A supervised Graph-based deep learning algorithm to detect and quantify clustered particles

Lucas A. Saavedra, Alejo Mosqueira and Francisco J. Barrantes

Supplementary Material

Transformation of localization datasets into Graphs

Since GNNs take a graph as input, localization datasets need to be converted into a graph format following a suitable criterion. One such condition is the spatiotemporal criterion (ST) proposed by Manzo and coworkers to study cell dynamics (Pineda et al., 2023). In this criterion, localizations (considered as "nodes") are linked with an edge if they are at a distance of r nanometers and less than w frames apart. However, as explained in the main text, the ST criterion may not be ideally suited for cluster analysis.

We chose instead to transform normalized localization datasets into a Graph using a Delaunay triangulation criterion (DT) for GNN1 and a multidimensional Delaunay Triangulation (MDT) for GNN2. Given their spatiotemporal nature, DT and MDT do not require high memory requirements and are not time consuming. In the DT criterion, the normalized localization dataset is transformed following the Delaunay triangulation on axes 'x', 'y', and time. Once the data is transformed, each node is assigned normalized coordinates in the region of interest (ROI) and each edge is assigned the spatial distance between adjacent localizations. As DT edges may not be sufficient to detect clusters once all non-clustered localizations are removed (remaining connected components after edge removal are the detected clusters), we then implemented the MDT criterion, which results in a graph with an increased number of edges. MDT is constituted by the union of the edges obtained with the Delaunay triangulation applied on four projections of the localization dataset: (i) on axes x and y, (ii) on x and time (t) axis, (iii) on y and t, and (iv) on axes x, y, and t. In this criterion, node attributes are the normalized coordinates of the localization and edge attributes are the spatial distance between localizations and the absolute difference of time between the two localizations.

Since the only encoded attribute of time is the difference between localizations, the method can analyze localization datasets with the different recorded frames used in the training process, as exemplified in Figures 1 in the main text. Figure 2 compares the computational and performance attributes of DT and MDT with spatiotemporal criteria.

Architecture details

We employed the MAGIK architecture (Pineda et al., 2023) as implemented in the DeepTrack software package (Midtvedt et al., 2021). For the input to the Graph Neural Networks (GNNs), GNN1 employs two node features (the 'x' and 'y' coordinates) and one edge feature (the spatial distance between localizations). GNN2 also uses the same node features as GNN1, but its edge features include both spatial and temporal distances between localizations, resulting in two edge features. The input is then processed by a node and edge encoder, which converts the node and edge features into latent representations of dimension 96, respectively. Each encoder is constructed using two normalized sequential dense layers with 64 and 96 units, employing the Gaussian Error Lineal Unit (GELU) activation function (Hendrycks, 2016). The latent representations are further updated by three repeated fingerprinting graph blocks (FGNNs) with an output dense layer with 96 units each, which update each edge and node latent representation through a gated-based attention mechanism comprised of 12 attention heads. The edge latent representations are weighted by the localization spatial distances through a learnable Gaussian attention mechanism (Pineda et al., 2021). Finally, the updated latent representations are decoded to obtain predictions for nodes and edges in GNN1 and GNN2, respectively. The Sigmoid activation function is used for the output in both GNNs. The architecture comprises approximately 400,000 trainable parameters, a complexity suitable for inference and training on standard desktop computers.

Training Procedure

We simulated 340 localization datasets of 1,000 frames each. Fifty of these datasets are simulations without clusters. One hundred and forty of the 340 datasets (~40%) were separated for validation and the remaining were used for training. Before starting the training, each training dataset was converted into a Graph following the transformation required for a given GNN. Classification labels were also prepared to train the GNNs. In the case of GNN1, each node was assigned a label 1 or 0 depending on whether or not the node corresponded to a localization in a cluster, respectively. In the case of GNN2, each edge was assigned a label 0 or 1 depending on whether the edge was an inter- or intra-cluster relationship, respectively. Next, the number of Graphs for GNN2 was duplicated: a filtered Graph and a non-filtered Graph were generated from a dataset to help GNN2 detect misclassified positive localizations from GNN1. Next, a minimum number of 512 graphs were generated from these Graphs through a data augmentation procedure: first, a Graph was selected and randomly partitioned into several nodes or edges for classification, as for the inference procedure. Partition sizes varied between 500 and 4,000, the step size was kept at 500, and GNNs were trained with each size separately. Next, the sub-Graphs were randomly rotated. In this way, one can generate a large variety of Graphs with a low number of localization datasets such that every epoch works with new Graphs during the entire training procedure, thus preventing overfitting and increasing model

generality. The network was trained on up to 100 epochs for GNN1 and 10 epochs for GNN2. No early stopping mechanism was applied in either case, and the batch size was set to 1 to prevent batches with Graphs with different selected partition sizes (batch size different from 1 implies artifactual graph modifications). Training GNN1 and GNN2 under a specific partition size required ~4 h and ~10 min, respectively.

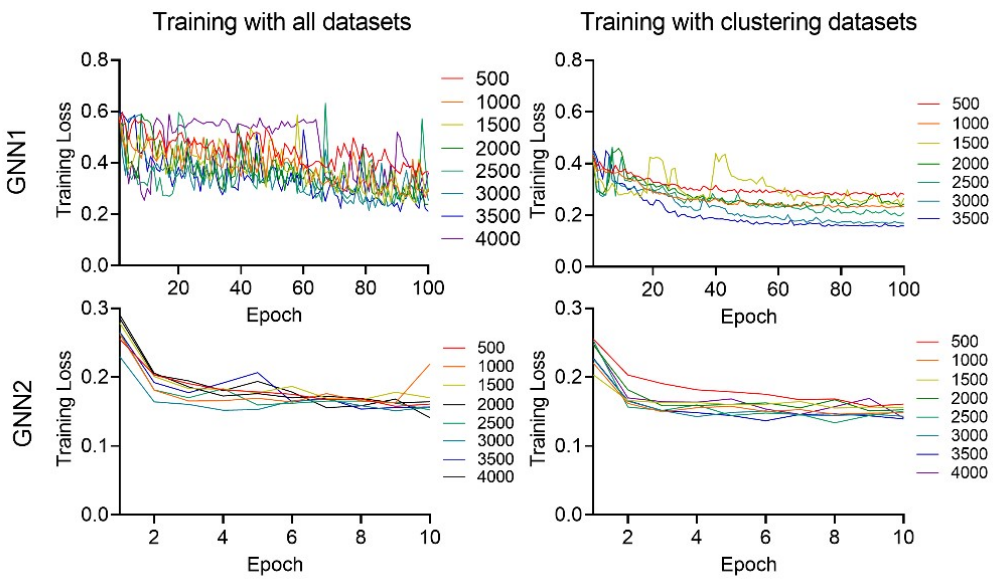
Both GNNs were subjected to two different training conditions. One condition included all the training datasets, and the second one excluded those simulated datasets with no clusters. The performance of these training conditions is shown in the Results section. Both architectures were trained using a loss binary cross-entropy function, and a value between 0 and 1 was predicted with a sigmoid function for every node and edge, depending on the task. To classify the nodes (GNN1) and edges (GNN2), if the predicted output of these was greater than a threshold, θ, it was considered positive. In the case of GNN1, automatic threshold selection was implemented using GHOST, an algorithm that determines which threshold fits better on imbalanced data (Esposito et al., 2021). To prevent overwhelmingly long training times and excessive memory consumption, for GNN2 we used a value of θ*=0.5*. All training and inference procedures were executed on a PC with an Nvidia TITAN V GPU with 64GB RAM. The proposed architecture and training were implemented with the TensorFlow package.

