THE UNIVERSITY *of* EDINBURGH

Phenotyping single cells of Saccharomyces cerevisiae using an end-to-end analysis of high-content time-lapse microscopy.

Alán Fernando Muñoz González

THE UNIVERSITY
of EDINBURGH

PhD in Quantitative Biology, Biochemistry and biotechnology

University of Edinburgh

December 2022

Declaration

I declare that the thesis has been composed by myself and that the work has not been submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included. My contribution and those of the other authors to this work have been explicitly indicated below. I confirm that appropriate credit has been given within this thesis where reference has been made to the work of others.

Alán Muñoz

Phenotyping single cells of Saccharomyces cerevisiae using an end-to-end analysis of high-content time-lapse microscopy.
Thesis, University of Edinburgh, Scotland, United Kingdom
Illustrated; with bibliographic information

Abstract

The field of systems biology has developed under the acknowledgement that quantitative approaches are necessary to fully understand the complexities that shape cells' structure and behaviour. The development of automated microscopy protocols has led to the accumulation of high-throughput data sets but the high resolution of modern imaging hardware compounded with time series generates an analysis bottleneck.

In this thesis I discuss the methods and tools we have developed to process live-cell imaging time series in a fully automated manner. I then showcase a situation in which such an analysis proves essential to extract biological insight on the protein aggregation dynamics of *Saccharomyces cerevisiae*.

In chapter 1 I develop a method that uses machine learning algorithms to track single-cell time-lapses based only on cell outlines, and is robust to experimental and technical noise, as well as varied experimental conditions. I explain the statistical considerations necessary to compare long-term tracking algorithms and discuss useful benchmark metrics for time-inclusive image analyses.

In chapter 2 I use the cell segmentation and tracking software as a base to develop an analysis tool that automates all steps in a reproducible manner. It is the fastest tool as far as we are aware to process data from pillar-style microfludic devices, improving on existing pipelines by at least an eight-fold factor, and thus changing the ways we can study cell decision-making at the single-cell level. In chapter 3 I review the current available computational tools that may be useful for developing data analysis software for biology. I then propose a system for sustainable scientific software development that uses modern development technologies and enables community-driven efforts.

In the last chapter I use the image-processing pipeline to study the protein aggregation dynamics of metabolic enzymes that form reversible aggregates via phase separation. We complement this with single-cell physiology information to gain a deeper understanding of the impact of these proteins on yeast cells' growth and division. Heterogeneous responses, distinguishable by using mutant strains with varying aggregation phenotypes, are studied using high-throughput correlations.

This work shows that the development of computational tools can make the acquisition of biological insight faster and increases reproducibility while providing a toolset to tackle a problem that had no practical alternatives before. It also furthers biological understanding of a conserved phenomenon such as reversible protein aggregation, in a way only possible by studying behaviour and physiology of thousands of cells at the same time, a feat that up until now required much correction by hand to be practical. My thesis aims to pave the way for more comprehensive analyses using aggregated high-quality data.

Lay summary

The robustness and adaptability of living organisms is remarkable, cells that are genetically identical can produce a wide variety of behaviours. This is true for organisms of any type or size. In recent decades many mechanisms of gene regulation have been uncovered, this led to the realisation of just how astoundingly complex living beings are.

To survive, unicellular organisms must be able to respond in a wide array of ways, deciding how based their environment and how hard it makes it for them to live, a frequently overlooked fact is that they are constantly surrounded by their environment. Their responses to external changes do not occur in isolation, before changes occur, they were adapted or adapting to live in a different one. To fully comprehend what that makes populations of microorganisms so robust, we must understand how cells plan to react to environmental changes. This requires information about what were cells up to before changes ensued.

The methods used to access information inside cells, such as DNA -the blueprint of life -or proteins -molecules that perform most intracellular processes -usually kill the cells, making them unsuitable for investigating cells' behaviour in changing environments. The main non-invasive method to study cells is looking at them using microscopes. It enables imaging cells long periods of time, which will be crucial to understand how they make decisions. Up until recently, it required a lot of time and effort, but automation tools for the acquisition of microscopy movies has become available for scientists to use.

These advances ended up changing the issue: biologists transitioned from having a lack of images to an excess of them. We focus in developing tools to access the information hidden inside microscopy movies. For this purpose budding yeast is a

very useful organism: it is relatively simple and grows fast; it also shares many genes and regulatory mechanisms with humans, thus making discoveries in yeast useful for multiple areas of biology. By building on top of previous tools developed within our research group, we developed new software that enables the study thousands of yeast cells over the course of their lifetimes, with minimal human effort. This allows us to acquire information on cell size and growth, in addition to signalling inside the cell. By automating processing and data aggregation, we reduce the need of analysing each experiment individually, empowers biologists to focus their efforts on questioning the data.

With the software I developed, I study proteins that form filaments as a reaction to lack of energy sources. I uncover and quantify variability within cell populations that are usually assumed to be identical, and understand how some changes in DNA that affect the likelihood of filament assembly. Our study also sheds light on the impact of such aggregation of proteins in the growth capabilities of individual cells, thus shaping populations.

Acknowledgements

First, I would like to start thanking my supervisor Peter. This work would not have been possible without him providing the right balance of both creative freedom and dedicated attention. He infused some sense of order to these projects; despite his suggestion against the usage of the word "betwixt", I decided to hide it in here.

I would like to thank all members of the Swain Lab, past and present. We could discuss anything, from deep scientific ideas to how to use Emacs as a cooking recipe manager; they helped the fun and exciting aspect of science, be palpable at times.

I would also like to thank my friends, who during these years have shaped me through our chats, walks and other trips all over Scotland and beyond. I appreciate their acceptance of my nonsensical late-night epiphanies and barely-planned adventures at face value.

To finish, I would like to thank my family: my parents, who taught me discipline and optimism, which were particularly useful these past few years, and my siblings, from whom I learned what it means to always be there for one another. Thanks to my Mamanina and Tío Jorge, who always rooted for me, but could not see me wrap this up; I like to think they would be happy and proud of what I have done so far.

Phenotyping single cells of Saccharomyces cerevisiae using an end-to-end analysis of high-content time-lapse microscopy.

Doctoral Thesis

to obtain the degree of doctor
from the University of Edinburgh
to be defended
on Wednesday 1st February, 2023
at 16:00 hours

by

Alán Fernando Muñoz González

Supervisor: Prof. Peter S. Swain

Doctoral Thesis Committee: Dr. Edward Wallace (Internal examiner)

University of Edinburgh

Prof. Carsten Marr

Helmholtz Zentrum München

Contents

1 Introduction

1.1 Single cell microcsopy tracking

1.1.1 Tracking as a computer vision task

Object tracking is a task in the field of computer vision in which an algorithm or model identifies discrete entities between frames in a time lapse of images. These objects result from other (generally) independent-but-related tasks: object detection and segmentation. These in turn consist in the ability to identify the location and boundaries of a specific type of object, such as a person or a car, in an image. Detection is only concerned in finding their location within the image, whilst segmentation differentiates the pixels that belong to a specific object and the ones that not.

A tracking algorithm must account for multiple transitions for detected and segmented objects: Their permanence between frames, their arrival into existence within the time series, as well as temporal and permanent disappearances - either due to real disappearance or detection and segmentation errors. The likelihood of these events, in combination with the shape and - if applicable - image values within the span of a given object define the difficulty a given tracking algorithm will face.

Single-cell analysis of image time lapses demands identification of individual cells over multiple frames. Correct tracks are thus crucial to produce valid biological results, because most cells whose identification is short-lived provide little to no useful information. Noise propagates from segmentation onto tracking, in such a way that high-quality tracking algorithm should be robust to segmentation noise while retaining robustness by avoid catering to any specific segmentation algorithm.

We can split cell-tracking methods in two groups, based on whether or not they split segmentation and tracking steps: The first group is contour evolution methods, in which we identify cells directly from experience (having seen an object previously) (Panels 1.1C,E); the second group is tracking by detection, in which we find all objects first (Panel 1.1D) and then proceed to establish identity relationships between these objects, as shown in Panel 1.1D (Ulman et al., 2017). The latter is more widely used, as most are time-independent and have become the to-go approach to identify cells in images. There are different ways to group cell tracking methods, but those are better covered by others (Emami et al., 2020).

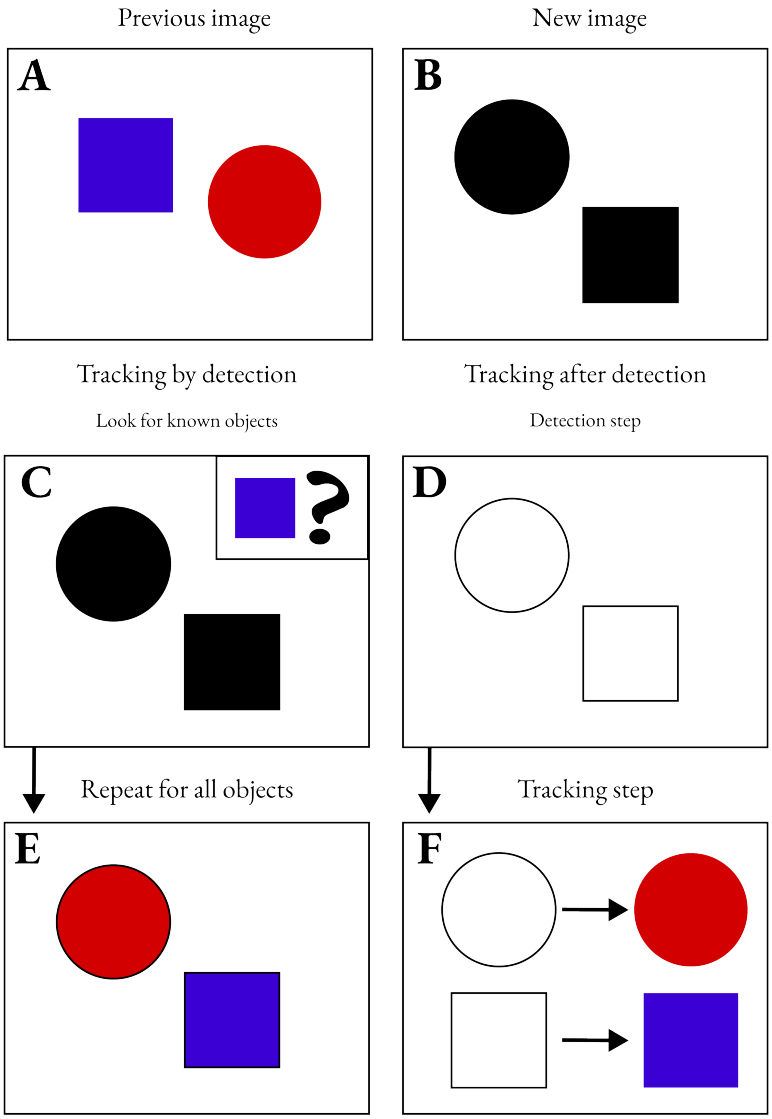

Figure 1.1: **There are two types of tracking based on how they deal with detection (or segmentation): tracking by detection and tracking after segmentation.** In this diagram we show tracking objects between two consecutive time points. **A.** First time point with labels that are annotated, the square is labelled blue and the circle is labelled red. **B.** Second time point with unlabelled entities. **C.** A tracking-by-detection system keeps searches for similarities between the existing (and labelled) entities and looks for them in the new image. **D.** A tracking-after-detection system first detects the existing entities in the new time point. **E.** To finish tracking by detection just search for all the known entities in the new image. **F.** Tracking after detection compares the identified outlines against the known entitites, thus transforming the task into solving an assignment problem.

Tracking by detection is generally contingent on segmentation, at the same time (and counter-intuitively), it is not heavily influenced by segmentation performance (Ulman et al., 2017). In practice, this means that as long as segmentation performs decently, the accuracy of tracking depends only on its own performance. The decoupling of segmentation and tracking as independent tasks makes different tracking algorithms comparable.

When using tracking after segmentation, we end up with multiple hypotheses of identity (which objects are which in two consecutive time points), this unequivocally becomes an assignment problem. The formal definition of such a problem is: Given two sets A and T with a weight function (pairwise probabilities) $C : AxT \rightarrow 0, 1$. Find a bijection $f : A \rightarrow T$ such that the following cost function is minimised:

$$\Sigma_{a \in A} C(a, f(a)) \tag{1.1}$$

Thus we need an algorithm to resolve the assignment problem, regardless of how the pairwise probabilities are defined. Unlike the traditional assignment problem, we must also differentiate reocurring from new entitites regardless of whether or not the sizes of A and T are equal.

Finally, we must be careful when translating "traditional" computer vision metrics into biology. Performance metrics used for "traditional" computer vision tracking may not be as useful for single-cell analyses. Tracking algorithms that perform accurately in multiple short time-spans may not work for longer series of frames. This is a consequence to the number of correct traces decreases rapidly with the duration of a time series experiment (Versari et al., 2017).

1.1.2 Live microscopy analyses rely upon cell segmentation and tracking

To quantify changes in single cells over time, the first challenge to solve is identifying where cells are and following them over consecutive time points. Cell segmentation

focuses on the first task and, as explained in the previous section, tracking aims to tackle the second task. These two problems be solved at the same time or sequentially, depending on which algorithm or model we use.

Multiple algorithms exist to solve the general cell-tracking task, each with varying degrees of success. For the sake of briefness, I will give a general overview of the most relevant ones in my opinion, favouring an automated or semi-automated approach over manual post-processing of results. In spite of my overview, it is worth noting that as of 2023, the most popular general cell segmentation algorithms are *Cellpose* (Stringer & Pachitariu, 2022) and *Stardist* (Schmidt et al., 2018), the latter now commonly used in combination with *Trackmate* for tracking (Fazeli et al., 2020; Tinevez et al., 2017).

The first tracking algorithms implemented a variation of the linear sum assignment algorithm (also referred to as the "Hungarian" algorithm) to minimise the sum of some weights that represented relationships between cells in consecutive frames (Falconnet et al., 2011; Versari et al., 2017). These work well under a combination of conditions: cells fixed to a pad and when stage drift is low, but they are deficient when pitted against machine and deep learning approaches (Ulman et al., 2017).

Latest developments favour deep learning as the state-of-the-art tool for segmentation, providing a variety of solutions: *Aspert et al.* avoid single-cell tracking using an event-based methodology (Aspert et al., 2022), where they are looking for division events instead; delegating tracking to existing tools and for semi-automation is an alternative (Kapoor & Carabaña, 2021).

Research groups concatenate multiple deep learning models and algorithms in a pipeline (Moen et al., 2019; O'Connor et al., 2022). Despite this being likely to outperform the usage of fewer deep learning models and algorithms, maintaining numerous deep learning models and algorithms is likely to become a practical challenge for long-term sustainability.

In a paper whose tracking algorithm deviates from "traditional" approaches, *Ulicna et al.* use the processed output of a U-Net (semantic segmentation CNN architec-

ture, popular for biological imaging) and the linear sum assignment algorithm on a Bayesian belief matrix to assign cell identities (Ulicna et al., 2021). Another noteworthy distinctive feature of this paper is that tracking is particularly challenging because their cells, unlike *S. cerevisiae*, are motile, rendering purely distance-based tracking unreliable.

Some segmentation and tracking pipelines result in regions of interest that characterise the bounds of an object within an image (masks) and require custom preprocessing. For instance, using watershed transforms to find a single point at each cell interior (Wood & Doncic, 2019), and those interior points serve as a seed for segmentation and tracking. This step automates their analyses at the cost of making it more specific to their use-case, and thus laborious to adapt for different setups.

1.1.3 Design decisions are dependant on the type of data

Cell segmentation can be performed in most type of images, the most standard being bright-field, Differential Interference Contrast (DIC) and fluorescence. Bright-field and DIC are consistent over time, but fluorescence-based segmentation only works when tagging fluorophores that are consistently cytosolic and homogeneously diluted (i.e., no localisation nor aggregation) during cell cycle and environmental conditions that enable normal cell growth and division. If tracking is performed after and not during segmentation, the type of data is unimportant for the former. When both occur at the same time, bright-field images - which are arguably harder to segment manually (Wood & Doncic, 2019) - may affect the tracking quality as a consequence of impaired segmentation. Additionally, in the case of fluorescence-driven segmentation, experimental acquisition settings (e.g., exposure, LED voltage) impact segmentation due to their effect on the value distribution of pixels for fluorescence at the edges of the cell membrane.

One last complication for dealing with cell microscopy data is our base assumption on overlap. Existing software packages assume objects in two-dimensional images, ignoring any potential overlap between cells (*Cellpose: A Generalist Algorithm for*

Cellular Segmentation | Nature Methods, n.d.; Padovani et al., 2022). This sacrifices generalisation, specially for setups in which overlaps occur, but greatly reduces computational cost and complexity and simplifies setups for image acquisition. Notably *Cellstar* is one of the popular computational tools that includes support for overlaping cells (Versari et al., 2017).

1.1.4 Cell retention probability depends on the trapping mechanism

Performance scalability is a challenge largely overlooked when dealing with long-term time lapses of single-cells. To formalise this within a computational complexity framework I will use the Big Oh notation (Chivers & Sleightholme, 2015), an algorithm analysis tool that greatly simplifies our ability to compare the efficiency of algorithms. Cells divide exponentially, as such an algorithm complexity's upper bound (Θ_{alg}) ranges from constant $\Theta_{const}(c)$, to linear $\Theta_{lin}(c)$, to quadratic $\Theta_{quad}(c^2)$, where c is the number of cells.

