The Journal of Open Source Software

DeLTA 3.0: A modern segmentation and tracking tool for microscopy images

Virgile Andreani [1,2¶], **Jean-Baptiste Lugagne** [1,3], **Owen M. O'Connor** [1], **and Mary J. Dunlop** [1¶]

1 Biomedical Engineering, Boston University, USA **2** EPI Lifeware, Inria Saclay, France **3** Department of Engineering Science, University of Oxford, United Kingdom ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 10 October 2025
Published: unpublished

Summary

Time-lapse microscopy is a popular and widely used approach for quantifying single-cell dynamics. Analysis of sequences of images requires two steps: segmentation—locating the cells in each frame—and tracking—identifying how cells in one frame relate to the next. DeLTA (Deep Learning for Time-lapse Analysis) is a Python (Perez et al., 2011) package that performs segmentation and tracking. It was initially developed to handle images of bacteria growing within the "mother machine," (Wang et al., 2010) a popular microfluidic device for single-cell analysis that constrains cells to grow in one dimension (Lugagne et al., 2020), and was later extended to handle cells growing in two dimensions (O'Connor et al., 2022). After three years of experience and feedback from our group and the broader community, we are releasing version 3.0—a major overhaul designed to achieve four key objectives: enhance usability for scientists, improve modularity and adaptability for developers, integrate common features like growth rate computation and lineage manipulation, and adopt modern open-source development practices.

Statement of need

DeLTA is a popular package for cell segmentation and tracking (Ahmadi et al., 2024; Allard et al., 2022; Gericke et al., 2025) that distinguishes itself in several ways from similar tools such as BACMMAN (Ollion et al., 2019), Cellpose (Stringer et al., 2021), CellProfiler (Stirling et al., 2021), MM3 (Thiermann et al., 2024), and Omnipose (Cutler et al., 2022). First, the simplicity of its implementation makes its code pedagogical and easily adaptable. Second, in this new version, utilities are provided for common analysis functions (e.g., calculating growth rate, correcting segmentation and tracking). In addition, in this version, deep learning models have been ported to keras (Chollet & Others, 2015), a high-level backend-agnostic library that allows users to select any of the three main deep learning backends (tensorflow (Abadi et al., 2015), torch (Ansel et al., 2024), or jax (Bradbury et al., 2018)).

