dynsight: An open Python platform for simulation and experimental trajectory data analysis

RESEARCH ARTICLE | FEBRUARY 24 2026

dynsight: An open Python platform for simulation and experimental trajectory data analysis

Simone Martino ; Matteo Becchi ; Andrew Tarzia ; Daniele Rapetti ; Chiara Lionello ; Giovanni M. Pavan

J. Chem. Phys. 164, 084114 (2026)
https://doi.org/10.1063/5.0309974

The Journal of Chemical Physics ARTICLE pubs.aip.org/aip/jcp

dynsight: An open Python platform for simulation and experimental trajectory data analysis

Cite as: J. Chem. Phys. 164, 084114 (2026); doi: 10.1063/5.0309974
Submitted: 30 October 2025 • Accepted: 3 February 2026 •
Published Online: 24 February 2026

Simone Martino, Matteo Becchi,[a] Andrew Tarzia,[b] Daniele Rapetti,[c] Chiara Lionello, and Giovanni M. Pavan[a]

AFFILIATIONS

Department of Applied Science and Technology, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

[a]**Authors to whom correspondence should be addressed:** matteo.becchi@polito.it and giovanni.pavan@polito.it
[b]**Present address:** School of Chemistry, University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom.
[c]**Present address:** Scuola Internazionale Superiore di Studi Avanzati (SISSA), Trieste, Italy.

ABSTRACT

The study of complex many-body systems via analysis of the trajectories of the units that dynamically move and interact within them is a non-trivial task. The workflow for extracting meaningful information from the raw trajectory data is often composed of a series of interconnected steps, such as (i) identifying and tracking the constitutive objects/particles, resolving their trajectories (e.g., in experimental cases, where these are not automatically available as in typical molecular simulations); (ii) translating the trajectories into data that are easier to handle/analyze by using well-suited descriptors; and (iii) extracting meaningful information from such data. Each of these different tasks often requires non-negligible programming skills, the use of various types of representations or methods, and the availability/development of an interface between them. Despite the considerable potential that new tools contributed to each of these individual steps, their integration under a common framework would decrease the barrier to usage (especially by diverse communities of users), avoid fragmentation, and ultimately facilitate the development of new approaches in data analysis. To this end, here we introduce `dynsight`, an open Python platform that streamlines the extraction and analysis of time-series data from simulation or experimentally resolved trajectories. `dynsight` simplifies workflows, enhances accessibility, and facilitates time-series and trajectories data analysis, offering a useful tool for unraveling the dynamic complexity of a variety of systems (or signals) across different scales. `dynsight` is open source (github.com/GMPavanLab/dynsight) and can be easily installed using `pip`.

I. INTRODUCTION

The analysis of complex systems is often non-trivial. Typical examples include systems composed of many units (atoms, molecules, objects, particles, etc.) that interact dynamically with each other and evolve collectively over time toward emergent functions. These systems typically exhibit rich and complex behaviors that can be examined at different levels of detail and/or across various lengths and time scales to be understood. However, collecting all the needed data and analyzing them across multiple scales often requires considerable expertise, proficiency in the use of different methods, and, eventually, programming activities that may hinder understanding in diverse settings.

In the past decades, trajectory analysis in molecular simulations has greatly benefited from the development of software, such as `MDAnalysis`,[1,2] `Ambertools`,[3] `PLUMED`,[4,5] `VMD`,[6] `OVITO`,[7] and `Chemiscope`[8] (to name a few). These software applications have been developed *ad hoc* to handle and analyze trajectories obtained from molecular simulations and provide simple and efficient interfaces for facilitating the extraction of relevant structural and dynamical information from them by integrating (more or less) standard descriptors.

Together with widely used descriptors, such as number of neighbors, distances, etc., recently, new types of advanced descriptors have also been developed, which can provide information-rich characterizations of the systems under study. These can essentially be subdivided into two categories: static (time-independent) and dynamical (time-dependent) descriptors. For the former family, some relevant examples include, e.g., the Smooth Overlap of Atomic Positions (SOAP)[9] and the Atomic Cluster Expansions (ACE),[10] to

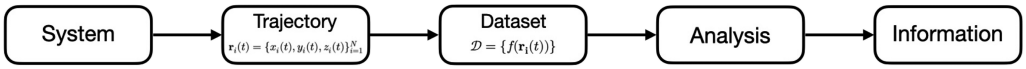

FIG. 1. The `dynsight` workflow: Typical `dynsight` pipeline, where trajectories, obtained from either simulations or experimental videos, are converted into datasets through the computation of descriptors and, subsequently, analyzed to extract physically relevant information.

name a few. Such advanced descriptors provide a high-dimensional embedding of the local structure, symmetry, and degree of order in the positions of neighbors around a central unit. The abstractness and generality of such descriptors offer the advantage of capturing many local structural details in the micro-domains that compose a system without heavy assumptions based, e.g., on prior knowledge of the system.

At the same time, pattern recognition approaches to detect dominant motifs in, e.g., SOAP datasets suffer in detecting rare events that have a negligible statistical weight in the dataset—these are only sparsely observed but are often key for the physics of the system. Particularly useful for detecting such important fluctuations are descriptors such as *Time*SOAP,[11] which tracks fluctuations of the SOAP spectra, capturing local structural changes, and the Local Environments and Neighbors Shuffling (LENS),[12] which allows the detection of local dynamical fluctuations (e.g., changes in fluidity or viscosity, permutations, neighbor exchange events, etc.). Coupled with well-suited clustering approaches that make it possible to systematically discriminate statistically relevant fluctuations from noise and to classify them based on their similarity and diversity (e.g., onion clustering[13,14]), these descriptors allow one to obtain deep insights into the local and collective dynamical events that may appear in a system[15] and to study their correlations in space and time, thereby elucidating the mechanisms behind physical phenomena in an exquisitely data-driven way.[16]

Thanks to their general and abstract character, such methods and approaches can, in principle, be applied to any kind of dynamically complex system. At the same time, all such advanced approaches use their own platform, codes, or software, which has, until now, made their cross-application for the analysis of trajectories (whether obtained from simulations or experiments) substantially fragmented. To streamline the use of these methods and extend them beyond molecular simulations, we have developed and here describe `dynsight`.