Training Loss

The binary cross-entropy function is the loss function used to train all the neural networks. In the case of edges, the function is defined as:

$$L(y_e,p_e) = y_e * \log(p_e) + (1 - y_e) * log(1 - p_e)$$

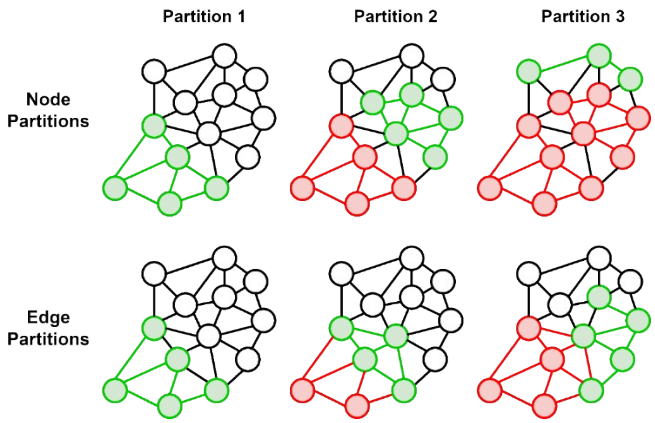
where y_e is the edge true label and p_e is the edge predicted value. In the case of nodes, it is the same function, except that nodes are evaluated. Supplementary Figure 1 shows how the average training loss (i.e., the averaged evaluation of the loss function across all edges or nodes, depending on the GNN) changes as training advances.



Supplementary Figure 1. Training loss of GNNs. The left column shows training of GNNs when all datasets are included in the training process and the right column the corresponding training process when datasets with no clusters are excluded. Notice that in the case of GNN1, training is more stable when datasets without clusters are not included during training. The partition size is colour-coded as indicated on the right of each figure.

Inference Procedure

For binary node classification, the localization dataset was transformed into a Graph using the DT criterion. Due to memory limitations, the Graph is partitioned into different subGraphs such that every node appears only once across all partitions and nodes are temporally sorted. The resultant Graph is fed into the GNN1 to classify nodes. For binary edge classification, the filtered localization dataset is transformed into a Graph using the MDT criterion, and the Graph is partitioned into different subGraphs such that every edge appears only once across all partitions. The resulting Graph is fed into the GNN2 to classify edges. Supplementary Figure 2 shows an example of how nodes and edges are considered through partitions for these tasks with a Graph of 13 nodes and 27 edges. It is important to note that the last node or edge partition will not necessarily have the size defined by the user, as the number of nodes/edges is not inevitably divisible by the selected partition size.



Supplementary Figure 2. Edge and node partitions. In the case of node partitions, the example shows sub-Graphs with 5 nodes (except for the last partition, which has only 3 nodes). The green objects correspond to the current partition, whereas the red objects represent objects that have already been iterated. In the case of edge partitions, all nodes are iterated and some of them may be re-iterated several times. In contrast, in node partitions some edges may not be iterated.

Graph Criterion Utility (GCU) definition

Here we developed two criteria based on DT to build Graphs from datasets involving space and time dimensions. As the selected criterion has a notable impact on the prediction performance, we further introduced a metric called Graph Criterion Utility (GCU) that depends on another metric, Adjusted Rand Index Score (ARI), representing the performance of the task to be accomplished if the algorithm classifies/infers perfectly under a specific criterion. In other words, GCU is the maximum performance reachable by the algorithm if a specific criterion is applied. In the specific task of this work, GCU_{ARI} measures how well clusters are detected if GNN1 and GNN2 classify with 100% accuracy using the DT criterion. To achieve this, we began by filtering unclustered localizations from ground-truth datasets, effectively treating each dataset as perfectly classified by GNN1. Subsequently, these filtered datasets were converted into a compatible graph for GNN2, where we eliminated all inter-cluster edges, assuming perfect classification by GNN2. The remaining connected components of this manipulated graph were considered the obtained clusters $C^{'}$. Finally, we computed GCU_{ARI} as $ARI(C,C')$ such that C represents the set of ground-truth clusters.

Measurement of dataset balance

To quantify the balance of a dataset (a perfectly balanced dataset has the same number of samples for every defined class), we applied the non-negative relative Shannon entropy (also

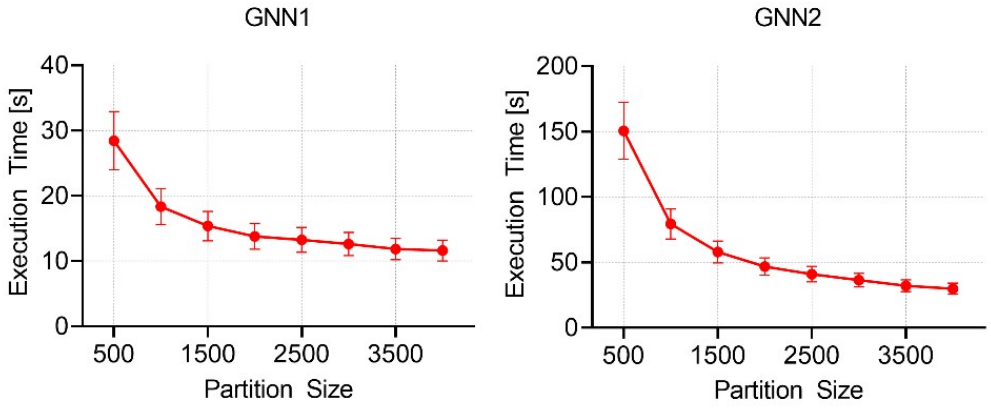
known as Kullback-Leibler divergence) which is useful to measure the distance between observed and expected distributions (Shannon, 1948). Given a dataset with m samples where each sample is assigned a class (or category) $k \in [1,2,\ldots,K]$, K is the number of classes, m_k is the number of samples belonging to class k, and $p_k = \frac{m_k}{m}$ is the proportion of samples in the dataset belonging to class k. The relative Shannon entropy is defined as:

$$\sum_{k=1}^{K} p_k * \log \frac{p_k}{q_k}$$

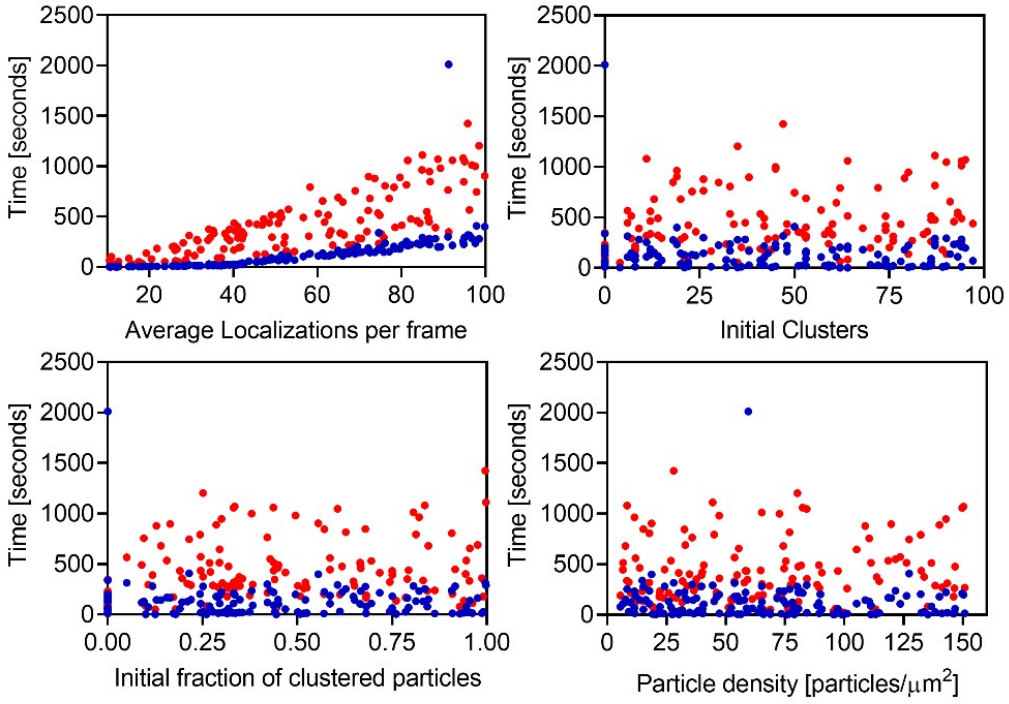
where q_k is the expected proportion of samples belonging to class k. If the dataset is balanced, observed proportions are expected to be $\frac{1}{K}$ (i.e., the number of samples for each class is uniformly distributed). The expression can be rewritten as:

$$\sum_{k=1}^{K} p_k * \log \frac{p_k}{q_k} = \sum_{k=1}^{K} p_k * \log \frac{p_k}{1/K} = \sum_{k=1}^{K} p_k * \log (p_k K) = Balance(p_1,p_2,\ldots,p_K)$$

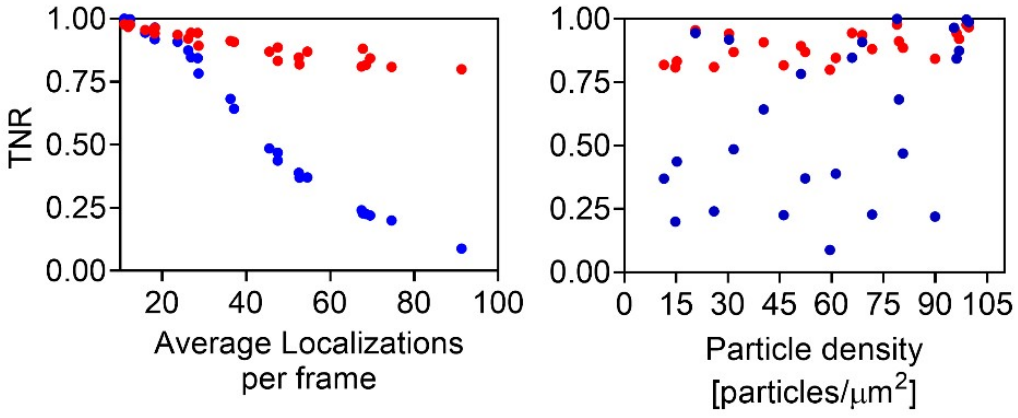
which is zero for a completely balanced dataset ($p_1 = p_2 = \ldots = p_K = 1/K$). Therefore, the more balanced a dataset is, the more the expression approaches 0. The balance was calculated with the relative Shannon entropy implemented in SciPy (Virtanen et al., 2020).



Supplementary Figure 3. GNN1 and GNN2 execution times for different partition sizes. Each red point represents the mean of the time within which GNNs executed each validation dataset of 1,000 frames and error bars represent the 95% confidence interval of the mean.



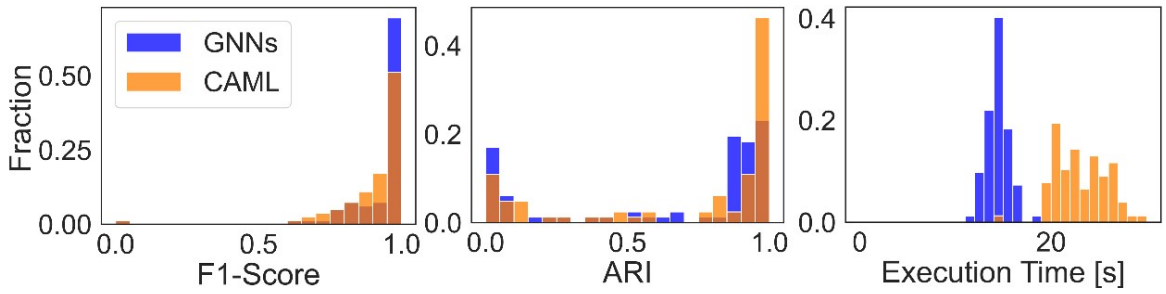
Supplementary Figure 4. Execution times of the proposed GNN algorithm (red dots) vs qSR (blue dots) on datasets of 1,000 frames across different simulation conditions. qSR is faster than the proposed algorithm and the speed linearly decreases with the average localization per frame in both cases ($p<0.0001$).



Supplementary Figure 5. TNR of the proposed algorithm (red dots) vs qSR (blue dots) on datasets of 1,000 frames across different simulation conditions on datasets without clusters. With both algorithms, TNR decreases as the average number of locations per frame increases ($p<0.0001$). With GNNs, we observed greater robustness in these situations.

