous diffusion is Fractional Brownian Motion (FBM). FBM extends the concept of Brownian motion by introducing correlations between the increments of a particle's path. This model is particularly useful for describing various complex processes observed in biological systems [1, 8, 9]. Mathematically, a FBM where displacement at time t is denoted by $B_H(t)$ is characterized by:

- $B_H(0) = 0$: The process starts at the origin.
- $\mathbb{E}[B_H(t)] = 0$: The expected value of the process is zero, indicating no systematic drift.
- $\mathbb{E}[B_H(t)B_H(s)] = K\left(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\right)$: This describes the correlation structure of FBM, where k is a constant and H is the Hurst exponent.

We recall that $\mathbb{E}[]$ denotes the expected value, and K is a constant with units length^2 * time^{-2H} [1]. Here, the Hurst exponent, which lies in the range $0 < H < 1$, determines the nature of the correlations:

- If $H = 0.5$, the movement reduces to a standard Brownian motion, with no correlations (i.e., a random uncorrelated motion).
- If $H > 0.5$, the process shows positive correlations in the increments, meaning that if a particle moves in one direction, it is likely to continue in that direction, leading to superdiffusion ($\alpha > 1$).
- If $H < 0.5$, the process shows negative correlations in the increments, meaning that if a particle moves in one direction, it is more likely to reverse its motion, resulting in subdiffusion ($\alpha < 1$).

The anomalous diffusion exponent α is related to the Hurst exponent as $\alpha = 2H$ [10]. For an unconstrained fractional Brownian motion (FBM) in 2D, the mean squared displacement (MSD) scales with time t according to

$$\mathrm{MSD}(t) = 4Kt^{\alpha}[1].$$

where K is the diffusion coefficient and $\alpha = 2H$ determines the type of diffusion exponent based on the value of the Hurst exponent [1].

In many physical systems, the environment's complexity leads to a range of diffusion behaviors among different particles, resulting in a distribution of exponents α and coefficients K_α. Some particles may exhibit subdiffusive behavior due to localized trapping, while others may demonstrate superdiffusion driven by persistent directional motion. These variations reflect the non-uniform and heterogeneous nature of the environments in which anomalous diffusion occurs. In the context of the challenge, 5 different physical models of motion and interactions are considered (Figure 2.1):

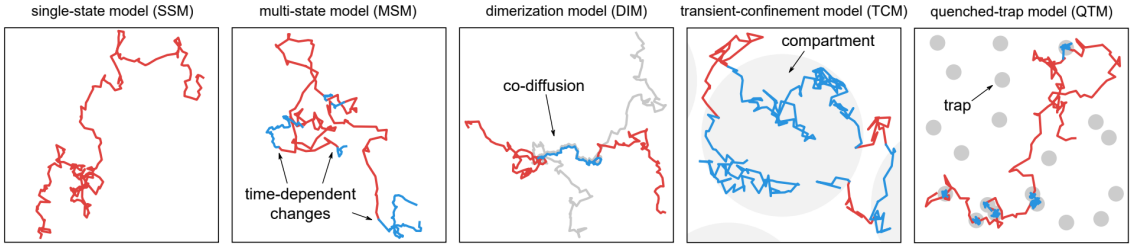

Figure 2.1: Visual representation of the physical models of motion and interactions. Source: [1].

- **Single-State Model (SSM)** — This model describes particles moving within a single diffusion state, similar to the behavior observed in certain lipids in the plasma membrane. It is often used as a negative control to assess the accuracy of detecting changes in diffusion. [11]

- **Multi-State Model (MSM)** — In this model, particles diffuse according to a time-dependent, multi-state diffusion process, involving two or more states. These states can exhibit temporary changes in the diffusion coefficient (K) and/or the anomalous exponent (α). Such changes in K are seen in proteins due to factors like allosteric changes or ligand interactions. [12]

- **Dimerization Model (DIM)** — This model describes how particles diffuse in a 2-state process, where interactions with other particles lead to transient changes in K and/or α. Such variations in K have been observed during protein dimerization and other protein-protein interactions. [13]

- **Transient-Confinement Model (TCM)** — This model depicts particles diffusing within a spatially-dependent, 2-state process. It is seen when proteins are temporarily confined in regions with different diffusion properties, such as those created by clathrin-coated pits on the cell membrane. When there is a high density of these confinement regions, the model can mimic the picket-and-fence effect, which explains how the actin cytoskeleton influences transmembrane protein movement. [14]

- **Quenched-Trap Model (QTM)** — This model involves particles diffusing in a space-dependent, 2-state process where proteins become temporarily immobilized at specific sites due to binding with immobile structures, like those caused by cytoskeletal pinning. [16]

CHAPTER 3
The 2nd Andi Challenge

This section offers a comprehensive overview of the 2nd Andi Challenge[1], outlining the various tasks involved, which are the main objectives of this bachelor thesis. It also includes details on the structure of the provided data and the evaluation methodology.

3.1 Tasks

The challenge is organized along two tracks. The organizers provide the data-generating models for training the machine-learning models and try to replicate live-cell molecule images. *Track 1* is based on the analysis of raw videos, and *Track 2* is based on the analysis of trajectories. In each track, participants can compete in two different tasks:

- **Ensemble Task** — ensemble-level predictions, providing for each experimental condition the model used to simulate the experiment, the number of states, and the fraction of time spent in each state. For each identified state, participants should determine the mean and standard deviation of the distribution of the generalized diffusion coefficients K, and the mean and standard deviation of the distribution of the anomalous diffusion exponent α corresponding to the underlying motion.

- **Single-trajectory Task** — trajectory-level predictions, providing for each trajectory a list of M inner CPs delimiting $M+1$ segments with different dynamic behavior. For each segment, participants should identify the generalized diffusion coefficient K, the anomalous diffusion exponent α corresponding to the underlying motion, and an identifier of the kind of constraint imposed by the environment (0 = immobile (QTM model), 1 = confined (TCM model), 2 = free (unconstrained), 3 = directed ($\alpha > 1.9$)). For Track 1, predictions must be provided for a subset of particles (we will refer to them as VIP, very important particles) identified through a label map of the first frame of the movie. For Track 2, predictions must be provided for all the trajectories.

[1] https://codalab.lisn.upsaclay.fr/competitions/16618

Figure 3.1 presents a visual representation of the challenge structure, illustrating the data utilized in each track and the associated tasks.

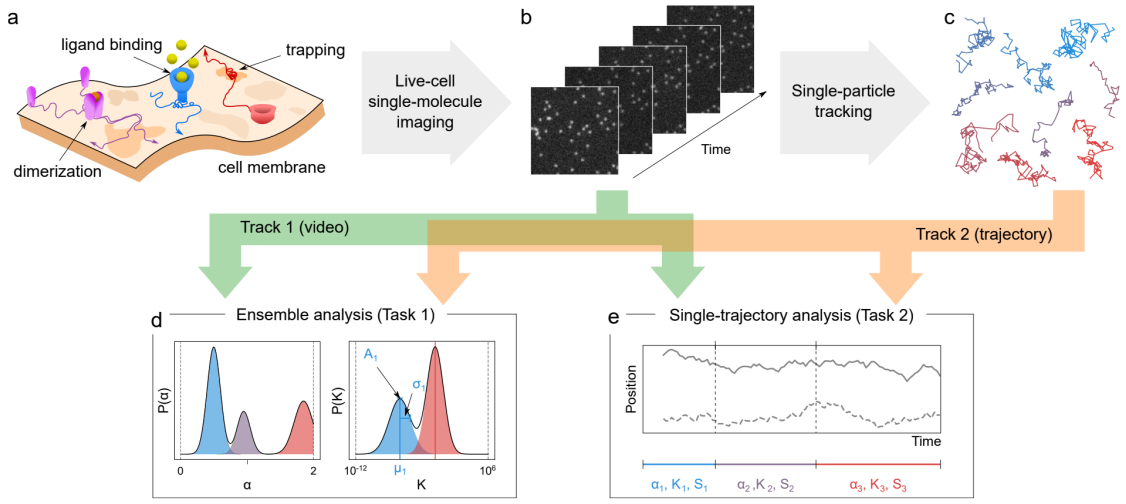

Figure 3.1: Illustrative representation of the 2nd AnDi Challenge tracks (b, c) and tasks (d, e). Source: [1].

3.2 Data

The datasets for the competition consist of various experiments, each stored in a folder labeled sequentially (e.g., EXP_1, EXP_2). These folders correspond to different models with fixed but unknown parameters. Inside each experiment folder, there are files named sequentially (e.g., FOV_1, FOV_2), each containing data from 30 fields of view (FOVs). A FOV is a specific area within the experiment where data on particle movement is captured as shown in figure 3.2. Each FOV captures data on particles diffusing within a 128 × 128 pixel area.

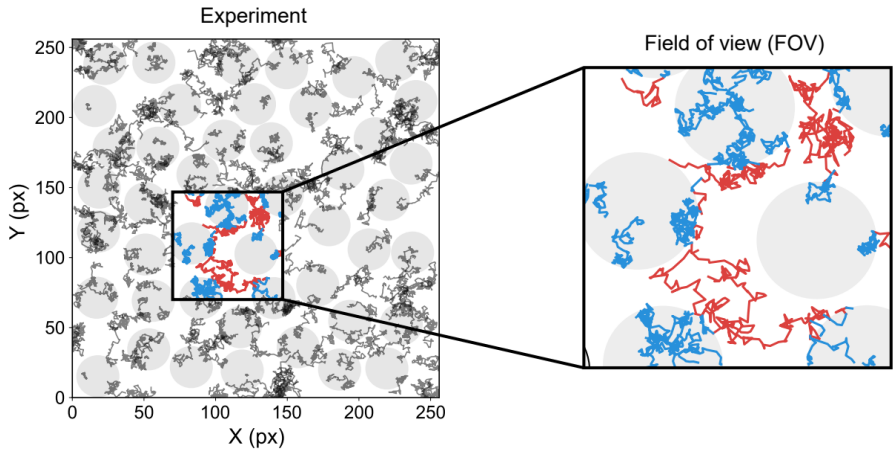

Figure 3.2: Illustrative representation of a field of view in an experiment. Source: [1].

For Track 1, the particle coordinates in each FOV are used to create 200-frame videos, stored as a series of 8-bit images in multi-tiff format. These synthetic images include noise to simulate background fluorescence and shot noise, following

a Poisson distribution. Additionally, there is an extra image representing a map of the VIP particles. In this track, only the 10 VIP particles need to be predicted, and they are identified by unique integer values presented in this frame.

For Track 2, each FOV has an associated CSV file containing a table with columns for trajectory index, time step, x-coordinate, and y-coordinate. The simulated trajectories are corrupted with Gaussian noise, reflecting the finite precision of localization, and can have a maximum length of 200 frames.

The competition provides a starter kit that includes labeled data from 5 different experiments, each containing 20 FOVs. Additionally, a Python library is available to generate datasets through simulation for both tracks [15]. The range of values considered for α is between 0 and 2, while for the k coefficient, the values are between 10^{-12} and 10^{6}.

3.3 Evaluation Methodology

3.3.1. Ensemble Tasks Evaluation

For Ensemble Tasks, participants were required to submit predictions, including the model type used for each experiment, the number of states S, and the parameters of each state. While the correctness of the model type was noted, it did not influence the ranking. The number of states S was evaluated by comparing the predictions to the ground truth. For both the generalized diffusion coefficient K and the anomalous diffusion exponent α, participants provided the mean, standard deviation, and relative weight for each state, from which multi-modal distributions $P(K)$ and $P(\alpha)$ were constructed. The similarity of these distributions to the ground-truth distributions $Q(K)$ and $Q(\alpha)$ was quantified using the first Wasserstein distance:

$$W_1(P, Q) = \int_{\text{supp}(Q)} \left|\text{CDF}_P(x) - \text{CDF}_Q(x)\right| \, dx$$

where CDF stands for the cumulative distribution function and $\text{supp}(Q)$ stands for the smaller closed set where Q do not anihilate.

3.3.2. Single-trajectory Tasks Evaluation

For single-trajectory tasks, participants have to predict the number of changepoints (CPs) M and the dynamic properties of the resulting $M + 1$ segments in which the changepoints split the trajectory, which include the generalized diffusion coefficient K, the anomalous exponent α, and the constraint imposed by the environment in each segment. The evaluation involved various metrics to assess the accuracy of these predictions, which were stated by the organizers of the challenge.

- **Changepoint Detection** Changepoint detection was evaluated by comparing predicted CPs to ground-truth CPs. It can happen that we were not able

to predict the same number of CPs. So that, the Hungarian algorithm was firstly employed to solve the assignment problem, minimizing the sum of distances between paired CPs:

$$d_{\mathrm{CP}} = \min\left(\sum d_{i,j}\right).$$

Once we have pared the i-th ground-truth CP at $t_{(\mathrm{GT}),i}$ and the j-th predicted CP at $t_{(\mathrm{P}),j}$, the gated absolute distance these CPs is defined as:

$$d_{i,j} = \min\left(|t_{(\mathrm{GT}),i} - t_{(\mathrm{P}),j}|, \epsilon_{\mathrm{CP}}\right),$$

where $\epsilon_{\mathrm{CP}} = 10$ was used as a maximum penalty for CPs located beyond this threshold.

As we can have correct and incorrectly determined CPs we compute true positives (TP), false positives (FP), and false negatives (FN) and then the Jaccard similarity coefficient (JSC) was computed as:

$$\mathrm{JSC} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP}}.$$

in order to measure the dissimilarity of the predictions respect to the ground truth.

For CPs classified as TP, the root mean square error (RMSE) was calculated as:

$$\mathrm{RMSE} = \sqrt{\frac{\sum\left(t_{(\mathrm{GT}),i} - t_{(\mathrm{P}),j}\right)^2}{N}},$$

where N represents the number of paired CPs.

- **Dynamic Property Estimation** The evaluation of dynamic properties involved pairing predicted and ground-truth segments based on their JSC. The Hungarian algorithm maximized the JSC sum, and only paired segments were considered for further metric calculations. For paired segments i from the ground truth GT and j from the predictions P, the mean squared logarithmic error (MSLE) for K was calculated as:

$$\mathrm{MSLE} = \frac{\sum_{\mathrm{pairs}(i,j)}\left[\left(\log(K_{(\mathrm{GT}),i}) + 1\right) - \log(K_{(\mathrm{P}),j} + 1)\right)^2\right]}{N}.$$

For the anomalous diffusion exponent α, the mean absolute error (MAE) was computed as:

$$\mathrm{MAE} = \frac{\sum|\alpha_{(\mathrm{GT}),i} - \alpha_{(\mathrm{P}),j}|}{N},$$

where N is the number of paired segments. The reason for using MSLE for k is that it handles a wide range of possible values, both small and large,

without biasing the model toward predicting higher values. The diffusion-type classification was evaluated using the F1-score:

$$\mathrm{F1} = \frac{2\mathrm{TP}_c}{2\mathrm{TP}_c + \mathrm{FP}_c + \mathrm{FN}_c},$$

where TP_c, FP_c, and FN_c denote true positives, false positives, and false negatives for the classification.

CHAPTER 4
State of the Art

4.1 Introduction

This chapter reviews the state of the art in 3 key areas for participation in the challenge: particle tracking algorithms, methods for detecting change points in time series, and machine learning models. These areas are the fundamental bases of the methodology showcased in the next chapter.

4.2 Particle Tracking

Particle tracking is an essential technique for studying the movement of particles within various media. The development of robust and accurate particle tracking algorithms is vital for analyzing experimental data, especially when dealing with noisy environments or crowded fields [17]. In the case of this work, it is essential to be able to extract the trajectories from the experiment videos. Over the years, several algorithms have been developed, each with its own strengths and limitations.