The typical workflow of a `dynsight` analysis is schematically illustrated in Fig. 1. In particular, it starts with the extraction of the trajectories of the system under study. While this first step is straightforward in typical computer simulations (where the individual trajectories of all units moving in the system are automatically recorded at every simulation time step), obtaining microscopic-level trajectories from experimental data often remains challenging. To this end, we have implemented algorithms based on computer vision, object detection, and particle tracking that automatically and efficiently detect and track the individual units that move over time in experimental videos, providing as output their trajectories. Once the trajectories are available, `dynsight` provides methods to compute a variety of advanced single-particle descriptors[11,12,14] and a flexible platform to eventually implement other desired ones. Useful tools to smooth signals and reduce noise in a physically meaningful way are also included,[17] as well as an efficient unsupervised clustering algorithm for single-point time-series analyses that can detect the different fluctuations and microscopic dynamical domains that compose a system or signal (i.e., onion clustering,[13] although many other clustering algorithms can be easily interfaced or integrated within `dynsight`).

By streamlining extraction, preprocessing, clustering, and analysis in a flexible and broadly applicable way, `dynsight` facilitates the understanding of datasets or signals, offering a robust and useful platform for advancing the study of complex systems and complex physical phenomena, as well as supporting the analysis of complex data and signals in general. The implementation of these methods in Python ensures seamless interaction and expandability with existing software, while the open-source nature of `dynsight` enables straightforward integration of additional descriptors and analysis methods.

II. SOFTWARE OVERVIEW

A. How `dynsight` unifies trajectory analysis

In its current version (v2026.2.16), `dynsight` is structured to support a wide range of tasks commonly encountered in the analysis of many-body dynamical systems. These tasks include handling trajectory data, computing single-particle descriptors, performing time-series clustering, and conducting various auxiliary analyses. To achieve this, `dynsight` is organized into specialized modules, each addressing a specific aspect of this workflow (documentation describing their use is available at dynsight.readthedocs.io/en/latest/workflow). In particular, the `dynsight.trajectory` module includes classes that streamline the entire analysis process into simple Python objects:

`Class Trj`: This object contains a many-body trajectory in the form of an `MDAnalysis.Universe`,[1] and offers, among others,

TABLE I. Selected list of `dynsight.trajectory` methods.

Method's type	Available methods
Trj → insight	Trj.get_coord_number() Trj.get_coordinates() Trj.get_lens() Trj.get_soap() Trj.get_timesoap() Trj.get_orientational_op() Trj.get_velocity_alignment()
Insight → insight	Insight.get_angular_velocity() Insight.spatial_average() Insight.get_tica()
Insight → clusterInsight	Insight.get_onion() Insight.get_onion_smooth()

methods for the calculation of several single-particle descriptors (see Table I).

`Class Insight`: This object contains a single-particle descriptor in the form of an `np.ndarray`[18] with shape `(n_atoms, n_frames, n_components)`. Methods include the computation of other descriptors, dimensionality-reduction techniques, and clustering algorithms (see Table I), together with fast load from/save to file methods.

`Class ClusterInsight`: This object contains a clustering analysis on a single-particle descriptor, in the form of an `np.ndarray`[18] of labels, with shape `(n_atoms, n_frames)`. Methods include functions for plotting the clustering results, together with fast load from/save to file methods.

B. Trajectory extraction from experimental videos: The `vision` and `track` modules

To uncover details about the motion and interactions of individual particles in a trajectory, we recognized the need to extract and track individual particles within trajectories, which may then be analyzed using descriptors. While this is trivial in computer simulations, finding and tracking particles with high accuracy in experimental trajectories is an open challenge.[19,20] To this end, we have implemented algorithms based on computer vision (`dynsight.vision`) and tracking (`dynsight.track`) that can automatically extract trajectories from experimental videos [see Fig. 2(b)]. The `dynsight.vision` module leverages convolutional neural network models from the widely used YOLO library[21] to detect objects in experimental trajectory videos (i.e., from a set of consecutive snapshots or frames) and save the position of each detected object. The `vision` module provides a standalone web application called `label_tool` with a user-friendly graphical user interface (GUI) that allows the user to manually label an initial sample of the object that the user wants to detect [Fig. 2(b), left]. This preliminary annotation is used to automatically generate a synthetic dataset that facilitates the first training phase of the detection model. Subsequently, an iterative process of detection and retraining using the experimental video itself further refines the model to recognize the specific objects [Fig. 2(b), center].

The `dynsight.track` module applies the tracking algorithm `trackpy`[22] to assign a unique ID to each particle detected by the `vision` module, enabling the reconstruction of individual trajectories across frames [Fig. 2(b), right]. In this current version of `dynsight` (v2026.2.16), the `dynsight.vision and dynsight.track` modules have implemented these methods for single object detection and tracking; nonetheless, given the flexibility and open character of the `dynsight` platform, other detection and tracking algorithms will eventually be implemented in the future.

C. Computing single-particle descriptors

`dynsight` facilitates the computation of several single-particle descriptors, which are functions of the particle trajectories $(x, y, z$ coordinates, velocities, etc.). The descriptors already available in the current release (v2026.2.16) of `dynsight` are listed below.