A Θ_{const} tracking algorithm is able to identify an increasing number of cells with zero (or, in practice, minimal) increases in computations needed, algorithms of this kind use heuristics to keep the number of computations low; Θ_{lin} algorithms process each cell individually and once, for instance, in watershed-based algorithms that deform a "seed" outline until it fits an expected (Falconnet et al., 2011; Versari et al., 2017; Wood & Doncic, 2019); lastly, a naive Θ_{quad} algorithm compares all cells against each other, assuming n is approximately constant between consecutive frames, thus complexity becomes c^2 (Lugagne et al., 2020; Ulicna et al., 2021).

The computational cost of quadratic algorithms is exacerbated when the experimental setup produces exponentially-growing numbers of cells. Recent protocols estimate the duration of segmentation, tracking and linage analysis within a 2-to-3 week time-span (Bheda et al., 2020). The gap in computational cost between automated pipelines with efficient algorithms, and either manual pipelines or automated ones using computationally inefficient algorithms delays the results and, indirectly, biological discoveries as well.

Some of these problems can be attenuated using microfluidics, a system to manipulate small amount of fluids to have a precise control of the environment where cells grow. Microfluidic single-cell analyses become more manageable is through the usage of devices that retain some cells and let others go, generally in constant media flow - sometimes called hydro-dynamically washed cells - such as the mother machine, microfluidic device to trap a bacteria in place for time lapse microscopy (Wang et al., 2010) and the ALCATRAS system, its equivalent for budding yeast (Crane et al., 2014). Unlike pad-based experimental setups, the majority of cells in a trap-based time lapse will be transient. Only trapped cells remain in-frame for a sizeable duration, restraining the exponential explosion of cells in an image by getting rid of a subset of cells. Due to asymmetrical division, buds (extension of mother yeast that grow in volume to become an independent cells) are flushed away by the constant flow of media.

Unlike pad-based systems, in trap-based devices we can modulate the number of cells we maintain in-frame by adjusting the media flow pressure for our microfluidic devices. I will evaluate a simplified example to show the difference. Let us say the rate of washed-out cells is $w(t) = p_{wash} \cdot c_{free}(t - \Delta t)$, where t is a continuous time point, p_{wash} is the probability of the media flow washing-out free cells, c_{free} is the number of non-trapped cells and μ is the average division rate of cells. Then if $c_{tot} = c_{trap} + c_{free}$, with c_{tot} and c_{trap} are the total number of cells and the number of trapped cells, respectively, we get equations (1.2) and (1.3):

$$c_{tot}(t) = c_{tot}(t - \Delta t) \cdot 2^{\Delta t \mu} - w(t) \tag{1.2}$$

$$= c_{tot}(t - \Delta t) \cdot 2^{\mu} - p_{wash} \cdot c_{free}(t - \Delta t) \tag{1.3}$$

In reality, there are also probabilities that change the $\frac{c_{free}}{c_{tot}}$ ratio, and free and trapped cells have different μ division times, but for simplicity we will ignore all this and assume the fraction of trapped cells k a constant, thus $c_{free} = k \cdot c_{tot}$ and the previous expression factors into equation (**??**).

$$\underline{c_{tot(t)}c_{tot}(t-\Delta t)(t) = (2^{\mu} - p_{wash} \cdot k)}$$

equation

This indicates that the rate at which cell number increase - and thus the input for our complexity function $\Theta(c)$ - results from the division rate μ, cell-washing rate p_{wash} and the fraction of free cells k.

All cells are equal in the eyes of tracking, but some cells' tracking is more important than others'. While a single frame transition error does not impact the total accuracy, it has a strong impact on the resulting biological insight. In a best-case scenario, the cell is not assigned to an existing track and fades away from the analysis automatically. In the worst-case scenario, we combine two independent signals, obfuscating the biology obtained from them. In the future, this model could be extended into a pressure control model for live experiments to calculate the media flow pressure necessary to keep cell numbers under control automatically and in parallel to the running experiment.

1.2 Challenges of high-throughput microscopy

1.2.1 Existing biological microscopy data is limited by existing software infrastructure

Most single-cell microscopy studies I found do not venture beyond the realm of looking at tens of cells for short (i.e., $<$ 10 hours) periods of time. I suggest three main underlying reasons: First, the demand for high-end microscopy imaging in biological research labs, as they are often shared among multiple research groups. Long experiments thus usually involve full control or ownership of a microscope. While this constraint does not prevent time series imaging experiments, it reduces the amount of feasible experiments by placing barriers in the unfettered access to microscopy infrastructure, limiting the biological insight that is available. The second reason is the technical challenges that scaling-up experiments entail. Thanks to automation

of image acquisition, such as *Micromanager* (Edelstein et al., 2014), scientists and engineers can produce bespoke tools to image multiple time series in parallel, with a multitude of custom settings and conditions. Recently *Pinkard et al.* tackled this challenge and thus there are available options to use and extend image-acquisition software, but despite these advances, the technical barrier is still not low for people without extensive programming experience (Pinkard et al., 2021). The third cause is the explosion in space and computational analysis costs of images resulting in more than tens of gigabytes in storage. High-throughput data poses a challenge by itself, but we must add more: these analyses require the usage of fully or semi-automated techniques, design and coordination of data structures, server-client interactions and the implementation of image processing algorithms.

The limited availability of technology that facilitates long live-microscopy experiments means that these type of experiments occurred within a subset of research laboratories. Each of these groups developed their own analysis pipelines. The lack of open source computational infrastructure, that is, software tools and techniques, limited the capacity of sharing them and set community standards.

Despite the recent release of multiple deep-learning models and algorithms for budding yeast segmentation (Dietler et al., 2020; O'Connor et al., 2022; Padovani et al., 2022; Salem et al., 2021; Wood & Doncic, 2019), there is little to no consistency in their usage and standardisation. On the other spectrum of an analysis pipeline, tools focused on visualisation of multidimensional imaging data do exist, are more standardised and better documented, the best example being *napari* (Sofroniew et al., 2022).

Scaling analyses of large multidimensional images thus requires solving multiple challenges. The main ones are: Reading data from different sources, from a desktop computer and from a server, pre-processing images, passing them to and through multi-step pipelines, formatting and storing results while preserving the results' reproducibility, and, finally, providing both low and high-level access interfaces to developers and users.

1.2.2 Limiting data footprint and being compatible with multiple sources

High-content screening for microscopy combines automated microscopy with quantitative image analysis (Mattiazzi Usaj et al., 2016). Such screenings and our imaging time lapses share multiple challenges, making OMERO (Li et al., 2016) an adequate tool to tackle data and metadata management. We must organise data through the implementation of multiple structures, such as projects, groups, experiments and/or tags. It is also good practice to store the experimental metadata and experimentalists' comments alongside the raw data.

Another important feature we require is filtering and selecting sections of these data sets or their entirety. OMERO covers most of these organisation tasks, complemented by additional processes downstream.

Dealing with multidimensional images requires a different approach from the ones used for smaller or non-image data sets, such as sequencing or simulation-derived data. The storage-size footprint of an average experiment is half a terabyte upon acquisition, compressing to half that size at best. Storing them on a personal-use computer is thus both impractical and unsustainable on the long-term, as we accumulate experiments. Even caching the data during processing might lead to excessive usage of storage space.

If we are to avoid storing data locally, we should download images as just before processing them, thus limiting the storage footprint to the output of our pipeline. We must then assume network issues are going to happen and deploy strategies to handle them; connection can drop due to a myriad of causes, such as a time-out event, where there was no interaction between the computers for too long or when an error in a data set that raises an exception, interrupts the connection. For our pipeline to be robust to failures it must handle network drops and reconnect automatically.

1.2.3 Transforming raw data into an efficient data structure

To assign unique identifiers to all cells, our indices have a hierarchical structure, inspired on both the structure of ALCATRAS devices and image size limitations. For any given experiment, each cell can be identified based on three levels: position, tile and cell label. The following list explains each one of them, and Figure 1.2 shows a graphical representation.

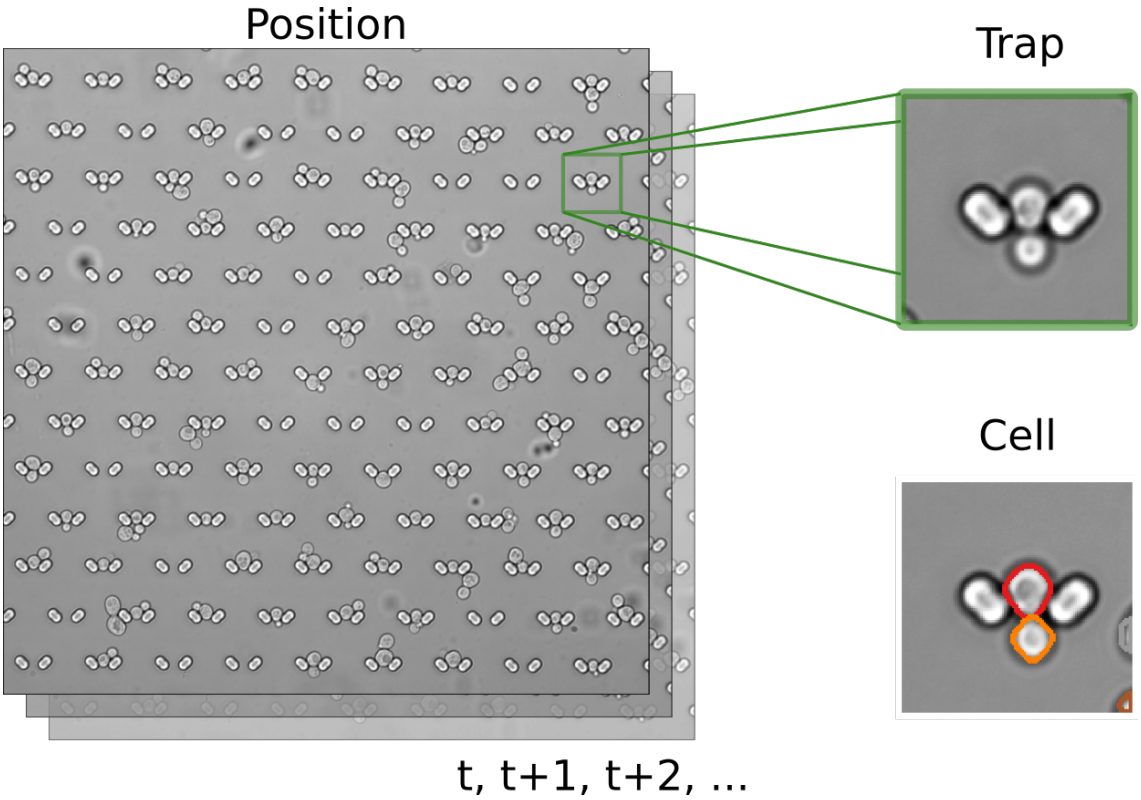

Figure 1.2: **Positions are composed of traps, which may contain zero or more cells.** Trap pillars fix cells in place, constraining the distance they can travel (left panel). In ALCATRAS devices each tile contains a single trap, because they are used to set the location of tiles. We can thus select a region surrounding that trap and trust that if a cell is trapped there we will image it (top-right panel). Generally more than one cell occupy a trap because they divide and produce daughters, but these are not always flushed away immediately. We thus define the last level as each cell outline (bottom-right panel). With these three levels, position, trap and (uniquely-labelled by trap) cell, we can pinpoint a specific cell anywhere and anytime within an experiment, regardless of how long they were within our imaging field.

- **Position**: The entire image taken by a camera, also referred to as a frame. It is unique for a given experiment.
- **Tile**: A region selected from an image. They are unique and enumerated from 0 to n, where n is the number of tiles for a given position For ALCATRAS devices, focusing on these regions also reduces the computational cost of all further computer vision analyses, as we trim about half the image while focusing in the cells that are trapped and their descendants.
- **Cell label**: A natural number identifying a given cell outline. It is unique for a given trap.

For any single experiment, the top level is position, defined as a frame in which we image a fixed region of our microfluidic device. Its size is determined by the camera resolution determines, its location is determined by the performer of the experiment via acquisition automation, but it must have a unique identifier. This uniqueness is necessary because it is the only value assigned during the experimental, thus out of our control during the processing pipeline. The number of positions in an experiment can vary from one to as many as the timing interval or device design allow. Their identifiers are a string indicating the position name, with a unique natural number at the end of the string to ensure uniquenes; for example, one position name could be "*wt_glucose_*001".

Given that our ALCATRAS system uses a tile-based system, it naturally produces indices: the *S. cerevisiae* trap pillars can fix individual cells in place, while flushing the surrounding cells. Thus the sections of the image with actually useful information are much smaller than the entirety of the image, and it is more efficient to analyse them separately instead. Their indices are integers ranging from zero to $n - 1$, where n is the number of tiles.

For the more general case of experiments without pillar traps, this second level is still useful to reduce the size of an image - making processing faster. It is equivalent to crop the regions where we expect cells to grow. Segmentation is possible on

raw images, but the complexity of models that apply operations over all contiguous pixels, such as convolutional neural networks, scales quadratically.

The main potential benefit of entire fields of vision is increasing the likelihood of finding cells that were flushed from one tile to another one downstream. Even in such case, this is not only a computationally expensive task, but it also requires completely overhauling our tracking strategy. Complex tracking algorithms would be necessary, such as the ones used to track motile cells (Ulicna et al., 2021); within our experimental setup, the ideal model or algorithm should consider that cells barely move most of the time, but every once in a while a tiny subset travels long distances. I do not consider it an objective worth pursuing, as the return on investment for solving such a task would require re-engineering the tracking methods to yield a marginal increase in the number of cells retained for a long time.

The last level differentiates cells within a single tile, with labels starting from one and that do not repeat in a single tile over the course of an experiment, even if a cell appeared for a single time point. We can then define every cell thriving - or dying - in any given experiment as a set of three values: The position name, which is a string, the tile identity, which is a natural number, and the cell label, which is a positive integer; for instance, as a hypothetical experiment, we could fetch a cell in position *wt_glucose_*001, tile 2 and cell label 5.

1.2.4 Selecting tiles and adjusting for stage drift

Existing algorithms identify traps using a manually-curated trap template, but this approach does not scale easily for varying experimental conditions and heterogeneity in yeast trap designs. These algorithms are susceptible to changes in the acquisition conditions. For previous setups we have to manually curate a trap for each combination of cameras, microfluidic device and, in some cases, set of bright field imaging parameters (Bakker et al., 2018).

We can use manually-curated trap templates, but these require additional processing before guaranteeing robust identification of traps. The template orientation may

not be the same as the image we fit it to. There are two options to choose from if we aim to use manually-curated traps: Data acquisition must be consistent in all future experiments, which is both impractical and inconvenient, because the person performing the experiment might not be aware of this. Alternatively, we can add an extra step where we find the best fit, increasing complexity and again requiring parameters.

During time lapse acquisition the microscope stage drifts over time when moving between consecutive frames (see Figure 1.3). The pipeline must account for this drift to keep our traps centred over the course of an experiment. Given that cells are static when between the pillars of a trap, it is of utmost importance to maintain the traps roughly in the same place in the image and, ideally, have the centre of our pillars be the centre of our region of interest.

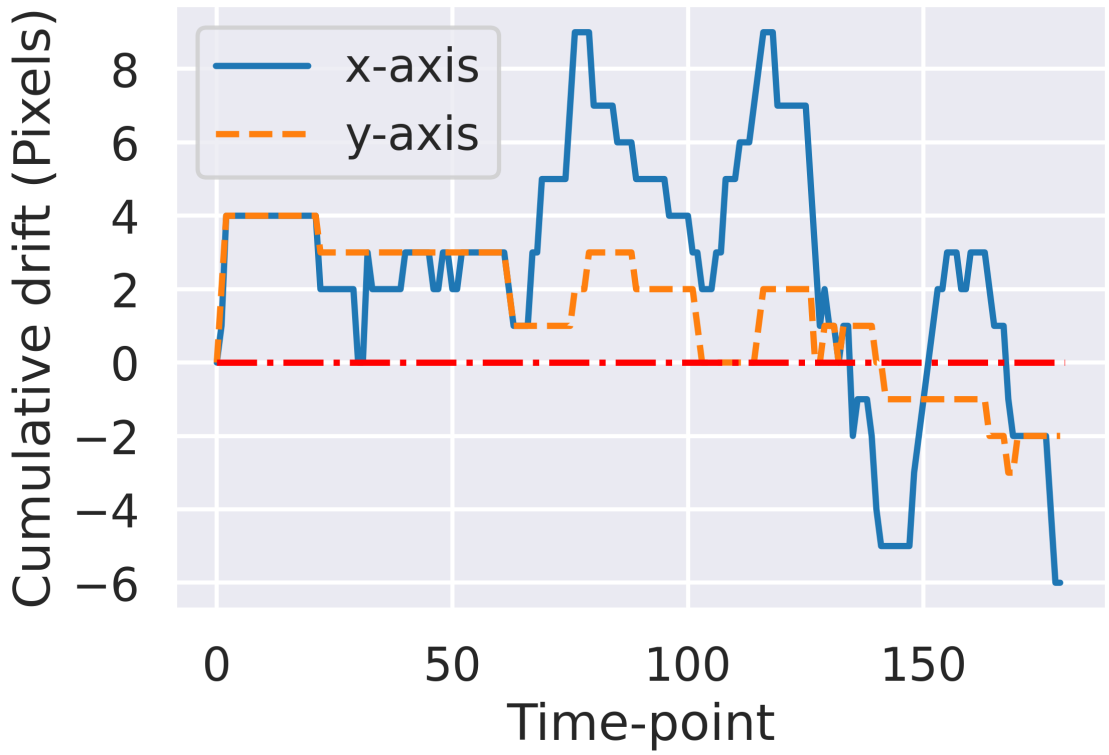

Figure 1.3: **During image acquisition stochastic drift occurs in both x and y directions.** Example of stage drift in a single position. We show the accumulation of drift over time for each axis. The line where $y = 0$ shows no drift. The choice of time points instead of time is intentional, as drift only occurs when the stage moves and not when it is waiting.