4.2.1. Centroid-Based Algorithms

Centroid-based algorithms enhance particle detection by improving upon simple thresholding methods. Instead of relying solely on intensity thresholds to detect particles, these algorithms calculate the center of mass, or centroid, of detected particles, based on the intensity distribution within a defined region of interest [18]. This approach makes the method more robust against noise, variations in particle intensity, and changes in illumination, as the centroid calculation inherently considers the full intensity profile of a particle. Additionally, centroid-based algorithms are computationally efficient, allowing for accurate particle localization without significant processing overhead, even in challenging imaging conditions.

One widely used example is the Crocker-Grier algorithm, developed by these authors in 1996 [19]. This method combines Gaussian filtering with local maxima detection to identify particles across a sequence of images. The Gaussian

filter smooths the image, reducing noise and it enhances particle-like features, making it easier to detect local intensity maxima corresponding to particle positions. Once particles are identified in individual frames, the algorithm links them into trajectories by calculating the displacement of particles between consecutive frames. It uses a cost matrix to find the optimal linking, minimizing the overall displacement while allowing for some tolerance to missing particles or false positives due to noise.

This algorithm is particularly effective in handling overlapping particles to a certain extent, making it robust in complex datasets. Its ability to maintain high accuracy while keeping computational demands low has made it popular in real-time applications and large datasets, where both precision and speed are crucial. The Crocker-Grier algorithm strikes a balance between handling noise and providing accurate particle tracking, making it adaptable to a variety of experimental conditions and imaging environments.

4.2.2. Feature-Based Tracking

Feature-based tracking algorithms offer a more sophisticated approach to particle tracking by extracting distinctive features, such as intensity, size, shape, and texture, from individual particles to track them across time. This method allows for greater flexibility in identifying and following particles in dynamic and complex environments. Unlike threshold-based methods that rely solely on intensity, feature-based tracking can distinguish between particles based on multiple characteristics, making it highly effective in scenarios where particles may overlap, occlude each other, or exhibit variations in appearance.

Popular techniques within this category include the Scale-Invariant Feature Transform (SIFT) [20] and Speeded-Up Robust Features (SURF) [21]. These algorithms extract unique and stable features from each particle, ensuring that the same particle can be identified and tracked across different frames, even under challenging conditions. SIFT, for instance, detects key points in an image that is invariant to scaling, rotation, and slight changes in illumination, making it ideal for complex environments. Similarly, SURF improves upon SIFT by offering faster computation while maintaining robust feature detection. Both techniques are particularly useful in crowded environments, where particles may overlap or occlude one another, as they focus on unique feature patterns rather than relying solely on intensity thresholds.

By leveraging these feature extraction methods, feature-based tracking algorithms can maintain accurate tracking even in cluttered or noisy datasets, where simpler methods might fail. These algorithms provide an effective solution for complex tracking problems, balancing robustness and computational efficiency in real-world applications.

4.3 Changing Points in Time Series

Detecting change points in time series data is crucial for identifying shifts in the underlying processes that govern the dynamics of particle diffusion. In the con-

text of anomalous diffusion, change points can indicate transitions between different diffusion regimes, such as from subdiffusive to superdiffusive behavior. Several methods and algorithms have been developed for detecting these change-points over time:

4.3.1. PELT (Pruned Exact Linear Time)

The PELT (Pruned Exact Linear Time) algorithm, introduced by [22], is widely recognized for its efficiency in detecting change points in time series data. The key advantage of PELT is its ability to identify the optimal segmentation of a time series while maintaining a linear computational complexity. This makes it a powerful tool in scenarios where both precision and scalability are required. The algorithm operates by assessing the entire time series and detecting points where significant changes in data distribution occur. Unlike simpler methods, PELT minimizes a cost function that balances the accuracy of the segmentation with a penalty for introducing too many change points, thereby avoiding overfitting. This makes the algorithm ideal for detecting meaningful shifts in data, especially in complex time series where abrupt changes might not be immediately obvious.

What makes PELT particularly effective is its use of pruning techniques to reduce the number of change points evaluated, thereby improving computational efficiency. As the algorithm processes each time step, it applies a pruning rule that eliminates certain points from further consideration if they are unlikely to improve the segmentation. This reduces the overall number of calculations needed, allowing PELT to operate more efficiently than traditional dynamic programming approaches. However, while PELT offers precise change point detection, its performance can still be a challenge when applied to extremely large datasets. As the size of the data increases, the number of potential change points can grow significantly, which might lead to higher memory usage and longer processing times. Despite this, PELT remains one of the most reliable algorithms for change point detection, particularly when both accuracy and a balance between computational speed and complexity are essential.

4.3.2. Window Sliding

The window sliding method is a straightforward yet powerful approach for detecting change points in time series data. This technique works by dividing the time series into either overlapping or non-overlapping windows, then applying change point detection algorithms within each segment to identify significant shifts in the data [23]. By breaking the data into smaller, manageable sections, the window sliding method is able to localize change points more effectively. One of its primary advantages lies in its low computational complexity, as the analysis is limited to smaller segments rather than the entire time series at once. Additionally, the method is easy to implement and highly adaptable, making it a popular choice in practical applications where large datasets need to be quickly and efficiently processed. Figure 4.1 presents a visual representation of window sliding works.

Moreover, the window sliding method's flexibility allows it to be combined with other advanced techniques to enhance its detection capabilities. For example, [24] applied this approach in conjunction with deep learning methods to identify change points in predicted models. By integrating deep learning with window sliding, they were able to capture more complex patterns and improve the precision of change point detection, even in challenging datasets with intricate dynamics. This hybrid approach demonstrates the versatility of the window sliding method, as it can be tailored to meet the demands of specific datasets and problem types, whether the goal is to detect abrupt changes or subtle shifts in data patterns. Overall, the window sliding method offers a balanced solution that combines computational efficiency with adaptability, making it a valuable tool for a wide range of time series analysis tasks.

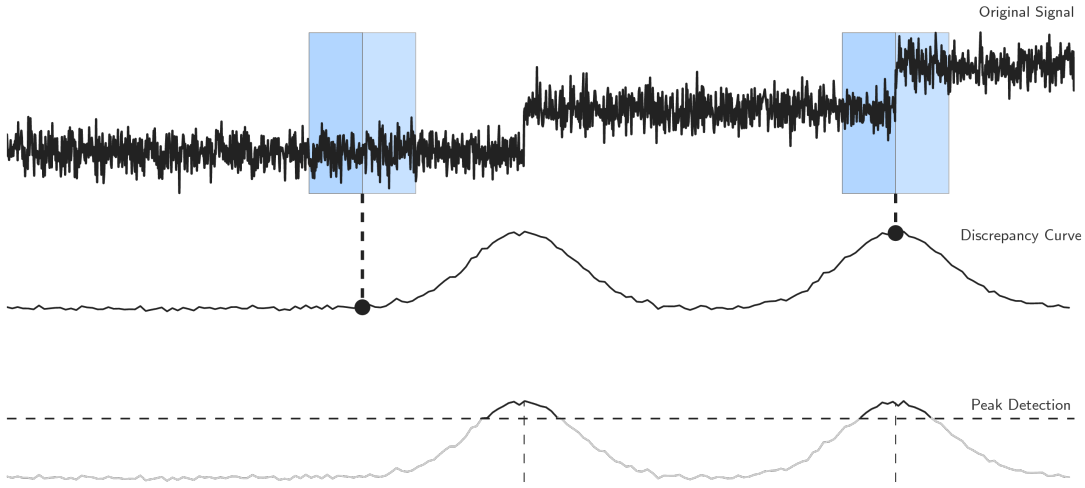

Figure 4.1: Illustrative representation of the Window Sliding method. Source: [23].

4.3.3. Binary Segmentation

Binary segmentation is a simple yet effective technique for detecting change points in time series data. The method works on an iteratively basis dividing the time series into two segments, identifying the most likely change point within each segment, as shown in figure 4.2 [23, 25, 26]. At each iteration, the algorithm minimizes a cost function—typically based on the difference between statistical properties such as the mean or variance—until it finds a significant change point. Once a change point is identified, the process repeats within the resulting segments, splitting them further until a predefined stopping criterion, such as a maximum number of change points or a threshold for statistical significance, is met. This makes binary segmentation conceptually straightforward and computationally efficient for many applications.

However, one limitation of binary segmentation is its sensitivity to the stopping criterion, which can impact the method's effectiveness in detecting subtle or small changes. If the criterion is too lenient, the algorithm may over-segment the data, detecting too many change points. Conversely, a strict criterion could

result in missing meaningful shifts. To address these limitations, several extensions of the basic technique have been proposed. For example, circular binary segmentation adapts the method for detecting changes in periodic data, while wild binary segmentation enhances its ability to detect multiple change points by introducing random subsampling of the data. These extensions make binary segmentation more versatile and capable of handling a wider variety of change point detection scenarios.

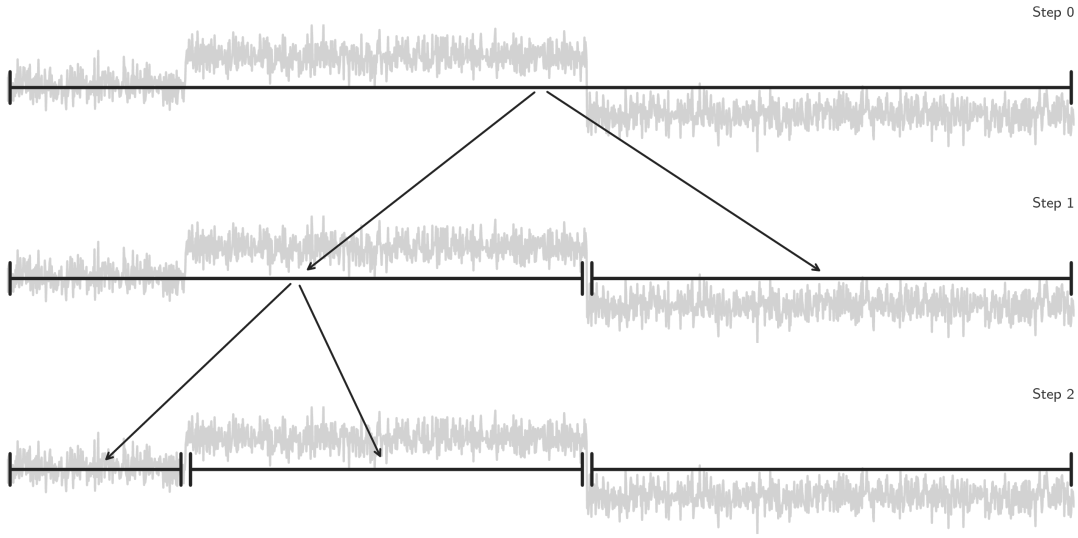

Figure 4.2: Illustrative representation of the Binary Segmentation algorithm. Source: [23].

4.3.4. Bottom-up Segmentation

In contrast to binary segmentation, bottom-up segmentation takes an opposite approach by first splitting the time series into many small segments and then progressively merging them based on a cost function that measures the quality of the segmentation. This method begins with a large number of segments and evaluates whether merging two adjacent segments reduces the overall cost. The merging process continues until no further improvements can be made or a predefined stopping criterion is met. This bottom-up approach is known for its linear computational complexity, making it efficient for large datasets, and its conceptual simplicity, which allows for easy implementation [23].

Despite its advantages, bottom-up segmentation can be unstable because it starts with very small segments. If the initial segmentation is not representative of the true underlying structure of the data, the merging process may result in inaccurate or suboptimal change point detection. The reliance on local information from small segments makes the algorithm sensitive to noise or minor fluctuations in the data, which can affect the stability of the results. This instability is a potential drawback in cases where a more global perspective of the time series is required to capture meaningful changes. Nonetheless, bottom-up segmentation remains a useful technique, particularly for applications where speed and simplicity are prioritized over fine-tuned precision.

4.4 Machine Learning

Machine learning plays a key role in analyzing complex systems like anomalous diffusion. By applying machine learning models, researchers can make predictions and better understand the mechanisms behind diffusion processes. This section is divided into two parts: the first part covers some models used in the initial Andi Challenge that was mainly based on predicting the diffusion exponent and classifying the trajectories according to five diffusion models: Annealed Transient Time Motion, Continuous Time Random Walk, Fractional Brownian Motion, Lévy Walk, and Scaled Brownian Motion [6]. The second part focuses on U-Nets which is the architecture in which we have based our models for the challenge.

4.4.1. The Andi Challenge

The following machine learning methods were developed in the frame of the first edition of the Andi Challenge. We give an overview of the different approaches considered in that challenge.

XGBoost

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of Gradient Boosting Trees (GBTs) designed for enhanced speed and performance in predictive modeling. Like GBTs, XGBoost builds an ensemble of decision trees, where each tree corrects the errors of the previous ones by minimizing a loss function through gradient descent [27].

One of its primary enhancements is the inclusion of L1 and L2 regularization, which helps prevent overfitting and makes the model more generalizable. XGBoost also supports parallel processing, allowing for the simultaneous construction of trees, which significantly speeds up training on large datasets. Additionally, XGBoost employs advanced tree pruning techniques to retain only the most relevant branches, further reducing the risk of overfitting. It also automatically handles missing data, improving accuracy even when data is incomplete. Moreover, XGBoost uses a weighted quantile sketch to accelerate tree learning, making it well-suited for large datasets.

The DeepSPT team introduced a model that integrated XGBoost with ResNet, which was applied in the 1st AnDi challenge for anomalous exponent inference and diffusion model classification in 1D trajectories [6]. Similarly, [28] utilized a GBT for the same tasks but extended their approach to also address 2D trajectories.

Feedforward Neural Networks

Feedforward Neural Networks (FNNs) are a category of artificial neural networks that aim to replicate the way the human brain processes information. As depicted in Figure 4.3, FNNs are composed of multiple layers of interconnected

nodes or neurons. These layers include an input layer, one or more hidden layers, and an output layer. Each connection between neurons carries a weight that is adjusted during the training process to minimize prediction errors.

In an FNN, neurons in each layer receive inputs, compute a weighted sum of these inputs, and then pass the result through a non-linear activation function. This process allows the network to learn and model complex, non-linear relationships between the input data and the output predictions. The non-linear activation functions are crucial as they introduce the ability to capture intricate patterns that linear models cannot.

The network's ability to adjust the weights of the connections between neurons through training makes FNNs highly versatile and effective for a wide range of predictive tasks, from classification to regression. Figure 4.3 visually illustrates this structure, highlighting how data flows through the network and how different layers and connections contribute to the final output.

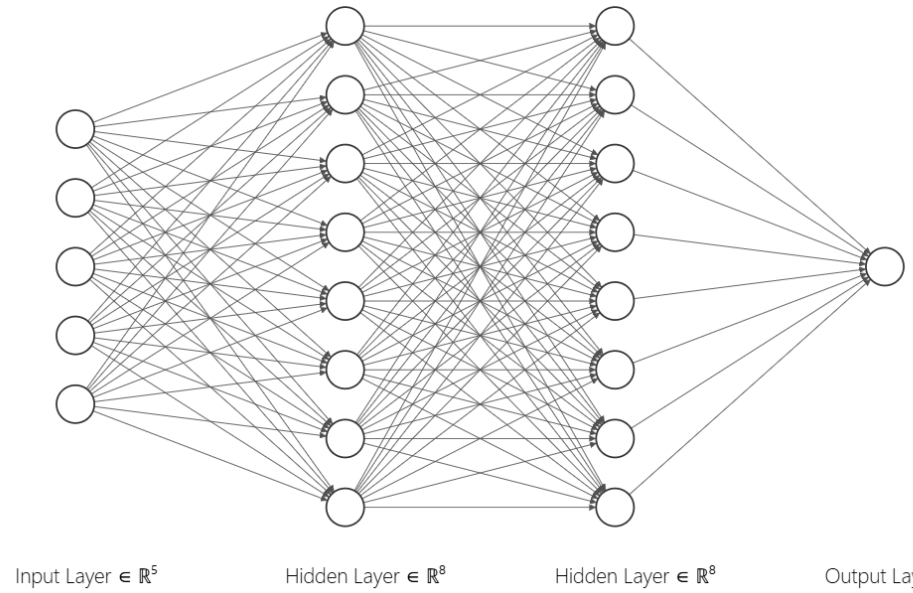

Figure 4.3: Illustrative representation of a Feedforward Neural Networks architecture.