GNNs applied to static samples

In order to assess the performance of the GNN-based algorithm on static cluster data, we resorted to datasets containing the type of information such as that reported in ref. (Williamson et al., 2020). Suppl. Figure 6 compares the CAML architecture and GNN in terms of F1-Score, ARI, and execution time. The GNN-based algorithm outperforms CAML both in terms of execution time and predictive performance.

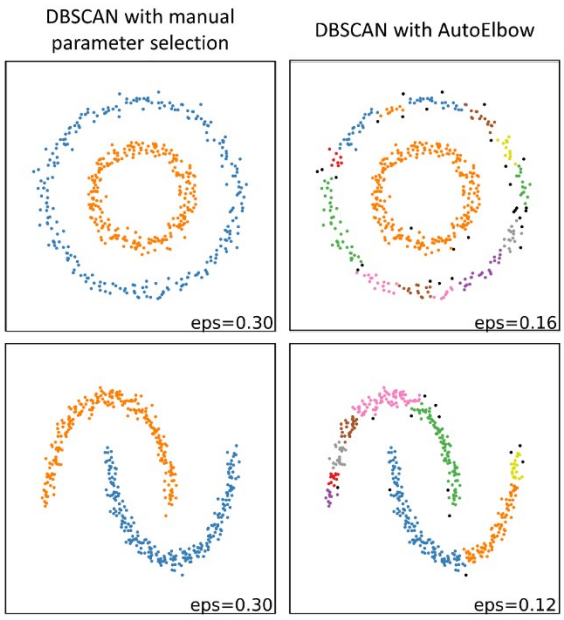


Supplementary Figure 6. GNNs vs CAML performance metrics on static datasets. Histograms of F1-Score (left), ARI (middle), and execution time (right) for GNN and CAML algorithms.

AutoElbow for DBSCAN

We applied some recently developed algorithms to automatically select parameters for cluster analysis. One of these is AutoElbow (Adeiza James Onumanyi et al., 2022)., which detects the elbow point of an elbow-based Graph and automatically selects parameters to further implement cluster analysis. These authors showed that the algorithm performs well for k-means that consider as evaluation measure the total WSD (TWSD), obtained as the sum of the distances between each point in a cluster and its centroid (Adeiza James Onumanyi et al., 2022) according to a set of values for k, the number of clusters. Ester and coworkers (Ester et al., 1996) explained how eps (maximum distance at which a set of points are considered as neighbours) can be selected according to a visual inspection of the k-distance Graph: the "valley" of this Graph is equivalent to the "elbow" in k-means. When we implemented AutoElbow for DBSCAN and executed it on 2 toy datasets with 2 clusters in each dataset, the number of clusters detected by automatic selection was overestimated in comparison to manual selection. Supplementary Figure 7 shows the results obtained with AutoElbow

compared to manual parameter selection on Scikit-Learn toy datasets (Pedregosa et al., 2011). We provide a Python implementation on the code repository of the work of AutoElbow.



Supplementary Figure 7. AutoElbow for DBSCAN. Each square represents the results of cluster analysis with DBSCAN, employing different eps. The left column shows the results under manual parameter selection and the right column under automatic parameter selection using AutoElbow. Each cluster is colour-coded, and the eps value is shown in the bottom-right corner of each example.

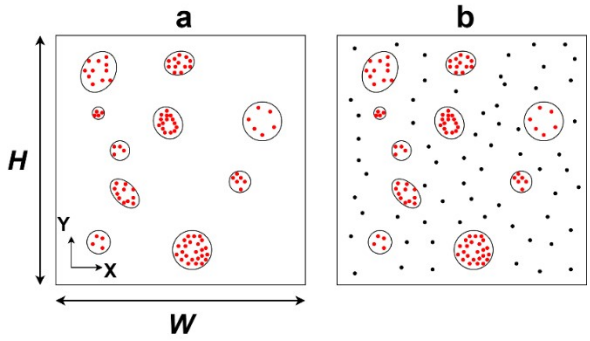
Simulation Details

As there are no available ground-truth datasets to train and validate the present algorithm, a simulation framework was designed and implemented with Python to recreate a wide range of experimental conditions. To accelerate simulations, we implemented CythonBuilder on the written code to compile Python scripts into Cython code (Behnel et al., 2011).

i) Cluster initialization

An experimental instance was created with a ROI of height H and width W. Next, a number between N_{INF} and N_{SUP} of elliptic clusters (as this is the typical 2D-shape in most experimentally found nAChR nanoclusters (Mosqueira et al., 2018, 2020)) was created, where each cluster had a different number of initial particles $n \in [n_{inf}, n_{sup}]$, width r_w and height r_h, such that $r_w, r_h \in [r_{inf}, r_{sup}]$, and cluster eccentricity < ε_{max}. Additionally, each cluster was oriented with an angle $\theta \in [0, 2\pi]$ and its centroid was randomly positioned across the ROI.

Each cluster centroid was assigned a diffusion coefficient $D_c \in [D_{c_{inf}}, D_{c_{sup}}]$, and a right-skewed randomly generated lifetime t_{life} with a minimum $t_{lifemin}$ and maximum $t_{lifemax}$. Once all clusters were generated inside the defined ROI (Supplementary Figure 8a), non-clustered particles were added until the percentage of clustered particles reached a value $\sim Clustered_{Percentage}$ (Supplementary Figure 8b). Due to memory limitations, the total number of particles in the simulated data cannot be larger than N_{MAX}.



Supplementary Figure 8. Initialization steps of the simulations. a) Appearance of the simulated datasets upon addition of all clustered particles (red) inside the ROI of width *W* and height *H*. b) After all clusters were created, non-clustered single particles (black) were added.

ii) Particle and cluster dynamics

Each clustered particle was allowed to move strictly inside its own cluster with a diffusion coefficient between $\frac{D_c}{2}$ and $D_c * 2$, thus allowing the particle to diffuse slower or faster than the cluster centroid. Also, each given particle was allowed to be retained with a probability of 0.05 (i.e., allowing some particles that have no chance of being retained) and remain inside the cluster $t_{residence}$ seconds from a hypo-exponential distributed time defined by $t_{residence}{}_1$ and $t_{residence}{}_2$. Particles non belonging to a cluster were allowed to move with a diffusion coefficient between D_{inf} and D_{sup}. Both clustered and non-clustered particles were assigned an anomalous exponent $\alpha \in [\alpha_{min}, \alpha_{max}]$ while cluster centroids were asigned to follow Brownian motion with a corresponding diffusion coefficient.