Our pipeline must also handle tiles that drift beyond the frame using a consistent approach. It is uncertain how far the stage will drift at the start of the experiment, and it is possible that tiles with cells end up beyond the frame edge. This poses a problem because we select tiles at the start of the experiment: adding tiles during the experiment increases architectural complexity of our software while providing a low amount of new data.

1.2.5 Segment and track regions of interest

The core of any such pipeline is a toolset of segmentation models or algorithms that identify outlines inside Region Of Interests (ROIs), as well as track them over time. Segmentation software converts information from images where outlines are visible, such as bright-field or DIC images, into labelled regions, such as cell outlines or organelles inside cells.

If these ROIs identify living cells, we may also want to include information on temporal relationships between them, such as relationship between mothers and buds. There are multiple algorithms and deep learning techniques for image segmentation, we use BABY to feed our image acquisition data structures into a segmentation software (Pietsch et al., 2022).

Additional considerations crop up when we want to generalise the analysis pipeline. Multiple sources and methods of processing images must be accounted for. A user may request images from a remote server or local files and even feed the new timepoint as the experiment runs (i.e., as a "live" experiment), with no knowledge on the length of an experiment; the ideal tool would aim to account for these cases. To further complicate matters, accidental or intentional interruptions are to be expected during processing.

Supporting the use of local files also increases the likelihood of other scientists using such software to analyse their own locally-stored data. While highly discouraged for research groups that generate large amounts of data, mainly due to how dependant it

is the organisation standards of an individual - a unique point of failure - it provides a low barrier of entry for testing how the software fares with a user's data.

1.2.6 Extracting information from cell outlines and fluorescence images

The next aim would be to quantify phenomena occurring inside ROIs, images, masks and labels (in this context, unique identifiers for each mask). We define this as the task of extracting data of individual cells from microscopy images via cell masks. We select the area corresponding to a cell (or any other region of interest, for that matter) and extract the pixels it contains to then apply a metric that offers information on that region of the image as a measure of fluorescence inhomogeneity. For instance, we can obtain the brightest five pixels inside a given cell divided by that cell's median fluorescence by using the fluorescence image and that cell's outline.

There are multiple available sources for both images and mask-label sets. We can fetch images from an OMERO server or from a local file, if we are working with local images. We may fetch masks and labels from a local file or, alternatively, pass the in-memory image to a "live" method or class instance that applies segmentation and extraction routines.

1.2.7 Use biological knowledge to improve results using heuristics

Time-independent neural networks are not sufficient to provide an accurate estimation of single-cell trajectories when using semantic segmentation - that is, the assignment of a categorical label to each pixel in an image (Guo et al., 2020). Recently, *Aspert et al* demonstrated that neural networks on their own are (at the moment) not enough to accurately reconstruct replicative life-spans from microscopy images (Aspert et al., 2022). In order to increase the accuracy of their predictions, the outputs from the segmentation network served as input to two different sub-pipelines: The first is a Long Short-Term Memory network, neural network architecture that accumulates information of recently processed images. The second one is a post-

processing algorithm that also helped clean the results to achieve more accurate birth predictions.

While BABY's time-agnosticism increases processing speed and simplifies training - as we are only required to provide images and targets to train the semantic neural network at its core - it can lead to segmentation limitations. We cannot correct for unsegmented cells, as it would require an additional segmentation algorithm, but we can fix tracking errors when BABY misidentifies a cell as new, when in reality it was present in the previous time point.

Using the term laid by *Ulicna et al.* (Ulicna et al., 2021) as a base, we define a tracklet as a list of numbers that represent a unique cell over the course of an experiment; for the sake of simplicity, let us assume that a tracklet represents one cell's volume over the course of its presence in the experiment. Following that definition, its edges are at the first time-point in which the cell appeared and in the time-point before it vanishes forever (within the experiment) - when the flow, for example, sweeps away the cell through a microfluidic device. This tracklet is not necessarily continuous, as cells can disappear - when the segmentation software fails to find them - and reappear a handful of time points later; a useful feature trackers could have is the capacity for linking outlines to a cell found in further in the past than a single step.

Merging tracklets has additional implications for data sets with lineage information, mainly because joining tracklets forces us to adjust their lineage relationships as well. We must ensure merge events are always applied, as merge events potentially correct lineage relationships. When there is a merge event involving multiple already-assigned daughters, we shall use a consistent conflict-resolution criteria for these cases.

Due to noise arising from a variety of sources, additional signal filtering is a commonplace necessity when processing data from microscopy videos. A standard filter, for instance, is the Savitsky-Golay filter; it is a digital filter that smooths data to increase accuracy without distorting the tendency. It is particularly useful when

processing fluorescence signals arising from a small set of pixels, such as proteins tagged with fluorescent probes that localise in an organelle.

1.2.8 Select tracklets

The selection of tracklets greatly influences the results. For instance, when dealing with cell outlines, the cells we are imaging often have different replicative or chronological age, as well as overall regulatory state - the collection of intracellular metabolic and signalling conditions at a given moment. Some might not even be real cells at all, thus this step is also necessary to filter-out dead cells or falsely-identified outlines. Lastly, it makes our processed data set more consistent by defining a set of (a feature is an individual measurable characteristic or attribute of the data used to make predictions in a machine learning model) as our ground truth on how cells that are alive are expected to behave (e.g., dead or fake cells do not change in volume when growing in glucose).

The main challenge when selecting a set of tracklets is maintaining awareness of how a given cell selection approach biases the final results. Averaging over all cells at a specific time point is not the same as measuring the daughter cells' growth rates or just the mothers'. Performing all analyses on multiple sets of cells becomes impractical due to all available combinations of cell sets and analyses. We therefore have to select a consistent set of criteria to select a subset of all objects.

To follow single-cell dynamics we can use a wide variety of criteria. The most straightforward one is cells that are present in an experiment for the majority of the experiment, for we can only be certain that those cells have experienced all changes in environmental conditions. Sometimes we might consider selecting the cells identified as their daughters, as they are important indicators of the mothers' state and growth. We must pick the default way of filtering cells and then add a simple interface to basic filters: mothers, daughters and families (mothers and daughters).

Lineage and presence-based selection is not enough, as some data sets require additional methods for selecting cells. For example, applying a threshold on minimum time-averaged growth rate is an efficient method to clean "fake" cells (e.g., false-positive cell outlines) or dead cells that are of no use to the analysis. The risk of this approach is biasing the cell selection against slow growing cells, an issue of particular concern in experiments where cells go through a period of starvation. Additional alternative methods are selecting cells based on the intensity of their fluorescence, either over the course of the experiment or at a particular stage (i.e. environment, such as a specific carbon source concentration).

1.2.9 Aggregating tools into modules to maximise software sustainability

After identifying all challenges the next step is to define a set of architectural design principles, so each individual component can tackle the previously-defined challenges as tasks. We must delimit these by defining "steps" (or stages) with precisely-defined purposes. Additionally, inputs and outputs must be well-defined, or we risk running into data-structure inconsistencies during posterior analyses.

Once we split the design into multiple independent components we have to further separate components based on their interactions, connecting the low and high-level tools. For example, we need the objects that fetch images from a server to hand them over to a higher-level component that uses that information. The interactions between these different classes determines parts of our software architecture, but in the end we must still make decisions about how they talk to each other. This requires us to define standardised Input/Output methods inside the low-level classes that directly interact with the server or results file.

To achieve high modularity and component independence we must group the previously defined components into independent entities. There are multiple advantages for splitting any complex software into sub-modules: First, it reduces class interdependency, objects thus make weak assumptions on what to expect from the other

components they depend on. If our objects strongly depend on each other's properties they become "stiff" (i.e., changing a method or property of a class breaks methods in others, so we can't change them too much without investing time). Second, some sub-modules have different purposes, for example, some users might not need segmentation tools, as segmentation occurred on a dedicated server, and they just want to analyse the data and produce figures. The main task for developing software architectures is finding the equilibrium, ensuring individual components are not massive while also avoiding fracturing components excessively, because excessive partitioning requires considerably more time and effort to maintain.

If we are to tackle a complex task, such as the automated processing of microscopy videos, all these considerations must be accounted for. Sensible software design - including, of course, scientific software – determines its future re-usability, and thus how useful it is to further scientific knowledge.

1.3 CTP synthase is a model to study protein aggregation

CTP synthase is a well-known catalytic enzyme that forms protein aggregates, linking gene regulation and growth to imageable phenomena - due to its localisation in elongated protein aggregates, a prime subject to test ALIBY (Chapter 3) and yield cell decision-making insight (Chapter 4). CTP synthase catalyses the final step of CTP biosynthesis (Equation 1.5).

$$\mathrm{ATP} + \mathrm{UTP} + \mathrm{glutamine} \xrightarrow{\mathrm{CTPS}} \mathrm{ADP} + \mathrm{P}_i + \mathrm{CTP} + \mathrm{glutamate} \tag{1.5}$$

The chemicals are:

- ATP (adenine triphosphate) is a high-energy molecule, main energy carrier in cells.
- UTP (Uridine Triphosphate (UTP)) is a Uridine Triphosphate.

- glutamine (glutamine) is a amino acid used in protein biosynthesis, key player in multiple biochemical functions.
- ADP (adenine diphosphate) is a high-energy molecule, main energy carrier in cells (in its low energy state).
- P_i(phosphate) is a inorganic phosphate, component of high-energy molecules.
- glutamine is a amino acid used in protein biosynthesis, key player in multiple biochemical functions.

Its biochemistry has been extensively studied. Is an essential enzyme due to its role in ribonucleotide production. CTP synthases are normally present as a dimer that oligomerise to form a tetramer. They are present in all domains of life, but its activity and regulation in both human and *Drosophila* works through a different mechanism when compared to yeast.

CTP synthase is involved in three main biosynthesis pathways: RNA, DNA and phospholipids (Figure 1.4). The production of CTP is not only a rate-limiting step for DNA duplication, but is also involved in RNA biosynthesis and the Kennedy and CDP-PAG phospholipid synthesis pathways (Yang et al., 2002). Research is focused on its DNA duplication pathway, but CTP synthase is also important for the regulation in the ratio neutral lipids and phospholipids. Protein kinases A and C phosphorilate it.

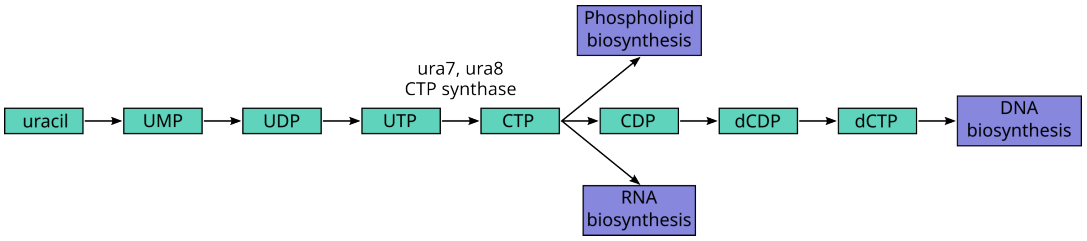

Figure 1.4: CTP synthase is a rate-limiting reaction for RNA, DNA and phospholipids biosynthesis. The blue, red and yellow colours represent different pathways in which CTP synthase is involved.

This protein is a model for aggregate assembly-driven regulation. Excessive aggregation of CTP synthase may lead to slower growth, and this aggregation reduces enzyme activity (Lynch et al., 2020). Filament assembly is a type of protein ag-

gregation in which the resulting polymer expands linearly towards opposite sides, it does not necessarily protect it from degradation. Filament assembly is predicted to happen as a two-step process: First individual filaments appear, and then these bundle together in thicker filaments. Pictures of different degrees of aggregation are visible in Figure 1.5.

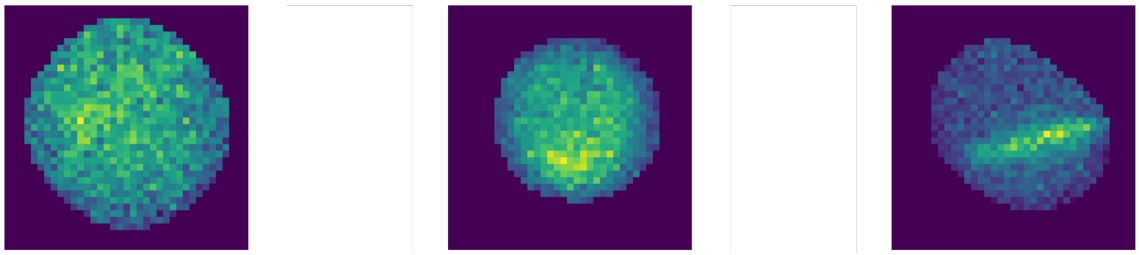

Figure 1.5: Three *Saccharomyces cerevisiae* cells are shown sorted by their degree of aggregation. On the left we see the dissolved state of CTP synthase, in the middle some aggregation appears to occur at medium levels. In the right image full aggregation of a filament becomes visible, accompanied by a decrease in overall intracellular fluorescence; most CTPS cluster together to form the filament. To visualise aggregation I used a strain whose *ura8* protein was tagged with Green Fluorescent Protein (GFP), and then imaged the cells in media with and without glucose.

CTP synthase can be produced by any of two paralog genes: *ura7* and *ura8*. The differences between these two isoforms are well-known: their function and regulation is almost identical. Both are maximally expressed during the exponential phase of growth (Nadkarni et al., 1995). Cells undergoing exponential growth have twice as many *ura7* mRNAs than *ura8*. They share 78% identical residues and deletion of either results in slow growth (Ozier-Kalogeropoulos et al., 1994). Additionally, they aggregate into supra-molecular structures in response to stress, and co-localise when forming filaments (Noree et al., 2010a; Shen et al., 2016). Previous work explored aggregation or dissolution speed in isolation, with no information on cells before, during, and after change in the environment (Franzmann & Alberti, 2019; Yang et al., 2002). Full control of environmental conditions coupled with long-term tracking is required to correlate cell behaviour across sequential changes external conditions.

1.3.1 Cytosolic pH acidification is necessary but not sufficient for CTP synthase filaments to assemble

Assembly of CTP synthase filaments depends on both pH and their substrates being present to form aggregates. Cytosolic availability of CTP is necessary for filament assembly to occur (Hansen et al., 2021; Petrovska et al., 2014). Due to the filament assembly upon carbon starvation depending on both CTP synthase and pH, its *in vivo* dynamics depend on both the media in which cells grow and for how long they have been starving. All the existing articles in which numbers of cells with filaments or foci during saturation growth phase are (manually) counted come to different values, both in normal growth conditions (See Table 1.1) and under cytosolic pH control (See Table 1.2).

Existing predictions on the population fraction that form aggregates are not fully coherent. There are two conditions under which population heterogeneity data is usually obtained, each with their own purposes: under "normal" starvation conditions - cells are starving in media in a fixed pH - and when cytosolic pH values are controlled by making the cell membrane permeable to protons. Starving in an environment with a pH between four and seven represents more accurately the environmental conditions in which yeast cells normally live. In the second experimental setup cytosolic pH can be adjusted manually.

Table 1.1: Percentage of aggregating cells in *in vivo* conditions. Notes indicate environmental condition and imaging time if provided. Rows indicate the tagged and columns indicate the source of the estimation: (Noree et al., 2010a, 2014; Petrovska et al., 2014). Media in all cases was (), notes indicate experimental conditions and starvation time if specified in the original work. Confidence interval shown if available.

Data source	*ura7* (%)	*ura8* (%)}	Notes
Noree (2014)	26.03 ± 1.72		Between 250 and 300 cells
Noree (2010)	15.20	16.40	More than 200 cells; buffer media was used
Hansen (2021)	89 ± 1.3	53 ± 7	4 hours of incubation

The quantitative inconsistency in the available literature regarding fraction of yeast cells that form protein aggregates hints at a complex behaviour at play. These

Table 1.2: Percentage of aggregating cells in membrane-permeable cells when using pH-calibrated buffer. Columns indicate the tagged enzyme and the source (Hansen et al., 2021; Petrovska et al., 2014). The media used was for all conditions, and notes indicate the time between the start of starvation and imaging.

pH (A.U.)	*ura8* (Petrovska)	*ura7* (Hansen)	*ura8* (Hansen)
5.0	100		
6.0	35	93 ± 3.8	50 ± 10
6.5		77 ± 5.3	34 ± 6.8
7.0	1	18 ± 7	7.2 ± 5.3
7.4		1.7 ± 1.4	0
Notes	2 hrs	4 hrs	4 hrs

options can further be explored by studying dynamic environments, and through a more stringent control of environmental conditions. While these discrepancies may result from the usage of different experimental protocols or specific strains, the dynamic nature of intracellular regulation could generate a variety of phenotypes from a single genotype.