1. LENS: Local environments and neighbors shuffling

LENS[12] [see Fig. 3(d)] is a single-particle descriptor designed to analyze local dynamics in complex systems by tracking changes in the immediate surroundings of individual units over time, thereby capturing the degree of local dynamical rearrangement of the neighborhood of each unit. For details of the LENS descriptor and its applicability, we refer interested readers to the original paper.[12] Shortly, LENS is a permutationally variant, structurally invariant descriptor: it is particularly well-suited to capture local dynamical fluctuations (rather than, e.g., local structural transitions, for which other descriptors are better suited; see below) even in cases where these are local, collective, or rare events.[12] LENS has proven particularly useful to identify and classify local fluctuations and dynamic domains, such as distinguishing between solid-like and liquid-like

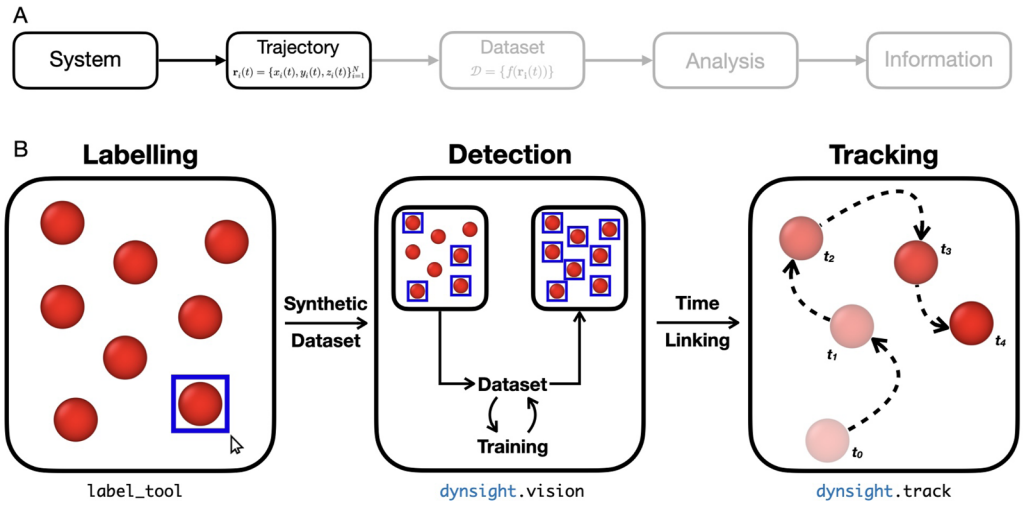

FIG. 2. The `vision` and `track` modules of `dynsight`. (a) A first operative phase aims to extract the trajectory data from experimental videos or simulations. (b) Schematic representation of the workflow to extract 2D trajectories from experimental videos. Procedure of labeling (left), training (center), and tracking (right) to obtain the trajectories of the moving objects from the input video in `dynsight`.

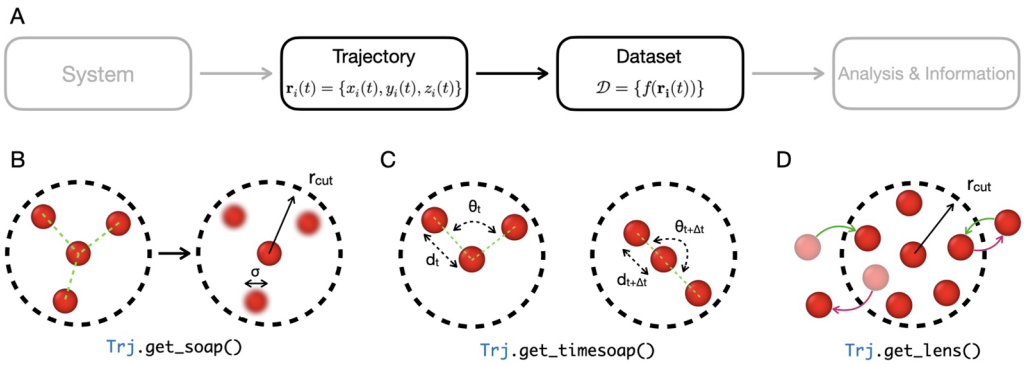

FIG. 3. Descriptors: Example descriptors in `dynsight` for analyzing trajectories to produce a dataset (a). (b) Smooth Overlap of Atomic Positions (SOAP),[9] a continuous, high-dimensional representation of the local particles' density around a particle. (c) *Time*SOAP,[11] a uni-dimensional measure of the temporal variation of the SOAP spectra. (d) Local Environments and Neighbors Shuffling (LENS),[12] a measure of the degree of neighbors' reshuffling around a particle.

regions in complex systems or identifying interfacial dynamics in phase transitions,[12] and in elucidating key mechanisms behind non-trivial dynamics.[16] The value of LENS between t and Δt is defined as

$$\delta_i(t) = \frac{\#(C_i^t \cup C_i^{t+\Delta t}) - \#(C_i^t \cap C_i^{t+\Delta t})}{\#(C_i^t) + \#(C_i^{t+\Delta t})},$$

and represents the set difference between the mathematical union and intersection of neighbor ID lists present within r_{cut} from particle i at the two consecutive time steps t and $t + \Delta t$, normalized by the total length of the neighbor ID lists. Here, C_i^t is the list of the IDs of the neighboring particles of particle i, and $\#(C_i^t)$ is its cardinality. In this way, LENS values range from 0 to 1, where a LENS = 0 between consecutive time steps means that the neighbors did not change in the time interval, while a value of 1 means that all neighbors changed in the sampled time interval (complete reshuffling).[12] With `dynsight`, LENS can be easily computed from a many-body trajectory using the `Trj.get_lens()` method [see Fig. 3(d)].