In the study of anomalous diffusion, [24] used an FNN architecture to characterize anomalous exponent and diffusion model prediction in 1D, 2D, and 3D trajectories.

CNN + biLSTM

Convolutional Neural Networks (CNNs) are designed to process grid-like data structures, applying convolutional layers with learned filters (kernels) to capture local patterns, such as edges in images or motifs in time series data. This allows the network to extract relevant spatial features from the input data automatically.

On the other hand, Bidirectional LSTM or biLSTM are an extension of Long Short-Term Memory Networks (LSTMs), which are specialized for capturing long-term dependencies in sequential data. Unlike standard LSTMs, which process sequences in one direction, biLSTMs process the input sequence in both forward

and backward directions, allowing the model to capture context from both past and future data points.

By combining a CNN with biLSTM, the model first extracts spatial features from the data through convolutional layers. These features are then fed into the biLSTM layers, which capture temporal dependencies in both directions.

In the context of anomalous diffusion, [29] presented an approach that combined CNN and biLSTM to infer the anomalous exponent and classify the diffusion model for trajectories of 1, 2, and 3 dimensions.

4.4.2. U-Net

Compared to the previous challenge, where the prediction was made on single trajectories, the trajectories of this new version of the ANDI challenge are presented as FOVs of an experiment with several parts that exhibit similar diffusion properties and diffusion coefficients. Additionally, some of these trajectories even interact within them. In this context, all the trajectories within the FOV can be represented as a 3-dimensional matrix with the structure $ParticlesxFramesxDimensions$.

U-Net is a convolutional network architecture originally developed for biomedical image segmentation and designed to have 3D matrix as input. The U-Net architecture has a symmetric encoder-decoder structure, making it well-suited for predicting coefficients and states for each frame of the original matrix. The encoder path (down-sampling) captures the context of the input data. In contrast, the decoder path (up-sampling) enables precise localization by combining feature maps from the encoder with upsampled data [30]. Figure 4.4 represents a visual example of a U-Net network where the compression is taking in powers of two by each leavel. The ability of U-Nets to combine context and localization is particularly valuable in applications requiring both high-resolution feature extraction and precise predictions.

Attention U-Nets extend the traditional U-Net architecture by incorporating attention mechanisms. Attention mechanisms allow the network to focus on the most relevant parts of the input data, effectively weighting the importance of different regions or features [32]. The attention mechanism typically involves calculating attention weights based on the input features and then applying these weights to the feature maps in the decoder path.

Figure 4.5 illustrates the attention mechanism in the upscaling concatenation process of an Attention U-Net. Let g represent the matrix input from the skip connection with the encoder, and x^l denote the upscaled image from the decoding process. The attention mechanism begins by transforming both g and x^l through a convolutional filter. These transformed features are then concatenated and passed through a ReLU activation function, followed by another convolutional filter and a sigmoid activation function. The output of this sequence, denoted as α, is rescaled to match the dimensions of x^l. Subsequently, α is concatenated with x^l and used as input for the convolutional filters in the decoder at that level.

This approach enhances the network's ability to focus on important regions and ignore less relevant ones. This merged representation is used to generate an

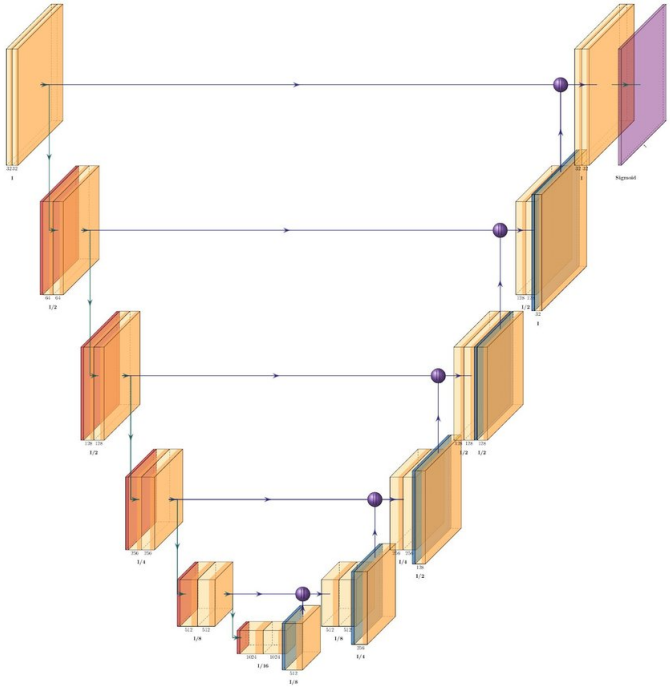

Figure 4.4: Illustrative representation of a U-Net model architecture. Source: [31].

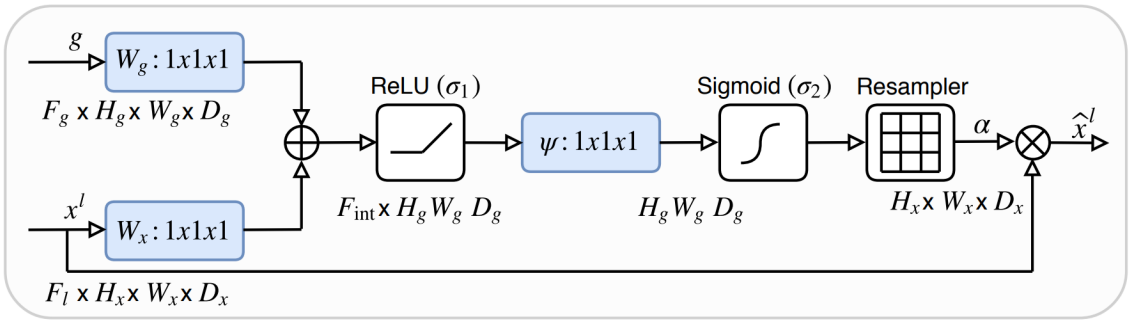

Figure 4.5: Illustrative representation of the attention mechanism. Source: [32].

attention map that highlights or suppresses certain features. Finally, the attention map recalibrates the original signal input, emphasizing the features deemed important by the attention mechanism.

4.5 Conclusion

This chapter has provided a comprehensive review of the current state of particle tracking, change point detection, and machine learning models, specifically in the context of studying anomalous diffusion. Importantly, some of the algorithms and models showcased in this review will serve as the fundamental base for developing the models presented later in this work.

CHAPTER 5
Methodology

Building on the theoretical background and cutting-edge research discussed earlier, this section outlines the methodology employed for participating in the challenge. It begins with an overview and rationale for the chosen approach (5.1). Next, it explains the generation of the database used to train the models (5.2), the development of the particle tracking algorithm (5.3), the model for inferring diffusion parameters and particle states (5.4), and the Change-Point algorithm for detecting changes in diffusion states/parameters (5.5). Finally, it details how these components are integrated and how predictions for the competition are made (5.6).

5.1 Overview

Given the tight timeframe (Mid-March to June), we propose a modular approach. This method allows different components to be developed separately and easily swapped with alternative algorithms, ensuring that changes do not affect the rest of the system and simplifying the process of making improvements. Additionally, it minimizes the computational resources required for both data generation and model training. This is achieved by using raw trajectories instead of videos for inference of parameters and states, as particles will be tracked with a particle tracking algorithm, allowing the models to be trained directly on raw trajectories. These trajectories require less computational resources to generate through simulation and train models than the videos.

The proposed methodology is illustrated in Figure 5.1. It comprises three main components: a particle tracking algorithm, Attention U-Net models, and a Change-Point algorithm. Initially, the video of the particle within the FOV (a) is processed by the particle tracking algorithm to extract the particle trajectories (b). These trajectories are then analyzed by three distinct Attention U-Net models to predict the α coefficient, the k coefficient, and the particle's state at each frame of the video (c). Using the resulting three different time series, a Change-Point algorithm is employed to identify changes in state and/or diffusion coefficients for the various particles (d).

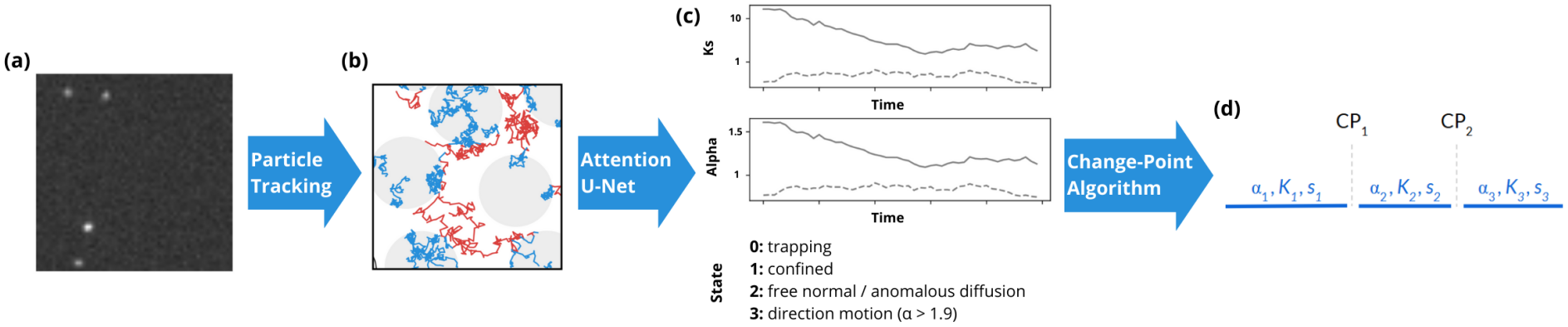

Figure 5.1: Illustrative representation of the proposed methodology.

5.2 Data Generation

Due to the limited size of the dataset offered by the competition, synthetic data is generated. To generate experiments with the different physical models of motion and interactions the Python library used is *andi-datasets* [15]. This library enables the simulation of experiments using five distinct phenomenological models. This library allows us to generate either videos of experiments or just the raw trajectories. However, as previously noted, since inference is separated from particle tracking, the model training will only require the raw trajectories.

The library allows us to generate experiments for any of the 5 models while controlling parameters such as the number of particles, trajectory lengths, α and k values, and state transition probabilities. It also includes built-in functions to generate parameter values.

An important consideration in experiment generation is the concept of FOVs. An FOV is a specific window within the experiment used to obtain trajectories (or videos in Track 1). For example, an experiment sized 512×512 can have FOVs of 128×128. To avoid boundary effects, FOVs are smaller than the full experiment size. Since all trajectories within a FOV are correlated, they are saved together in a matrix of size $64 \times 208 \times 2$, where 64 represents the number of particles (experiments can have more particles, in this case, several matrices are generated), 208 is the trajectory length, and 2 denotes the X and Y positions. The particles can move in and out of the FOV, so when converting the trajectories into a matrix, a 0 is assigned whenever a particle is outside the FOV.

Generating a large number of simulations demands significant computational resources and time. Due to constraints on both computational and financial resources, we chose not to use high-demand services such as AWS, Azure, or Google Cloud because of their costs. Instead, we opted for Google Colab, which provides a more cost-effective solution for accessing computational resources. Throughout the project, we acquired 1,600 computational credits for data generation and model training at a cost of €136.00 and 200GB of shared storage on Google Drive for €11.96.

On the technical side, the data generation library does not utilize multiprocessing techniques, and implementing these would require significant time and resources. To accelerate the data generation process, multiple computers were used in parallel, sharing memory. These computers ran the data generation tasks using low-cost CPU-only instances with extended RAM (Basic CPU, with 51 GB), as the library does not benefit from GPU acceleration.

The data generation process involved running between 2 and 5 instances simultaneously, simulating 4,000 experiments, each containing 5 FOVs, resulting in a total of 20,000 FOVs. For each FOV, we generated three matrices and one tag: the matrices represent raw trajectories, coefficients, and particle states, with dimensions $64 \times 208 \times 2$ for the first two matrices and 64×208 for the state matrix. The tag corresponds to the physical model of motion and interactions. Each batch was compressed and saved using Numpy, containing 3 arrays of matrices and one array of tags. Figure 5.2 showcases the data generation process. In the Listing A.1, the last version of the script used to generate the data is showcased.

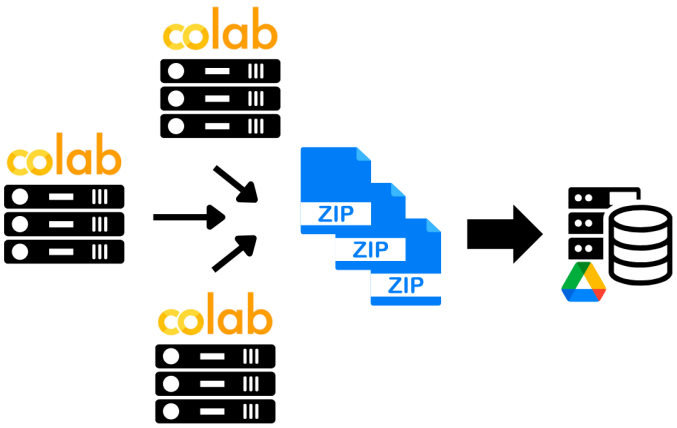

Figure 5.2: Illustrative representation of the data generation.

The final database generated is composed of approximately 775,000 FOVs with 49 million trajectories, corresponding to 155,000 independent experiments, where each physical model constitutes one-fifth of the database. To determine the α and k coefficients, as well as other experiment parameters, we used the library's built-in functions to generate random values. For the α and k coefficients, we combined values from the library with random sampling from a uniform distribution for large k values. This approach addresses the function's limitation in generating only small k values (between $(0, 2]$), whereas actual values can range from 10^{-12} to 10^{6}. High values of k and α are not practical for all models in experiments of this scale, so we limited k to a range between 10^{-6} as the minimum and 10^{2} as the maximum.

5.3 Particle Tracking

Particle tracking is a technique used to follow the movement of particles across successive frames in a video, enabling the analysis of their trajectories over time. In our specific task, we need to track particles in videos with dimensions of 128x128 pixels over 200 frames[1], and identify specific particles of interest (VIP particles) by their locations in the first frame. To achieve this, we use the TrackPy library [33], which implements the Crocker-Grier algorithm for efficient particle detection and tracking [19].

Before processing with TrackPy, the video frames undergo a preprocessing step where their borders are expanded. This expansion helps in mitigating edge effects that could affect the accuracy of particle detection near the boundaries of the frame. After preprocessing, we use the `trackpy.batch` function to detect particles across all 200 frames. This function is fine-tuned with parameters such as particle diameter, intensity thresholds, and separation distance using the videos presented in the development database provided by the competition (100

[1]The discrepancy in the number of frames between the trajectories and videos provided by the competition and the generated data is intentional. The use of 208 frames is due to the specific nature of the models employed. For predictions based on competition data, padding with zeros will be applied.

videos). Figure 5.3 shows visually the detection of particles made by the algorithm.