1.4 Summary

First, I summarise what - in my opinion - are the most useful computational frameworks and tools for more open, efficient and reproducible science. Then, I explain how we solved the tracking issue for our use-case and developed a robust and modular tracking system. Afterwards, I use tools that our lab has recently developed as a base to design and implement automated end-to-end analysis pipeline for microscopy videos. Finally, using all the tools both I and the Swain Lab have developed over the past few years I show how we can study formation of protein aggregates and its correlation to cell growth and division.

2 A hitchhiker's guide to modern scientific software development

2.1 Introduction

Bio-sciences are producing more data than ever before, both due to the increase in availability and reduction in price of sequencing technologies, as well as automation infrastructure and tools becoming more accessible for scientists who study biology and other related fields. In principle more data should help increase our understanding of biology, but to do so we must first face a set of new challenges that this increase in data brought along.

I have set to highlight and describe tools that improve software development efficiency and sustainability. Based on trial-and-error of tools aimed to solve the practical challenges we face when studying cell decision-making using imaging time series. Not all tools are necessary for all use-case scenarios, but most of them may come of use to a wide variety of scientific software developers - or even to general software developers.

As a general rule, I favour tools that are free and open-source - FOSS, following the Free Software Foundation definition (Stallman, 2009), explained by *Fortunato and Galassi* (Fortunato & Galassi, 2021) - or, in some cases, the ones more widely used and thus likely to have long-term support. Open-source code, models and data are easier to share, maintain and, arguably, licence due to the rise of hosting services and community-driven efforts to use and keep them alive when they are a vital component of a development or scientific ecosystem.

This is not a comprehensive listing of all available tools, but a basic guide on challenges and the technological solutions we have chosen to use, as well as other popular alternatives where applicable.

Tools may come handy in a case-specific basis. A set of them might be unnecessary for simple tasks, but after a certain degree of complexity - either in analysis or in the size and structure of experimental data - new challenges arise and if not dealt with they will consume scientists' time and effort. These tools aim to reduce the burden on the scientific software developer while increasing the likelihood that the software will remain relevant over time; in other words, increasing development efficiency and sustainability.

For my selection, all software is available at no cost and most have at least one FOSS version. Due to the fast pace at which software evolves, the specific tools are not guaranteed to exist indefinitely, but they usually tackle common challenges. Even if they become unmaintained in the future, a replacement will arise because the original problem that motivated its development is unlikely to change.

I do not include pipeline software such as *snakemake* (Köster & Rahmann, 2012) despite its usefulness for bioinformatic pipelines dealing with (relatively) small data - such as text or uni-dimensional numerical signals . Storing intermediate steps for data such as DNA sequences or -omics datasets may be viable, but that is not the case for others, either because of the amount or type of data (e.g., multidimensional image time series). This list is not extensive and it is based on the experience of developing of a code-base of more than fifteen thousand lines of Python code, point at which the interactions between its components make maintenance challenging.

Table 2.1 shows challenges and their fragmentation into smaller sub-fragments. It also organises the tools explained further in the chapter into potential solution to each of the challenges.

Table 2.1: Summary of general challenges of scientific software development, their breakdown into sub-challenges and the tools that we review in this chapter and I have deemed helpful to simplify or solve these challenges.

Goal	Challenges	Available tools
Advanced text editing	Auto-completion and checks	LSP (VIM/Emacs/VS-Code), Pycharm, Rstudio
	Debugging	PDB (VIM/Emacs), VS-Code plugins, Pycharm, Rstudio
Reproducible environment	Development dependencies	Poetry
	Isolation and management	pyenv, venv, Anaconda
	System-wide dependencies	containers, Anaconda, system package managers
	Python-specific dependencies	Poetry, pip, pipenv, PDM
Code quality	Homogenising style	black
	Finding inconsistencies	flake8, mypy
Hosting and management	Integrate contributions	Codeberg, GitLab, Github
	Manage issues/features	Codeberg, GitLab, Github
Bug mitigation/control	Testing and code coverage	pytest, UnitTest, Coverage
	Automation	pre-commit hooks, containers, GitLab CI, Github Workflows, Jenkins
Publishing	Packaging and release	Poetry, twine
	Managing local server	TLJH, containers
Documentation	Hosting	ReadTheDocs, self-hosting
	Automation	containers, Sphinx, ReadTheDocs
Data availability	Data sets and training	Kaggle
	Models	HuggingFace, Kaggle
	Notebooks	Kaggle, Git hosting services

2.2 A literature review of software

I have tried these tools and chosen the set that better suits our workflow. Most of these tools are available in all operative systems, but we focus on UNIX-like environments, which nowadays are accessible from most computers. While both R and Python are widely-used languages with their own community each, we add a section for Python-specific tools at the end because that is our language of choice for development and machine/deep learning, and thus we have tried out more tools. Even then, a large fraction of the tools I include are language-agnostic.

2.2.1 Integrated Development Environments are at the core of efficient software development

A common saying in software engineering is that using the right tool for the job is half the work. To write computational programs a text editor is generally a prerequisite. While there is a sea of options with winners and losers that change in popularity over the years, a small amount of them have either consistent high popularity or test of time. The general-purpose Integrated Development Environments (IDEs) are:

- *Vim* is a modal editor - that is, its paradigm follows different states each with their own way to interact with text - that includes its own scripting language for users to develop extensions. *NeoVim* is a newer implementation that adds extensibility.
- *Emacs* is a highly customisable text editor (Stallman, 1981), there are multiple layers that provide an out-of-the-box experience and modal editing style ("Doom Emacs," n.d., n.d.).
- *VSCode* is the most popular editor in 2021, it is open-source and most practical functionality arises from installing extensions.
- *RStudio* is the most popular R IDE, it is free and open source (RACINE, 2012) and has multiple useful features thanks to its consistent ecosystem.

- *PyCharm* is a widely-used Python IDE. In addition, it can process other languages as well. There are two versions: community, *edu* and professional; the first two are open source, but some features are only available in the proprietary professional version.

Emacs and (Neo)Vim can use the GNU Debugger - GDB, a debugger is a tool used to stop a running program at any point to perform a thorough analysis of its state; it is the essential tool for software development to identify regions of code that fail to provide the expected functionality (bugs). Debuggers are available to use in the command line as well. The other three Integrated Development environments (IDEs) either include their own or complement with other some open-source debugging tools. Additionally, all of them have Language Server Protocol (LSP) implementations, which runs a server in the background and provides features such as auto-complete or find all references, which improves coding speed and efficiency.

2.2.2 Version control software improves collaboration and resistance to errors

A version control system is software that facilitates storing our software and keep track of its changes, regardless of what these changes are or who makes them. It facilitates the incremental evolution of software, collaboration between multiple developers in parallel and works as a fail-safe against bugs. We chose git over other version control systems due to its ubiquity in the open source software community (Blischak et al., 2016). It is arguably the dominant version control system for scientific software. The publicly-available git-based services are:

- *GitHub* is the hegemonic - although closed-source - provider of version control services.
- *GitLab* is an alternative version control software with multiple out-of-the-box features. There are two versions: the community edition, which is open-source and anyone has access to the code and may host it in their servers, and the

premium edition (which is the one publicly available as a service), although it is closed-source, but contains more features.

- *Gitea* is a fully open-source git client that anyone can host, it contains fewer out-of-the-box features than *GitLab*, but it is possible to reproduce most of those using other open-source software. The reduced overhead leads to lower consumption of computational resources. The free public non-profit instance *Codeberg* is the main public alternative based on it.

2.2.3 Pre-commit helps to spot errors before adding them to the code-base

The fastest way to identify simple issues is before committing them to the version control system. Hooks are the main solution for preemptive checks; they are computational processes triggered by some change in a system state or other processes. *Pre-commit* hooks manage the installation of these tools and run tests in the background before committing changes to the local repository (*Pre-Commit/Pre-Commit*, 2022).

Pre-commit hooks identify and may fix problems that tend to be trivial but are easy to slip between the cracks when editing a code-base. Using them limits the risk of contaminating the code-base and thus reduces the likelihood of typos or trivial mistakes percolating into other developers' setups.

Some of the errors that pre-commit can catch are automatically fixed, such as formatting or the order of module imports. Others, such as breaking conventions, require manual input, but these are useful to identify sections of the code that may be in need of refactoring. Among all these tools it has one of the best cost-to-yield ratio.

2.2.4 Continuous Integration automates checks and deployment

Continuous Integration (CI) is one of the continuous practices that accelerate software development without sacrificing quality (Shahin et al., 2017). It consists in the

automation of testing the integration of multiple components of a software package. When a change happens in the code-base, a new and independent environment is built, then tests and checks are run in this isolated environment to ensure the software still fulfils its tasks after the changes. Continuous Integration reduces the time developers spend debugging software, as it reduces the time it takes to identify new bugs or regressions (recurrence of previously fixed bugs).

Continuous Integration has synergism with atomic incremental changes to the code, that is, small changes that alter a very limited set of components. By running frequent automated tests it facilitates spotting sections of code that cause regressions or brand-new bugs into the code base. Multiple tools can implement CI, we consider two of the most popular ones:

- *GitLab* includes CI tools, and runners programs - processes that run the tests in any computer anywhere with internet access.
- *Jenkins* is the most popular stand-alone continuous integration tool, with a collection of plugins to automate tasks (Minas et al., 2017).

2.2.5 Literate programming provides reproducible pipelines and tutorials

Literate programming represents a change in the way of constructing programs: Instead of writing to instruct a computer what to do, it focuses on explaining human beings what we want the computer to do (Knuth, 1984). In such format the author weaves a narrative between blocks of code, both improving readability and reproducibility (Kery et al., 2018), the latter as defined by *Gundersen* (Gundersen, 2021). It is not the best option for "active" software development though - that is, iterative modifications of snippets and their frequent recurrent evaluation. Multiple evaluations produce multiple independent results, becoming difficult to follow or display within its sequential structure. It can quickly devolve into a tangled web of interconnected blocks of code that may not work when re-run from the start.

Literate programming may be of use, however, to provide the backbone of a custom analysis pipeline or to explain how to use some software functionality. There are multiple tools to use them:

- *Jupyter* is a Python-derived framework for literate programming and data science development tools (Kluyver et al., 2016). It is widely used and can work with multiple languages in independent notebooks.
- *R markdown* is R's literate programming approach, commonly used within R-lang's ecosystem, some other languages supported.
- *Org-mode* is a markdown language tightly linked to *Emacs*, but available in *VIM* as well. It allows for multi-language literate programming (Schulte et al., 2012), resulting in multiple languages being usable within a single notebook. Its main downside is being highly dependant on the *Emacs* ecosystem.

2.2.6 The Littlest Jupyter Notebook facilitates intragroup deployment of environment for data analysis

During the first stages of software development it is usually tested by a small group of users within the research group or collaborators. Using local installations in personal computers may work for programs that run on small amounts of data, but for larger data sets (i.e. larger than tens or hundreds of GigaBytes) it is more practical to relay these processes to a bigger and more powerful server. The downside of this is that research groups depend on the IT infrastructure of their institute or centre, onto which they exert only limited control.

The Littlest Jupyter Notebook TLJH is a software that combines multiple tools to provide a consistent virtual environment in a single computer (usually one with high technical specifications) for multiple users. Through the combination of Jupyter notebooks and additional system and networking tools it provides a Jupyter-based computing environment (Ochkov et al., 2022). An important side-note of TLJH is being careful with the security setup. The details are beyond the scope of this

work, but it is worth reading attentively the documentation section on secure multi-account management.

2.2.7 *ReadTheDocs* hosts documentation and provides tools for updating it upon changes

Clear and informative documentation is of utmost important. Since most of the open-science user base is the scientific community, wide adoption depends on the users - potential collaborators or testers - being aware of the available functionality and how to best use it. While some developers may be comfortable reading the source code (provided it is well-documented), having easy-to-read information of classes or routines on a website lowers the barrier of entry for new users.

ReadTheDocs provides free hosting and enough computing power to produce and publish documentation. In addition its free hosting of documentation, it provides tools to update it whenever changes occur in a publicly software repository.

For scientific software, documentation may also include different literate programming notebooks as examples. We can convert example notebooks or scripts within the code-base into markdown and these automatically read as usage examples.

2.2.8 Publishing notebooks, data sets and models

For reproducibility of models and results it is important to bundle trained models and their source data together. While models may have a small footprint, the size of curated training data sets varies considerably between fields and groups. Besides hosting both of these in a group-controlled server, there are public alternatives. The only caveat is that none of these are open-source. Regardless, this is in my opinion the most pragmatic way to make results and tool available to the public.

- *Kaggle* is an online community for data scientists. On their website you can share notebooks as examples, enough space available to host most machine and deep-learning models.

- *Google Colab*: Depends on storage available on the Google Drive service. For computing time usage efficiency, there is a risk of automatic disconnection upon inactivity, thus risking data loss. On the other hand, it has higher (Random Access) memory limits than *Kaggle*.
- *Face hugging*: Widely-popular database for models. You can access, host and share models. Allows for group and semantic search of models. Unlike the previous two, it does not provide computing resources, only hosting of data and code. It does provide more space for data sets than the previous two.

2.2.9 Containers guarantee environment reproducibility and fast deployment

Containerisation is a software developed to automatically deploy a consistent environment for software trials, testing or to have a consistent development environment. It is similar to a virtual machine that shares some internals with the host computer, making it smaller and more efficient than a full-blown virtualisation. They are usually used to perform Continuous Integration, but can also provide a reproducible virtual environment for data analysis.

- *Docker* is a popular containerisation software. The Graphical User Interface requires a licence, but when used from command line this is not necessary.
- *Podman* is an almost-perfect replacement for *Docker*. It requires fewer system permissions, and thus it is easier to set up in UNIX-like systems (e.g., MacOS, Linux). Its main advantage is being open-source and working almost exactly the same way as *Docker*.

2.3 The Python ecosystem

Python is one of the most widely-used and fastest-growing programming languages (Srinath, 2017). It has mature and documented data science and machine and deep

learning libraries; it is also free and open-source, making it one of the top choices for the open-science tools development, regarding both data acquisition or analysis.

2.3.1 Virtual environment and dependency management

A common frustration for software developers occurs when setting up the development environment is a problem by itself. Analogous to performing a biological experiment, keeping the conditions surrounding individual tests is essential for the sake of reproducibility. There are two main processes required to ensure consistent development: virtual environment and dependency resolution.

Virtual environments are the surrounding tools that provide a basic set of tools for development, allowing for isolated environments (Figure 2.1). In the case of Python the version and its location embody the basic virtual environment; after setting those two things it is technically possible to develop python packages. Some libraries (or modules, in the specific case of Python) require additional tools from the host environment, but most Python virtual environments do not deal with OS-wide tools.

Two options that solve the virtual environment task are:

- *venv*: Module from the Python standard library - tools always included with a Python installation - to create lightweight virtual environments (Van Rossum & Drake, 2009). It is possible to manage virtual environments using only this, but it can be cumbersome and more time-consuming than the alternatives.
- *pyenv* (and *pyenv-virtualenv*, its add-on for virtual environment management) is available in UNIX-like operating systems (*Simple Python Version Management*, 2022). There is also an installer for Windows (*Pyenv for Windows*, 2022). In my opinion this hits the right notes between simplicity and functionality, even if it takes slightly more effort to install for Windows computers. It is a mature project, widely used due to it focusing on a single problem: Swiftly switching between multiple python instances installed in the host system, regardless of folder structure or location.

As a complement to virtual environment management, dependency managers aim to solve the problem of how can we ensure consistent builds of a project, but focus on using the versions of multiple libraries that are compatible with each other - data science software development happens at a fast pace, so versions may change. While it is possible to install most packages with just the package installer for Python (*pip*), it does not guarantee inter-module compatibility nor reproducibility between different computers, even when recording the specific versions of all software.

There are multiple virtual environment managers that also include dependency solving tools:

- *Anaconda*: A suite of development tools widely used which includes *conda*, a virtual and dependency management framework (*Conda/Conda*, 2022). It is widely used and robust, but in most use-cases it might add unnecessary tools. It contains a graphical user interfaces to open its available tools, which is useful for users preferring such interface over the command line. It contains reproducible environments and a graphical interface, but it tends to use more disk space. Dependency solving usually takes longer than the alternatives.

These three options are lighter than *conda* and use lock-files (a file that fixes packages into specific versions to produce deterministic builds).

- *poetry* aims to tackle multiple tasks for Python development in addition to virtual environment management (*Poetry*, 2022). It can manage virtual environments (or delegate that to another tool), solve Python dependencies, and publish packages to p ublic repositories, a feature explained in further sections.
- *Pipenv* Alternative that combines *pip* and *venv*, developed by the Python Packaging Authority. It is compatible with other virtual environment managers and places its focus on supporting Windows first.
- Python Development Master (*PDM*) is a younger dependency manager that does not use virtual environments at all (*PDM*, 2022). Instead, it contains all libraries necessary for a given project inside it, following a recent Python Enhancement Proposal.