iii) Particle states

Let us consider the topological condition of a particle as its "state". There are 4 possible states that a particle can assume as a function of time: (i) *freely diffusing* outside a cluster, i.e.,

not clustered; (ii) *clustered*; iii) *leaving* a cluster, or iv) *immobile*. When a particle stays in a cluster (state (ii)) for a period > $t_{residence}$ seconds, it transits to state (iii). When the particle moves further away from the cluster centroid and leaves the cluster, it reacquires state (i), i.e., *non-clustered*, diffusing freely in a Brownian fashion. Each cluster is assigned an immobilization function, $f_{immobilization}$, which determines the probability that a particle is immobilized in the cluster as a function of its position. If the particle remains immobile (i.e., in state (iv)), it eventually leaves the cluster. Once it leaves the cluster, the particle has a 0.5 probability of staying immobile or regaining free motion as an isolated particle. When a free moving particle collides with a cluster, it instantly transitions from state (i) to state (ii). When a cluster concludes its assigned lifetime, all particles transition back to state (i), and when all particles leave the cluster, the latter disappears.

iv) Probability immobilization functions

To model the probability of immobilization of a given particle depending on its position in the cluster, we considered three probabilistic functions: (i) f_{linear} which models linear probability increases as the particle moves towards the cluster centroid; (ii) $f_{quadratic}$, which models quadratic behaviour, and (iii) $f_{discrete}$, which models region-dependent behaviour. These three functions depend on cluster shape and on the maximum and minimum probability of immobilization, annotated as p_{max} and p_{min} respectively. f_{linear} is defined as:

$$f_{linear}(x,y) = -\sqrt{\left(\frac{x*(p_{max}-p_{min})}{\frac{r_w}{2}}\right)^2+\left(\frac{y*(p_{max}-p_{min})}{\frac{r_h}{2}}\right)^2}+p_{max}$$

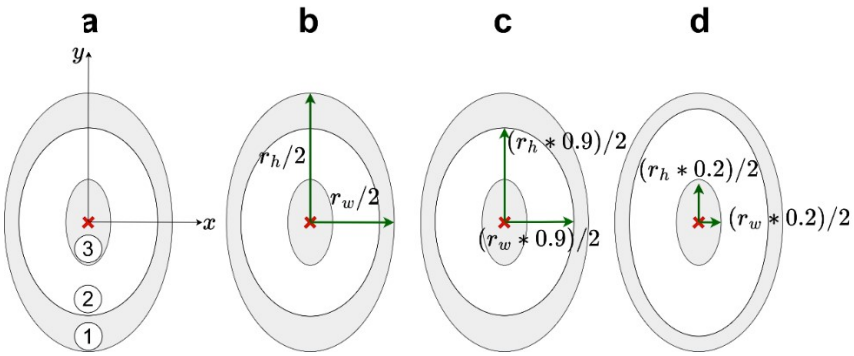
where x, y are the horizontal and vertical position of the particle relative to the cluster centroid position, r_w and r_h are the width and height of the cluster, respectively.

Then, $f_{quadratic}$ is defined as:

$$f_{retention_j}(x,y) = -\frac{\left(\frac{x}{\sqrt{\frac{1}{(p_{max}-p_{min})}}}\right)^2}{\left(\frac{r_w}{2}\right)^2}-\frac{\left(\frac{y}{\sqrt{\frac{1}{(p_{max}-p_{min})}}}\right)^2}{\left(\frac{r_h}{2}\right)^2}+p_{max}$$

Finally, $f_{discrete}$ is defined as a composite function:

$$f_{discrete}(x,y) = f(x) = \begin{cases} p_{min}, & if\ (x,y)\ is\ inside\ region\ 1 \\ p_{min} + \dfrac{p_{max} - p_{min}}{2}, & if\ (x,y)\ is\ inside\ region\ 2 \\ p_{max}, & if\ (x,y)\ is\ inside\ region\ 3 \\ 0, & otherwise \end{cases}$$



Supplementary Figure 9. Regions of the discrete function. a) shows the regions 1, 2 and 3 defined in the composite function. b), c) and d) show the dimensions of the ellipses that define regions 1, 2, and 3, respectively. r_h and r_w are the length of the major and minor axis of the defined ellipse of a nanocluster, respectively.

v) Cluster merging

Two clusters were allowed to merge if they overlapped with each other. For the sake of simplicity, the overlap is the number of particles that belong to the intersection of the twoclusters. Such overlap is defined as:

$$S(C_j, C_i) = \frac{|C_j \cap C_i|}{|C_j \cup C_i|}$$

where C_j and C_i are clusters. In this way, the overlap is the percentage of particles that belong to both clusters simultaneously. It is a Jaccard index (Jaccard, 1901). If $S(C_j, C_i) > S_{min}$, the two clusters are merged.

vi) Modified Hosking method to simulate anomalous diffusion

The Hosking method is an exact method to simulate trajectories that behave according to fractional Brownian motion (fBm) (Hosking, 1984). This algorithm considers a trajectory length L and an anomalous exponent α and returns displacements $[\Delta_1, \Delta_2, \ldots, \Delta_L]$ as output

parameters. However, this algorithm is not applicable to the framework presented here, because the particle is tracked and controlled step-by-step, thus allowing the generation of new clusters, incorporation of particles into clusters, and particles leaving the clusters. Hence, the trajectory cannot be generated instantaneously but step-by-step. We therefore introduced a modified Hosking algorithm that considers the maximum length of the trajectory L_{MAX}, a step l, and an anomalous exponent α and returns the displacements $[\Delta_1, \Delta_2,\ldots,\Delta_l]$ such that Δ_l depends on the previous stored displacements. To implement this modified algorithm, we used the Python modules *stochastic* and *fbm*. Trajectories were scaled as in the Andi Datasets software package (Muñoz-Gil, 2023; Muñoz-Gil et al., 2021; Muñoz-Gil et al., 2020).

vii) Formation of new clusters

At each step of the simulation, the framework scanned the ROI in search of new clusters. Once all clusters and particles moved one step, a virtual and empty cluster was initialized as described above. The procedure was iterated on the non-clustered particles to create new clusters. If P_i is a non-clustered particle, it was incorporated into a given cluster. Next, the nearest non-clustered particle, P_{i+1}, was selected and added to the same virtual cluster, re-setting the cluster centroid as the center of mass of *both* particles. If both particles fell inside this newly generated cluster, the process was iterated, selecting more neighbouring particles and repeating the process until a new particle could not be incorporated into the cluster because it did not comply with the average centroid of the cluster. In this case, the non-fitting particle was removed. The number of particles in the cluster was checked next: if it was greater than n_{inf}, the cluster was no longer considered virtual, and its occurrence in the simulation was confirmed. The entire cluster generation procedure was iterated until all non-clustered particles were taken into account.

viii) Localization dataset generation

In STORM experiments, particles exhibit non-continuous blinking throughout the entire duration of the imaging experiment. In the simulation process, particles were allowed to blink between $frames_{consecutives_{min}}$ and $frames_{consecutives_{max}}$ consecutive frames. An average between $localization_{averagemin}$ and $localization_{averagemax}$ of localizations was considered per frame. Finally, a Gaussian-distributed noise was added to all localizations, with mean μ_{noise} and standard deviation σ_{noise}.