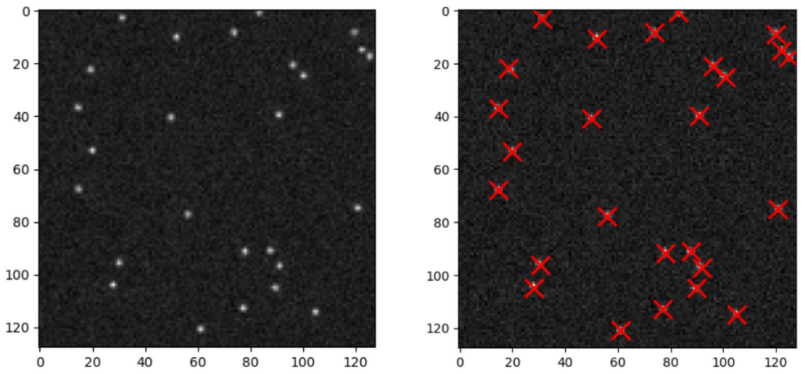

Figure 5.3: Particle detection with TrackPy.

Once the particles are detected, we link them across frames using `trackpy.link`. This step connects the detected particles from one frame to the next, creating continuous trajectories that represent the movement of each particle throughout the video. The algorithm accounts for potential challenges like particles leaving and reentering the frame or temporarily overlapping with other particles. Figure 5.4 presents a visual comparison of the real trajectories and those derived using `trackpy.link`. Note that the colors do not consistently identify the same particles across the plots. Nevertheless, the two plots align very closely, showcasing a high degree of similarity.

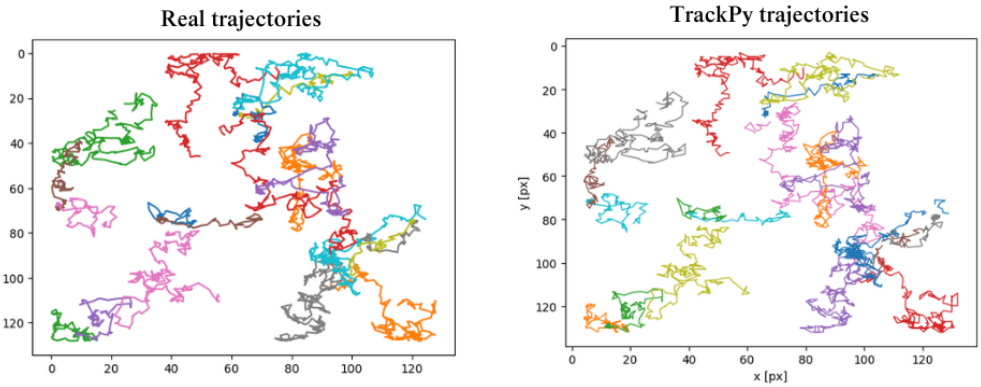

Figure 5.4: Comparison of real trajectories and trajectories extracted with TrackPy.

To connect these trajectories with their corresponding VIP particles, we implemented a function that matches the detected particles with the known locations of VIP particles in the first frame. The function first identifies the positions of all particles in the VIP location matrix (or first frame) after border expansion. It then computes the distance between these positions and the detected particle positions in the first frame with trackpy using the Hungarian algorithm (linear sum assignment). By minimizing the distance, we assign each detected particle a unique identifier corresponding to the VIP particle it most likely represents. The Python implementation is shown in Listing 5.1.

In Figure 5.5, the visual representation of the tasks of the particle tracking module is shown from the video of the particles moving to a 3D matrix used as input of the Attention U-Net.

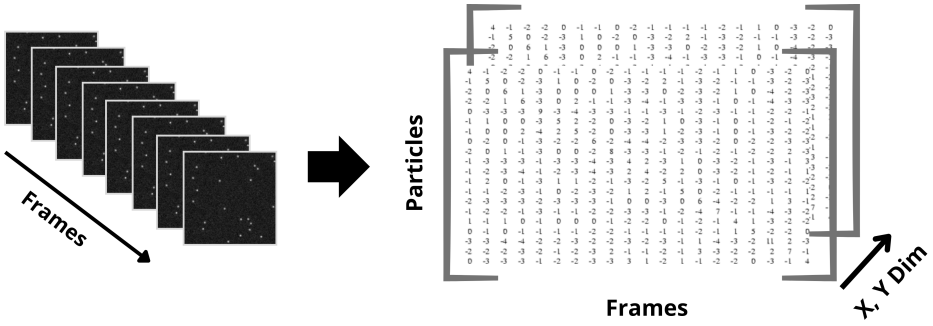

Figure 5.5: Illustrative representation of the video to trajectory transformation.

```python
def loc_particles(firstframe, df_video):
    """
    Maps detected particles VIP particles in the first frame.

    This function identifies and assigns unique IDs to the VIP
       particles based on their positions in the first frame. It
       matches these particles with those detected previously with
       Trackpy using the Hungarian algorithm for optimal assignment
       .

    Parameters:
    - firstframe (numpy.ndarray): 2D array of the first video frame
        with unique integer IDs for particles (0 for background).
    - df_video (pandas.DataFrame): DataFrame with columns "frame",
       "x", and "y" representing particle positions in each frame.

    Returns:
    - dict: Mapping of VIP particle indices by Trackpy to their
       unique IDs.

    """
    indxparticles = {}
    df_firstframe = df_video[df_video["frame"] == 0]
    firstframe = expand_borders([firstframe])[0]
    items = np.unique(firstframe)[:-1]
    positions = [np.mean(np.where(firstframe == itm), axis=1) for
       itm in items]
    positions_array = np.array(positions)
    detected_array = df_firstframe[['y', "x"]].values
    distances = np.sqrt(((detected_array[:, np.newaxis, :] -
       positions_array) ** 2).sum(axis=2))
    row_ind, col_ind = linear_sum_assignment(distances)

    for detected, assigned in zip(row_ind, col_ind):
        if len(np.where(firstframe == items[assigned])[0]) >= 3:
            indxparticles[int(detected)] = int(items[assigned])

    return indxparticles
```

Listing 5.1: Python function for VIP particle location.

5.4 Inference of Diffusion Parameters and State Classification

This section details our methodology for inferring the diffusion parameters and classifying states using an Attention U-Net model for each task. Figure 5.6 illustrates the input and output dimensions before and after the data passes through the α, k, and state Attention U-Net models. The Attention U-Net architecture is an enhancement of the traditional U-Net with attention mechanisms, as mentioned in the literature review.

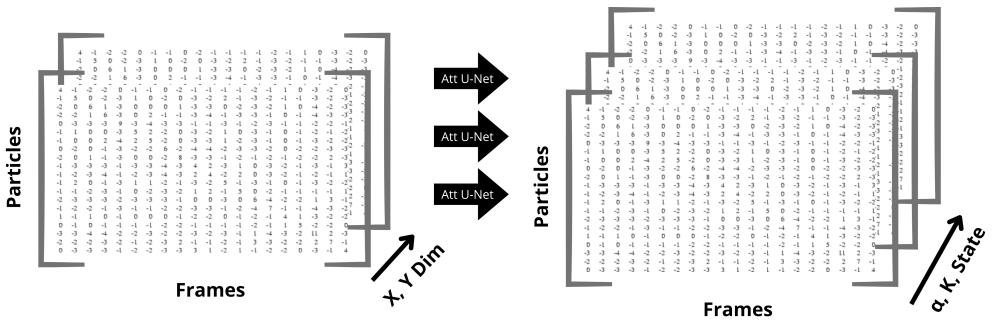

Figure 5.6: Illustrative representation of the prediction structure.

The Attention U-Net architecture is designed to manage symmetric predictions, where both the input and output have the first two dimensions of equal size. This is achieved through a combination of convolutional and decoding operations. The model starts with convolutional blocks, which include a series of convolutional layers combined with batch normalization and ReLU activation to extract detailed features from the input data. These blocks are arranged into encoder stages that sequentially capture hierarchical features while decreasing the spatial dimensions of the input via max pooling layers. To enhance feature relevance, the network uses attention gates to selectively emphasize important features by integrating information from both the encoder and decoder paths. In the decoder stages, upsampling layers restore the spatial dimensions and merge features from the encoder path, refined by attention mechanisms. The final segmentation output is produced through a convolutional layer with a Softmax activation function for classification tasks or ReLU for regression tasks. The model is optimized using the Adam optimizer. The Python implementation of an Attention U-Net for multi-class classification, featuring two encoder levels and two successive convolutional layers at each level, is detailed in Listing 5.2.

```
import tensorflow as tf
from tensorflow.keras import layers as L
from tensorflow.keras.models import Model

def conv_block(x, num_filters):
    """Convolutional block with two Conv2D layers followed by
       BatchNormalization and ReLU activation."""
    x = L.Conv2D(num_filters, (3, 3), padding="same")(x)
    x = L.BatchNormalization()(x)
    x = L.Activation("relu")(x)

```

```
    x = L.Conv2D(num_filters, (3, 3), padding="same")(x)
    x = L.BatchNormalization()(x)
    x = L.Activation("relu")(x)

    return x

def encoder_block(x, num_filters):
    """Encoder block with a convolutional block followed by max
       pooling."""
    x = conv_block(x, num_filters)
    p = L.MaxPool2D((2, 2))(x)
    return x, p

def attention_gate(g, s, num_filters):
    """Attention gate for selective feature merging."""
    Wg = L.Conv2D(num_filters, 1, padding="same")(g)
    Wg = L.BatchNormalization()(Wg)

    Ws = L.Conv2D(num_filters, 1, padding="same")(s)
    Ws = L.BatchNormalization()(Ws)

    out = L.Activation("relu")(Wg + Ws)
    out = L.Conv2D(num_filters, 1, padding="same")(out)
    out = L.Activation("sigmoid")(out)

    return out * s

def decoder_block(x, s, num_filters):
    """Decoder block with upsampling, attention gate, concatenation
       , and convolutional block."""
    x = L.UpSampling2D(interpolation="nearest")(x)
    s = attention_gate(x, s, num_filters)
    x = L.Concatenate()([x, s])
    x = conv_block(x, num_filters)
    return x

def attention_unet(input_shape):
    """Builds the Attention U-Net model."""
    inputs = L.Input(input_shape)

    # Encoder
    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 96)
    b1 = conv_block(p2, 128)

    # Decoder
    d1 = decoder_block(b1, s2, 96)
    d2 = decoder_block(d1, s1, 64)

    # Output layer
    outputs = L.Conv2D(5, 1, padding="same", activation="softmax")(
       d2)

    # Model definition
    model = Model(inputs, outputs, name="Attention-UNET")
    model.compile(
        optimizer='adam',
        loss=tf.keras.losses.CategoricalCrossentropy(),
        metrics=['accuracy']
```

```
    )

    return model
```

Listing 5.2: Python implementation with Tensorflow/Keras of a Attention U-Net for a multi-classification taks.

Using this architecture, we tested multiple versions of the models for each task by varying the number of encoder levels, filters, and convolutional blocks. The dataset was split 80-20, with 620,000 FOVs used for training and 155,000 for validation. The validation set was used to compare different architectures. The final architectures were selected not only based on performance but also on the time taken to train until early-stopping[2] stops the training occurred.

To train models with such a large database of matrices, we had to load the data in blocks and perform the training in multiple stages. The GPUs used for training were the NVIDIA L4 and A100, selected based on the complexity of the configuration. These GPUs provide 24 GB and 40 GB of VRAM, respectively, allowing us to maximize the block sizes loaded into memory. The batch size varied between 16 and 128, depending on the model's complexity and corresponding memory limitations.

As a preprocessing step for the database, we calculate the displacement at each position between t and t_{-1} for each dimensional space. The new values of the matrix in the dimension X and Y are calculated as:

$$\Delta x_i^{'}(t) = 128 \cdot (x_i(t) - x_i(t-1))$$
$$\Delta y_i^{'}(t) = 128 \cdot (y_i(t) - y_i(t-1))$$

The multiplication by 128 compensates for the normalization applied during data generation. This preprocessing step converts absolute positions into relative movements.

In Table 5.1, we provide a summary of the final configurations selected. The α and ks models, which represent numerical values, are handled as regression tasks. For these, the error functions optimized are MAE (Mean Absolute Error) and MSLE (Mean Squared Logarithmic Error), respectively, as per the competition's evaluation metrics. The output of these models corresponds to a parameter for each particle at each frame, resulting in a matrix of size $64 \times 208x\times$.

On the other hand, predicting the particle state is treated as a multiclass classification problem, where categorical cross-entropy is used as the loss function. The model predicts 4 classes based on environmental constraints, along with an additional class for cases when the particle is absent from the FOV in a given frame (i.e., $X_{ij} = 0$). The output is a matrix of size $64 \times 208 \times 5$ that will transform after the prediction to $64 \times 208 \times 5$ using *argmax* to select the class with higher probability and normalizing the preditions, since a particle can not change of state for at least 3 frames.

[2]Early stopping is a technique used to prevent overfitting by halting the training of a model once its performance on a validation set starts to degrade. It monitors a specific metric, such as validation loss, and stops training when no improvement is observed after a set number of iterations.

Table 5.1: Description of Models with Task Types, Architecture Details, and Performance Metrics

Model	Task	Encoder Levels	Conv. Blocks	Filters per level	Parameters	Epoch	Output Size	Output Activation Function	Batch Size	Metric Error	Error
Alpha's	Regression	3	2	128-512-1024	28,753,665	2	(64x208x1)	ReLu	16	MAE	0.041
K's	Regression	3	2	128-512-1024	28,753,665	1	(64x208x1)	ReLu	16	MSLE	0.043
State	Classification	3	6	64-96-128	2,448,677	10	(64x208x5)	SoftMax	64	Categorical Cross-Entropy	0.116

5.5 Change Point Detection

After generating the predictions, we will have three distinct time series for each particle in the field of view (FOV): α, k, and the state at each position. To detect changes in state, we will focus mainly on the α and k time series while using the state series to fine-tune the detection process. Specifically, particles in the "trapped" state exhibit α and k values close to zero, which helps us refine the change point detection method.

For detecting change points, we use the sliding window method implemented in the `ruptures` Python library [23], with the **l2 model** incorporated. The l2 model detects changes by minimizing the squared differences between different segments of the time series, allowing it to identify significant variations in the data. The method works by sliding a window of fixed width across the data and analyzing the segments to determine if any notable change has occurred.

The key parameters of the sliding window function in the ruptures library are:

1. **Width**: This sets the size of the window that moves across the data. A smaller width detects rapid changes, while a larger width captures broader trends. For trajectories without trapped states, we set the width to 12. For trajectories with trapped states, we use a wider window of 18 to account for smoother transitions.

2. **Jump**: This controls how much the window shifts after each iteration. We use a jump of 1 in both cases to ensure high precision by evaluating every possible position along the time series.

3. **Min_size**: This parameter sets the minimum number of time steps needed for a state change to be detected. In our case, it is set to 3, which reflects the minimum span for a particle to change states.

4. **Penalization (pen)**: This influences the algorithm's sensitivity to detecting changes. A larger penalization discourages the detection of small or insignificant changes, while a smaller penalization makes the algorithm more sensitive to minor changes. For trajectories with no trapped states, we use a penalization value of 29 to focus on significant transitions. For trajectories with trapped states, we use a smaller penalization of 5, making the algorithm more responsive to subtle changes when the particle is in a low-variation state.

Table 5.2: Parameter Configurations for State Change Detection

Trajectory Type	Model	Width	Jump	Min_size	Penalization (pen)
No Trapped State	"l2"	12	1	3	29
With Trapped States	"l2"	18	1	3	5

This configuration ensures that the sliding window method is tuned to accurately detect both rapid state transitions and the more gradual changes associated with trapped states. By adjusting the width, jump size, minimum size, and penalization values, we improve the accuracy and reliability of the change point detection across different particle behaviors.

