The Journal of Open Source Software

GEFF: Graph Exchange File Format

Morgan Schwartz [1*], **Caroline Malin-Mayor** [1*], **Talley Lambert** [2], **Teun Huijben** [3], **Jan Funke** [1], **Laura Xénard** [4,13], **Mark Kittisopikul** [1], **Draga Doncila Pop** [5], **Yohsuke Fukai** [6], **Jordão Bragantini** [3], **Melisande Croft** [7], **Anniek Stokkermans** [8], **Georgeos Hardo** [9,12], **Benjamin Gallusser** [3], **Kasia Kedziora** [10], **Jean-Yves Tinevez** [4], **Ko Sugawara** [11], **Tobias Pietzsch** [14], **and Ilan Theodoro** [3]

1 Janelia Research Campus, Howard Hughes Medical Institute, Ashburn, VA, USA **2** Harvard Medical School, Boston, MA, USA **3** Biohub, San Francisco, CA, USA **4** Institut Pasteur, Université Paris Cité, Image Analysis Hub, Paris, France **5** Monash University, Melbourne, Australia **6** RIKEN Pioneering Research Institute, Kobe, Hyōgo, Japan **7** Human Technopole, Milan, Italy **8** Hubrecht Institute, Utrecht, Netherlands **9** United Arab Emirates University, Al Ain, UAE **10** University of Pittsburgh, Pittsburg, PA, USA **11** RIKEN Center for Biosystems Dynamics Research, Kobe, Japan **12** University of Cambridge, Cambridge, UK **13** Institut Pasteur, Université Paris Cité, INSERM U1225, Paris, France **14** Independent Consultant, Germany ***** These authors contributed equally.

DOI: 10.21105/joss.10143

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @Puneetha-Ramachandra
- @Pablo1990

Submitted: 20 November 2025
Published: 13 April 2026

Summary

GEFF (Graph Exchange File Format) is a file format specification for exchanging graph data. Its main application today is focused on exchange of animal, cell, and organelle tracking data in the life sciences. It is not intended to be mutable, editable, chunked, or optimized for use in an application setting. As an exchange format with a strict specification, GEFF enables interoperability between tools written in various programming languages.

The geff repository contains two Python packages: `geff-spec`, the specification of GEFF metadata written with `pydantic.BaseModels`, which are exported to a JSON schema for use in other languages, and `geff`, the Python library that reads and writes GEFF files to and from several Python in-memory graph data structures (`networkx` (Hagberg et al., 2008), `rustworkx` (Treinish et al., 2022), and `spatial-graph`). A Java implementation of the GEFF v1 spec, `geff-java` is in progress in a separate repository.

Statement of Need

Cell and organelle tracking is an active area of research with many tools for performing tracking and visualizing results. At the 2023 Janelia Trackathon, a two-week workshop gathering cell tracking researchers, there was widespread agreement that a common graph file format would benefit the field by reducing code duplication and increasing standardization between projects. However, attempts made at that time were intended to be mutable and optimized, which introduced barriers to code generation and adoption by tools with different types of optimization needs. The 2025 Janelia Trackathon brought together all the authors of GEFF to decide on the specification and initial implementation, which was accomplished in a week-long hackathon.

Research Impact Statement

GEFF allows different research tools to all track and visualize the same data, reducing barriers to pipelining analysis and visualization tools, even across languages. As of submission time, the following tools all support either saving and/or loading GEFF files: `motile-tracker`, `traccuracy`, `ultrack`, `track_gardener`, `laptrack`, `trackastra`, `TrackMate`, `InTRACKtive`, `tracksdata`, and `napari-geff`. The developers have already fielded inquiries through GitHub and email from researchers, both developers and end users, about how best to use GEFF to accelerate their research workflow. We hope that GEFF will become the standard for storing and exchanging tracking information in the bio-image analysis community, and potentially even other fields that require exchanging graph-based information.