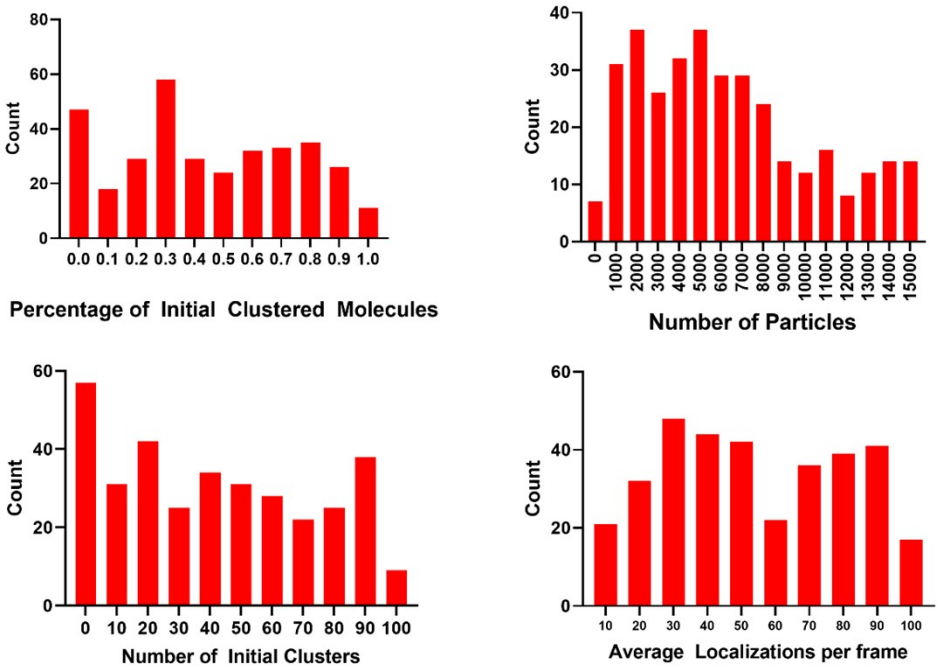
ix) Selected range of simulation parameters

Some of the parameters used in the simulations (except for probability values, S_{min}, and $t_{residence_i}$) are based on previous experimental work from our laboratory (Mosqueira et al., 2018, 2020). Supplementary Table 1 shows the value of the simulation parameters. The frequency distributions of simulation parameters at the beginning of each simulated experiment (Supplementary Figure 10).

Supplementary Table 1. List of parameters used in the simulations

Parameter	Value
W	$10\ \mu m$
H	$10\ \mu m$
N_{MAX}	10,000 *particles*
$N_{C_{INF}}$	10 *particles*
$N_{C_{SUP}}$	100 *particles*
r_{inf}	$20\ nm$
r_{sup}	$200\ nm$
D_{cinf}	$1 * 10^{-5}\ \mu m^2/s$
D_{csup}	$0.01\ \mu m^2/s$
D_{inf}	$1 * 10^{-5}\ \mu m^2/s$
D_{sup}	$0.7\ \mu m^2/s$
$Clustered_{Percentagemin}$	$\sim 0\%$
$Clustered_{Percentagemax}$	$\sim 100\%$
t_{frame}	$10\ ms$
$t_{residence_1}$	*0.50 s*
$t_{residence_2}$	*0.02 s*
ε_{max}	0.6
$t_{life_{min}}$	25 *frames*
$t_{life_{max}}$	7,000 *frames*
p_{min}	0.01
p_{max}	0.5

α_{min}	0.1
α_{max}	1.9
S_{min}	10%
$\bar{e}$	40 nm
σ_e	10 nm
$frames_{consecutives_{min}}$	2 frames
$frames_{consecutives_{max}}$	5 frames
$localization_{averagemin}$	10
$localization_{averagemax}$	100



Supplementary Figure 10. Frequency distributions of initial values of some key variables in the simulated datasets. The distribution of the initial percentage of molecules within a cluster is depicted in the top-left histogram. The top-right histogram illustrates the distribution of the number of particles in the experiment. The bottom-left histogram depicts the distribution of the initial number of clusters. Finally, the distribution of the average number of locations per frame is presented in the bottom-right histogram.

qSR implementation

qSR is a MATLAB software designed to identify clusters in single-molecule data in fixed- or live-cell samples (Andrews et al., 2018) using the DBSCAN algorithm. qSR v1.1.0 (https://www.github.com/cisselab/qSR/releases/tag/v1.1.0) was implemented as part of the GNN algorithm benchmarking. A length scale of 0.1 µm, a temporal tolerance of 1 s, and a minimum cluster size of 20 were chosen. The selected parameters are like those used in (Mosqueira et al., 2020) for STORM and (Escamilla-Ayala et al., 2020) for PALM.

CAML implementation

CAML v1.0 (https://gitlab.com/quokka79/caml/-/releases/v1.0) was implemented for comparison with the GNN-based approach. We used the model 07VEJJ provided in the v1.0 release. We reutilized the validation and training datasets provided in https://osf.io/xa4zj/ .