5.6 Prediction

After identifying the change points, we can obtain predictions for the single-particle task for the video task by computing the average of α and k between each pair of consecutive change points, CP_i and CP_{i+1}. We then select the class that appears most frequently in the state time series. For each particle, we generate a string formatted as follows: "$ParticleIDinVIPimage$, K_1, α_1, $State_1$, CP_1, ..., K_n, α_n, $State_n$, CP_n, $TrajectoryLength$". Then, all the predictions made in an FOV are saved in a .txt.

For the ensemble task, we need to extract data for an experiment across all its fields of view (FOVs). If the number of detected change points across the entire FOVs is very low, we consider only one state and calculate the mean and standard deviation of both α and k. Conversely, if the number of change points is sufficient, we apply K-Means clustering with $K = 2$ to categorize the detected segments into two groups. We then compute the mean and standard deviation of α and k for each group, as well as the relative weight of each state. Table 5.3 outlines the submission format for the ensemble task. In the header labeled *model*, replace *modelxxx* with the physical models of motion used in the experiment, selected based on state predictions. In the header *num_state*, replace YYY with the number of states. Each column corresponds to a state, with rows representing the average of α (μ_α^n), the standard deviation of α (σ_α^n), the average of k (μ_k^n), the standard deviation of k (σ_k^n), and the relative weight of each state (N_n) for the column n.

Table 5.3: Ensemble Task Prediction Format

model: modelXXX;	**num_state: YYY**		
μ_α^1;	μ_α^2;	μ_α^3;	...
σ_α^1;	σ_α^2;	σ_α^3;	...
μ_K^1;	μ_K^2;	μ_K^3;	...
σ_K^1;	σ_K^2;	σ_K^3;	...
N_1;	N_2;	N_3;	...

The Python script used to generate predictions for the challenge—encompassing particle tracking, inference with Attention U-Nets, change point detection, and prediction formatting for both the ensemble and single trajectories tasks is shown in Listing A.2.

CHAPTER 6
Validation

This chapter explores the crucial process of model validation, a fundamental step in assessing how well a model performs with new, unseen data. Validation is essential for identifying and addressing issues such as overfitting and underfitting, ensuring that models generalize effectively beyond their training datasets.

The first dataset used to validate the performance of the models is the validation file provided by the competition. Using this file for validation serves two key purposes: it introduces an unknown data structure and generation method, and allows for comparison with other models.

6.1 Competition Validation

Although the primary objective of this thesis is to participate in the video track, the modular approach also enables us to submit predictions for the raw trajectories in the Trajectory track.

Table 6.1 presents the ensemble results for the video track. Our implementation (**ICSO UPV**) successfully submitted one of the two valid predictions, showcasing the hardness of the challenge and achieving the top position in the ranking with the best W1 for various K values. For the Single Trajectory Task, Table 6.2 shows the ranking. Here, only a few teams submitted valid results. Our metrics are highly competitive, though we secured second place, but with a bad result in the MAE for the α.

Table 6.1: Video Track: Ensemble Task Results During the Validation Phase

Ranking	**Team Name**	**MRR**	**Metrics**	
			W1 (K)	**W1 (alpha)**
1	***ICSO UPV***	***0.750***	***0.50 (1)***	***0.43 (2)***
2	SPT-HIT	0.750	0.72 (2)	0.21 (1)
3	SU-FIONA	0.333	1000000.00 (3)	2.00 (3)
4	HNU	0.333	1000000.00 (3)	2.00 (3)

On the other hand, the Trajectory track had a higher number of valid submissions, which facilitated a more thorough comparison of the results. For the

Table 6.2: Video Track: Single Trajectory Task Results During the Validation Phase

Ranking	Team Name	MRR	Metrics				
			RMSE (CP)	JSC (CP)	MSLE (K)	MAE (alpha)	F1 (diffusion type)
1	SU-FIONA	0.867	3.05 (3)	0.50 (1)	0.07 (1)	0.20 (1)	0.82 (1)
2	***ICSO UPV***	***0.533***	***0.80 (1)***	***0.44 (2)***	***0.13 (2)***	***0.48 (3)***	***0.77 (3)***
3	SPT-HIT	0.433	2.27 (2)	0.44 (3)	0.80 (3)	0.26 (2)	0.78 (2)
4	HNU	0.250	10.00 (4)	0.00 (4)	100000.00 (4)	2.00 (4)	0.00 (4)

Ensemble task, detailed in Table 6.3, the performance of the models was impressive, with our approach achieving a Top 3 position. Specifically, it secured the second-best W1 score for inferring the parameter K and demonstrated competitive performance with the W1 score for α.

Regarding the Single Trajectory Task, the results presented in Table 6.4 highlight the model's robust performance across various metrics. Our approach achieved 7th place overall, with metrics such as MSLE and MAE being closely aligned with those of the leading entries on the leaderboard.

Table 6.3: Trajectory Track: Ensemble Task Results During the Validation Phase

Ranking	Team Name	MRR	Metrics	
			W1 (K)	W1 (alpha)
1	Unfriendly AI	1.000	0.10 (1)	0.12 (1)
2	KCL	0.375	0.36 (4)	0.16 (2)
3	***ICSO UPV***	***0.350***	***0.28 (2)***	***0.17 (5)***
4	HSC AI	0.250	0.31 (3)	0.18 (6)
4	SPT-HIT	0.250	0.43 (6)	0.16 (3)
5	Nanoninjas	0.188	1.03 (8)	0.16 (4)
6	UCL SAM	0.171	0.38 (5)	0.20 (7)
7	Sk	0.121	0.59 (7)	0.33 (10)
8	BIOMED-UCA	0.108	10.78 (11)	0.25 (8)
9	DeepSPT	0.106	1.34 (10)	0.31 (9)
10	D.AnDi	0.101	1.34 (9)	0.39 (11)
11	far_naz	0.083	37.13 (12)	0.59 (12)

6.2 Local Validation

This subsection details the validation of machine learning models using a set of 155,000 Fields of View (FOVs) and approximately 10 million trajectories. The results from this validation set will be compared with those from the competition's validation, as described in the previous subsection. This comparison aims to evaluate the models' performance and accuracy relative to established benchmarks.

In Figure 6.1, a hexagonal binning plot is presented to compare the real versus predicted alpha values generated by the model. This plot reveals a notable concentration of data points along the diagonal line, indicating that predictions

Table 6.4: Trajectory Track: Single Trajectory Task Results During the Validation Phase

Ranking	Team Name	MRR	Metrics				
			RMSE (CP)	JSC (CP)	MSLE (K)	MAE (alpha)	F1 (diffusion type)
1	SPT-HIT	0.583	1.52 (3)	0.75 (1)	0.03 (3)	0.15 (1)	0.86 (4)
2	HNU	0.562	0.12 (1)	0.61 (5)	0.14 (9)	0.16 (2)	0.89 (1)
3	Unfriendly AI	0.457	2.32 (7)	0.69 (2)	0.02 (1)	0.17 (7)	0.87 (2)
4	SU-FIONA	0.289	2.84 (13)	0.69 (3)	0.02 (2)	0.16 (3)	0.85 (5)
5	UCL SAM	0.241	1.24 (2)	0.67 (4)	0.03 (5)	0.22 (9)	0.81 (7)
6	KCL	0.204	1.98 (4)	0.60 (6)	0.09 (7)	0.18 (8)	0.86 (3)
7	***ICSO UPV***	***0.183***	***2.34 (8)***	***0.59 (8)***	***0.03 (4)***	***0.16 (4)***	***0.83 (6)***
8	M3	0.134	2.09 (5)	0.50 (13)	0.12 (8)	0.17 (6)	0.80 (10)
9	BIOMED-UCA	0.128	2.42 (9)	0.52 (12)	0.08 (6)	0.17 (5)	0.78 (13)
10	KNU-ON	0.102	2.99 (15)	0.60 (7)	0.18 (10)	0.31 (11)	0.80 (9)
11	Nanoninjas	0.096	2.55 (10)	0.28 (16)	0.31 (11)	0.25 (10)	0.81 (8)
12	AIntgonnawork	0.082	2.13 (6)	0.47 (14)	0.86 (16)	1.02 (19)	0.43 (17)
13	HSC AI	0.081	2.67 (12)	0.53 (10)	2.57 (19)	0.37 (12)	0.78 (12)
14	bjyong	0.075	3.02 (16)	0.58 (9)	1.42 (18)	0.53 (18)	0.79 (11)
15	DeepSPT	0.073	2.65 (11)	0.23 (17)	0.62 (14)	0.39 (13)	0.71 (15)
16	D.AnDi	0.073	3.49 (18)	0.52 (11)	0.56 (12)	0.43 (14)	0.45 (16)
17	Sk	0.065	2.94 (14)	0.32 (15)	0.80 (15)	0.44 (15)	0.35 (18)
18	EmetBrown	0.065	3.40 (17)	0.05 (18)	0.61 (13)	0.47 (16)	0.77 (14)
19	far_naz	0.055	3.79 (19)	0.05 (19)	1.11 (17)	0.51 (17)	0.04 (19)

closely match the actual values. This alignment suggests that the model exhibits strong performance in predicting alpha values.

To assess whether this behavior is consistent across different models, Figure 6.2 displays similar hexagonal binning plots, each separated by model type. The comparison demonstrates minimal differences among the models, with each plot showing a similar diagonal trend as observed in the previous figure. This consistency across models further supports the reliability of the prediction performance.

In Figure 6.3, we plot the MAE of the model as a function of the real α value. The plot shows a similar error for α values between 0 and 1.5, with the error increasing linearly until it reaches 2. This may be due to the lower frequency of higher α values in the experiments, which could be influenced by the limitations of the FOV space (128×128). Particles diffusing too quickly are likely to exit the FOV boundaries faster, reducing their observation time.

Figure 6.4 presents the same comparison but separated by motion model. Here, the error patterns are similar across models, except for MSM, which tends to have a lower error after $\alpha > 1$.

Figure 6.5 shows the MSLE as a function of the real value of K, where we observe a consistent error across the spectrum, with slightly lower values for $K < 10$. When separating the results by motion model in Figure 6.6, the error profiles remain consistent, except for MSM, which again shows some variation.

Both the MAE and MSLE results are in line with the validation presented in Table 6.4.

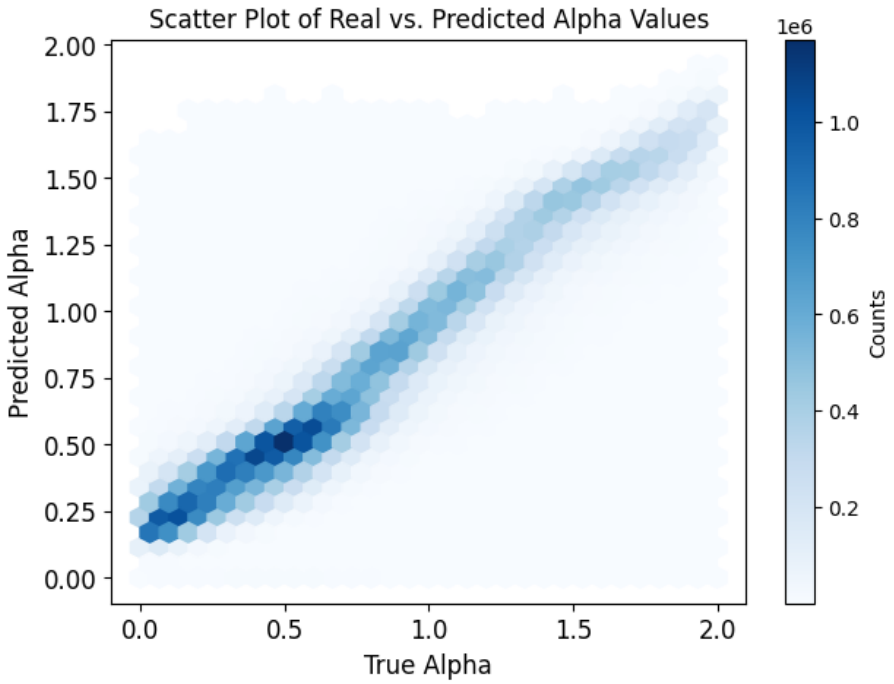

Figure 6.1: Hexagonal binning plot of real versus predicted alpha values. The plot visualizes the distribution of prediction errors by grouping data points into hexagonal bins, with color shading indicating the frequency of data point occurrences. Darker hexagons indicate higher density and potentially more reliable predictions.

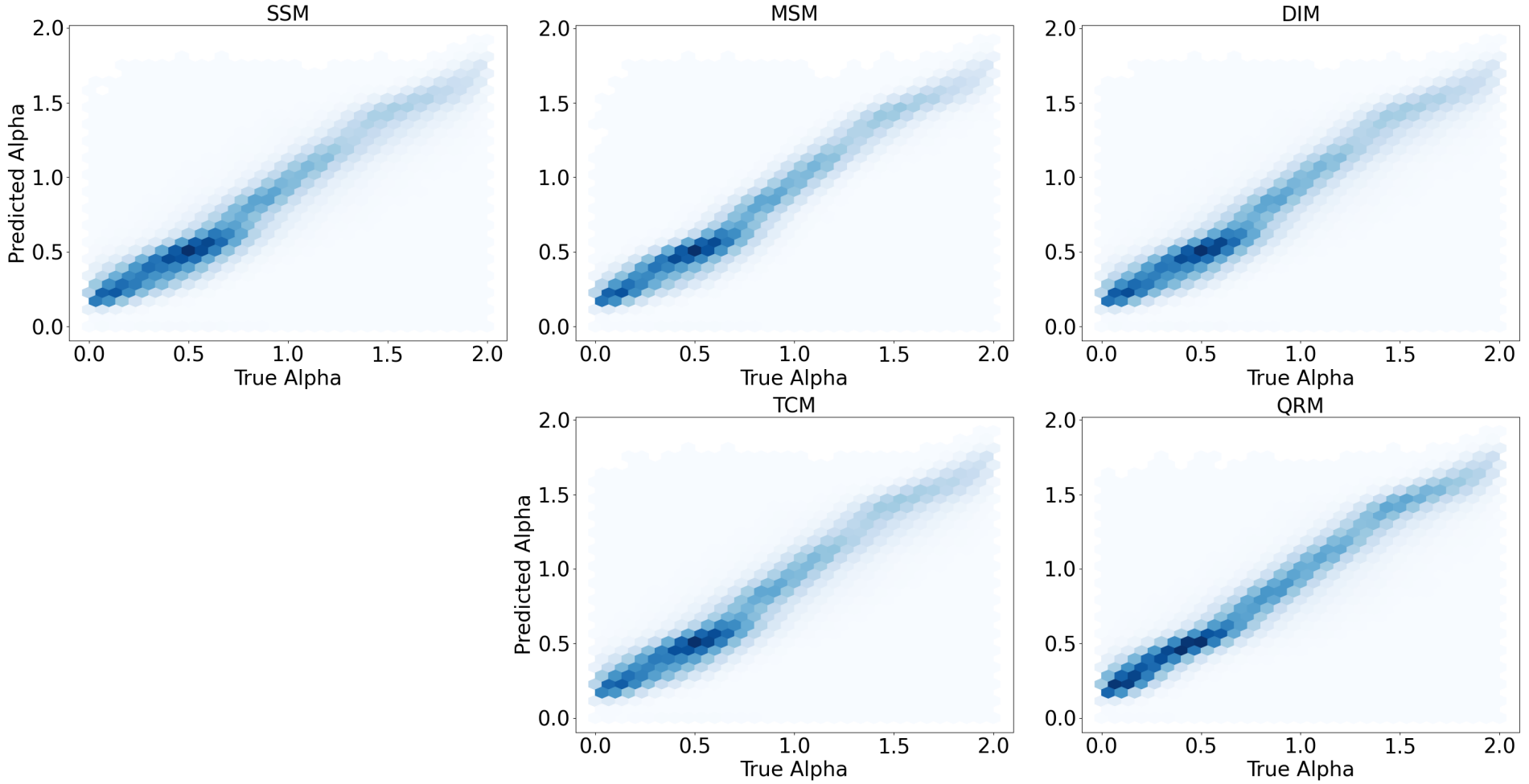

Figure 6.2: Hexagonal binning plot showing the relationship between real and predicted alpha values based on the motion model. The models include: Single-State Model (SSM), Multi-State Model (MSM), Dimerization Model (DIM), Transient-Confinement Model (TCM), and Quenched-Trap Model (QTM).