State of the Field

There are many formats used to store and exchange tracking solutions. A commonly used one is the Cell Tracking Challenge (CTC) (Maška et al., 2014) format, which combines TIFF files with segmentation masks and a CSV file to provide division edges. However, some tracking applications such as particle tracking do not operate on segmentations, but instead utilize point detections, making this format not applicable. As such, individual tracking tools often define their own format for saving tracking results; for example, TrackMate (Tinevez et al., 2017) has a specific XML file format, Mastodon (Pietzsch et al., 2025) saves and loads from a binary file, the Motile Tracker exports and loads to CSV files with specific node ID, parent ID, and location columns, and Ultrack (Bragantini et al., 2025) has a custom SQL database. In these existing file formats, there is limited support for storing additional properties on either nodes or edges. Additionally, none of these tools shared a common file format, which prevented interactions between them and strongly limited the scope and ambition of track analysis pipelines. In contrast to prior efforts to create a common file format (Gonzalez-Beltran et al., 2020) (Kyoda, 2020), we worked with tool authors to integrate support for GEFF directly into the tools themselves rather than only creating a third-party library. Each of them can now export to and import from GEFF in addition to their custom formats, enabling interoperability with minimal code change in each library.

Implementation

GEFF is built on `zarr` (Zarr Developers, 2019), a common file format used in bioimage analysis. Graphs are represented as an array of node IDs and an array of edge IDs where each edge ID is a tuple of two node IDs. Nodes and edges can have properties, which are stored in a properties array with corresponding indices. The specification includes support for nodes and edges with missing properties, as well as variable-length properties. To support the cell tracking community, the GEFF specification also provides specific metadata with standardized meaning, including positional axes, tracklet and lineage IDs, and linking to related objects such as image and segmentation arrays.

GEFF's object specification supports a wide range of shapes, enabling its application across diverse fields in the life sciences. The library integrates multiple representation formats, from binary masks (2D/3D), commonly used in cell biology, developmental biology, and natural image tracking, to geometric primitives (points, circles, ellipses, spheres, and ellipsoids), which are essential in super-resolution microscopy, virology, and developmental studies. It also includes polygons and meshes for detailed structural analysis in cell biology, microbiology, and complex shape modeling, as well as pose-based representations for markerless tracking of anatomical keypoints in multi-animal and behavioral research.

Each object, track, or lineage is represented by a simple directed graph, where each edge is a link from one biological object detected in a frame to its detection in the earliest next frame.

GEFF can therefore be used to harness tracking data with gaps (an edge extends over more than two adjacent time points, because of a missing detection or object exit and reentry), object divisions and objects fusions. Multiple GEFFs can refer to the same set of segmentations or detections, which makes it possible to easily store multiple tracking solutions or hypotheses without duplicating the underlying data, which is a limitation of the commonly-used CTC format.

By integrating these diverse object and link representations, GEFF facilitates seamless data exchange between segmentation algorithms, morphometric analysis tools, and tracking pipelines, ensuring compatibility with both established and emerging imaging workflows. This versatility makes the library a valuable resource for researchers working across disciplines, from high-throughput cell biology to fine-grained anatomical studies in developmental and computational biology.

Software Design

The GEFF specification emphasizes simplicity over optimization. It would be infeasible to make a single graph format, or even tracking format, that was optimized for all use cases. Implementing a simple exchange format allows a variety of tools with different goals to exchange information, while retaining their optimized internal formats. Additionally, focusing on an exchange format allows cross-language compatibility. In addition to a hand-written specification, the GEFF specification package provides Pydantic classes and a JSON schema for creating and validating the metadata properties.

The Python reference implementation has a highly modular design. The `core_io` module is the heart of the implementation, with core read and write implementations that can be used by any graph library, converting from an `InMemoryGeff` data structure to an on-disk `zarr` and back. This crucial abstraction improves maintainability as well as extensibility; new graph library implementations only need to convert to and from the `InMemoryGeff` data structure, and any improvements to performance or bugfixes in the `core_io` module are automatically propagated to the individual graph library implementations. GEFF supports Zarr specification v2 and v3, and has minimal dependencies, making it a lightweight dependency for other libraries.

The `_graph_libs` currently implements three graph backends. `networkx` and `rustworkx` are two of the most common Python graph libraries, making GEFF adoption simple for most Python programmers. `spatial_graph` is a newer graph library with improved efficiency when searching for all graph elements in a spatial region. While the Python implementation contains significant internal typing and abstraction logic that reduces code duplication and enhances maintainability, the external API is quite simple; most users will only need to use the public `read` and `write` functions.