2. SOAP: Smooth overlap of atomic positions

The SOAP[9] [see Fig. 3(b)] descriptor is a structural descriptor that provides a high-dimensional representation of the local structure around a particle by encoding the relative spatial arrangement of neighboring particles into a smooth and continuous representation. In this sense, the SOAP power spectrum serves as a descriptor of the degree of local order or disorder in the relative displacements of the weights around a center (symmetry, distances, etc.). For details concerning the SOAP descriptor, we refer interested readers to the original paper[9] or to dedicated works focusing on the analysis of the SOAP dataset[23–26] or time-series data.[14,27] In brief, the SOAP descriptor provides a rich representation of the atomic neighbor density around a particle,

$$\rho(\mathbf{r}) = \sum_{i=1}^{n_{cut}} \delta(\mathbf{r} - \mathbf{r}_i),$$

where i runs over the n_{cut} particles closer than a cutoff radius r_{cut} from the central particle, and $\mathbf{r}_i$ is the position of the i−th particle. $\rho(\mathbf{r})$ can be approximated in terms of radial basis functions $g_n(r)$ and spherical harmonics $Y_{lm}(\theta, \phi)$ as

$$\rho(\mathbf{r}) \approx \sum_{n=0}^{n_{\max}-1} \sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} c_{nlm} \cdot g_n(r) \cdot Y_{lm}(\theta, \phi),$$

where c_{nlm} are the expansion coefficients. From this expansion, it is possible to compute a rotational-invariant power spectrum $\mathbf{p}$, whose components are

$$p_{nn'l} = \pi \sqrt{\frac{8}{2l+1}} \sum_m (c_{nlm})^\dagger c_{n'lm},$$

which describes the structural information of the local particle's environment. `dynsight` uses the `DScribe` package[28] to perform SOAP calculations, making it easier to streamline these calculations within a single Python workflow via the `Trj.get_soap()` method.

3. TimeSOAP: Tracking time-variations in SOAP spectra

*Time*SOAP[11] [*t*SOAP, see Fig. 3(c)] is a time-dependent descriptor that, starting from the structural description of local environments provided by SOAP, detects and tracks high-dimensional fluctuations over time in the SOAP spectra. It captures local structural changes or transitions in the neighborhood of every unit. In this way, *t*SOAP enables the identification of critical events or a microscopic-level characterization of complex phenomena such as phase coexistence and structural transformations[11] that may be lost in standard pattern recognition analyses of SOAP datasets or when using simpler metrics. *t*SOAP is a structurally variant, permutationally invariant descriptor: in this sense, it is complementary to LENS and well-suited to capture local structural fluctuations.[16] For more details on the *t*SOAP descriptor, we refer interested readers to the dedicated papers.[11,16] For a given particle with index i, *Time*SOAP quantifies the distance between the SOAP spectra of particle i at two consecutive frames, t and $t + \Delta t$ (i.e., capturing the variation of the SOAP spectrum in Δt). The distance between SOAP spectra is defined as

$$d(\vec{a},\vec{b}) = \sqrt{2 - 2\,\frac{\vec{a}\cdot\vec{b}}{||\vec{a}|| \cdot ||\vec{b}||}},$$

where $\vec{a}$ and $\vec{b}$ represent the SOAP spectra of particle i at the consecutive time frames t and $t + \Delta t$, respectively. With `dynsight`, *Time*SOAP can be computed from a SOAP spectrum time-series using the `Insight.get_angular_velocity()` method or the `dynsight.soap.timesoap()` function [see Fig. 3(c)].

4. Miscellaneous descriptors

The `dynsight.Trj` class offers methods to compute, from a many-body trajectory, the number of neighbor particles, velocity, local velocity alignment,[29] and the orientational order parameter.[30] Moreover, the Time-lagged Independent Component Analysis (tICA)[31] dimensionality reduction algorithm is also implemented using `deeptime`[32] to simplify the treatment of high-dimensional descriptors. It is worth noting that, while such descriptors were originally developed for atomistic or molecular systems, their abstract character makes them well-suited to study systems at virtually any scale.

In the `dynsight` framework, adding new descriptors or `Insights` is simplified for users. The computation of any scalar or vector single-particle quantity can be easily implemented within `dynsight`, making it a viable platform for method development.

D. Unsupervised clustering for single-point time-series analysis: For example, *onion clustering*

The current version of `dynsight` (v2026.2.16) implements *onion clustering*[13] as an efficient unsupervised clustering method for single-point time-series analysis. Notably, the flexibility and open character of the `dynsight` platform make it easy to also implement or interface with other clustering methods that can then be used to analyze the trajectories and datasets obtained when using the other modules.

For a detailed description of the *onion clustering* method, we refer interested readers to the original paper[13] and to other related applied works that clearly explain the potential and advantages of this method.[14,15,27,33] Shortly, *onion clustering* is an efficient algorithm for single-point clustering of time-series data that allows to identify and classify all fluctuations and local dynamical domains (including sparse or hidden ones, which may be obscured by the noise of the dominant ones) that can be characterized in a statistically robust manner as a function of the temporal resolution Δt used in the analysis. More in detail, *onion clustering* performs a series of clustering analyses, each with a different time resolution Δt (i.e., different minimum lifetime required for the identified states), and outputs the microscopic dynamical domains that can be effectively distinguished as a function of Δt [the typical output is the so-called *onion plot*: Fig. 4(c)]. The clustering exploits an iterative procedure where, at the end of the process, each data point is either classified in one of the identified states or labeled as "unclassified"