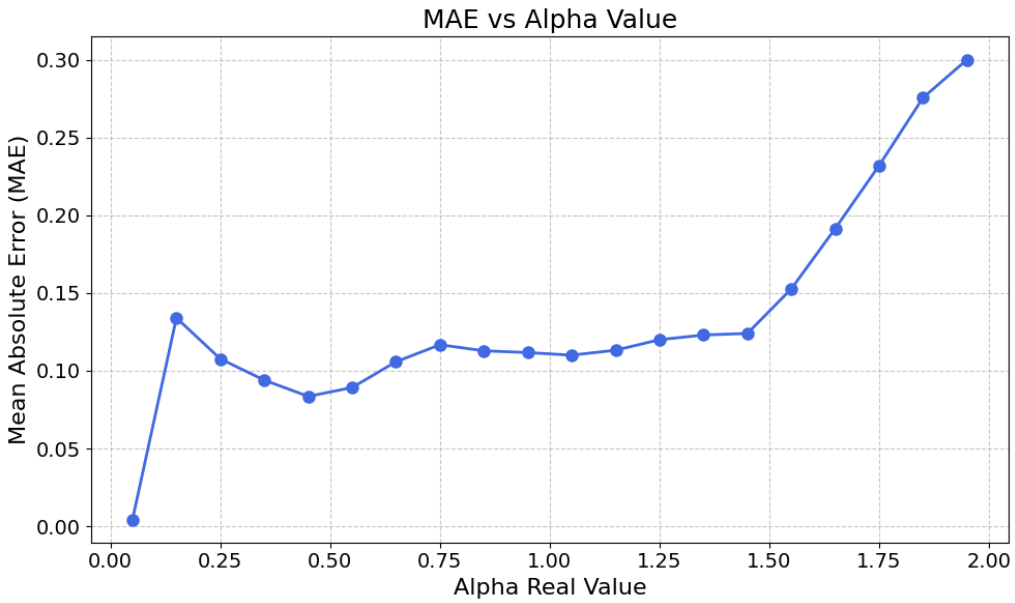

Figure 6.3: Comparison of Mean Absolute Error (MAE) as a function of feal flpha V values.

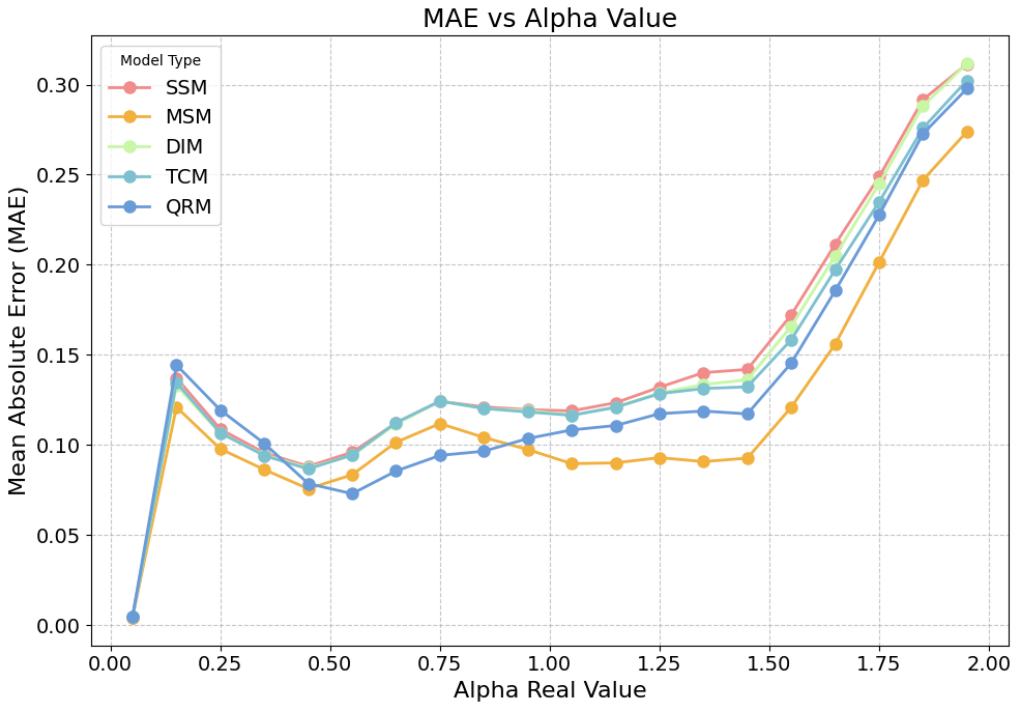

Figure 6.4: Comparison of Mean Absolute Error (MAE) as a function of real alpha values by motion model. The models include: Single-State Model (SSM), Multi-State Model (MSM), Dimerization Model (DIM), Transient-Confinement Model (TCM), and Quenched-Trap Model (QTM).

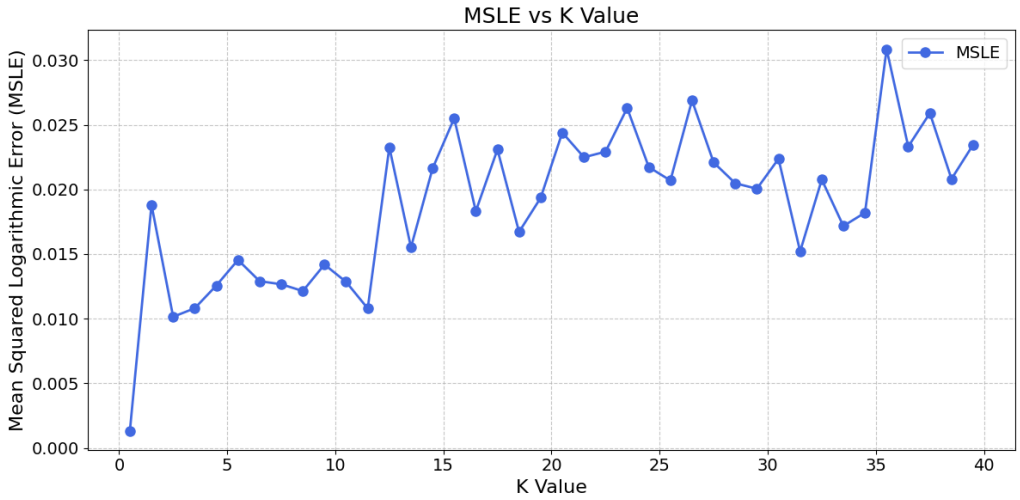

Figure 6.5: Comparison of Mean Squared Logarithmic Error (MSLE) as a function of real K values

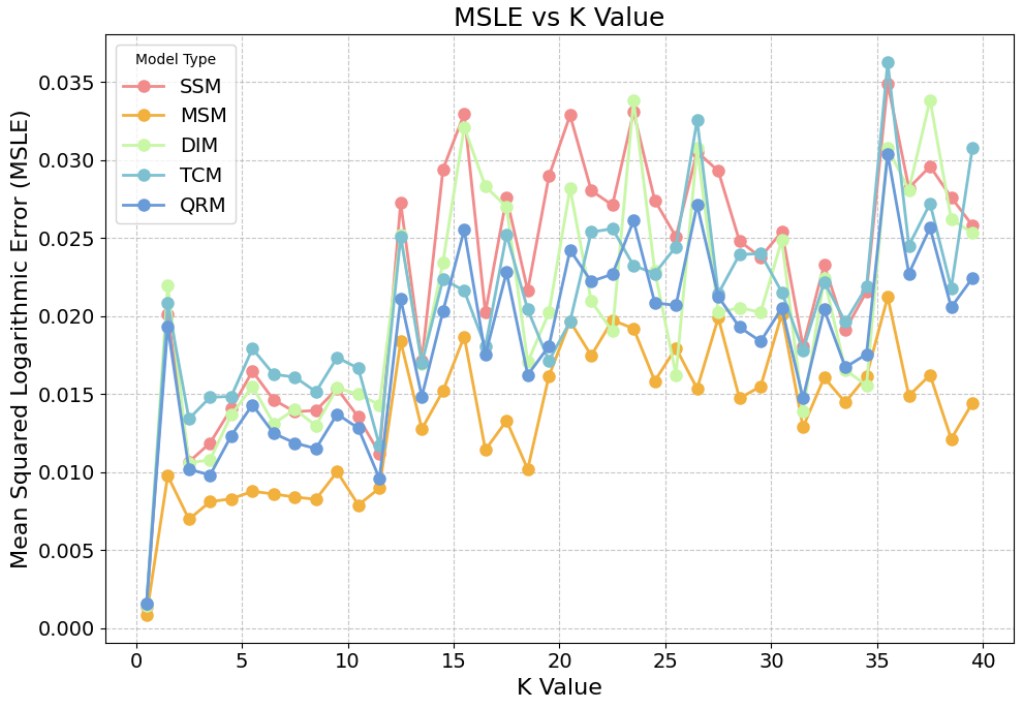

Figure 6.6: Comparison of Mean Squared Logarithmic Error (MSLE) as a function of real K values by motion model. The models include: Single-State Model (SSM), Multi-State Model (MSM), Dimerization Model (DIM), Transient-Confinement Model (TCM), and Quenched-Trap Model (QTM).

6.3 Conclusion

After evaluating the system and model performance through both the competition platform and local validation, it can be confirmed that the models deliver strong performance across all tracks and tasks. The results were competitive with other models in the validation phase, achieving similar outcomes. Furthermore, the alignment between local validation metrics and competition results demon-

strates the high quality of the generated data, the effective separation of train and test sets, and the robustness of the model training process.

CHAPTER 7
Results

In this section, the final results of the competition are discussed in detail. Following the completion of the 2nd Andi Challenge on July 15, 2024, the models developed as part of this Bachelor's Thesis by the **ICSO UPV** team achieved second place in both video tracks, as illustrated in Table 7.1 and Table **??**. Among more than 20 participating teams, only two teams successfully submitted models for all tasks and tracks, covering both video and simple trajectory categories, with **ICSO UPV** being one of them.

Table 7.1: Final Result of the Ensemble Task on the Video Track.

Ranking	Team	MRR	W1 (K)	alpha rank	K rank	W1 (alpha)
1	SPT-HIT	0.400	0.058	1	1	0.259
2	BIOMED-UCA	0.167	0.330	2	3	0.273
2	***ICSO UPV***	***0.167***	***0.143***	***3***	***2***	***0.380***

Table 7.2: Final Result of the Single Trajectory Task on the Video Track.

Ranking	Team	RMSE (CP)	JSC (CP)	MAE (alpha)	MSLE (K)	F1 (diff. type)	RMSE rank	JI rank	MAE rank	MSLE rank	F1 rank	MRR
1	SU-FIONA	2.948	0.301	0.224	0.578	0.910	3	1	1	3	1	0.733
2	***ICSO UPV***	***2.508***	***0.216***	***0.425***	***0.185***	***0.827***	***1***	***3***	***3***	***1***	***3***	***0.600***
3	SPT-HIT	2.800	0.272	0.297	0.288	0.848	2	2	2	2	2	0.500

A key factor in this success was the design choice to make all components of the models modular. This approach provided the flexibility needed to adapt the models to the specific requirements of each track, enabling participation in all categories. The modular structure was particularly advantageous in overcoming several technical challenges, such as the difficulty in identifying particles from the VIP matrix, which generated multiple errors. By isolating and addressing these issues within the modular framework, the models maintained high performance across all tasks.

The models performed strongly in the video track, with results closely aligned with the other two competitors. This consistent performance highlights the robustness and adaptability of the models developed, as well as the effectiveness of the methodologies employed.

As a result of achieving a **Top 5** position in one of the competition tracks, the methodology developed in this Bachelor's Thesis will be included in an article that is "in principle accepted" for publication in the journal **Nature Communications**, with co-authorship being granted. Additionally, a presentation on the methodology will be delivered at the Andi Workshop, scheduled to take place at the University of Gothenburg in June 2025.

CHAPTER 8
Conclusions

8.1 Objetives

In conclusion, the objectives defined at the outset of this Bachelor's Thesis have been successfully achieved, as demonstrated in the results chapter. The outcomes not only met but exceeded expectations, showcasing the effectiveness of the methodologies employed. With the results presented, it is hoped that this work will make a significant contribution to the field of anomalous diffusion research and push the boundaries of scientific understanding. By advancing knowledge and providing new insights, this thesis aims to foster further exploration and innovation in the study of the diffusion processes.

8.2 Relation with Study Porgram

The following courses, taken as part of the Bachelor's in Data Science, provided the essential knowledge and practical experience that made this work possible:

- **Programming Fundamentals (14002), Programming (14003), Algorithmics (14007):** These courses provided the foundational programming skills necessary for integrating the various modules of the model, including prediction and data generation.
- **Evaluation, Deployment, and Monitoring of Models (14028), Descriptive and Predictive Models I (14010), Descriptive and Predictive Models II (14011):** The machine learning and deep learning techniques covered in these courses enabled the successful training of the models used in the thesis.
- **Project III, Data Analysis (14021):** This course's emphasis on scientific rigor allowed for high-level collaboration with researchers from various research centers as part of the AnDi Challenge.
- **Infrastructure for Data Processing (14016):** This course provided knowledge on different cloud infrastructures and guided the selection of Google Colab to optimize resource utilization.

Special recognition goes to the course **Artificial Neural Networks and Deep Learning (054307)**, which I attended during my exchange semester at Politecnico di Milano. In this course, I was introduced to the use of U-Nets and explored various neural network architectures in depth.

Bibliography

[1] G. Muñoz-Gil *et al.*, "Quantitative evaluation of methods to analyze motion changes in single-particle experiments," *In-principle accepted at Nature Communications (Registered Report Phase 1)*, 2023. doi: 10.48550/arXiv.2311.18100.

[2] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1-6. doi: 10.1109/ICCUBEA.2018.8697857.

[3] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the graphics processing unit (GPU)," *IEEE Micro*, vol. 41, no. 6, pp. 42-51, 2021. doi: 10.1109/MM.2021.3113475.

[4] P. Rajendra, A. Girisha, and T. G. Naidu, "Advancement of machine learning in materials science," *Materials Today: Proceedings*, vol. 62, pp. 5503-5507, 2022. doi: 10.1016/j.matpr.2022.04.238.

[5] Y. Kumar and M. Mahajan, "Recent advancement of machine learning and deep learning in the field of healthcare system," in *Computational Intelligence for Machine Learning and Healthcare Informatics*, vol. 1, 2020, pp. 77. doi: 10.1515/9783110648195.

[6] G. Muñoz-Gil *et al.*, "Objective comparison of methods to decode anomalous diffusion," *Nature Communications*, vol. 12, no. 1, pp. 6253, 2021. doi: 10.1038/s41467-021-26320-w.

[7] A. Einstein, "Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen," *Annalen der Physik*, vol. 4, 1905. doi: 10.1002/andp.19053220806.

[8] M. Weiss, "Single-particle tracking data reveal anticorrelated fractional Brownian motion in crowded fluids," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 88, no. 1, pp. 010101, 2013. doi: 10.1103/PhysRevE.88.010101.