The Journal of Open Source Software

General principles of the package

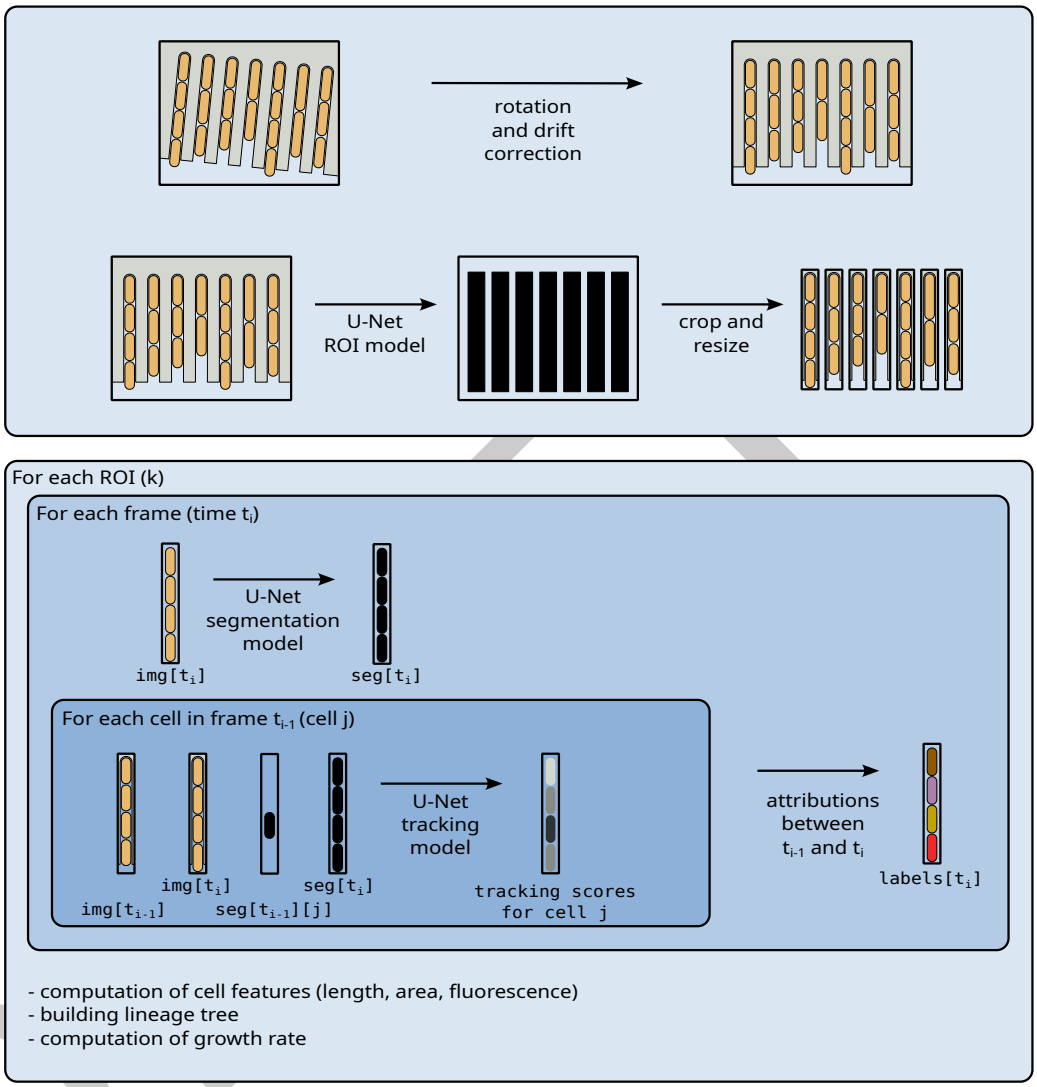

Figure 1: Schematic principles of operation of DeLTA. First, any rotation and drift are removed from the movie. Then, in the case of a mother machine experiment, a region of interest (ROI) model detects and crops individual mother machine channels. Then, for each ROI, each frame is segmented with a segmentation model, and then passed to a tracking model with the previous frame and the mask of a single cell in the previous frame, in order to compute tracking scores between the cells of the previous frame and the cells of the current frame. Attributions are decided based on these tracking scores. Finally, morphological cell features are computed, the lineage tree is built, and growth rates are computed. This approach is common across all versions of DeLTA.

DeLTA uses three different U-Net (Ronneberger et al., 2015) models, one for region of interest (ROI) identification, one for segmentation, and one for tracking. The ROI identification is only used in the 'mother machine' setting, to detect mother machine chambers, which are then analyzed separately; in the '2D' setting, the ROI is the whole frame. Once ROIs have been identified and resized, the segmentation model is run frame-by-frame to determine the location of cells. Finally, for every consecutive pair of frames, the tracking model is run once per cell in the earlier frame, to determine the probabilities that this cell corresponds to each cell in the later frame. After mother-daughter relationships have been determined, a lineage

tree is constructed (see Lineage section) where cell features are stored: area, length, average fluorescence, and growth rate (see Growth Rate section). Finally, these results are saved as a netCDF file that can be accompanied with a segmented movie for direct visualization.