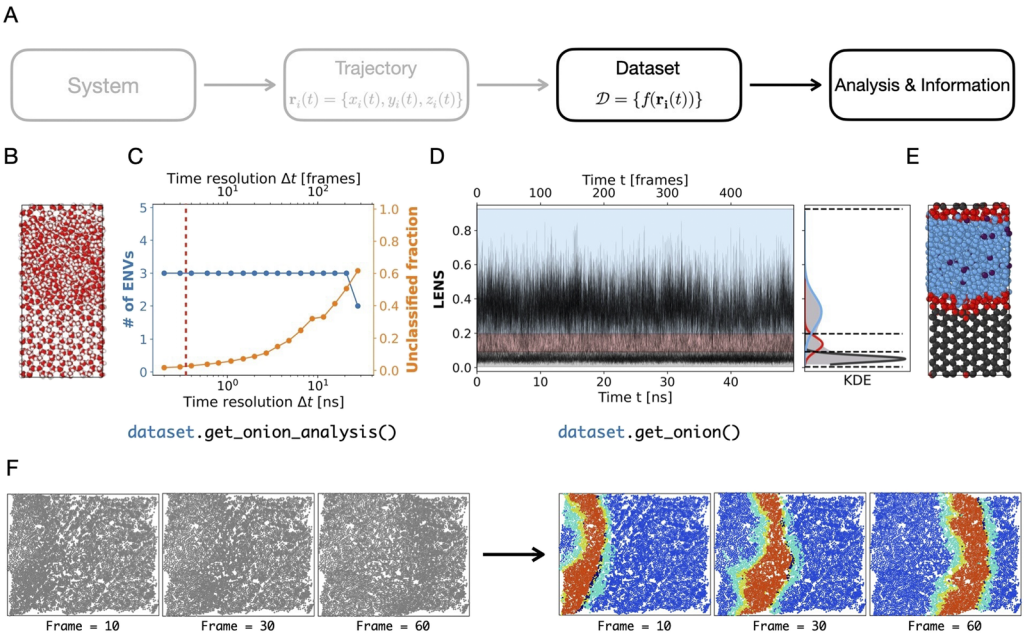

FIG. 4. Time-series clustering: Example of `dynsight` usage for information extraction through the *onion clustering* algorithm from descriptor based datasets (a). (b) Prototypical example of a water–ice molecular dynamics simulation. (c) Number of identified environments (blue) and unclassified data (orange) as a function of Δt used by *onion clustering*. (d) Time-series of the LENS descriptor classified by *onion clustering*, with corresponding kernel density estimation (KDE) on the right. (e) Snapshot colored according to the detected clusters. (c)–(e) Adapted from Ref. 14, licensed under a Creative Commons Attribution (CC BY) license. (f) Representative frames of a trajectory resolved from an experimental video of Quincke rollers evolving into a wave pulse,[34] analyzed using the velocity alignment descriptor and *onion clustering* to detect, track, and quantify the wave motion.[13]

at the selected time resolution [Fig. 4(c): orange—the fluctuations occurring faster than the time resolution used]. At each step, the global maximum of the cumulative distribution of the data points is identified and fitted as a Gaussian state (i.e., a microscopic state) characterized by its own mean value and the variance of the signal (own characteristic noise). The time-series is partitioned into consecutive windows of duration Δt, and individual units that continuously stay within that Gaussian for the entire Δt are assigned to that first main system's micro-environment (ENV1). The data classified in ENV1 are then removed from the original time-series, which are, as a consequence, purged from their noise. A second maximum is then fitted in the new "epured" data distribution, and the approach is iterated, detecting at every step new ENVs composed of units that reside in them continuously for Δt. At each successive iteration, the method discovers new dynamically different and statistically relevant environments, including those hidden by the noise of the dominant ones (particularly useful, e.g., for non-uniform and heterogeneous systems and for dynamical systems dominated by rare events).[13] Performing this analysis at different values of the time resolution Δt allows automatic identification of the optimal choice of Δt, the one that maximizes the statistically robust subdivision of the time-series data in the largest number of microscopic environments [Figs. 4(c) and 4(d)]; i.e., this is the time resolution that guarantees maximum information extraction from the data.[15,33] Onion clustering can be performed on virtually any time-series using the `Insight.get_onion()` method and its variations. Note that while this method offers considerable advantages in statistical robustness and interpretability,[13,33] many other clustering methods can be implemented or interfaced with other modules of `dynsight`, thanks to the open character of this platform. The standalone source code and its corresponding documentation are available at github.com/GMPavanLab/onion_clustering (for a complete discussion and description of the *onion clustering* method, we refer the readers to the dedicated paper[13]).

E. Miscellaneous modules useful for data analysis: The `analysis` module

The `dynsight.analysis` module provides other key tools for pre- and post-processing and statistical analysis of many-body trajectories, including:

- Denoising tools: the present version (v2026.2.16) of `dynsight` implements a spatial denoising approach,[17] which can be applied to time-series computed from trajectories in order to enhance their signal-to-noise ratio. Noise filtering tools are currently being optimized in our group, which will also be available in `dynsight`.
- The computation of the radial distribution function $g(r)$ from many-body trajectories, which quantifies how particle density varies as a function of distance from a reference particle.
- The computation of several information-theory quantities, such as Shannon entropy,[35] sample entropy,[36,37] and information gain achieved by clustering algorithms,[33] which allow a quantitative evaluation of the information content and extraction during the analyses.

While these additional tools are included in the present version of `dynsight` (v2026.2.16), the open character of this platform makes it straightforward to implement, interface with, or patch other utilities that may be needed.

F. Fostering and facilitating open-data archiving

By design, `dynsight` is a platform that promotes transparent data management and open data practices. We have implemented an integrated logging module that records a human-readable log of all performed analyses. The `dynsight.logger` can also automatically generate an archive containing all the relevant data and metadata. This standardized archive is thus immediately ready for deposition in common open data repositories (e.g., *Zenodo*), which comply with the FAIR principles (Findable, Accessible, Interoperable, Reusable). This feature is useful not only for tracking the analysis workflow but also for facilitating the reproducibility of results, ultimately contributing to the advancement of research within the community.

III. HOW TO USE `dynsight`

`dynsight` is available on the PyPI repository at pypi.org/project/dynsight and can be installed with `pip install dynsight`. The package is installed in the `site-packages` directory of the active Python environment and can be accessed by importing it within any Python script or interactive session by typing `import dynsight`. The code is open-source under the MIT license, and the documentation is available at dynsight.readthedocs.io/en/latest.