[9] W. Wang, A. G. Cherstvy, X. Liu, and R. Metzler, "Anomalous diffusion and nonergodicity for heterogeneous diffusion processes with fractional Gaussian noise," *Physical Review E*, vol. 102, no. 1, pp. 012146, 2020. doi: 10.1103/PhysRevE.102.012146.

[10] B. B. Mandelbrot and J. W. Van Ness, "Fractional Brownian motions, fractional noises and applications," *SIAM Review*, vol. 10, no. 4, pp. 422-437, 1968. doi: 10.1137/1010093.

[11] C. Eggeling *et al.*, "Direct observation of the nanoscale dynamics of membrane lipids in a living cell," *Nature*, vol. 457, no. 7233, pp. 1159-1162, 2009. doi: 10.1038/nature07596.

[12] A. I. Shushin, "Anomalous two-state model for anomalous diffusion," *Physical Review E*, vol. 64, no. 5, pp. 051108, 2001. doi: 10.1103/PhysRevE.64.051108.

[13] T. Sungkaworn *et al.*, "Single-molecule imaging reveals receptor–G protein interactions at cell surface hot spots," *Nature*, vol. 550, no. 7677, pp. 543-547, 2017. doi: 10.1038/nature24264.

[14] Y. Chen, B. Yang, and K. Jacobson, "Transient confinement zones: a type of lipid raft?," *Lipids*, vol. 39, no. 11, pp. 1115-1119, 2004. doi: 10.1007/s11745-004-1337-9.

[15] G. Muñoz-Gil, B. Requena, G. Fernández Fernández, H. Bachimanchi, J. Pineda, and C. Manzo, "AnDiChallenge/andi_datasets: AnDi Challenge 2". *Zenodo*, dic. 05, 2023. doi: 10.5281/zenodo.10259556.

[16] T. Miyaguchi and T. Akimoto, "Anomalous diffusion in a quenched-trap model on fractal lattices," *Physical Review E*, vol. 91, no. 1, pp. 010102, 2015. doi: 10.1103/PhysRevE.91.010102.

[17] C. Manzo, and M.F. Garcia-Parajo. "A review of progress in single particle tracking: from methods to biophysical insights". *Reports on Progress in Physics*, 78(12), 124601. doi:10.1088/0034-4885/78/12/124601

[18] M. K. Cheezum, W. F. Walker, and W. H. Guilford, "Quantitative comparison of algorithms for tracking single fluorescent particles," *Biophysical Journal*, vol. 81, no. 4, pp. 2378-2388, 2001, doi: 10.1016/S0006-3495(01)75884-5.

[19] J. C. Crocker and D. G. Grier, "Methods of digital video microscopy for colloidal studies," *J. Colloid Interface Sci.*, vol. 179, no. 1, pp. 298-310, 1996, doi: 10.1006/jcis.1996.0217.

[20] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91-110, 2004, doi: 10.1023/B.0000029664.99615.94.

[21] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer, 2006, vol. 3951, pp. 404-417, doi: 10.1007/11744023_32.

[22] R. Killick, P. Fearnhead, and I. A. Eckley, "Optimal detection of changepoints with a linear computational cost," *Journal of the American Statistical Association*, vol. 107, no. 500, pp. 1590-1598, 2012. doi: 10.1080/01621459.2012.737745.

[23] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, pp. 107299, 2020. doi: 10.1016/j.sigpro.2019.107299.

[24] A. Gentili and G. Volpe, "Characterization of anomalous diffusion classical statistics powered by deep learning (CONDOR)," *Journal of Physics A: Mathematical and Theoretical*, vol. 54, no. 31, pp. 314003, 2021. doi: 10.1088/1751-8121/ac0c5d.

[25] A. J. Scott and M. Knott, "A cluster analysis method for grouping means in the analysis of variance," *Biometrics*, pp. 507-512, 1974. doi: 10.2307/2529204.

[26] J. Bai, "Estimating multiple breaks one at a time," *Econometric Theory*, vol. 13, no. 3, pp. 315-352, 1997. doi: 10.1017/S0266466600005831.

[27] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 2016, pp. 785-794. doi: 10.1145/2939672.2939785.

[28] J. Janczura *et al.*, "Classification of particle trajectories in living cells: Machine learning versus statistical testing hypothesis for fractional anomalous diffusion," *Physical Review E*, vol. 102, no. 3, pp. 032402, 2020. doi: 10.1103/PhysRevE.102.032402.

[29] Ò. Garibo-i-Orts *et al.*, "Efficient recurrent neural network methods for anomalously diffusing single particle short and noisy trajectories," *Journal of Physics A: Mathematical and Theoretical*, vol. 54, no. 50, pp. 504002, 2021. doi: 10.1088/1751-8121/ac3707.

[30] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference*, Munich, Germany, October 5-9, 2015, Proceedings, Part III, vol. 18, Springer International Publishing, 2015, pp. 234-241. doi: 10.1007/978-3-319-24574-4_28.

[31] R. Booto Tokime, X. Maldague, and L. Perron, "Automatic Defect Detection for X-Ray Inspection: A U-Net Approach for Defect Segmentation," *Digital Imaging and Ultrasonics for NDT 2019*, July 2019.

[32] O. Oktay et al., "Attention U-Net: Learning Where to Look for the Pancreas," *arXiv*, 2018. doi: 10.48550/arXiv.1804.03999.

[33] D. B. Allan, T. Caswell, N. C. Keim, C. M. van der Wely R. W. Verweij, "soft-matter/trackpy: v0.6.4". Zenodo, jul. 10, 2024. doi: 10.5281/zenodo.12708864.

[34] D. Domingo-Calabuig, S. Hoyas, R. Vinuesa, and J.A. Conejero, "1(5), 810-812, ACS Sust. Res. Manag." 2024. doi: 10.1021/acssusresmgt.4c00074.

APPENDIX A
Python Code

```
1
2  import pandas as pd
3  import numpy as np
4  import random
5  from andi_datasets.datasets_challenge import
       challenge_phenom_dataset, _get_dic_andi2, _defaults_andi2
6  from google.colab import drive
7  drive.mount('/content/drive')
8
9  def single_state(alpha):
10   values = _get_dic_andi2(1)
11   values["T"] = 208
12   values["N"] = random.randint(80, 120)
13   values["Ds"] = np.array([random.choice([random.uniform(0.0001, 1)
         , random.uniform(1,10), random.uniform(10,40)]), 0.01])
14   values["alphas"] = np.array([alpha, 0.01])
15   return values
16
17 def multi_state(alpha):
18   values = _get_dic_andi2(2)
19   values["T"] = 208
20   values["N"] = random.randint(60, 120)
21   alpha2 = random.uniform(0.001, 1.99999)
22   while abs(alpha2 - alpha) < 0.2:
23     alpha2 = random.uniform(0.001, 1.99999)
24   if random.random() > 0.5:
25     k1 = random.choice([random.uniform(1,5), random.uniform(5,10),
           random.uniform(10,40)])
26     k2 = random.uniform(0.01, 2)
27     values["alphas"] = np.array([[alpha, 0.01],[alpha2, 0.01]])
28     values["Ds"] = np.array([[k1, 0.01],[k2, 0.01]])
29   else:
30     k2 = random.choice([random.uniform(1,5), random.uniform(5,10),
           random.uniform(10,40)])
31     k1 = random.uniform(0.01, 2)
32     values["alphas"] = np.array([[alpha2, 0.01],[alpha, 0.01]])
33     values["Ds"] = np.array([[k2, 0.01],[k1, 0.01]])
34
35   return values
36
37 def inmobile_traps(alpha):
38   values = _get_dic_andi2(3)
39   values["T"] = 208
```

```
  values["N"] =  random.randint(80, 120)
  ks = random.choice([random.uniform(0.5,2), random.uniform(2,10),
      random.uniform(10,40)])
  values["Ds"] = np.array([ks, 0.01])
  values["alphas"] = np.array([alpha, 0.01])
  values["Nt"] = random.randint(250, 400)
  values["r"] = random.uniform(0.35, 0.6)
  values["Pu"] = random.uniform(0.01, 0.02)
  return values

def dimerization(alpha):
  values = _get_dic_andi2(4)
  values["T"] = 208
  values["N"] =  random.randint(80, 120)
  alpha2 = random.uniform(0.001, 1.99999)
  while abs(alpha2 - alpha) < 0.2:
    alpha2 = random.uniform(0.001, 1.99999)

  k1 = random.choice([random.uniform(1,4), random.uniform(4,10),
      random.uniform(10,60)])

  values["alphas"] = np.array([[alpha, 0.01],[alpha2, 0.01]])
  values["Ds"][0][0] = k1
  values["Pu"] = random.uniform(0.01, 0.03)
  values["Pb"] = random.uniform(0.01, 0.03)

  return values

def confinement(alpha):
  values = _get_dic_andi2(5)
  values["T"] = 208
  values["N"] =  random.randint(60, 120)
  alpha2 = random.uniform(0.001, 1.99999)

  k1 = random.choice([random.uniform(1,4), random.uniform(4,10),
      random.uniform(10,60)])
  if alpha < values["alphas"][0][0]:
    if alpha >= (values["alphas"][0][0]-0.2):
      values["alphas"][1][0] = alpha-0.35
    else:
      values["alphas"][1][0] = alpha
  elif alpha <= (values["alphas"][0][0]+0.2):
    values["alphas"][0][0] = alpha + 0.35
  else:
    values["alphas"][0][0] = alpha

  values["Ds"][0][0] = k1

  values["Nc"] = random.randint(30, 45)
  values["trans"] = random.uniform(0.1, 0.15)
  values["r"] = random.uniform(4.5, 6)
  return values

for generation_batch in [1,2,3,4,5]:
  experiments = []
  model_used = []
  model_type = []
  for i in np.linspace(1.9, 0.01, 20):
    for _ in range(40):
```

```
96        alpha = random.uniform(i, i+0.09)
97        if alpha >= 1.9:
98          model_type.extend([1]*5)
99        else:
100         model_type.extend([0]*5)
101
102       experiments.append(single_state(alpha))
103       model_used.extend([1])
104       experiments.append(multi_state(alpha))
105       model_used.extend([2])
106       experiments.append(inmobile_traps(alpha))
107       model_used.extend([3])
108       experiments.append(dimerization(alpha))
109       model_used.extend([4])
110       experiments.append(confinement(alpha))
111       model_used.extend([5])
112
113   dfs = []
114   models = []
115   modeltype = []
116   for i, experiment in enumerate(experiments):
117     try:
118       df, _, _ = challenge_phenom_dataset(save_data = False, # If
              to save the files
119                                           dics = [experiment], #
                                                 Dictionaries with
                                                 the info of each
                                                 experiment (and FOV
                                                 in this case)
120                                           return_timestep_labs =
                                                 True, get_video =
                                                 False,
121                                           num_fovs = 5, # Number
                                                 of FOVs
122                                           files_reorg = False)
123       dfs.extend(df)
124       models.extend([model_used[i]]*5)
125       modeltype.extend([model_type[i]]*5)
126     except:
127       print(i, " error")
128
129
130   trajs_list = []
131   coef_list = []
132   states_list = []
133   for df in dfs:
134       input = np.zeros((64,208, 2))
135       coef = np.zeros((64,208, 2))
136       state = np.zeros((64,208))
137       df["state"] = df["state"] + 1
138       grouped = df.groupby(['traj_idx'])
139       for _,group in grouped:
140           part = int(_[0])
141           if part<64:
142             org, dest = int(group["frame"].iloc[0]), int(group["
                    frame"].iloc[-1])
143             input[part, org:dest+1, 0] = group["x"]/128
144             input[part, org:dest+1, 1] = group["y"]/128
145             coef[part, org:dest+1, 0] = group["alpha"]
```

```
            coef[part, org:dest+1, 1] = group["D"]
            state[part, org:dest+1] = group["state"]

      trajs_list.append(input)
      coef_list.append(coef)
      states_list.append(state)

  np.savez_compressed(f"/content/drive/MyDrive/data/trajs_train_{generation_batch}.npz", np.array(trajs_list))
  np.savez_compressed(f"/content/drive/MyDrive/data/coef_train_{generation_batch}.npz", np.array(coef_list))
  np.savez_compressed(f"/content/drive/MyDrive/data/state_train_{generation_batch}.npz", np.array(states_list))
  np.savez_compressed(f"/content/drive/MyDrive/data/modelgen_train_{generation_batch}.npz", np.array(models))
  np.savez_compressed(f"/content/drive/MyDrive/data/modeltype_train_{generation_batch}.npz", np.array(modeltype))
```