Lineage

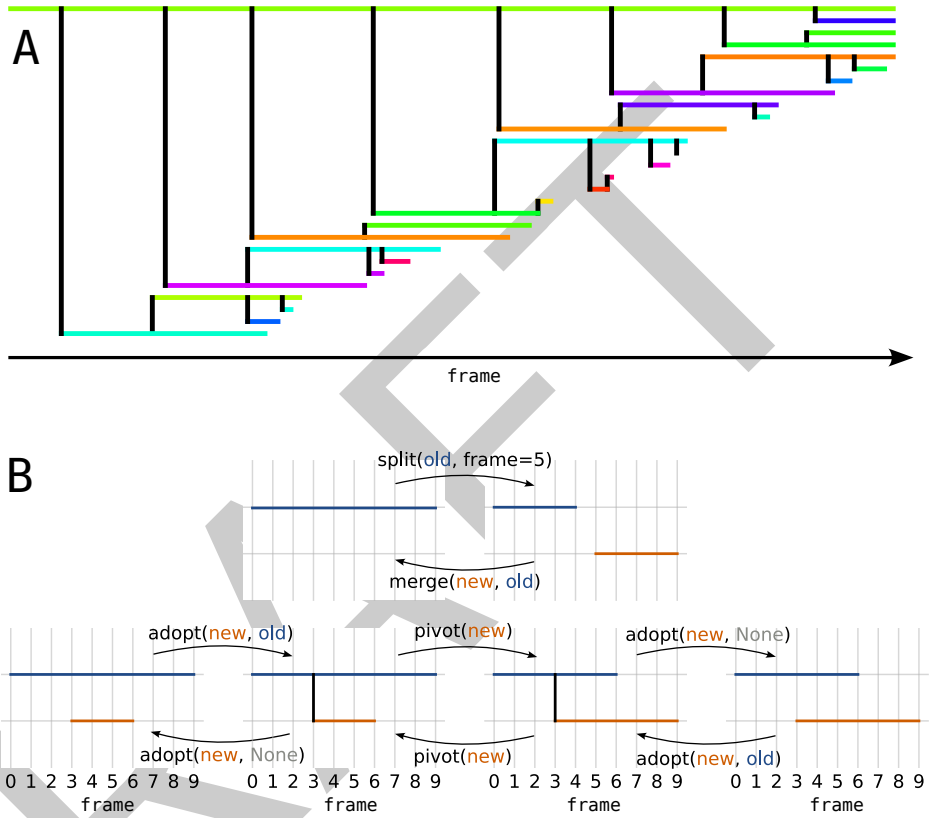

Figure 2: Lineage tree manipulation. **A**: Partial lineage tree of one channel of a mother machine experiment. Horizontal lines represent individual cells, while vertical black lines represent divisions. Cells exit at the bottom of the mother machine, so some lineages terminate before the end of the figure. One can see that the top cell is trapped and remains in the mother machine throughout the time range shown here. **B**: This set of functions can be used to correct any tracking errors. If DeLTA fails to associate one cell with itself at the next frame, the `merge` function can be used to mend the line. The `split` function does the opposite. The `adopt` function can set or unset a mother-daughter relationship, while `pivot` switches a mother and a daughter (the mother being by convention the cell with the oldest pole).

The tracking functionality of DeLTA detects cell divisions, linking mother cells to their daughters. Bacterial cell divisions are typically symmetrical, however pole age has measurable effects on cell physiology, where the daughter cell that inherits the older end of the mother cell grows more slowly than the other daughter (Stewart et al., 2005). For this reason, we consider that the mother cell persists through divisions, giving birth to one daughter (instead of the mother cell disappearing and giving rise to two daughter cells). Thus, upon cell division, the mother cell retains its ID number, while the new cell is given a new ID.

DeLTA constructs the lineage tree frame-by-frame, by extending existing cells and creating new daughter cells at divisions. However, tracking errors are possible, caused for example by a temporary air bubble in the field of view, or by a segmentation or a tracking mistake by the neural networks. These cases can easily be spotted by experimentalists, who might want to correct them by relabeling the cells involved. However, relabeling of a cell requires updating

its whole lineage subtree. This operation, depending on the nature of the error to be fixed, can be error-prone. For this reason, we implemented a set of elementary operations on the tree, whose combined effect allows for arbitrarily complex lineage tree manipulations so that this process is straightforward for users [Figure 2].