NASTIC and segNASTIC implementation

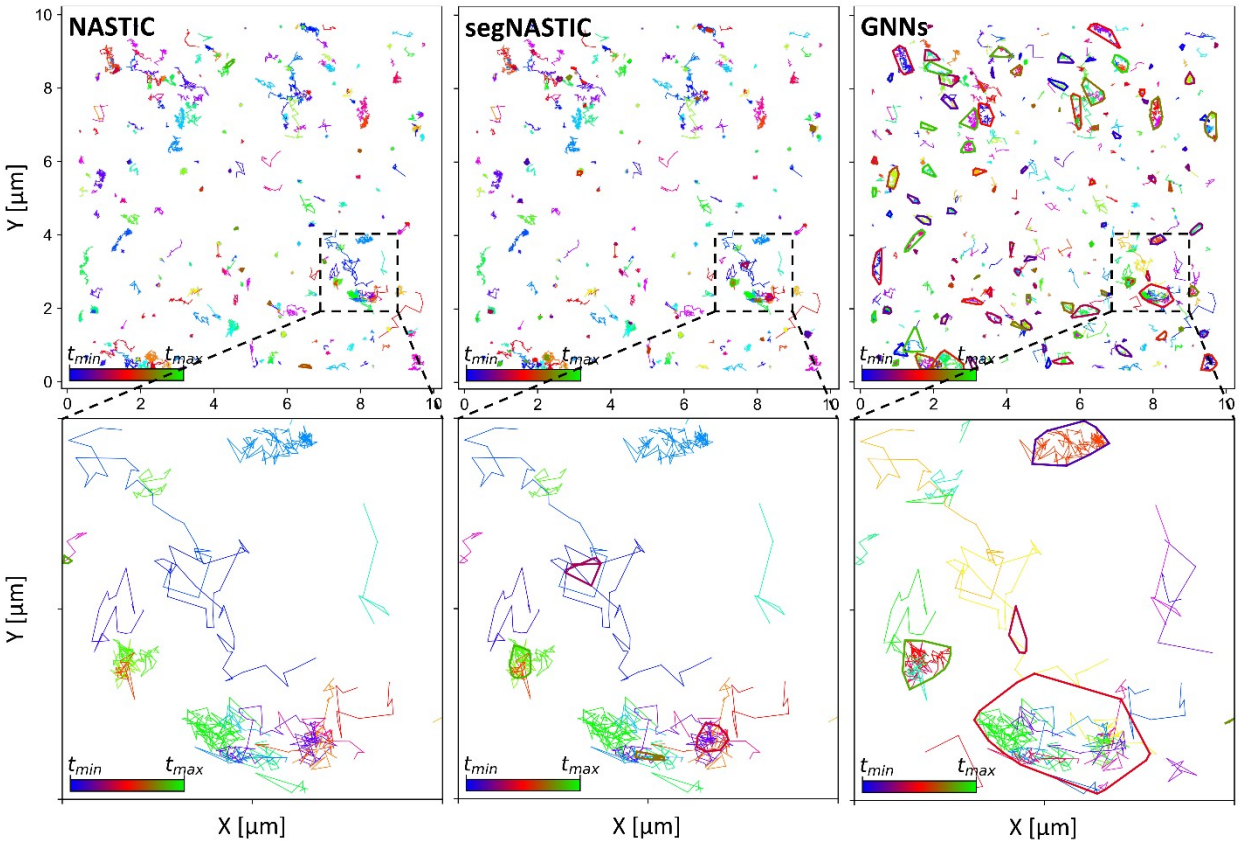
Parameter-based algorithms are easy to implement and settings to execute them are easy to select. However, the selected parameters may not be constant across different datasets and experimental conditions and results strongly rely on the initial choice of parameters and are thus more prone to parameter-related biases, which may be subjective in nature. Some algorithms are available, however, to automatically select parameters (Adeiza J. Onumanyi et al., 2022). However, this strategy is not foolproof and may lead to overestimation of the number of clusters (Suppl. Figure 7). In addition, parameter-based algorithms are generally distance-based, and may not work correctly as the number of dimensions in the dataset increases (Keogh & Mueen, 2017; Scurll, 2022).

NASTIC and segNASTIC, two recently introduced algorithms to analyze clusters based on trajectories' overlap, are examples of methodologies where the parameters condition the results (Wallis et al., 2023). NASTIC and segNASTIC require, along with other parameters, time thresholds. It is not possible to detect cluster durations below half the time threshold in NASTIC (minimum cluster duration obtained was 0.62 s with a threshold of 1 s). Decreasing this threshold -and hence the overlap- led to total failure in detecting clusters, whereas increasing the threshold led to detection of only spatial clustering (Wallis et al., 2023). In contrast, segNASTIC detected shorter cluster durations with the same threshold time (minimum cluster duration obtained was 0.1 ms, below the frame rate). Clearly, results are conditioned by the selected parameters and parameter selection is hard to determine. Additionally, the GNN-based algorithm does not require experimental trajectory data, a step that is required by both NASTIC and segNASTIC.

Suppl. Figure 11 depicts in graphic form the results obtained with NASTIC and segNASTIC and those resulting from the current GNN-based approach on a representative STORM experimental dataset. Application of NASTIC and segNASTIC to the present experimental

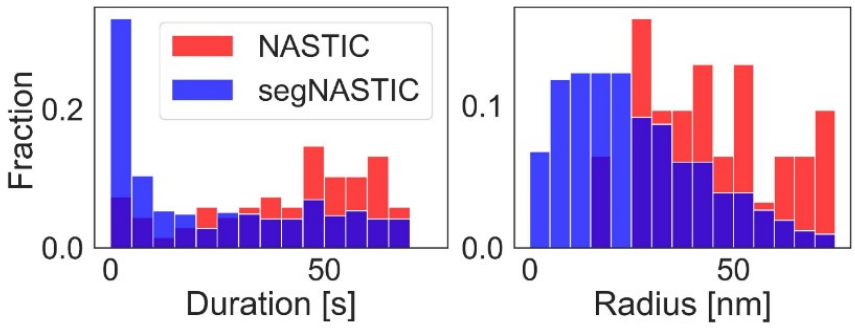
STORM and TIRF datasets produced clusters with durations of 41.39 ± 7.16 s and 23.95 ± 5.55 s, respectively, which appear to be overestimated. In addition, the calculated radii in STORM datasets were 78 nm ± 4nm (NASTIC) and 30 nm ± 3nm (segNASTIC), two values that appear to be underestimated.

Suppl. Figure 12 further illustrates they key spatiotemporal cluster metrics (duration and size) resulting from application of NASTIC and segNASTIC to STORM datasets.



Supplementary Figure 11. Comparison of the NASTIC and segNASTIC spatiotemporal cluster indexing detection technique (Wallis et al., 2023) with the current GNN-based method. The upper row shows an experimental dataset of nAChR particles labelled with fluorescent bungarotoxin. The lower rows expand the ROIs outlined by the black dashed boxes in the upper row. The contours that surround clusters are their convex hulls. Each convex hull is assigned a colour according to the average acquisition time of the particles contained in them. The coloured bar in each lower-left corner shows t_{min} and t_{max} (15 s and 20 s, respectively). NASTIC and segNASTIC fall short of detecting cluster processes that the GNN algorithm identifies successfully. NASTIC v1.0.6 analysis (https://github.com/tristanwallis/smlm_clustering/releases/tag/v1.0.6) was performed with a time threshold of 1 s and a radius factor 1.2. In the case of segNASTIC (which is included in NASTIC v1.0.6), time was set with a threshold

of 1 s, a radius factor 1.2, a segment threshold of 5, an overlap threshold override of 0, and cluster size screen of 0.15 µm.



Supplementary Figure 12. Cluster metrics resulting from application of NASTIC and segNASTIC on STORM datasets.