Listing A.1: Python script to generate simulate experiments

```

from sklearn.cluster import KMeans
import numpy as np
import ruptures as rpt
from collections import Counter
from scipy.optimize import linear_sum_assignment
from andi_datasets.utils_videos import import_tiff_video
import trackpy as tp
import pandas as pd
import keras
import os

# Function to expand the borders of each frame in the video
def expand_borders(v, fm=False):
    expanded_images = []
    for frame in v:
        # Set the median pixel value for padding
        img_median = 255 if fm else np.median(frame)

        # Expand borders by padding the edges of the frame
        img_borders = np.vstack((frame[-4:, :], frame, frame[:4, :]))
        img_borders = np.hstack((img_borders[:, -4:], img_borders, img_borders[:, :4]))

        # Set the expanded border areas to the median value
        img_borders[:4, :] = img_median
        img_borders[-4:, :] = img_median
        img_borders[:, :4] = img_median
        img_borders[:, -4:] = img_median

        expanded_images.append(img_borders)

    return np.array(expanded_images)

# Function to track particles in a video and return trajectories
def video2traj(raw_video):
    # Expand borders of video frames to avoid boundary issues
    raw_video = expand_borders(raw_video)
```

```
38
39      # Detect particles in the video frames
40      f = tp.batch(raw_video, diameter=3, invert=True, minmass=13,
            separation=2.6, max_iterations=10)
41
42      # Link particles to form trajectories
43      try:
44          traj = tp.link(f, 25, memory=10, neighbor_strategy="BTree",
                link_strategy="auto")
45      except:
46          traj = tp.link(f, 4, memory=10, neighbor_strategy="BTree",
                link_strategy="auto")
47
48      # Return sorted trajectories
49      return traj[["particle", "frame", "x", "y"]].sort_values(by=["
            particle", "frame"], ignore_index=True)
50
51  # Function to locate particles in the first frame and map them to
        detected particles
52  def loc_particles(firstframe, df_video):
53      indxparticles = {}
54
55      # Extract the first frame data
56      df_firstframe = df_video[df_video["frame"] == 0]
57      firstframe = expand_borders([firstframe])[0]
58
59      # Identify unique particle IDs in the first frame
60      items = np.unique(firstframe)[:-1]
61      positions = []
62      for itm in items:
63          pos = np.where(firstframe == itm)
64          positions.append((np.mean(pos[0]), np.mean(pos[1])))
65
66      positions_array = np.array(positions)
67      detected_array = df_firstframe[['y', "x"]].values
68      distances = np.sqrt(((detected_array[:, np.newaxis, :] -
            positions_array) ** 2).sum(axis=2))
69
70      # Solve the assignment problem to match detected particles to
            identified particles
71      row_ind, col_ind = linear_sum_assignment(distances)
72      indxparticles = {}
73
74      for detected, assigned in zip(row_ind, col_ind):
75          if len(np.where(firstframe == items[assigned])[0]) >= 3:
76              indxparticles[int(detected)] = int(items[assigned])
77
78      return indxparticles
79
80  # Function to load videos, track particles, and locate them
81  def load_videos(dir, n_fov):
82      fovs = []
83      vipfovs = []
84      print("Loading videos ...")
85      for fov in range(n_fov):
86          print(f"Video {fov}")
87          video = import_tiff_video(dir + f"videos_fov_{fov}.tiff")
88          vidtrajs = video2traj(video[1:])
89          locpart = loc_particles(video[0], vidtrajs)
```

```
        fovs.append(vidtrajs)
        vipfovs.append(locpart)
    print("Videos loaded")
    return fovs, vipfovs

# Function to read fields of view (FOVs) and prepare data for prediction
def read_fovs(nfov, dir_data):
    fovs_trajs = []
    fovs_trajs_2 = []
    fovs_id = []
    fovs_rang_trajs = []
    fovs_ind_trajs = []

    # Load videos and trajectories
    fovs_video, vipfovs = load_videos(dir_data, nfov)

    for fov, df in enumerate(fovs_video):
        fovs_id.append(fov)
        input = np.zeros((64, 208, 2))
        rang_trajs = []
        real_ind = []
        grouped = df.groupby(['particle'])
        indvips = 0

        # Process each particle group
        for _, group in grouped:
            part = int(_[0])
            org = int(group["frame"].iloc[0])
            dest = org + len(group["x"])
            if part in vipfovs[fov].keys():
                rang_trajs.append((org, dest))
                real_ind.append(vipfovs[fov][part])
                input[indvips, org:dest, 0] = group["x"]
                input[indvips, org:dest, 1] = group["y"]
                indvips += 1

        # Compute the difference between consecutive frames
        for i in range(1, 208):
            input[:, -i] = input[:, -i] - input[:, -i - 1]

        if len(real_ind) != 10:
            print(f"len real {len(real_ind)}")

        fovs_ind_trajs.append(real_ind)
        fovs_rang_trajs.append(rang_trajs)
        input_copy = input.copy()
        input[:, 0] = np.zeros((64, 2))
        fovs_trajs.append(input)
        fovs_trajs_2.append(input_copy)

    return fovs_trajs, fovs_trajs_2, fovs_rang_trajs, fovs_id, vipfovs, fovs_ind_trajs

# Function to normalize the predicted signal
def normalizar_pred(signal):
    for i in range(1, len(signal) - 2):
        if signal[i - 1] != signal[i] and (signal[i + 1] != signal[i] or signal[i + 2] != signal[i]):
```

```
146             signal[i] = signal[i - 1]
147     return signal
148 
149 # Function to detect change points in a signal using rupture
150 def change_points_model(pred_states, org, dest):
151     try:
152         signal = normalizar_pred(pred_states[org:dest])
153         model = "l2"  # Model used for segmentation
154         algo = rpt.Window(width=10, model=model, jump=1, min_size
                =3).fit(signal)
155         cp = algo.predict(pen=1)
156         return cp
157     except:
158         return []
159 
160 # Function to detect change points in a signal using rupture
    without normalization
161 def change_pointsnolog(pred_coef, org, dest):
162     try:
163         signal = pred_coef[org:dest]
164         model = "l2"  # Model used for segmentation
165         algo = rpt.Window(width=18, model=model, jump=1, min_size
                =3).fit(signal)
166         cp = algo.predict(pen=5)
167         return cp
168     except:
169         return []
170 
171 # Function to perform K-Means clustering on two lists of values
172 def k_means_clusters(lista1, lista2):
173     # Combine the two lists into a 2D array
174     data = list(zip(lista1, lista2))
175 
176     # Apply K-Means with K=2
177     kmeans = KMeans(n_clusters=2, random_state=0, n_init="auto").
            fit(data)
178 
179     # Get cluster labels
180     labels = kmeans.labels_
181 
182     # Group data by cluster
183     clusters = [[[], []], [[], []]]
184     for i, label in enumerate(labels):
185         clusters[label][0].append(lista1[i])
186         clusters[label][1].append(lista2[i])
187     return clusters
188 
189 # Function to handle missing parts in prediction files
190 def missing_parts(archivo, fov, exp, dir_data):
191     files_missing = []
192     try:
193         with open(archivo, 'r', encoding='utf-8') as file:
194             lineas = file.readlines()
195             if len(lineas) != 10:
196                 files_missing.append((archivo, dir_data + ""))
197     except:
198         pass
199 
200 # Function to predict trajectories and save results
```

```
def pred_trajs_fov_video(nfov, dir_pred, dir_data, unet_alpha,
    unet_ks, unet_states):
    # Read FOVs and prepare data
    fovs_trajs, fovs_trajs2, fovs_rang_trajs, fovs_id, vips,
        fovs_ind_trajs = read_fovs(nfov, dir_data)

    print("Predicting videos")

    # Predict alpha and k values using the models
    pred_alphas = unet_alpha.predict(np.array(fovs_trajs), verbose
        =0)
    pred_ks = unet_ks.predict(np.array(fovs_trajs), verbose=0)
    pred_trajs = np.concatenate((pred_ks, pred_alphas), axis=-1)
    pred_states = np.argmax(unet_states.predict(np.array(
        fovs_trajs2), verbose=0), axis=-1)

    states = 1
    ensembles_alphas = []
    ensembles_ks = []
    preb = -1

    # Check the mode of predicted states
    states_stats = np.unique(pred_states, return_counts=True)
    ch_mode = 0
    if len(states_stats[1]) == 3:
        if states_stats[1][1] > 200:
            ch_mode = 1

    if not os.path.exists(dir_pred):
        os.makedirs(dir_pred)

    # Process each FOV and write results to files
    for fov in range(nfov):
        submission_file = dir_pred + f'/fov_{fov}.txt'
        with open(submission_file, "w") as f:
            for id_part, ran in enumerate(fovs_rang_trajs[fov]):
                img = fov
                if ch_mode == 0:
                    cp_part = change_pointsnolog(pred_trajs[img,
                        id_part], ran[0], ran[1])
                else:
                    cp_part = change_points_model(pred_states[img,
                        id_part], ran[0], ran[1])

                if len(cp_part) > 0:
                    if cp_part[-1] == ran[1] - ran[0]:
                        cp_part = cp_part[:-1]

                if len(cp_part) == 0:
                    alpha = np.median(pred_alphas[img, id_part, ran
                        [0]:ran[1]])
                    ks = np.median(pred_ks[img, id_part, ran[0]:ran
                        [1]])
                    state = Counter(pred_states[img, id_part, ran
                        [0]:ran[1]]).most_common(1)[0][0]
                    state = state - 1
                    if state == 1:
                        alpha = 0
                        ks = 0
```

```
                    if state < 0:
                        print(state)
                    ensembles_alphas += [alpha] * int(ran[1] - ran
                        [0] + 1)
                    ensembles_ks += [ks] * int(ran[1] - ran[0] + 1)
                    res = [fovs_ind_trajs[fov][id_part], ks, alpha,
                        int(state), int(ran[1] - ran[0] + 1)]
                    formatted_numbers = ','.join(map(str, res))
                    f.write(formatted_numbers + '\n')
                else:
                    states = 2
                    alpha = np.median(pred_alphas[img, id_part, ran
                        [0]:cp_part[0] + ran[0]])
                    ks = np.median(pred_ks[img, id_part, ran[0]:
                        cp_part[0] + ran[0]])
                    state = Counter(pred_states[img, id_part, ran
                        [0]:cp_part[0] + ran[0]]).most_common(1)
                        [0][0]
                    state = state - 1
                    if state == 1:
                        alpha = 0
                        ks = 0

                    ensembles_alphas += [alpha] * int(cp_part[0] +
                        1)
                    ensembles_ks += [ks] * int(cp_part[0] + 1)
                    preb_chcount = int(cp_part[0] + 1)

                    res = [fovs_ind_trajs[fov][id_part], ks, alpha,
                        state, int(cp_part[0] + 1)]
                    cp_part = cp_part + [int(ran[1] - ran[0])]
                    for i in range(len(cp_part) - 1):
                        state = Counter(pred_states[img, id_part,
                            int(cp_part[i] + ran[0]):int(cp_part[i +
                             1] + ran[0])]).most_common(1)[0][0]
                        state = state - 1
                        alpha = np.median(pred_alphas[img, id_part,
                            int(cp_part[i] + ran[0]):int(cp_part[i
                            + 1] + ran[0])])
                        ks = np.median(pred_ks[img, id_part, int(
                            cp_part[i] + ran[0]):int(cp_part[i + 1]
                            + ran[0])])
                        if state == 1 and alpha > 1.7:
                            alpha = 0
                            ks = 0

                        ensembles_alphas += [alpha] * (int(cp_part[
                            i + 1] + 1) - preb_chcount)
                        ensembles_ks += [ks] * (int(cp_part[i + 1]
                            + 1) - preb_chcount)
                        preb_chcount = int(cp_part[i + 1] + 1)

                        res.extend([ks, alpha, state, int(cp_part[i
                            + 1] + 1)])
                    formatted_numbers = ','.join(map(str, res))
                    f.write(formatted_numbers + '\n')

    # Save ensemble labels and statistics
    with open(dir_pred + "/ensemble_labels.txt", 'w') as f:
```

```
293         if states == 1:
294             f.write(f'model: single_state; num_state: {int(states)}
                    \n')
295             data = []
296             data.append(np.array([np.mean(np.array(ensembles_alphas
                    )), np.std(np.array(ensembles_alphas)),
297                                   np.mean(np.array(ensembles_ks)),
                                          np.std(np.array(ensembles_ks))
                                          , len(ensembles_alphas)]))
298         else:
299             state_type = "multi_state"
300 
301             if ch_mode == 1:
302                 if states_stats[0][1] == 1:
303                     state_type = "immobile_traps"
304                 if states_stats[0][1] == 2:
305                     state_type = "confinement"
306             f.write(f'model: multi_state; num_state: {int(states)}
                    \n')
307             cluster_states = k_means_clusters(ensembles_alphas,
                    ensembles_ks)
308             data = []
309             data.append(np.array([np.mean(np.array(cluster_states
                    [0][0])), np.std(np.array(cluster_states[0][0])),
310                                   np.mean(np.array(cluster_states
                                          [0][1])), np.std(np.array(
                                          cluster_states[0][1])), len(
                                          cluster_states[0][0])]))
311             data.append(np.array([np.mean(np.array(cluster_states
                    [1][0])), np.std(np.array(cluster_states[1][0])),
312                                   np.mean(np.array(cluster_states
                                          [1][1])), np.std(np.array(
                                          cluster_states[1][1])), len(
                                          cluster_states[1][0])]))
313 
314         data = np.transpose(np.array(data))
315         data[-1, :] /= data[-1, :].sum()
316         # Save the data in the corresponding ensemble file
317         np.savetxt(f, data, delimiter=';')
318 
319 # Function to check if a file has the expected number of lines
320 def comprobar_lineas(archivo):
321     with open(archivo, 'r', encoding='utf-8') as file:
322         lineas = file.readlines()
323         return len(lineas) != 10
324 
325 # Function to solve missing particles by updating prediction files
326 def solve_missing_particles(pred_dir, data_dir, fovs):
327     for fov in range(fovs):
328         misspart = comprobar_lineas(pred_dir + f'/fov_{fov}.txt')
329         if misspart:
330             video = import_tiff_video(data_dir + f"/videos_fov_{fov
                    }.tiff")
331             particles = np.unique(video[0])[:-1]
332             with open(pred_dir + f'/fov_{fov}.txt', 'r', encoding='
                    utf-8') as file:
333                 lineas = file.readlines()
334                 par_in = [int(l.split(",")[0]) for l in lineas]
```

```
                lost_ids = [i for i in particles if i not in par_in
                    ]

            with open(pred_dir + f'/fov_{fov}.txt', 'a', encoding='
                utf-8') as f:
                for part in lost_ids:
                    print(part)
                    res = [part, 0, 0, 2, 5]
                    formatted_numbers = ','.join(map(str, res))
                    f.write(formatted_numbers + '\n')

# Load pre-trained models
unet_alpha = keras.models.load_model("/models/alphas-3-2-1024-v5-
    epoch-2.keras", compile=False) # CHANGE DIRECTORY
unet_ks = keras.models.load_model(f"/models/ks-3-2-1024-newdata-
    epoch-3.keras", compile=False) # CHANGE DIRECTORY
unet_states = keras.models.load_model(f"/models/states-3-6-128-v1-
    epoch-10.keras", compile=False) # CHANGE DIRECTORY

# Process each experiment
for exp in range(12):
    print("Pred. file ", exp)
    dir_data = rf"/data/public_data_challenge_v0/track_1/exp_{exp}/
        "  # CHANGE DIRECTORY
    dir_pred = rf"/data/track_1/exp_{exp}"  # CHANGE DIRECTORY
    pred_trajs_fov_video(30, dir_pred, dir_data, unet_alpha,
        unet_ks, unet_states)
    solve_missing_particles(dir_pred, dir_data, 30)
```

Listing A.2: Python script to make the predictions of the validation and competition data for the 2nd Andi Challenge.

APPENDIX B
Sustainable Development Goals

Degree of Relation of the Work with the Sustainable Development Goals (SDGs).

Sustainable Development Goals	High	Medium	Low	Not Related
SDG 1. **No Poverty.**				X
SDG 2. **Zero Hunger.**				X
SDG 3. **Good Health and Well-being.**		X		
SDG 4. **Quality Education.**		X		
SDG 5. **Gender Equality.**				X
SDG 6. **Clean Water and Sanitation.**				X
SDG 7. **Affordable and Clean Energy.**				X
SDG 8. **Decent Work and Economic Growth.**				X
SDG 9. **Industry, Innovation.**	X			
SDG 10. **Reduced Inequality.**				X
SDG 11. **Sustainable Cities and Communities.**				X
SDG 12. **Responsible Consumption and Production.**				X
SDG 13. **Climate Action.**				X
SDG 14. **Life Below Water.**				X
SDG 15. **Life on Land.**				X
SDG 16. **Peace, Justice, and Strong Institutions.**				X
SDG 17. **Partnerships for the Goals.**		x		

Alignment with Sustainable Development Goals (SDGs)

This Bachelor Thesis, titled Machine Learning-based Characterization of Single-Particle Behavior with Synthetic Experiment Videos, aligns closely with several Sustainable Development Goals (SDGs), demonstrating its broader impact beyond the immediate scope of particle tracking and video analysis [34].

SDG 3: Good Health and Well-Being The study's focus on understanding particle behavior within living cells has direct implications for improving health outcomes. By accurately characterizing single-particle dynamics, we enhance our comprehension of cellular processes and their anomalies, which can lead to more precise diagnostic tools and targeted therapies. Understanding how particles move and interact in biological systems is crucial for identifying early biomarkers for diseases, developing new treatments, and improving overall health diagnostics. This research thus contributes to the advancement of medical science and public health, supporting SDG 3.

SDG 9: Industry, Innovation, and Infrastructure The application of Machine Learning to analyze experimental videos represents a significant innovation in the field of biophysics and cell biology. By leveraging advanced algorithms to track and characterize particle behavior, this work exemplifies how cutting-edge technology can drive scientific progress. The integration of Machine Learning models in this context not only enhances our ability to interpret complex biological data but also sets a precedent for future research methodologies. This contributes to the broader goals of SDG 9 by promoting the use of technology and innovation to solve real-world problems.

SDG 4: Quality Education The methodologies developed and the insights gained from this thesis have educational value. They contribute to the field of computational biology and machine learning by providing new tools and techniques for analyzing biological data. Sharing these findings with the scientific community supports SDG 4 by enhancing knowledge dissemination and fostering a deeper understanding of cellular dynamics among researchers, educators, and students. The techniques and models used in this research can serve as valuable educational resources for those studying biophysics, data science, and related fields.

SDG 17: Partnerships for the Goals Participation in the 2nd Anomalous Diffusion (AnDi) Challenge reflects a commitment to collaborative efforts in advancing scientific research. Such challenges often involve collaboration between researchers, institutions, and industries, fostering partnerships that drive innovation and progress. By contributing to this challenge, the thesis supports SDG 17 by emphasizing the importance of cooperation and knowledge exchange in achieving common goals.