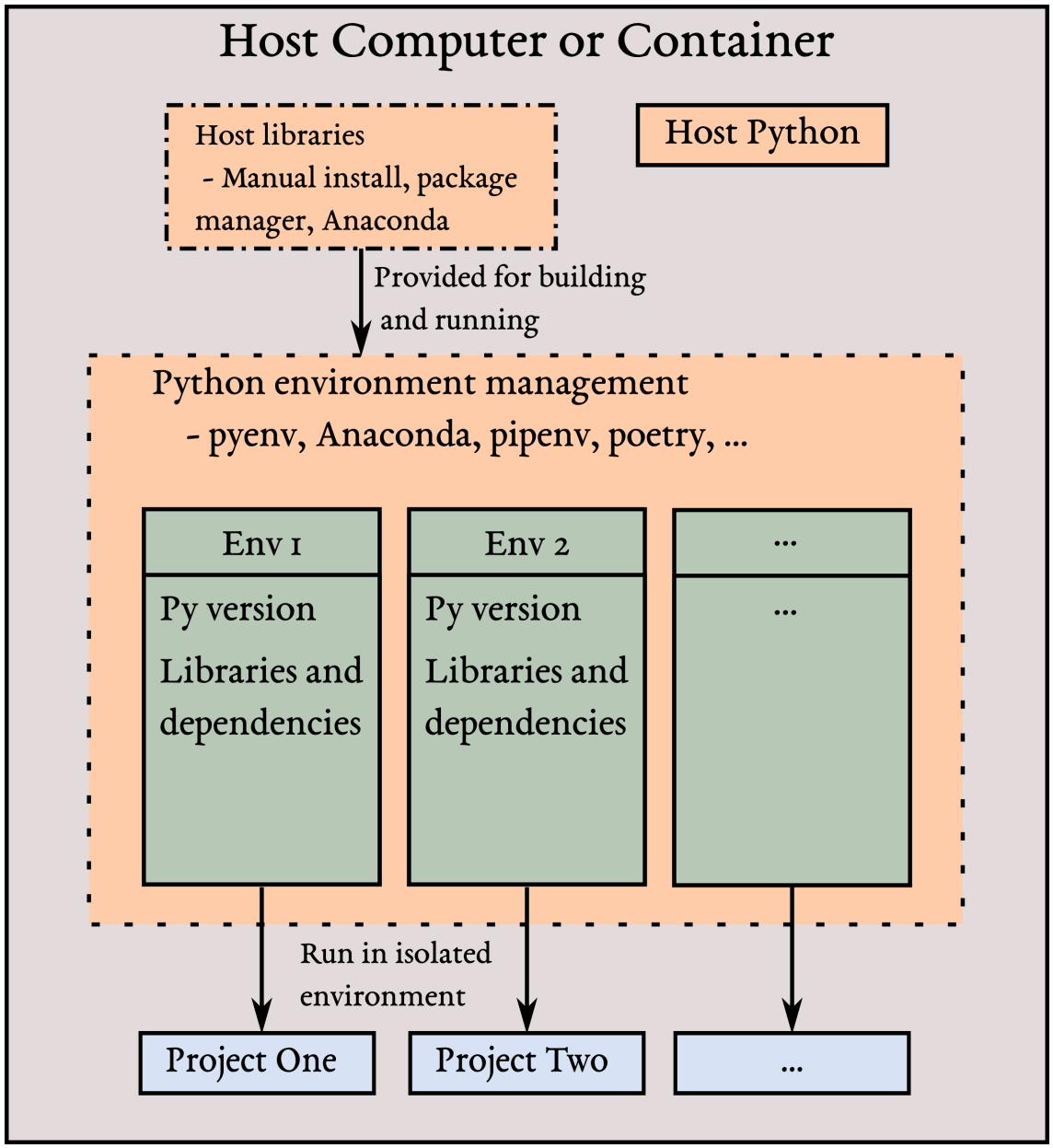

Figure 2.1: Diagram showing the structure of generic development isolation. The host system contains a *Python* environment management tool, independently from its own system-wide installation (in the case of UNIX-like systems). To build environments some basic libraries are needed (e.g., C compilation libraries); thus the environment management tool either contains (Anaconda) or retrieves these from the host computer. Once an environment is set with a given *Python* version we can install our dependencies, ideally aided by a dependency management tool. At last, we can use an environment to test our software or scripts, keeping our host system clean and each one of our environments independent from each other. Crucially, we leave our system-wide *Python* version untouched.

2.3.2 Code homogenisation and structure

Consistent code is easier to read and maintain. Removing personal styles and being strict with code requires more effort from the developers but in return facilitates collaboration within and between developing teams.

- *black* formats the code consistently. It facilitates identifying code changes when using version control systems and saves mental effort making style decisions (*Psf/Black*, 2022).
- *flake8* enforces (in a lenient way) Python style guidelines. It only warns of changes, as there are multiple ways to fix these and thus the fix cannot be automatic(*PyCQA/Flake8*, 2022).
- *isort* automatically organises the imports to other libraries to follow the Python convention. This is to differentiate python standard libraries from third-party ones and also elements from the current project(*PyCQA/Isort*, 2022).
- *mypy* performs analysis of data type., Setting types in Python is not a strict requirement for code to run, but it helps find logical errors during development (*GitHub - Python/Mypy: Optional Static Typing for Python*, n.d.).
- *Darglint* checks that the types defined in functions match to their documentation strings - usually called *docstrings*. It is particularly useful when coupled with automatic documentation, because the documentation strings are generally used to generate software documentation (Reilly, 2022).

2.3.3 Code testing and test coverage

Testing, alongside documentation, is one of the main drivers for sustainability of non-trivial software. As the code-base of a given project grows, the likelihood of new features or refactoring code (restructuring existing code without changing its function or interfaces) introducing bugs increases exponentially with the code-base size. Tests check if new changes break normal functionality, and, particularly when automated, are a time and effort-saving technique.

Python software testing usually boils down to one of two flavours:

- *UnitTest* is a testing suite part of the Python standard library (Van Rossum & Drake, 2009). While technically the official method and widely documented, it is not the most commonly used one.
- *pytest* is a widely used third-party testing framework for python code. It tends to require less code than *UnitTest*, because it allows for component recycling: setting up an environment and sharing it between multiple tests.

In an ideal scenario, tests should access every line of code, to make sure changes in one component do not break another component's functionality. A coverage of 100% indicates tests checked all functions in a project. The package *coverage* can use either *UnitTest* or *pytest* to produce a coverage report to direct the development of new tests and account for as much of the code-base as possible.

2.3.4 Building and publishing packages

Packaging is the task of reorganising all code files and their metadata to make them portable to another independent system. Publishing is making this ready-to-install package available to the public in a centralised repository.

Packaging and publication is a frequent procedure in young and/or popular software. Publication gives users access to the software after fixing bugs or adding new features. There are a couple of popular tools to perform this step:

- *poetry*, in addition to it virtual environment and dependency management capabilities, is also a publishing tool. It uses the now standard format of a single file for configuration and metadata for a given project. It is compatible with other virtual environments, if the use-case requires sharing it between multiple projects. It can publish to *PyPI* - the public Python Package Index and other public repositories.
- *twine* is a utility for publishing python packages to *PyPI* developed by the Python Packaging Authority (PyPA) (*Pypa/Twine*, 2022). It uses an older - yet more established - Python data organisation paradigm.

2.3.5 Automated documentation

There is only one viable software for automating documentation in Python. *Sphinx* is arguably the only tool used to produce documentation from Python code. It builds index-based searchable documentation from documentation comments (docstrings) at the top of methods or classes, through plugins it can interpret different docstrings and markdown and do all this recursively for entire projects (*Welcome Sphinx Documentation*, n.d.). Its usage encourages high-quality documentation during development, for functions properly documented inside the code-base will not need additional work afterwards.

2.4 Working case and conclusions

When dealing with software more complicated than a set of scripts, the task at hand transitions from how to obtain a result into how to accumulate multiple results and make them reproducible; the second task is specially important for the advancement of multiple scientific fields. I assume two kinds of users: normal users, whose goal is analysing their data in a standardised and reproducible way, and user-developers, who intend to edit the parameters and even tweak software internals to fit their purpose. I expect the former type of users to use the public release channels for subsequent features or bug-fixes, while the latter would usually get the latest development changes from the version control system, and only if it affects the section of the code-base they are working on.

In our group we have a system set up as shown in figure 2.2. The goals are attaining reproducible scientific results in an efficient, semi-automated and sustainable way. For this we have chosen a specific set of tools: *pyenv* + *virtualenv* for flexible virtual environments, *GitLab* as a version control system and Continuous Integration environment. All the python tools mentioned in section 2.3.2. We then release our latest changes onto *PyPI* and update the documentation on *ReadTheDocs* automatically. Finally, we have *TLJH* servers set up to provide computing time to our lab

members, and we install the version from *PyPI* instead of the version control one to reduce the risk of bugs affecting users.

Among all the tools reviewed, *Poetry* solves the most tasks at once, which is why I chose it as our dependency and publishing tool. For the remaining software choices I steered towards convenience and tool compatibility.

Unless there is a major overhaul in software development techniques, languages, or the tech industry as a whole, most of these challenges and tools - or analogous ones - should still be in the picture in years to come. In my opinion this is a setup good enough for most complex software development that, once set up, improves efficiency and reduces the scientists' time spent on non-experimental monotonous tasks.

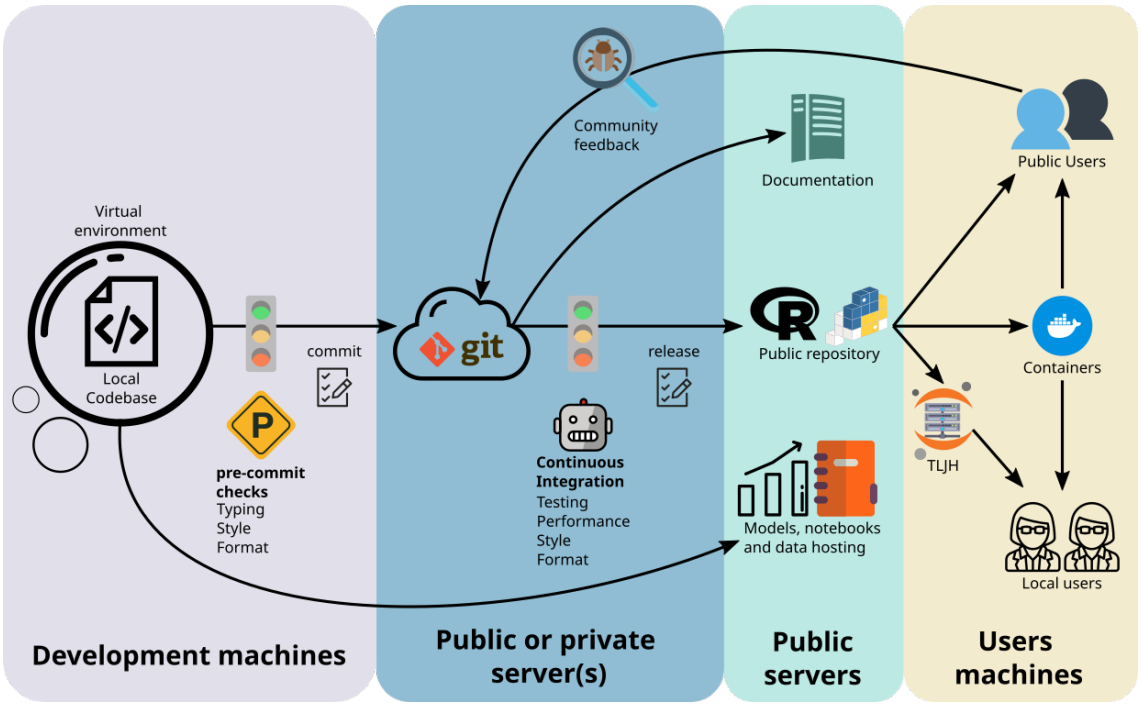

Figure 2.2: Diagram of use-case for software development and release steps. The flow starts in development machines that run checks before committing changes. If - and only if - these checks pass, the developer can commit and push changes onto the remote instance of git, either public or self-hosted. If hosted publicly, *ReadTheDocs* hooks can pull and regenerate the documentation automatically. The developer may manually upload models, notebooks and (generally) training data onto a cloud service for other users to reproduce the models or retrain their own. CI tools test the changes in isolated environments. If these tests pass as well, either the developer or an automated system releases the new version to the public repositories, making it accessible to the users. There are two types of users: The ones from the same group, who can access the latest changes from a shared server running *TLJH*, or other users, who can either use the public (packaged) repositories or the latest in-development version from a version control hosting service (e.g., *GitLab*). It is also possible for any user to use containers to deploy the latest version in a controlled environment. Once the latest version of a software is in circulation, if any bug arises or there is a feature request, the community of users may provide feedback - or pull requests, fixes and or changes to the code - on the remote version control instance, as they generally provide an issue tracker system alongside version control.

3 Outline-based automated cell tracking

A high accuracy tracking algorithm is crucial to study cell populations at the single cell level. It would be impossible to quantify single cell behaviour without identifying individuals over the course of an experiment. The validity of the numerical results produced is thus tightly linked to the accuracy of tracking.

There are multiple technical challenges an ideal tracking system ought to account for. By technical challenges I am referring to potential problems with the input data that may not always occur but they happen at with enough frequency that they remain a risk to the data quality. Some examples are drift of the microscope stage and temporary loss of focus or imaging artifacts. A software that naively assumes these never happen would be unsuitable for to deal with the issues commonly reported during image acquisition.

A tracking system that is based exclusively on segmentation results (i.e., cell outlines) would provide increased modularity for a complex segmentation and tracking software. This allows to combine multiple segmentation and tracking algorithms. It also facilitates measuring accuracy for each process individually. Modularity is important to be able to test current and future tools without writing compatibility layers, which is a costly and slow process for developers, let alone scientists-developers.

3.1 Computational methods

3.1.1 Random Forest

Random Forest is a machine learning algorithm that combines the predictions of multiple decision trees to make accurate predictions or classifications (Pedregosa et al., 2011). A decision tree is a tree-like model that represents decisions and their possible consequences: decision tree is trained on a random subset of the training data and uses random feature subsets, to ensure diversity among the trees. Predictions of all trees are then aggregated to arrive at the final result.

3.1.2 Support Vector Machine

Support Vector Machine -and, by extension, Support Vector Classifier -is a machine learning algorithm used for both classification and regression tasks (Pedregosa et al., 2011). It works by finding an optimal hyperplane that separates different classes or predicts a continuous value. It aims to maximise the margin, or the distance between the hyperplane and the closest data points from each class. By mapping the data into a higher-dimensional space, SVM can effectively find nonlinear decision boundaries. It is known for its ability to handle complex datasets, deal with high-dimensional feature spaces, and handle both linear and nonlinear classification problems.

3.1.3 eXtreme Gradient Boosting

eXtreme Gradient Boosting -often abbreviated as XGBoost, or, in my case, XGB -is a gradient boosting algorithm that excels in predicting or classifying tabular data. It utilises an ensemble of weak prediction models, typically decision trees, to create a strong predictive model. XGBoost sequentially builds the models, with each subsequent model correcting the errors made by the previous ones. What makes it unique is that it leverages gradient descent optimisation to minimise a loss function and improve the model's accuracy. It is widely used in competitions and real-world applications for tasks such as classification, regression, and ranking, where high

accuracy and efficiency are crucial, that is why I use it despite it not being part of the standard scientific libraries (Pedregosa et al., 2011).

3.1.4 Linear sum assignment algorithm

The assignment problem aims to find the optimal assignment of agents to tasks. Each task has a specific cost or benefit associated with it, and each agent can perform only one task. The goal is to determine the assignment that minimises the total cost or maximises the total benefit. This problem often arises in situations where there is a need to match resources or individuals to specific tasks or responsibilities. For example, it can involve assigning workers to different shifts, allocating students to different projects, or assigning vehicles to different delivery routes.

This implies finding the optimal assignment of tasks to agents with associated costs or benefits. The algorithm iteratively builds a solution by modifying a cost matrix. It reduces the matrix by subtracting the minimum value in each row from every element in that row and subtracting the minimum value in each column from every element in that column. This reduction step is represented as:

$$\text{cost}(i, j) \leftarrow \text{cost}(i, j) - \min(\text{row}_i) - \min(\text{column}_j) \tag{3.1}$$

After reducing the matrix, the algorithm proceeds to find the optimal assignment by identifying a series of alternating rows and columns containing zeros. It searches for a complete set of such zeros that do not share the same row or column, known as a "zero-sum submatrix." If a zero-sum submatrix is found, the algorithm updates the assignment. If not, it adjusts the cost matrix and repeats the process until an optimal assignment is achieved.

The linear sum assignment algorithm guarantees finding the optimal solution for the assignment problem. Its time complexity is approximately $O(n^3)$, where n represents the number of tasks or agents. The linear sum assignment algorithm, also known as the Hungarian algorithm, is usually used to solve the assignment problem (Dietler

et al., 2020). In this work, the assignment problem will arise when trying to match pairs of cell outlines to each other. It is explained using code further down in this chapter, in case that suits the reader better.

3.2 *CellTracker* system

To assign births with high accuracy, after segmentation we must complete two tasks: The first one is to link cell outlines from one time point to the next one (figure 3.1A,B). The second is to identify mother-daughter relationships (figure 3.1C). To solve both challenges we use machine learning classifiers that convert information from cell masks into the probabilities, and then post-process the probabilities they return.

Classifiers are computational models to which we feed a 1-dimensional array of numerical values and return a probability of this set of numbers belonging to a given class based on previous training. Both our cell tracker and lineage assignment classifiers transform our input array to an array of probabilities for a predetermined number of classes. The classifier's job is to convert a "feature vector", representing quantitatively the relationship between two cell masks, into a probability for each class. In our setup the only possible classes are either one or zero, and because all probabilities must add up to one, we only need a single probability to represent the class prediction.

The output classes represent the relationship between pairs of cells. For cell tracking (Figure 3.1A) the output shows the likelihood of two cells in different time points being the same cell. For lineage assignment (Figure 3.1A) the output indicates the probability of two cells having a mother-daughter relationship, and, unlike the previous one, it is asymmetric; that is, the probability of cell A being cell B's mother is not the same as the probability of cell B being cell A's mother.