Growth rate

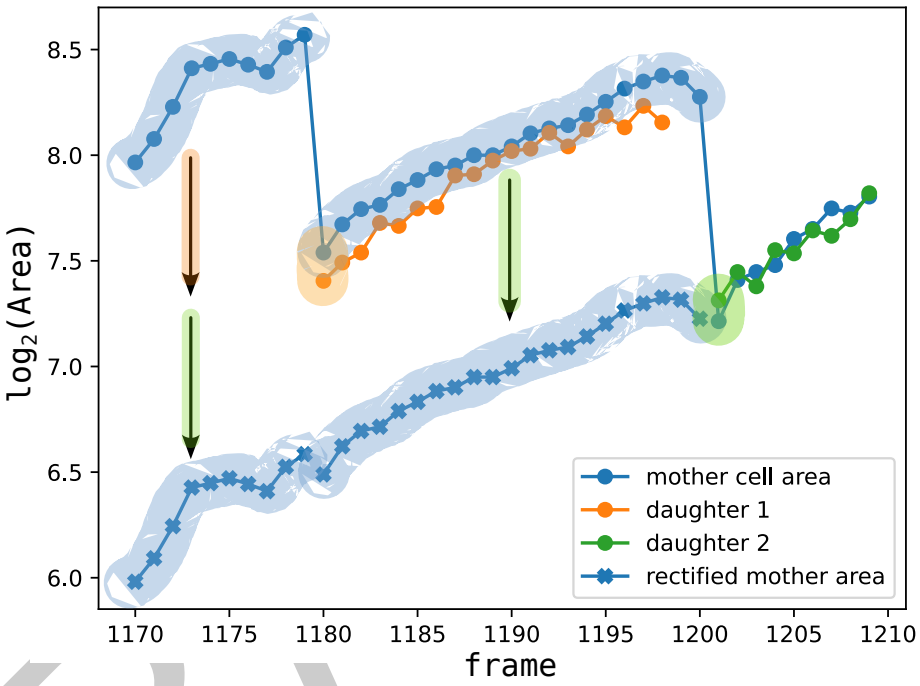

Figure 3: Alignment of cell areas for growth rate computation. Circle markers show the logarithm of the area (number of pixels) of a mother cell and two of its daughters. Jumps can be seen when the cell divides. Cross markers show entire cell cycles that are offset (colored arrows) as a function of the relative sizes of the cells after division (highlighted pair of points). The offset is $\log_2\left(1+\frac{\text{Area}_{\text{mother}}}{\text{Area}_{\text{daughter}}}\right)$. After offsetting, the growth rate can be computed at any point, even with a centered difference scheme.

Single-cell growth rate is often used as a proxy for cell fitness, which may depend on cell genotype or the surrounding environment. Within a given population, individual cells have different growth rates, which may also change even within one cell cycle. The single-cell growth rate is often defined as a function $\mu(t)$ that satisfies the following differential equation where $\ell(t)$ is the cell length:

$$\frac{\mathrm{d}\ell}{\mathrm{d}t} = \mu(t)\ell(t)$$

DeLTA can calculate growth rates based on cell area or length. Area-based growth rates rely on direct counts of the number of pixels per cell, while length-based growth rates derive length estimates from the bacterial shape. Growth rates based on volume are not possible to calculate with images at a single focal plane, however length, area, and volume growth rates are generally equivalent within a few percent.

Instantaneous growth rate is computed in DeLTA as the derivative of the logarithm of cell length (or area). Because segmentation noise makes this time series noisy, we use a Savitsky-Golay

filter (Savitzky & Golay, 1964) to simultaneously smooth and differentiate the signal. It is a centered finite difference scheme, which is a deliberate choice to avoid discretization biases.

To compute the growth rate, the Savitsky-Golay filter requires at least one data point for the previous frame and one for the next (or more than one for larger window sizes). To avoid apparent jumps in growth rate around cell division events, we offset the areas before and after division by extrapolation of the lineage. This amounts to considering that the area of a mother cell after division is the sum of its own area and that of its daughter. This allows us to efficiently compute growth rates of entire cell lineages, across divisions (Figure 3).

File format of results

Analysis results produce image-like data (segmentation masks and label masks), as well as scalar data associated with every cell (ID of mother cell), or with every cell and frame (features such as length, area, average fluorescence, growth rate). In DeLTA 2.0 we used the `pickle` library to serialize the Python object representing the results into a file. While this allows for the saving of arbitrary Python objects, these files can only be opened with the same version of DeLTA that created them.

To store results, we are now using netCDF4 (Rew et al., 1989), an open file format based on HDF5 (The HDF Group, 2025), popular for storing rectangular scientific data. netCDF4 libraries are available in many programming languages, including Python and MATLAB. A limitation of this approach, compared to pickle files, is that the HDF5 data format only supports collections of rectangular data arrays, which required us to reorganize the data describing cell features and lineages in a rectangular fashion, at the expanse of file size, although this can be mitigated by compression or by using sparse arrays, which is a future goal. The change to netCDF4 is an important improvement for compatibility across future version of DeLTA.

Conclusion

DeLTA 3.0 enhances function for both novice and experienced users with new features (including lineage tree manipulation and growth rate calculation), better interoperability (results stored as netCDF files), and backend-agnostic deep-learning models (implemented in pure keras).

Acknowledgements

The authors would like to acknowledge the contributions of Idris Kempf, Caroline Blassick, and Eric Bueno to the development of version 3.0 of DeLTA; feedback from the rest of the Dunlop lab, and input from the wider community through opening issues, support, and feature requests on DeLTA's gitlab repository.

This work was supported by the National Science Foundation (MCB-2143289).