A. The `label_tool` GUI of the `vision` module

Figure 5 shows the integration of the `label_tool` application with `dynsight.vision` and `dynsight.track` for particle trajectory extraction from experimental videos. Here, we used a preprocessed Quincke rollers experimental video[34] [see Fig. 5(a)], where color correction was applied to enhance particle visibility while suppressing background noise. The `label_tool` graphical user interface (GUI) enables users to efficiently create initial training data by selecting representative examples of target particles directly from single video frames. These user-provided annotations are then automatically expanded into a synthetic dataset by randomly redistributing the labeled particles over multiple artificial images, effectively increasing the diversity and robustness of the training dataset. In the following, we present a deliberately simple demonstrative case of a system composed of multiple particles of the same type (i.e., a single-type-object many-body system). Nonetheless, it is worth noting that the `label_tool` allows the definition and annotation of multiple particle classes, enabling model training for the simultaneous detection of diverse objects with different sizes, shapes, or colors in single frames, following the same iterative machinery described herein for this simplified illustrative case. As shown in Fig. 2(b), this synthetic dataset serves as the initial input for `dynsight.vision`, which performs iterative learning and detection to identify all bounding boxes corresponding to particles across the entire frame sequence. The detected positions are subsequently processed by `dynsight.track`, which associates detections over consecutive frames to reconstruct the full trajectories of individual particles [see Fig. 5(c)]. This workflow enables the semi-automated generation of accurate and reproducible tracking data, providing a robust starting point for the analysis methods presented above.

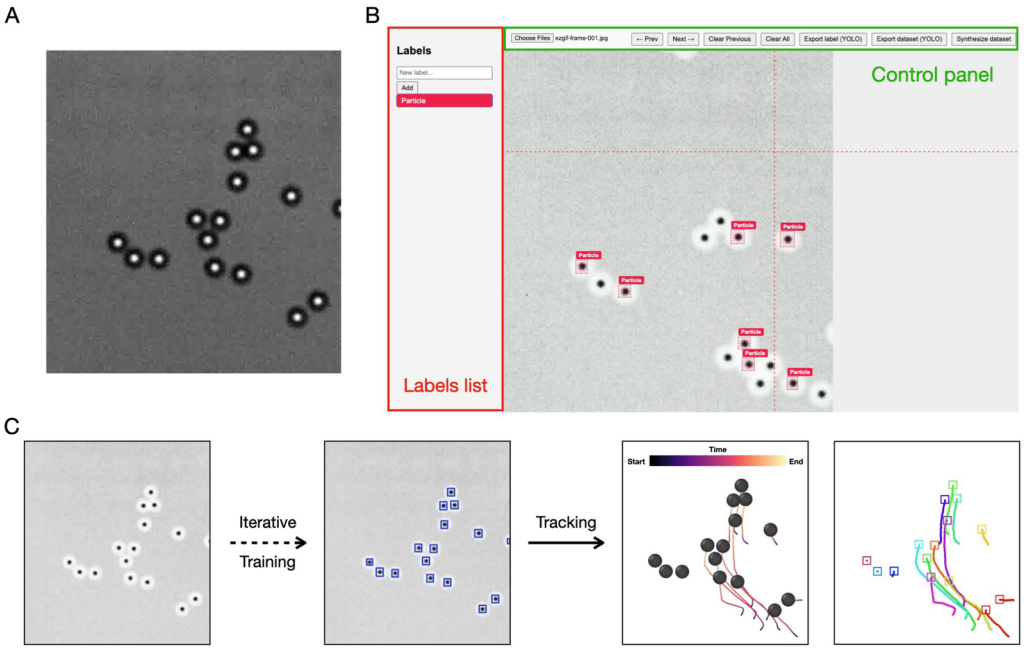

FIG. 5. Workflow of `Label_tool` and `vision/track` modules. (a) Single frame taken from a representative experimental video of an example complex colloidal system.[34] For visual purposes, a relatively simple many-body system is used here as a representative example, while it is worth noting that such approaches become particularly useful when the complexity of the system increases.[13,15,16,27] Frame extracted from Movie S11B in Ref. 34, with the permission of the Proceedings of the National Academy of Sciences (PNAS). (b) The `Label_tool` GUI allows the user to manually select prototypical examples of target particles from a color-corrected frame. (c) Using these annotations, `dynsight.vision` performs iterative training and detection to locate all particles across frames, while `dynsight.track` associates each detection with its corresponding trajectory.

B. An example of a typical `dynsight` analysis workflow

The first step is usually to create a `trajectory.Trj` object from some trajectory files. In this example, we are using a water/ice coexistence trajectory downloadable in the tutorial page of `dynsight` (dynsight.readthedocs.io/en/latest/tutorials_menu) [see Fig. 4(b)].

Listing 1. Loading trajectory files in `dynsight`.

```
from pathlib import Path
from dynsight.trajectory import Trj

files_path = Path("dynsight/examples/
    analysis_workflow")
trj = Trj.init_from_xtc(
    traj_file=files_path / "oxygens.xtc",
    topo_file=files_path / "oxygens.gro",
)
```

The variable `trj` now contains the trajectory (as an `MDAnalysis.Universe`, also allowing analysis using that software), and using the methods of the `trajectory.Trj` class, we can perform all the desired analyses on this trajectory. For instance, we can compute LENS [see Fig. 4(d)]:

Listing 2. Compute the LENS descriptor with a cutoff radius r_{cut} = 7.5~Å.

```
lens = trj.get_lens(r_cut=7.5)
```

Note that the units used for the calculation of the various descriptors depend on the units used in the input trajectory. We refer readers interested in data-driven methods to assess optimal space and time analysis resolutions to dedicated papers on this topic.[15,33]

The method `Trj.get_lens()` returns a `trajectory.Insight` object, which in its `.dataset` attribute contains the LENS values computed on the `trj` trajectory. The `trajectory.Insight` class offers its own methods for further analysis. For instance, one can perform spatial averaging[14,17] of the LENS values to reduce its internal noise:

Listing 3. Compute spatial average as an example of possible data processing.

```
trj_lens = trj.with_slice(slice(0, -1, 1))
lens_smooth = lens.spatial_average(
    trj=trj_lens,
    r_cut=7.5,
    num_processes=6,
)
```

Note that, since LENS is computed for intervals between frames, we need to use a sliced trajectory, which we get with the `Trj.with_slice()` method. Finally, we can perform clustering on the `lens_smooth.dataset`, using, for instance, the `Insight.get_onion_smooth()` method [see Figs. 4(c)–4(f)], and plot the results:

Listing 4. Onion Clustering computing and plotting.

```
lens_onion = lens_smooth.get_onion_smooth(delta_t
    =10)

lens_onion.plot_output(
    file_path=files_path / "tmp_fig1.png",
    data_insight=lens_smooth,
)

lens_onion.dump_colored_trj(
    trj=trj_lens,
    file_path=files_path / "colored_trj.xyz",
)
```

C. Examples and tutorials

Example scripts has been designed to help users to smoothly integrate the `dynsight` tools into their own code. These are provided as open-access in the dedicated example folder of the `dynsight` GitHub page available at github.com/GMPavanLab/dynsight/tree/main/examples.[15,33] In addition, a dedicated, detailed, and user-friendly tutorials webpage is organized to be followed step by step, with all related materials readily available for download. The explanation of the typical analysis workflow presented in this paper is already available in tutorial form and can be consulted together with the other tutorials at the following webpage: dynsight.readthedocs.io/en/latest/tutorials_menu.html. This page will be continuously updated with the development of new content.

D. How to contribute

The authors welcome any contribution to improve and expand the `dynsight` code and its modules. Any user can submit comments, requests, or additions by opening an issue or submitting a pull request in the `dynsight` GitHub repository github.com/GMPavanLab/dynsight.

IV. CONCLUSIONS

We have presented `dynsight`, an open Python platform for resolving, processing, and analyzing trajectory data from both experimental and simulated systems. By combining trajectory extraction, descriptor computation, and unsupervised clustering methods in a single platform, `dynsight` streamlines workflows that would otherwise require substantial expertise and familiarity with diverse tools. The flexible design of the platform allows application across diverse systems and scales, enabling detailed microscopic-level analyses as well as broader studies of local and collective dynamics and expected behaviors. Overall, `dynsight` lowers the barrier to trajectory-based analysis, promoting reproducibility, consistency, and wider adoption of advanced dynamic descriptors. The possibility of using a single common platform, as well as the same methods and analysis tools, to study systems from atomic simulations to macroscopic experimental setups also opens unique opportunities to study complex behaviors and emergent properties through different scales in virtually any complex system for which a trajectory of the constitutive units can be resolved. We believe that this will constitute a flexible and solid foundation for advancing the study of complex systems as well as the analysis of static and dynamic data (or signals) in general.

ACKNOWLEDGMENTS

G.M.P. acknowledges the funding received by the European Research Council under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 818776-DYNAPOL). A.T. also acknowledges the support received by the European Union under the Next Generation EU program (Mission 4 Component 1 CUP E13C22002930006). The authors acknowledge (in alphabetical order) Lucrezia Baldo, Cristina Caruso, Matteo Cioni, Martina Crippa, Massimo Delle Piane, and Domiziano Doria for the useful testing, review, and feedback.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Simone Martino: Data curation (equal); Formal analysis (equal); Investigation (equal); Methodology (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Matteo Becchi**: Data curation (equal); Formal analysis (equal); Methodology (equal); Software (equal); Supervision (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Andrew Tarzia**: Funding acquisition (supporting); Software (equal); Supervision (supporting); Writing – review & editing (equal). **Daniele Rapetti**: Software (supporting); Writing – review & editing (supporting). **Chiara Lionello**: Writing – review & editing (equal). **Giovanni M. Pavan**: Conceptualization (lead); Funding acquisition (lead); Methodology (lead); Project administration (lead); Resources (lead); Supervision (lead); Validation (equal); Writing – original draft (equal); Writing – review & editing (lead).

DATA AVAILABILITY

A copy of the current version (v2026.2.16) of the code is available as Zenodo repository at https://doi.org/10.5281/zenodo.18668433.

REFERENCES

[1]N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, "MDAnalysis: A toolkit for the analysis of molecular dynamics simulations," J. Comput. Chem. **32**(10), 2319–2327 (2011).

[2]R. J. Gowers *et al.*, "MDAnalysis: A Python package for the rapid analysis of molecular dynamics simulations," in *Proceedings of the 15th Python in Science Conference*, edited by S. Benthall and S. Rostrup (SciPy, 2016), pp. 98–105.

[3]D. A. Case *et al.*, "AmberTools," J. Chem. Inf. Model. **63**(20), 6183–6191 (2023).

[4]M. Bonomi *et al.*, "PLUMED: A portable plugin for free-energy calculations with molecular dynamics," Comput. Phys. Commun. **180**(10), 1961–1972 (2009).