This chapter covers the first task, but refer to (Pietsch et al., 2022) for the details on lineage assignment.

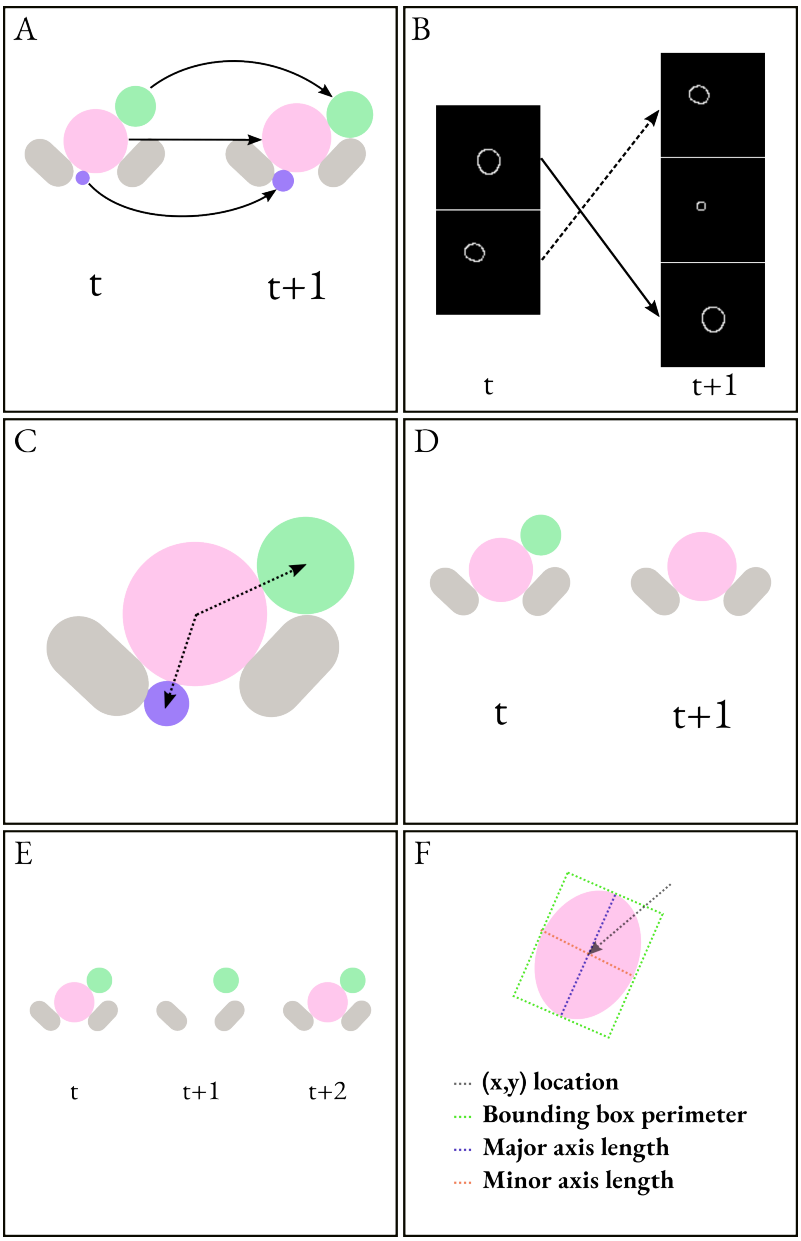

Figure 3.1: **Task overview and important concepts. A**. We must identify cells across time points regardless of how they grow and move within the images. **C**. In practice this is challenging due to the amount of information contained in cell outlines, which is low when compared to bright field data. **C**. We then have to find the mother-daughter relationship between cells at every time point. The first task requires multiple time-points to work, while the mother-daughter relationship predictions obtains predictions from individual time-points, as all cells and mothers are present (if existent) at any given time-point. **D**. Due to the ALCATRAS system, buds are flushed away by the media after cytokinesis. **E**. Outline-based tracking must also deal with the posibility of some cells disappearing transiently due to segmentation mistakes. **F**. To perform outline-based tracking I aim to use acell features, the ones pictured are major axis length (dotted blue line) , minor axis length (dotted red line), bounding box (dotted green square) and centroid (dotted black line).

We face multiple challenges that are unique to our experimental setup. Daughter cells are commonly washed out of the microfluidic device and so disappear from one time point to the next (Figure 3.1D). Additionally, if we decouple the segmentation and tracking step we must also cosider the posibility of segmentation-only errors (Figure 3.1E).

To track cells we use the changes in their masks over time as an indicator of identity. We use features extracted from cell masks (Figure 3.1F and Figure 3.2A), such as area, major axis length - herein called attributes - and train a classifier, either a random forest or gradient booster, to assign a probability to the pairwise changes in numerical features between cells/time-points sets -feature subtraction of two cell masks, where masks are in consecutive time points; I define this as a change vector and a set of these vectors comprises a tensor of features (Figure 3.2B).

By feeding the change vector to a classifier we obtain a probability (Figure 3.2B). I define this as cell-to-cell probability: The probability of a cell (or outline) in one time point being the same as another one in the following time point. Finally, we apply heuristics to the resulting probability matrix to assign cell labels and identify new cells (Figure 3.2).

The second half of the *CellTracker* algorithm uses these probabilities to differentiate existing from new cells. First, it aggregates the data from the last three time points (Panel 3.2D), then obtains the highest cell-to-cell probability for every known-and-new cell pairs (Panel 3.2E). After selecting the highest probability from the last three time points, it selects the maximum probability for every new cell (Panel 3.2 F). Finally, it applies a threshold to differentiate between new and existing cells (Panel. 3.2G).

As mentioned before, daughter cells are commonly washed out of the microfluidic device and so disappear from one time point to the next. These absences undermine traditional approaches for tracking, such as the Hungarian algorithm. We therefore compared three standard algorithms (Chen & Guestrin, 2016; Pedregosa et al., 2011): , Random forest and Gradient boosting for classification by their accuracy.

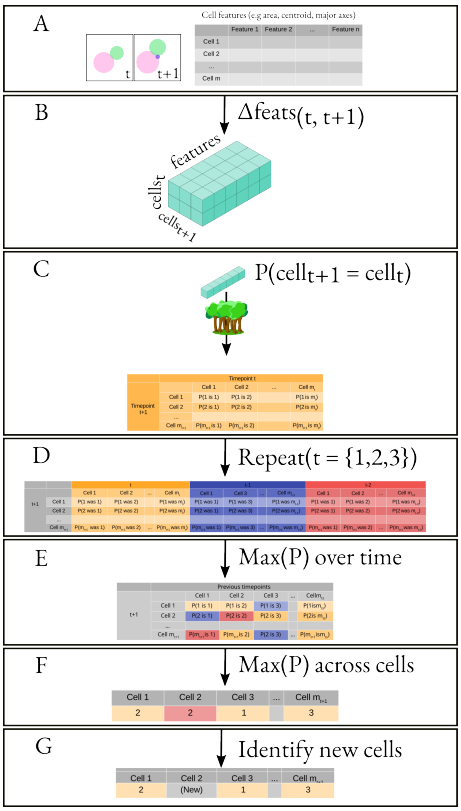

Figure 3.2: **Cell tracking algorithm.** **A**. We obtain the attributes of all cells at t and $t-1$. This results in two matrices of shape (m_t, n) and (m_{t+1}, n), where m_x is the number of cells at time x and n is the number of attributes. **B**. We perform an element-wise subtraction of all features for every pair of cells (c_t^i, c_{t-1}^j), where i,j are cell identifiers for each frame. In other words, we subtract the attributes of all cells in time point t from the attributes of all cells in time point $t-1$. This generates a feature vector for every cell pair between time points. **C**. We apply a classifier (random forest by default) to the feature vector corresponding to a pair (i,j), generating a matrix with dimensions C_t and C_{t-1}, where C_t is the number of cells found at time t. This matrix represents the probability of every cell in time t to be the same cell in time $t-1$. **D**. We perform the same operations using $t-2$ and $t-3$ instead of $t-1$. **E**. We pick the maximum probability for every pair of cells in all probability matrices. Cells are not guaranteed to be found at every time-point, we thus check cells from previous frames and select the highest probability between the last previous occurrences of a cell. **f, g**. Finally, we apply a minimisation algorithm to pick the optimal cell pairs and use a threshold to identify new cells (Algorithm 1).

Both random forest and gradient boosting (specifically Xtreme Gradient Boosting - XGB for short) performed similarly, and better than SVC (data not shown).

3.2.1 We use curated data to train our classifiers

To train a classifier we used a data set composed of independent time lapses with manually labelled cells. We calculate the feature vectors for all pairs of cells between consecutive time points. Afterwards we label as class one all vectors that link cells with the same label and zero the remaining ones. This set vector-label become the training data set for training our classifiers.

3.2.2 Masks are converted into features to train a Machine Learning Model

We select two sets of attributes (or features, as we call them when in a feature vector, i.e. after we subtract them between cell masks) to develop a main and a backup cell-tracking classifier, to one of which we applied feature engineering: using existing features to generate more - thus generating one set of features with explicit distance and one without distance explicitly accounted for. The first set contains area, minor axis length, major axis length, convex area, centre location on the x-axis (centroid-0) and centre location on the y-axis (centroid-1), and an additional cardinal distance obtained from x and y-axis location. The second set of features contains no explicit distance-based attributes: area, minor axis length, major axis length, convex area and bounding box area.

Once we have our features and labels ready, we train the three classifiers (SVC, Random Forest and Gradient boosting) with a grid of learning hyper-parameters. *Scikit-learn* explores the parameter space and chooses the best set of hyper-parameters. We trained a SVC with a regularisation parameter C of 0,1, 10 and 100; Γ kernel coefficient of 1, 1e-2 and 1e-4; we used no shrinking heuristic which speeds up training ; we also trained explored two different kernels: Radial Basis Function (*rbf*) and sigmoid. The training of the Random Forest classifier explored different number of

estimations, specifically between 10 and 80 estimators and a depth between 2 and 10 levels. We trained XGBC with a maximum depth of either 2,4 or 8 levels; the minimum child weight, which is a threshold below which the sample size will not be further split, is between 1, 5 and 10; gamma, which is the minimum loss reduction to partition a leaf node, between 0.5, 1, 1.5, 2 and 5; and sub-sampling ratios of 0.6, 0.8 and 1.

After training, we looked into the interpretability of features fed to our random forest, as it is generally insightful to look at which features the classifiers take into account to make their decisions. We chose random forest as a proxy for the other classifiers because it is easier to interpret the individual trees that compose it. The weight distribution given to different features depends on whether distance is explicitly taken into account or not. In figure 3.3 the first classifier places most of its weighting for deciding on the explicit distance between cells between time points. The second classifier, lacking information on location of masks along the axes, distributes the weight of decision among most available features more evenly. We expect the other classifiers (XGB, SVC) to follow a similar trend.

3.2.3 Metric-based units facilitate scaling

The pixel properties of tracked cells depend on three main experimental settings: the size of the cells, the microscope's objective and the camera's pixel size. A tracking algorithm may generalise correctly to one or multiple setups, but the robustness of such an approach and quantifying its accuracy is practically absent in the literature. None of the existing tracking algorithms (nor segmentation+tracking algorithms) show aggregated statistics of data sets from multiple experimental setups, although some of them expect re-scaling of the existing training data and addition of new data, but this is seldom documented - as seen on the public repository derived from *O'Connor et al.*'s DeLTA (O'Connor et al., 2022) - issue number 50.

There are two options for scaling the features upon which we can train our classifier: Using relative or absolute units. Relative units produces models trained on raw

pixels. In such case the model's internal definition of 'outlines that are too different to be the same cell' results from the size, shape and motility of the microorganism under surveillance. To avoid the complications that come with relative units, I chose to use "absolute" units for cell tracking - that is, metric-based units. When using absolute units the models "learn" to make decisions based on metric distances instead. If we are to scale it up or down to study a different organism, there is no clear frame of reference to do so, I thus decided to use the most portable measurement system for models to learn.

To account for this lack of standardisation and the low availability of public annotated data (Liu et al., 2021), I re-scale the feature properties that have a physical unit (e.g. length or area) using the image pixel size if it is available. I chose the pixel size over normalisation to an arbitrary range because is a real-life unit that researchers are familiar with. We maintain physical scaling when changing the source of our input data, such as using a different microscope objective that changes the pixel size, (e.g. linear for major axis length and quadratic for area) making the model robust to organisms living at different physical scales. The wide variety of setups, leading to different cell and pixel sizes, may impact tracking in a variety of ways. Setting re-scaling manually, if we were to change our data sets from our (default) *S. cerevisiae* scale to time-series of bacteria with 1.5x the pixel size, we would just re-scale the images by this factor. The interpretability eases correction of classifier errors and cell features.

This ease-of-use also facilitates the identification of errors when interpreting the tracking predictions. We can observe if units do not fit our expected values, and if tracking is misbehaving we can couple it with tools that make the decision-making of our models clearer by quantifying the statistical importance of individual features, even if they are not *Random Forest*, such as SHapley Additive exPlanations - SHAP - or Logical Interpretable Model-agnostic Explanations - LIME (Bowen & Ungar, 2020; Visani et al., 2022).

3.2.4 Our algorithm is accurate in multiple experimental setups

Our algorithm's bottleneck changes depending on the number of cells: At low numbers it is the calculation of features from outlines; at large numbers it is the cell-vs-cell subtraction operation and predictions. The computational complexity is proportional to the number of cells in a quadratic manner. Additionally, processing speed and algorithm implementation changes depending on whether cell outlines overlap or not. Most existing software assume cells do not overlap, with a few exceptions (Pietsch et al., 2022; Versari et al., 2017).

I developed a minimal-distance heuristic, that only compares cells that are no more than 3 micro-meters apart (this parameter can be adjusted). I can then limit the complexity increase generated by exponentially-growing cells. For cases where cell cells accumulate inside the frame without constraint the minimum-distance heuristic provides a speed boost (figure 3.5). Hundreds of cells are still manageable, thus facilitating live-processing of experiments

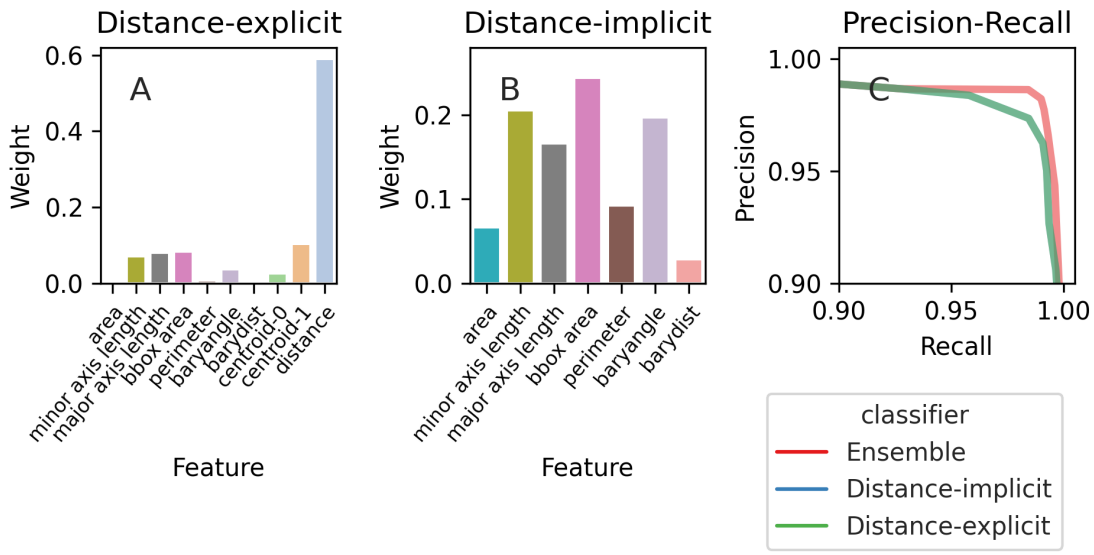

Figure 3.3: **Feature importance with and without distance metrics** Depending on which features we train the classifier with, the weights (or feature importance) are more evenly spread. **A** If we train the classifier using a distance metric, distance drives the decisions, and the remaining features are only used for marginal cases. **B** If we train the classifier using distance-implicit features, feature weights are more uniform. **C** Our precision-recall curve shows high accuracy for both distance-explicit and non-explicit random forest classifiers, at the point in which the implicit model becomes unreliable it "jumps" to catch the trajectory of the explicit model.

3.3 Data augmentation and heuristics

To generate additional training data, we use multiple time points backwards in time. For example, for time t we generate not only feature vectors by comparing those cells with the ones at $t-1$, but also $t-2$ and $t-3$. This acts to increase robustness and improve accuracy across a wider range of imaging intervals and growth rates.

For the purpose of training our classifiers, we treat these additional data sets as if they were consecutive time points, regardless of how far in time they really are. For instance, we train using the feature vector $\Delta(F_t, F_{t-1})$ but also $\Delta(F_t, F_{t-2})$ and $\Delta(F_t, F_{t-3})$, where F_t are all cell-features in t. This in principle produces more data at no additional cost, working as a "time-driven augmentation" of sorts.