Statistical Analysis

All statistical analyses were implemented using the Prism GraphPad 8 software. Correlations were measured with a two-tailed nonparametric Spearman correlation with a confidence interval of 95% and a significance threshold of 0.05 (level at which the null hypothesis is rejected). To compare two distributions, we used the Kolmogorov-Smirnov (KS) test for two samples. To compare more than two distributions, we used the Kruskal-Wallis (KW) test. Mean ± 95% confidence intervals are shown unless otherwise stated. No data points were excluded. ARI and F1-Score were measured using Scikit-Learn Python library implementation.

References

Andrews, J. O., Conway, W., Cho, W. K., Narayanan, A., Spille, J. H., Jayanth, N., Inoue, T., Mullen, S., Thaler, J., & Cissé, I. I. (2018). qSR: a quantitative super-resolution analysis tool reveals the cell-cycle dependent organization of RNA Polymerase I in live human cells. *Sci Rep*, *8*(1), 7424. https://doi.org/10.1038/s41598-018-25454-0

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2011). Cython: The Best of Both Worlds. *Computing in Science & Engineering*, *13*(2), 31-39. https://doi.org/10.1109/mcse.2010.118

Escamilla-Ayala, A. A., Sannerud, R., Mondin, M., Poersch, K., Vermeire, W., Paparelli, L., Berlage, C., Koenig, M., Chavez-Gutierrez, L., Ulbrich, M. H., Munck, S., Mizuno, H., & Annaert, W. (2020). Super-resolution microscopy reveals majorly mono- and dimeric presenilin1/γ-secretase at the cell surface. *eLife*, *9*, e56679. https://doi.org/10.7554/eLife.56679

Esposito, C., Landrum, G. A., Schneider, N., Stiefl, N., & Riniker, S. (2021). GHOST: Adjusting the Decision Threshold to Handle Imbalanced Data in Machine Learning. *Journal of Chemical Information and Modeling*, *61*(6), 2623-2640. https://doi.org/10.1021/acs.jcim.1c00160

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise* Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon.

Hendrycks, D., Gimpel, K. (2016). Gaussian error linear units (GELUs). *arXiv E-prints*, *arXiv:1606.08415*.

Hosking, J. R. M. (1984). Modeling persistence in hydrological time series using fractional differencing. *Water Resources Research*, *20*(12), 1898-1908. https://doi.org/https://doi.org/10.1029/WR020i012p01898

Keogh, E., & Mueen, A. (2017). Curse of Dimensionality. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning and Data Mining* (pp. 314-315). Springer US. https://doi.org/10.1007/978-1-4899-7687-1_192

Midtvedt, B., Helgadottir, S., Argun, A., Pineda, J., Midtvedt, D., & Volpe, G. (2021). Quantitative digital microscopy with deep learning. *Applied Physics Reviews*, *8*(1). https://doi.org/10.1063/5.0034891

Mosqueira, A., Camino, P. A., & Barrantes, F. J. (2018). Cholesterol modulates acetylcholine receptor diffusion by tuning confinement sojourns and nanocluster stability. *Sci Rep*, *8*(1), 11974. https://doi.org/10.1038/s41598-018-30384-y

Mosqueira, A., Camino, P. A., & Barrantes, F. J. (2020). Antibody-induced crosslinking and cholesterol-sensitive, anomalous diffusion of nicotinic acetylcholine receptors. *J Neurochem*, *152*(6), 663-674. https://doi.org/10.1111/jnc.14905

Muñoz-Gil, G. (2023). *AnDiChallenge/andi_datasets: andi_datasets 2.0.0 release*. In (Version v.2.0) Zenodo. https://doi.org/10.5281/zenodo.8005576http://dx.doi.org/10.5281/zenodo.8005576

Muñoz-Gil, G., Volpe, G., Garcia-March, M. A., Aghion, E., Argun, A., Hong, C. B., Bland, T., Bo, S., Conejero, J. A., Firbas, N., Garibo i Orts, Ò., Gentili, A., Huang, Z., Jeon, J.-H., Kabbech, H., Kim, Y., Kowalek, P., Krapf, D., Loch-Olszewska, H., . . . Manzo, C. (2021). Objective comparison of methods to decode anomalous diffusion. *Nature Communications*, *12*(1), 6253. https://doi.org/10.1038/s41467-021-26320-w

Muñoz-Gil, G., Volpe, G., García-March, M. A., Metzler, R., Lewenstein, M., & Manzo, C. (2020). *The anomalous diffusion challenge: single trajectory characterisation as a competition* (Vol. 11469). SPIE. https://doi.org/10.1117/12.2567914

Onumanyi, A. J., Molokomme, D. N., Isaac, S. J., & Abu-Mahfouz, A. M. (2022). AutoElbow: An Automatic Elbow Detection Method for Estimating the Number of Clusters in a Dataset. *Applied Sciences*, *12*(15), 7515. https://doi.org/10.3390/app12157515

Onumanyi, A. J., Molokomme, D. N., Isaac, S. J., & Abu-Mahfouz, A. M. (2022). AutoElbow: An Automatic Elbow Detection Method for Estimating the Number of Clusters in a Dataset. *Applied Sciences*, *12*(15).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825-2830. http://jmlr.org/papers/v12/pedregosa11a.html

Pineda, B., Pérez de la Cruz, V., Hernández Pando, R., & Sotelo, J. (2021). Quinacrine as a potential treatment for COVID-19 virus infection. *Eur Rev Med Pharmacol Sci*, *25*(1), 556-566. https://doi.org/10.26355/eurrev_202101_24428

Pineda, J., Midtvedt, B., Bachimanchi, H., Noé, S., Midtvedt, D., Volpe, G., & Manzo, C. (2023). Geometric deep learning reveals the spatiotemporal features of microscopic motion. *Nature Machine Intelligence*, *5*(1), 71-82. https://doi.org/10.1038/s42256-022-00595-0

Scurll, J. M. (2022). Measuring inter-cluster similarities with Alpha Shape TRIangulation in loCal Subspaces (ASTRICS) facilitates visualization and clustering of high-dimensional data [preprint]. *arXiv*.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379-423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., . . . SciPy, C. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat Methods*, *17*(3), 261-272. https://doi.org/10.1038/s41592-019-0686-2

Wallis, T. P., Jiang, A., Young, K., Hou, H., Kudo, K., McCann, A. J., Durisic, N., Joensuu, M., Oelz, D., Nguyen, H., Gormal, R. S., & Meunier, F. A. (2023). Super-resolved trajectory-derived nanoclustering analysis using spatiotemporal indexing. *Nature Communications*, *14*(1), 3353. https://doi.org/10.1038/s41467-023-38866-y

Williamson, D. J., Burn, G. L., Simoncelli, S., Griffié, J., Peters, R., Davis, D. M., & Owen, D. M. (2020). Machine learning for cluster analysis of localization microscopy data. *Nat Commun*, *11*(1), 1493. https://doi.org/10.1038/s41467-020-15293-x