The other three modules provide additional functionality for new users and developers. The `convert` module contains a CLI tool for converting existing on-disk formats to GEFF, including TrackMate XML, the Cell Tracking Challenge format, and any CSV-like format that can be loaded into a Pandas dataframe. The `testing` module provides `InMemoryGeff` objects with a variety of valid GEFF data combinations for testing both existing and new implementations. Finally, the `validate` module provides helper functions for testing the validity of on-disk GEFFs with varying levels of inspection, from fast, structure-only validation to intensive data validation.

Extensibility

While GEFF was developed by the cell tracking research community, it is a generic graph exchange format that could be easily extended to other use cases with additional metadata to specify the meaning of standard properties.

Acknowledgments

We would like to thank HHMI Janelia Research Campus for hosting the 2025 Janelia Trackathon and the other attendees of the trackathon for their discussion. LX and JYT acknowledge support from the French National Research Agency (France BioImaging, ANR-24-INBS-0005 FBI BIOGEN). LX acknowledges support from the INCEPTION project (PIA/ANR-16-CONV-0005) and the FIRE PhD program funded by the Bettencourt Schueller foundation and the EURIP graduate program (ANR-17-EURE-0012). KS acknowledges support from the JST CREST, Japan (JPMJCR2124).

Competing interests

KS is employed part-time by LPIXEL Inc.

AI Usage Disclosure

All specification and paper content was written manually and reflects the careful thought and input of the community. GEFF is an open source project, and as such contributors are free to use any tools, AI or otherwise, to generate code contained in pull requests. All pull requests are reviewed by a core developer and often iterated on multiple times; therefore, all content in the repository represents the effort and judgment of the authors.

References

Bragantini, J., Theodoro, I., Zhao, X., Huijben, T. A., Hirata-Miyasaki, E., VijayKumar, S., Balasubramanian, A., Lao, T., Agrawal, R., Xiao, S., & others. (2025). Ultrack: Pushing the limits of cell tracking across biological scales. *Nature Methods*, 1–14. https://doi.org/10.1101/2024.09.02.610652

Gonzalez-Beltran, A. N., Masuzzo, P., Ampe, C., Bakker, G.-J., Bésson, S., Eibl, R. H., Friedl, P., Gunzer, M., Kittisopikul, M., Le Dévédec, S. E., Leo, S., Moore, J., Paran, Y., Prilusky, J., Rocca-Serra, P., Roudot, P., Schuster, M., Sergeant, G., Strömblad, S., … Martens, L. (2020). Community standards for open cell migration data. *GigaScience*, *9*(5), giaa041. https://doi.org/10.1093/gigascience/giaa041

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th Python in science conference* (pp. 11–15). https://doi.org/10.25080/tcwv9851

Kyoda, K. H. L. A. T., Koji AND Ho. (2020). BD5: An open HDF5-based data format to represent quantitative biological dynamics data. *PLOS ONE*, *15*(8), 1–11. https://doi.org/10.1371/journal.pone.0237468

Maška, M., Ulman, V., Svoboda, D., Matula, P., Matula, P., Ederra, C., Urbiola, A., España, T., Venkatesan, S., Balak, D. M., & others. (2014). A benchmark for comparison of cell tracking algorithms. *Bioinformatics*, *30*(11), 1609–1617. https://doi.org/10.1093/bioinformatics/btu080

Pietzsch, T., Tinevez, J.-Y., Sugawara, K., Arzt, M., Ulman, V., & Hahmann, S. (2025). *Mastodon – a large-scale tracking and track-editing framework for large, multi-view images*. https://mastodon.readthedocs.io/en/latest/

Tinevez, J.-Y., Perry, N., Schindelin, J., Hoopes, G. M., Reynolds, G. D., Laplantine, E., Bednarek, S. Y., Shorte, S. L., & Eliceiri, K. W. (2017). TrackMate: An open and extensible platform for single-particle tracking. *Methods*, *115*, 80–90. https://doi.org/10.1016/j.ymeth.2016.09.016

Treinish, M., Carvalho, I., Tsilimigkounakis, G., & Sá, N. (2022). Rustworkx: A high-performance graph library for Python. *Journal of Open Source Software*, *7*(79), 3968. https://doi.org/10.21105/joss.03968

Zarr Developers. (2019). *Zarr-specs*. https://github.com/zarr-developers/zarr-specs