For tracking to become as robust as possible we require an unbiased subset of data for both training and testing. Our training data consists of a set of yeast cell time-lapses, with their identities annotated over time. The number of time points for each time-lapse can be variable. Additionally, the number of cells or position, on the axes in a single trap does not change dramatically between time points, therefore the feature vectors linking cells between them will be more similar within a time-lapse set than between sets of time lapses. Knowing that longer time lapses will have a stronger impact on the (hidden) decision probabilities in the training set, I limit model biases towards training sets containing more time points by averaging the accuracy of our classifiers over the number of frames. I call this trap normalisation, for time lapses in our training data are trap-specific. I then use the trap accuracy as the metric we aim to improve accuracy as shown by our precision-recall curve.

The precision-recall curve indicates that the distance-explicit classifier fares better than the non-explicit, but the latter still has high accuracy (fig 3.3c). Due to having access to more information, the distance-explicit model will perform generally better, but it will likely fail more often than the distance implicit model when encountering pivoting and other cell-displacement events, which are rare events, but seem to happen often once for a fraction of all time series (see an example in Figure 3.4). For this reason, I chose to the distance-implicit classifier as the main model, and if

its result is an ambiguous probability we use the backup model - processing more information - to try and resolve the ambiguity.

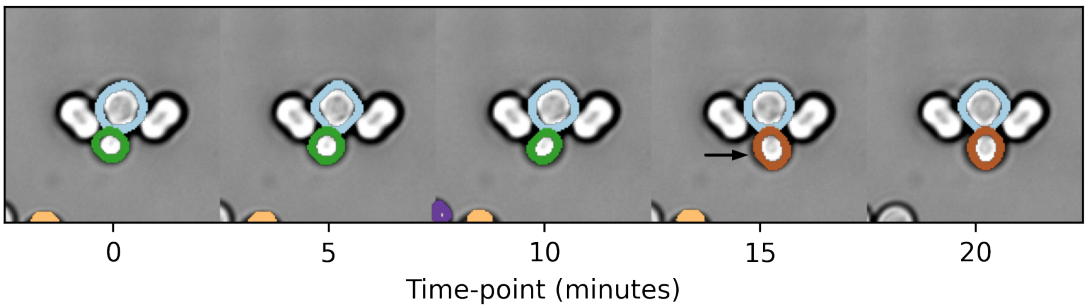

Figure 3.4: Example of a tracking error caused by a cell pivoting. The black arrow indicates the point at which tracking made a mistake caused by the rotation of the mother changing the bud's position. These type of events require manual curation to quantify, and are thus challenging to identify automatically

The ensemble model performs similarly to the distance-implicit classifier, but in the rightmost part of the curve it behaves more similarly to the distance-implicit one 3.5. This means that if we use a stringent threshold for assuming cell identities our results will be more similar to the distance-explicit model. As we would expect, the ensemble approach stands in-between the other two.

3.3.1 I use information multiple time points and the linear sum assignment algorithm to optimise cell assignment labels

To maximise robustness we select data from multiple previous time points. First, we compare cells between the current time-point and each of the last three time

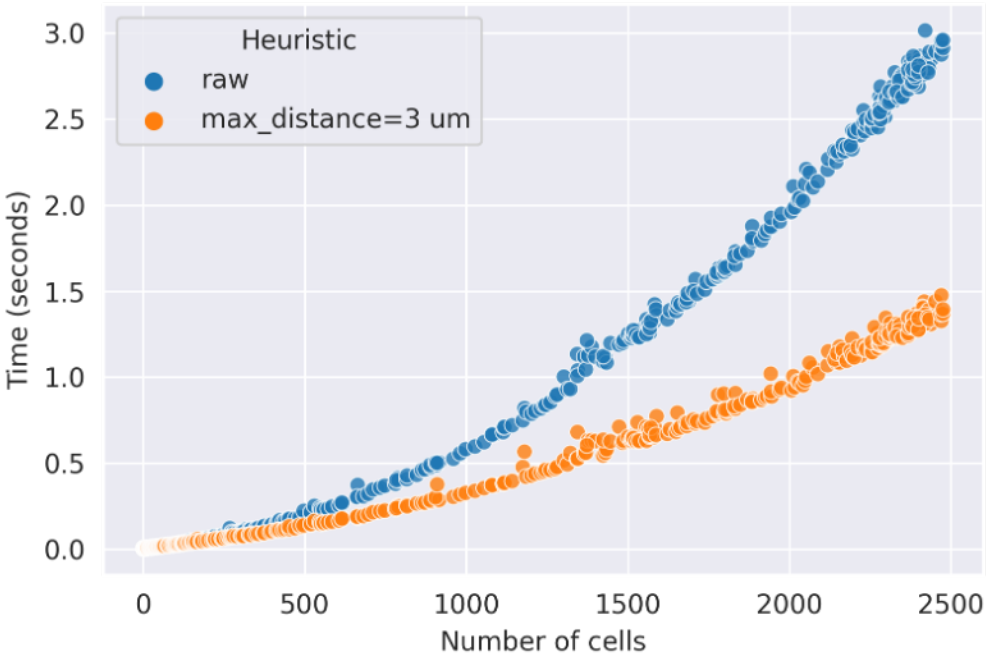

Figure 3.5: A distance based heuristic greatly improves processing speed at no performance cost. Cell-to-cell comparisons bear considerable responsibility on the exponential complexity increase as more cells appear subsequent frames. The *y-axis* represents the time it takes to process a pair of images and the *x-axis* represents the total number of cells at the given time-point.

points backwards in time. We then pick the maximum probability for each pair of new and "old" labelled cells, and finally split new cells from existing ones by using a threshold. The last two steps, which assume we already have a matrix of cell-to-cell probability - sometimes referred to as a Bayesian belief matrix (Ulicna et al., 2021) - are shown in algorithm 1.

Algorithm 1: Cell labeling

Our cell assignment algorithm makes use of the linear sum assignment algorithm, also called *Hungarian algorithm* to maximise the overall probability of cell-pair assignments. **Data:** $probMat$, $threshold$, $oldLabels$, $maxLabel$

Result: New cell labels ($newLabels$)

let $newLabels$ be zeros(ncols($ProbMat$));

for $old, new \in linearSumAssignment(-ProbMat)$ **do**

  if $probMat[old, new] > threshold$ **then**

    $newLabels[new] \leftarrow oldLabels[old]$;

  end

end

for $label \in newLabels$ **do**

  if $label \neq 0$ **then**

    $label = maxLabel + 1$;

    $maxLabel =$ label;

  end

end

return $newLabels$

First, we acquire the features for all cells in the current (t) and past two time points $t-1, t-2$. This generates feature vectors between t and independently $t-1$ and $t-2$. We then feed these vectors to the trained classifier and get a probability as a result. I use the classification model with no explicit distance-based features by default. If the probability it returns is between 0.1 and 0.9 it runs the backup model and uses the resulting probability.

Within the cell labelling algorithm (Algorithm 1), I use the linear sum assignment - also called Hungarian - algorithm on our probability matrix to select pairs of cells using their pairwise probabilities of being the same. The linear sum assignment algorithm guarantees at most one label per cell; instead of just maximising cell masks' probabilities on an individual basis, we select set of probabilities whose total sum is

the highest; this distinction is necessary only for fringe cases because the classifier usually returns high or low probabilities, making the identification of potential pairs a relatively simple task. Finally, we apply a threshold to these probabilities and assign new labels to the new cell masks - daughter cells or cells that just wandered in-frame - with probabilities lower than the threshold, which is 0.5 by default.

3.4 Tracking accuracy benchmarks

To accurately compare tracking techniques, a general framework that defines the "perfect tracking". I define the perfect tracking algorithm if it solves the task of following all the objects in the video during every frame they are present; this means to correctly identify the time interval that corresponds to their presence. Let us now accept the fact that such an algorithm is an unrealistic goal, and redefine a goal that uses research goals as main drivers.

In the tracking diagram shown as Figure 3.6, I expressed the ground truth as lines connecting objects that are the same - also represented by colours and a letter s indicating the start of a track. We omit fragmentation (time-points where cells were not detected) because the *CellTracker* algorithm skips those time-points, as they are not informative.

Precision and recall are two performance metrics we used in tandem to evaluate the accuracy of a classifier (or any other system that makes data-driven predictions). Precision indicates the number of instances that are correctly identified as relevant - where relevant means the real positive values - out of the total instances retrieved. Recall is the number of instances identified as relevant by the model out of the total relevant instances. Put in more formal terms, precision P is:

$$P = \frac{TP}{TP + FP} \tag{3.2}$$

And recall R:

$$R = \frac{TP}{TP + FN} \tag{3.3}$$

Where TP are True Positives, FP is False Positives and FN is False Negatives. In our tracking context, positives are pairs of cell outlines correctly identified as the same and negatives are pairs of cells outlines correctly identified as not being the same cell; this renders our definition of precision as the fraction of outline pairs correctly linked from all the cells linked. The recall metric is the fraction of outline pairs correctly identified as pairs divided by the outline pairs that actually are the same cell.

The most common tracking metric is Multi-Object Tracking Accuracy (Dendorfer et al., 2020). Formally defined as:

$$MOTA = 1 - \frac{\Sigma_t(FN_t + FP_t + SW_t)}{\Sigma_t GT_t} \tag{3.4}$$

Where t is the frame, FN_t and FP_t are the number of false negatives and false positives at frame t, SW_t is the number of identity switches (a merge of two independent tracklets) at frame t and GT_t is the number of ground-truth objects at frame t.

3.4.1 The tracking-only benchmarks require assigning weights

Quantifying accuracy is not an endeavour free of parameterising, despite what certain existing claims (Ulman et al., 2017). As shown in Figure 3.6, there are different types of errors that may happen. If we aim to penalise tracks that are not correct in their totality we have to account for the propagation error (the orange lines in Figure 3.6, panels B-E). In Figure 3.6D a single error (linking the grey object to the pink one) results automatically in an additional error, as the pink tracking link is automatically lost. In Figure 3.6E, the problem is even worse, because the false prediction condemns two tracks to being incorrect: The source at $t = 0$ and the target at $t = 1$.

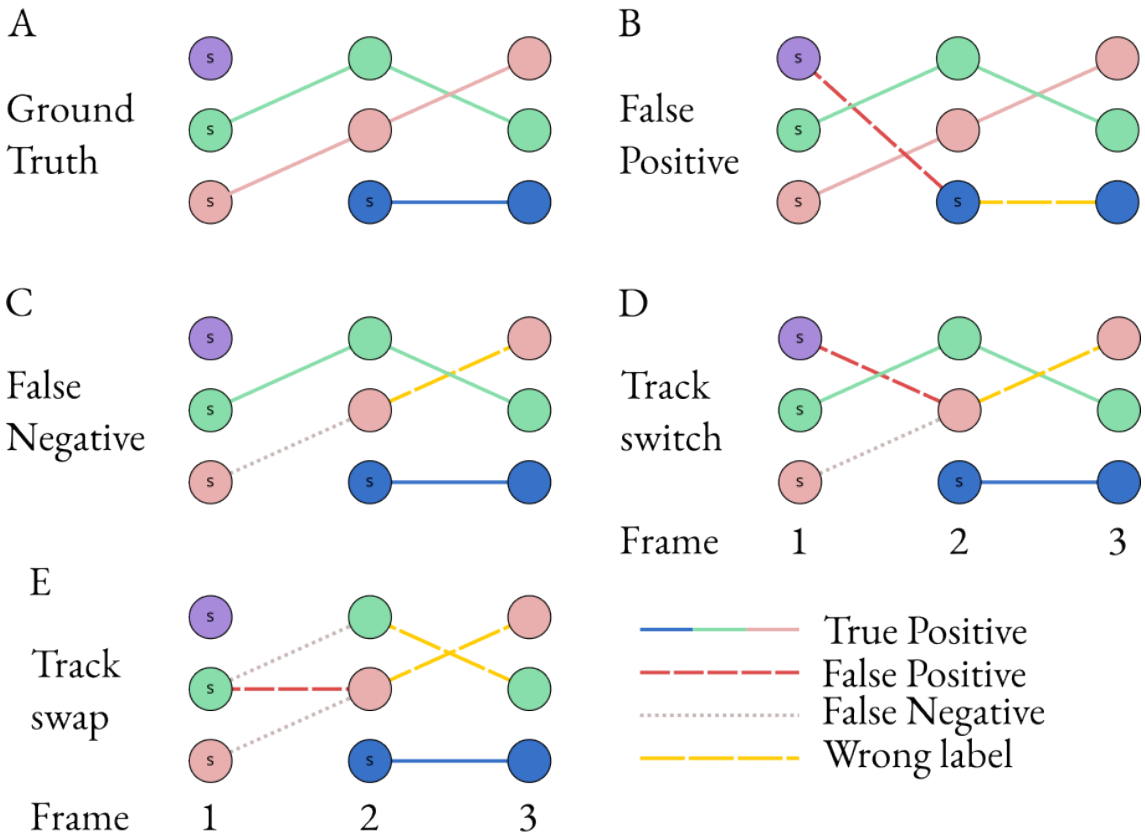

Figure 3.6: **Multiple elements must be considered to calculate tracking accuracy.** Circles represent individual cells at the frame indicated at the bottom. Their colour determines identity, new tracks display an *s* on the frame they first appear. Panel A shows the tracks as they would look if tracking was perfect. Panels B and C show how we define False Positives and False Negatives. Red and orange line are two type of errors, direct and propagated: Direct errors are False Positives (panel B) or False Negatives (Panel C); propagated errors arise when the local assignment is correct (from t to $t+1$), but the original label is incorrect and thus the final cell label will be too. We represent False Positives as solid lines and False Negatives as dotted lines. Given that tracks link instances between consecutive time-points, the task of *CellTracker* is to recreate the links in the ground truth as closely as possible. Our tracking algorithm assumes perfect detection, thus our False Positives or False Negatives are not exactly the same as conventional Multi-Object Tracking accuracy (MOTA). The closest approach to a track switch in MOTA are shown in panels D and E; each is a single mistake in the MOTA framework, but when looking into the details they produce two and three direct errors, respectively. You can get the number of errors by counting grey dotted lines; if we include indirect errors in the calculation (solid orange lines) Panel D has three errors and Panel E has five.

For conventional *MOTA*, False Positives and False Negatives are detection-based: detection of fake cells and missing detection of real cells, respectively. Given that we assume perfect detection, all our errors are "id switches". Thus quantifying accuracy thus becomes the fraction of correct assignments at each time-point and the non-assignments (tracklets that end in the previous time-point) correctly predicted (Figure 3.6B).

To separate detection from tracking, we calculate tracking accuracy under the assumption of fully correct detection of cells, that is, we assume that we detected all cells correctly in the entirety of the time series. This is unlikely to be the case, but it is necessary to test the tracking algorithm with multiple cell segmentation tools. Our tracking of false positives and false negatives does not correspond to the widely-used Multi-Object Tracking Accuracy (MOTA). We need an accuracy metric that instead focuses on switching events and predictions of new tracks, because the tracking algorithm will never know whether an object is real or not, it is not its task.

Not all training data sets follow the same shape or structure. Another peculiarity of our training data sets is the relative short length compared to the size of an experiment. This stems from the fact that having a wide variety of curated images to train a neural network is more efficient than longer time-series - they return-on-investment decreases as we add more images showing the same cells in different frames. Curating long time-series is ideal for tracking training, but resources must be diverted for the crucial task of curating data for the segmentation model or algorithm.

A different benchmark is using the assignment matrix as a set of predictions (i.e. the direct output of the machine learning classifier, for every pairwise arrangement of cell at t and $t+1$), but that is not really measuring the overall tracking accuracy, but only the one for each pair of time-points. Due to the size of the product of all outlines in t and all outlines in $t+1$, using the correct prediction of cell-to-cell pairs may lead to distorted results because as the number of cells increase, most of them

are True Negatives, although there are ways to correct for this, such as the ones used with acyclic graphs (Matula et al., 2015).

Standard MOT accuracy is not ideal for our setup due to the discrepancy of benchmarking goals between tracking quality vs long-term tracking (Versari et al., 2017); the probability of a single error occurring increases over time, but a single error results in misidentifying an entire track. Defining minor and major errors is an alternative solution (Wood & Doncic, 2019). There is no tracking method specific for a system like ours in which some objects are (practically) guaranteed to stay in-frame, while most others will disappear. It is unclear whether this makes a big difference when compared to other "conventional" tracking tasks, making this a potential branch for future research. Notwithstanding, others have also suggested that benchmarks should be adjusted according to the biological domain of study (Reinke et al., 2022).

3.5 Conclusion

I developed a robust, versatile and fast cell tracking system that depends only on cell contours. It uses machine learning classifiers to transform changes in cell contours into probabilities, and then treat the task as a quasi-standard assignment problem. *CellTracker* thus makes as few assumptions as needed to sustain modularity, making it compatible with any segmentation tool that is currently available and future tools that I expect will be published in coming years.

4 Automated single-cell microscopy analysis

4.1 Introduction

In the last decade, the field of biology has taken a turn towards data-driven approaches. At first Deoxyribonucleic Acid (DNA) sequencing what became more accessble, kickstarting a revolution in bioinformatics as a consequence. In recent years, novel techniques in fields that are adjacent to biology, such as microscopy or applied physics, provided a nursery for new ways to acquire unprecedented volumes of biological data.

Studying cell decision making is vital to understand key biological processes. Cells make crucial choices that impact stem cell development and disease progression (Bowsher & Swain, 2014). Investigating these decisions unveils molecular pathways and signalling mechanisms. Single cells studies provide insights into tumout heterogeneity and mechanisms underlying therapy resistance.