As of this version, direct dependencies of DeLTA include numpy (Harris et al., 2020), scipy (Virtanen et al., 2020), scikit-image (Walt et al., 2014), xarray (Hoyer & Hamman, 2017), pooch (Uieda et al., 2020), opencv (Bradski, 2000), tqdm (*Tqdm/Tqdm*, 2025), ffmpeg (Tomar, 2006), netCDF4 (Rew et al., 1989), bioio (Brown et al., 2023), termcolor (*Termcolor/Termcolor*, 2025), keras (Chollet & Others, 2015), and elasticdeform (Tulder & Fringeli, 2024).

Utilities that we use in the development process include ruff (*Astral-Sh/Ruff*, 2025), mypy (*Python/Mypy*, 2025), deptry (Maas, 2025), pixi (Arts et al., 2025), pip (*Pypa/Pip*, 2025), pytest (Krekel et al., 2004), pytest-cov (*Pytest-Dev/Pytest-Cov*, 2025), pre-commit (*Pre-Commit/Pre-Commit*, 2025), matplotlib (Hunter, 2007), sphinx (*Sphinx-Doc/Sphinx*, 2025), numpydoc (*Numpy/Numpydoc*, 2025), sphinx-design (*Executablebooks/Sphinx-Design*, 2025), and furo (Gedam, 2025).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., … Zheng, X. (2015). *TensorFlow, Large-scale machine learning on heterogeneous systems*. https://doi.org/10.5281/zenodo.4724125

Ahmadi, A., Courtney, M., Ren, C., & Ingalls, B. (2024). A benchmarked comparison of software packages for time-lapse image processing of monolayer bacterial population dynamics. *Microbiology Spectrum*. https://doi.org/10.1128/spectrum.00032-24

Allard, P., Papazotos, F., & Potvin-Trottier, L. (2022). Microfluidics for long-term single-cell time-lapse microscopy: Advances and applications. *Frontiers in Bioengineering and Biotechnology*, *10*. https://doi.org/10.3389/fbioe.2022.968342

Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., … Chintala, S. (2024, April). PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. https://doi.org/10.1145/3620665.3640366

Arts, R., Zalmstra, B., Vollprecht, W., Jager, T. de, Morcotilo, N., & Hofer, J. (2025). *Pixi*. https://github.com/prefix-dev/pixi/releases/tag/v0.53.0

Astral-sh/ruff. (2025). Astral. https://github.com/astral-sh/ruff

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal of Software Tools*.

Brown, E. M., Toloudis, D., Sherman, J., Swain-Bowden, M., Lambert, T., Meharry, S., Whitney, B., & BioIO Contributors. (2023). *BioIO: Image reading, metadata conversion, and image writing for microscopy images in pure python*. GitHub. https://github.com/bioio-devs/bioio

Chollet, F., & Others. (2015). *Keras*. https://keras.io

Cutler, K. J., Stringer, C., Lo, T. W., Rappez, L., Stroustrup, N., Brook Peterson, S., Wiggins, P. A., & Mougous, J. D. (2022). Omnipose: A high-precision morphology-independent solution for bacterial cell segmentation. *Nature Methods*, *19*(11), 1438–1448. https://doi.org/10.1038/s41592-022-01639-4

Executablebooks/sphinx-design. (2025). Executable Books. https://github.com/executablebooks/sphinx-design

Gedam, P. (2025). *Pradyunsg/furo*. https://github.com/pradyunsg/furo

Gericke, B., Degner, F., Hüttmann, T., Werth, S., & Fortmann-Grote, C. (2025). *Performance review of retraining and transfer learning of DeLTA2 for image segmentation for pseudomonas fluorescens SBW25*. 273–280. ISBN: 978-989-758-688-0

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hoyer, S., & Hamman, J. (2017). Xarray: N-d labeled arrays and datasets in python. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.148

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B., & Bruhin, F. (2004). *Pytest 7.5* (Version 7.5). https://github.com/pytest-dev/pytest

Lugagne, J.-B., Lin, H., & Dunlop, M. J. (2020). DeLTA: Automated cell segmentation, tracking, and lineage reconstruction using deep learning. *PLOS Computational Biology*, *16*(4), e1007673. https://doi.org/10.1371/journal.pcbi.1007673

Maas, F. (2025). *Fpgmaas/deptry*. https://github.com/fpgmaas/deptry

Numpy/numpydoc. (2025). NumPy. https://github.com/numpy/numpydoc

O'Connor, O. M., Alnahhas, R. N., Lugagne, J.-B., & Dunlop, M. J. (2022). DeLTA 2.0: A deep learning pipeline for quantifying single-cell spatial and temporal dynamics. *PLOS Computational Biology*, *18*(1), e1009797. https://doi.org/10.1371/journal.pcbi.1009797