[5]G. A. Tribello, M. Bonomi, D. Branduardi, C. Camilloni, and G. Bussi, "PLUMED 2: New feathers for an old bird," Comput. Phys. Commun. **185**(2), 604–613 (2014).
[6]W. Humphrey, A. Dalke, and K. Schulten, "VMD: Visual molecular dynamics," J. Mol. Graphics **14**(1), 33–38 (1996).
[7]S. Alexander, "Visualization and analysis of atomistic simulation data with OVITO–the open visualization tool," Modell. Simul. Mater. Sci. Eng. **18**(1), 015012 (2009).
[8]G. Fraux, R. Cersonsky, and M. Ceriotti, "Chemiscope: Interactive structure-property explorer for materials and molecules," J. Open Source Softw. **5**(51), 2117 (2020).
[9]A. P. Bartók, R. Kondor, and G. Csányi, "On representing chemical environments," Phys. Rev. B **87**(18), 184115 (2013).
[10]D. Ralf, "Atomic cluster expansion for accurate and transferable interatomic potentials," Phys. Rev. B **99**(1), 014104 (2019).
[11]C. Caruso, A. Cardellini, M. Crippa, D. Rapetti, and G. M. Pavan, "*Time*SOAP: Tracking high-dimensional fluctuations in complex molecular systems via time variations of SOAP spectra," J. Chem. Phys. **158**, 214302 (2023).
[12]M. Crippa, A. Cardellini, C. Caruso, and G. M. Pavan, "Detecting dynamic domains and local fluctuations in complex molecular systems via timelapse neighbors shuffling," Proc. Natl. Acad. Sci. U. S. A. **120**(30), e2300565120 (2023).
[13]M. Becchi, F. Fantolino, and G. M. Pavan, "Layer-by-layer unsupervised clustering of statistically relevant fluctuations in noisy time-series data of complex dynamical systems," Proc. Natl. Acad. Sci. U. S. A. **121**(33), e2403771121 (2024).
[14]S. Martino, D. Doria, C. Lionello, M. Becchi, and G. M. Pavan, "A data driven approach to classify descriptors based on their efficiency in translating noisy trajectories into physically-relevant information," Mach. Learn.: Sci. Technol. **6**(3), 035039 (2025).
[15]D. Doria, S. Martino, M. Becchi, and G. M. Pavan, "Data-driven assessment of optimal spatiotemporal resolutions for information extraction in noisy time series data," J. Chem. Phys. **162**, 234110 (2025).
[16]C. Caruso *et al.*, "Classification and spatiotemporal correlation of dominant fluctuations in complex dynamical systems," PNAS Nexus **4**(2), pgaf038 (2025).
[17]E. D. Donkor, A. Offei-Danso, A. Rodriguez, F. Sciortino, and H. Ali, "Beyond local structures in critical supercooled water through unsupervised learning," J. Phys. Chem. Lett. **15**(15), 3996–4005 (2024).
[18]C. R. Harris *et al.*, "Array programming with NumPy," Nature **585**, 357–362 (2020).
[19]C. Zhang, G. Brügger, and F. Scheffold, "Tracking of colloids close to contact," Opt. Express **23**(17), 22579–22586 (2015).
[20]L. Yao, Z. Ou, B. Luo, C. Xu, and Q. Chen, "Machine learning to reveal nanoparticle dynamics from liquid-phase TEM videos," ACS Cent. Sci. **6**(8), 1421–1430 (2020).
[21]G. Jocher and J. Qiu, Ultralytics YOLO11, Version 11.0.0, https://github.com/ultralytics/ultralytics, 2024.
[22]D. B. Allan, T. Caswell, N. C. Keim, C. M. van der Wel, and R. W. Verweij (2024). "soft-matter/trackpy: v0.6.4. Version v0.6.4," Zenodo. https://doi.org/10.5281/zenodo.12708864, URL: github.com/soft-matter/trackpy.
[23]P. Gasparotto, D. Bochicchio, M. Ceriotti, and G. M. Pavan, "Identifying and tracking defects in dynamic supramolecular polymers," J. Phys. Chem. B **124**(3), 589–599 (2020).
[24]R. Capelli, F. Muniz-Miranda, and G. M. Pavan, "Ephemeral ice-like local environments in classical rigid models of liquid water," J. Chem. Phys. **156**, 214503 (2022).
[25]V. L. Deringer *et al.*, "Computational surface chemistry of tetrahedral amorphous carbon by combining machine learning and density functional theory," Chem. Mater. **30**(21), 7438–7445 (2018).
[26]B. Monserrat, J. G. Brandenburg, E. A. Engel, and B. Cheng, "Liquid water contains the building blocks of diverse ice phases," Nat. Commun. **11**(1), 5757 (2020).
[27]C. Lionello, M. Becchi, S. Martino, and G. M. Pavan, "Relevant, hidden, and frustrated information in high-dimensional analyses of complex dynamical systems with internal noise," J. Chem. Theor. Comput. **21**(14), 6683–6697 (2025).
[28]L. Himanen *et al.*, "DScribe: Library of descriptors for machine learning in materials science," Comput. Phys. Commun. **247**, 106949 (2020).
[29]T. Vicsek and A. Zafeiris, "Collective motion," Phys. Rep. **517**(3–4), 71–140 (2012).
[30]G. Aeppli and R. Bruinsma, "Hexatic order and liquid density fluctuations," Phys. Rev. Lett. **53**(22), 2133 (1984).
[31]L. Molgedey and H. G. Schuster, "Separation of a mixture of independent signals using time delayed correlations," Phys. Rev. Lett. **72**(23), 3634 (1994).
[32]M. Hoffmann *et al.*, "Deeptime: A python library for machine learning dynamical models from time series data," Mach. Learn.: Sci. Technol. **3**(1), 015009 (2021).
[33]M. Becchi and G. M. Pavan, "Maximum information extraction via clustering and minimization of Shannon entropy," Mach. Learn.: Sci. Technol. **6**(4), 045074 (2025).
[34]Z. T. Liu *et al.*, "Activity waves and freestanding vortices in populations of subcritical Quincke rollers," Proc. Natl. Acad. Sci. U. S. A. **118**(40), e2104724118 (2021).
[35]C. E. Shannon, "A mathematical theory of communication," Bell Syst. Tech. J. **27**(3), 379–423 (1948).
[36]J. S. Richman and J. R. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," Am. J. Physiol. - Heart Circ. Physiol. **278**(6), H2039–H2049 (2000).
[37]J. S. Richman, D. E. Lake, and J. R. Moorman, "Sample entropy," Methods Enzymol. **384**, 172–184 (2004).