Ollion, J., Elez, M., & Robert, L. (2019). High-throughput detection and tracking of cells and intracellular spots in mother machine experiments. *Nature Protocols*, *14*(11), 3144–3161. https://doi.org/10.1038/s41596-019-0216-9

Perez, F., Granger, B. E., & Hunter, J. D. (2011). Python: An ecosystem for scientific computing. *Computing in Science & Engineering*, *13*(2), 13–21. https://doi.org/10.1109/MCSE.2010.119

Pre-commit/pre-commit. (2025). pre-commit. https://github.com/pre-commit/pre-commit

Pypa/pip. (2025). Python Packaging Authority. https://github.com/pypa/pip

Pytest-dev/pytest-cov. (2025). pytest-dev. https://github.com/pytest-dev/pytest-cov

Python/mypy. (2025). Python. https://github.com/python/mypy

Rew, R., Davis, G., Emmerson, S., Cormack, C., Caron, J., Pincus, R., Hartnett, E., Heimbigner, D., Appel, L., & Fisher, W. (1989). *Unidata NetCDF*. UCAR/NCAR - Unidata. https://doi.org/10.5065/D6H70CW6

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical image computing and computer-assisted intervention – MICCAI 2015* (Vol. 9351, pp. 234–241). Springer International Publishing. https://doi.org/10.1007/978-3-319-24574-4_28

Savitzky, Abraham., & Golay, M. J. E. (1964). Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, *36*(8), 1627–1639. https://doi.org/10.1021/ac60214a047

Sphinx-doc/sphinx. (2025). Sphinx. https://github.com/sphinx-doc/sphinx

Stewart, E. J., Madden, R., Paul, G., & Taddei, F. (2005). Aging and death in an organism that reproduces by morphologically symmetric division. *PLOS Biology*, *3*(2), e45. https://doi.org/10.1371/journal.pbio.0030045

Stirling, D. R., Swain-Bowden, M. J., Lucas, A. M., Carpenter, A. E., Cimini, B. A., & Goodman, A. (2021). CellProfiler 4: Improvements in speed, utility and usability. *BMC Bioinformatics*, *22*(1), 1–11. https://doi.org/10.1186/s12859-021-04344-9

Stringer, C., Wang, T., Michaelos, M., & Pachitariu, M. (2021). Cellpose: A generalist algorithm for cellular segmentation. *Nature Methods*, *18*(1), 100–106. https://doi.org/10.1038/s41592-020-01018-x

Termcolor/termcolor. (2025). termcolor. https://github.com/termcolor/termcolor

The HDF Group. (2025). *Hierarchical data format, version 5*. https://github.com/HDFGroup/hdf5

Thiermann, R., Sandler, M., Ahir, G., Sauls, J. T., Schroeder, J., Brown, S., Le Treut, G., Si, F., Li, D., Wang, J. D., & Jun, S. (2024). Tools and methods for high-throughput single-cell imaging with the mother machine. *eLife*, *12*, RP88463. https://doi.org/10.7554/eLife.88463

Tomar, S. (2006). Converting video formats with FFmpeg. *Linux J.*, *2006*(146), 10.

Tqdm/tqdm. (2025). tqdm developers. https://github.com/tqdm/tqdm

Tulder, G. van, & Fringeli, G. (2024). *Gvtulder/elasticdeform: Version v0.5.1*. Zenodo. https://doi.org/10.5281/zenodo.11267397

Uieda, L., Soler, S. R., Rampin, R., Kemenade, H. van, Turk, M., Shapero, D., Banihirwe, A., & Leeman, J. (2020). Pooch: A friend to fetch your data files. *Journal of Open Source Software*, *5*(45), 1943. https://doi.org/10.21105/joss.01943

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

Walt, S. van der, Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., & Yu, T. (2014). Scikit-image: Image processing in python. *PeerJ*, *2*, e453. https://doi.org/10.7717/peerj.453

Wang, P., Robert, L., Pelletier, J., Dang, W. L., Taddei, F., Wright, A., & Jun, S. (2010). Robust growth of escherichia coli. *Current Biology*, *20*(12), 1099–1103. https://doi.org/10.1016/j.cub.2010.04.045