High throughput techniques unveiled the fact that multiple cell behaviours are stochastic (Elowitz et al., 2002). Traditionally, bulk analysis techniques average out the molecular information from a large population of cells, masking the heterogeneity. Advances in automation techniques enabled the study of thousands of cells in parallel. Among these technologies, an important non-destructive method is live-microscopy assays, used in a variety of biological subfields (Ponsford et al., 2020).

Bioimaging is a neglected field of study when compared to more widespread bioinformatic techniques, such as sequencing or proteomics. To bridge the gap between single-cell decision making and make full use of novel image acquisition technologies, novel tools must bring synergy with data analysis pipelines and methods. These and

other challenges we faced when tackling the study of single cell decision-making by means of high throughput micorscopy.

A key distinction between our standard experimental setup, ALCATRAS (Crane et al., 2014), and most other live-microscopy environments is the constant flow of media, removing daughter cells once they are independent from their mother cell; this difference has key implications on the assumptions we can make about the resulting data, and thus on some components of the posterior analysis. Some cells leave the field of view between frames; newborn buds usually do because they are washed away by the media. Furthermore, the media changes brings biases in the way cells move, sometimes resulting in buds pivoting around the mother. Like most other live single cell analyses, we cannot assume a fixed number of cells. Unlike them, we also cannot assume that existing cells will stay there from one frame to the next.

The task at hand demands a simple and flexible data structure to store information on individual cells over multiple time points, metrics and fluorescence channels. I strive to build a system that assigns a unique fingerprint for every cell and that is able to access them efficiently. For ease of use, this data structure should seamlessly transform into a visually informative shape. Following conventions set by existing imaging management software, such as OMERO (Li et al., 2016), we represent multi-dimensional images as different axes: Time, channel, and x, y and z-axes (or z-stacks). Defining time as the first dimension shows its precedence for time series analysis.

4.2 Technical terms

4.2.1 OMERO

OMERO (Open Microscopy Environment) is a software platform designed for managing, analysing, and sharing biological imaging data. It provides a solution to

handle large-scale microscopy datasets and enables researchers to organise, annotate, and analyse their images in a collaborative environment (Li et al., 2016).

4.2.2 Otsu threshold

The Otsu threshold method calculates an optimal threshold value by maximising the between-class variance of the grey-scale image. The method assumes that the image contains two classes of pixels: foreground and background. The goal is to find a threshold that minimises the intra-class variance (variance within each class) and maximises the inter-class variance (variance between the classes) (Otsu, 1979).

4.2.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a deep learning model specifically designed for processing and analysing visual data, such as images and videos. It consists of multiple layers of interconnected neurons that perform operations like convolution, pooling, and non-linear activation to extract relevant features from the input data. CNNs are particularly effective at capturing spatial patterns and hierarchies of information, enabling them to automatically learn and recognise complex patterns and objects in images (Goodfellow et al., 2016).

4.2.4 Abstract Base Class (ABC)

An abstract base class, often referred to as an ABC, is a concept in object-oriented programming where a class is designed to serve as a blueprint or template for other classes. Unlike regular classes that can be instantiated, an abstract base class cannot be directly instantiated but is meant to be subclassed by other classes. It provides a common interface and a set of methods that subclasses must implement. The purpose of an abstract base class is to define a common structure, behavior, or contract that its subclasses should adhere to, ensuring consistency and promoting code reusability. It allows for the creation of a hierarchy of related classes, with

the abstract base class providing a foundation for shared functionality while leaving specific implementations to its subclasses.

4.3 Architecture

I designed ALIBY as a software suite that generates analysis pipelines to process microscopy movies while minimising human input; it aims to produce automated end-to-end analysis. Its main goal is streamlining solutions for multiple challenges biologists face when phenotyping cells. Its long-term goal is facilitate data accumulation and improve reproducibility in analyses of single-cell live-imaging. The process is shown in Figure 4.1.

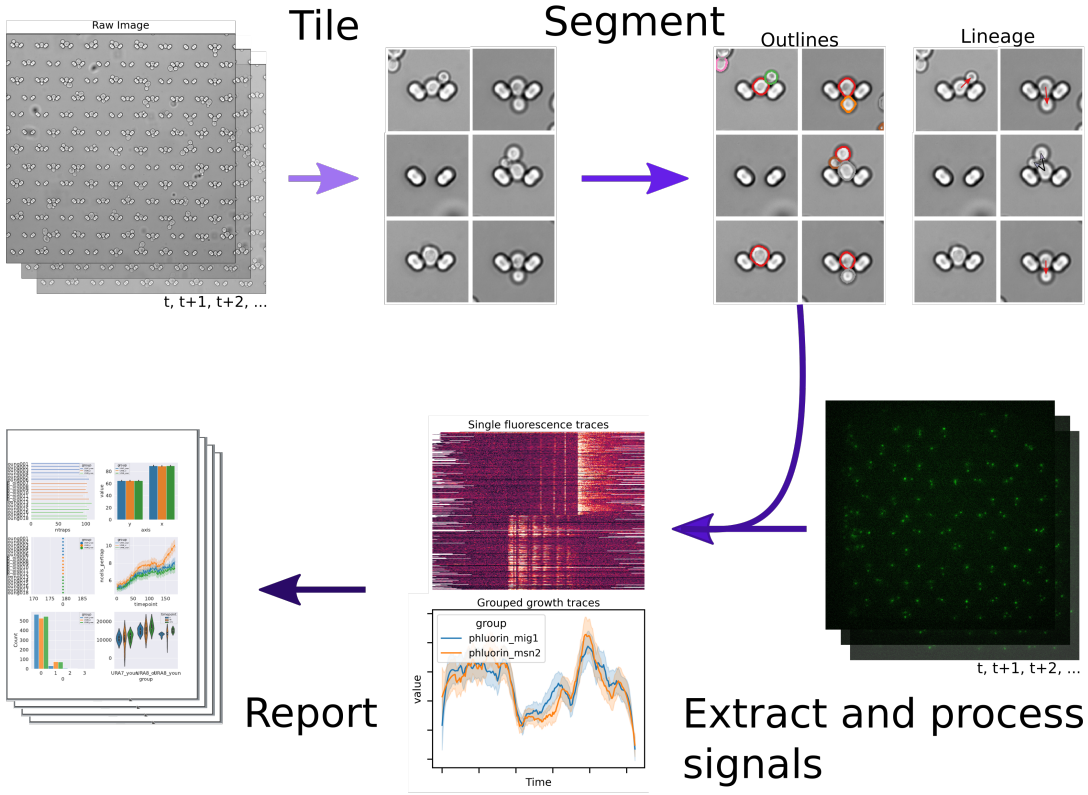

Figure 4.1: ALIBY aims to speed up most common tasks done when analysing high-content live-microscopy imaging.* It coordinates all steps of a live-microscopy processing pipeline. First cropping regions of interest where cells are, then using deep learning for cell segmentation and tracking and lineage assignment. Afterwards it calculates metrics for cell properties from images and further processes data in a reproducible manner to, at last, produces an automated report.

To maintain a "large" code base -understood within software engineering circles as more than 10k lines of code -modularisation is a necessity. I split ALIBY into four different components, based on a combination of conceptual abstraction (how to represent individual elements and their interactions) and pragmatism, based on the advantages and limitations of modern development toolsets.

agora contains the foundations, tools that are used by other modules (its name means marketplace in Greek, because it is where base ingredients are shared). It includes classes that handle I/O, such as data frames, cell masks and identifiers. It hosts the Abstract Base Classes (ABCs), which determine the shared structure of *parameters*, *process*es and other abstractions. This module contains the necessary bricks and bridges that conform and connect higher-level components, alongside tools that interact with storage structures. It is completely independent of any other component of ALIBY.

BABY (Birth Assignment for Budding Yeast) is a custom version the original segmentation package (Pietsch et al., 2022) adapted for more compatibility with ALIBY. My only functional change is the removal of a clogging interruption, I delegated the function to the pipeline level for more refined control.

postprocessor orchestrates all functions that operate on non-image data, generally extracted time series. These so-called *postprocess*es generally alter time series without changing their shape. It depends on agora to provide metadata and I/O.

ALIBY is the heart of the eponymous suite. It orchestrates the pipeline from start to finish by coordinating all top-level *process*es - processes that are only used by a *pipeline process*. It also hosts the facultative networking tools that may operate on images from remote locations, namely *tiler* and *extractor*. Given that it coordinates all other modules, it depends on all three (Figure 4.2).

The internal organisation of each module is shown in the following list; it shows the main class locations within sub-modules, including an overview of BABY, as it is the default segmentation tool:

- ALIBY

  - *Pipeline* coordinates all pipeline steps.
  - *Tiler*: Tiles and tracks tiles.
  - *Extractor*: Obtains numerical information from images and cell outlines.
- agora
  - *Reader*: Accesses *hdf5* files.
  - *Writer*: Writes results and metadata into *hdf5* files.
  - Metadata parser: Returns metadata logs and additional files.
- BABY
  - *BabyBrain*: Coordinator for the segmentation and tracking models. Contains an instance of MasterTracker.
  - *MasterTracker*: Orchestrates both tracking and lineage assignment.
    - *CellTracker*: Identifies cells across multiple frames.
    - *BudTracker*: Predicts lineage relationships from BABY.
- *Postprocessing*
  - *Postprocessor*: Coordinator that applies non-image operations to existing data. It may use lineage information.
  - *postprocess*: Apply an individual operation on a dataset.
  - *grouper*: Accumulates data from multiple positions organise results as groups, based on names or metadata.
  - *reporter*: Produces a graphical report from standard experimental metrics.

4.3.1 Automatic tiling

A tile is a rectangular section of the image, usually containing useful information. It is not essential to tile, as entire images could be directly processed, but it offers advantages, such as increased download speed when analysing remote data, or enabling parallel processing within a position. When using ALCATRAS devices tiles are defined by assigning one tile to each pillar-based tile.

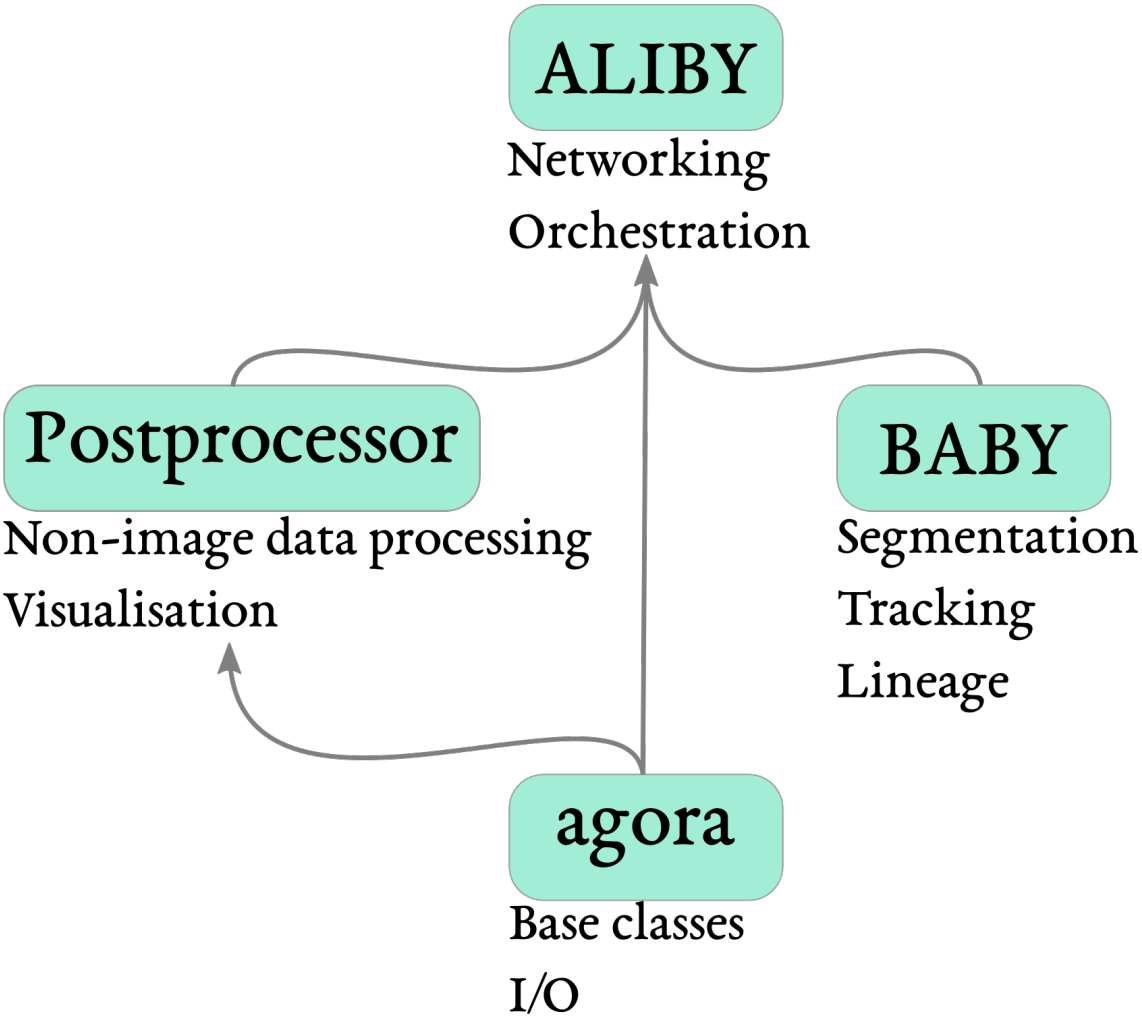

Figure 4.2: **I split ALIBY into multiple modules and defined a dependency hierarchy for efficient test automation and component reusability**. agora and BABY are independent, *postprocessor* depends on agora and ALIBY depends on all of them. Arrows indicate the dependency direction, the recipient of an arrow calls one or more elements from its origin. As the orchestrator, ALIBY uses components from all other sub-modules. Given that only ALIBY requires network tools, which just so happen to be much slower to install than all the rest, modularising the components makes automated testing of individual components faster. The reasoning behind this split is as follows: *postprocessor* can analyse any time series dataset and produce figures without the need of networking, image analysis tools (ALIBY orchestration module) nor deep learning tools. BABY is its own independent Python library. Both ALIBY (orchestration) and *postprocessor* use the are built on top of agora, whose purpose is providing tools to build more complex components.

Dividing our images in tiles offers two big advantages. In addition to reducing the amount of memory necessary, it speeds up segmentation and tracking; tracking, and, more notably, segmentation, do not scale well as image size increases. Convolutional Neural Networks operate on sets of contiguous pixels, because, counterintuitively, processing 9 5x5 matrices is faster than one 15x15 matrix, explaining why tiling is an important enabler of high-speed computing.

The objectives of *tiler* are selecting regions of interest for analysis from an image, tracking and correcting the drift of the microscope stage over time, handling errors. It also provides low-level access to tiled images for ALIBY's image-processing steps.

In addition to the tasks it actively performs, *tiler* is a middle-level tool that provides tiled images to the relevant steps. It does not decide whether the data is remote or local, that task is delegated to a different family of low-level tools.

In the future, multiple tilers can extend support to other experimental setups; its main goal is thus removing sections of an input image that are not informative. One particular case where this modification would be useful is when tiling images with no pillar traps, in which tiling means selecting the areas that can contain cells. As long as a *tiler* instance respects the interface - providing subsections of the original image - it is replaceable by any other tiling tools.

We developed an algorithm for self-templating tiling

To find tiles within a position we use a two-step process: First we analyse the original bright field image to produce the template of a tile, then we fit this template on the original image. Finding tiles is a time-consuming process when manually done, and can be straightforwardly automated using a combination of texture-based and algorithmic heuristics. Its main problems arise when the boundaries of foreground objects are not separated background, which is likely to happen, as microfluidic experiments are an artisan-like craft that is still prone to technical issues such as loading too many cells or debris accumulation. To bypass these issues when defining

tile locations in a trap-based device, we limit the use of texture-based algorithms to find a template - ideally an empty trap - which is then fit to the original image to find the definitive tile locations.

The main quality of most of our bright field images is the repetition of a pattern of the tile locations. For all experiments within this thesis we imaged Polydimethylsiloxane (PDMS) devices with pillars repeated in a regular pattern (Figure 4.3A). I have tested the tiling algorithm with different trap-based designs; as long a non-overlapping repeating pattern exists, few to no parameter adjustments is all that is needed for this type of tiling to work.

For the first step, we use texture-based segmentation to split our image into two different textures. In computer vision, a texture is a set of pixels that share similar values and together form patches different from their surroundings: for ALCATRAS devices the two textures we define are cells and tiles or background (Figure 4.3B). It uses information between adjacent pixels in an image to identify zones with high entropy, or heterogeneity. In our data, the zones of high entropy are the edges of cells and tiles, as they provide a transition between background and foreground regions. Lastly, we produce some tile templates from these regions, fit them to the original image and recover the best result.

We define which pixels are foreground and which ones background using the Otsu filter, a non-parametric thresholding technique that minimises intraclass variance (Otsu, 1979). In other words, it splits a distribution of pixels into two groups. This is the reason why it is important for the image to not have an excessive amount of cells: If the entropy-filtered image does not show a bimodal distribution (clear edges between background and foreground), our filter will not yield a sensible threshold.

The resulting Otsu threshold splits the image in two sets: Pixels that are edges and pixels that are not: An image whose pixels are either one or zero. We then apply a morphological closing operation to this binary image, which removes dark gaps between bright features, in other words, it fills the area inside edges, reconstructing

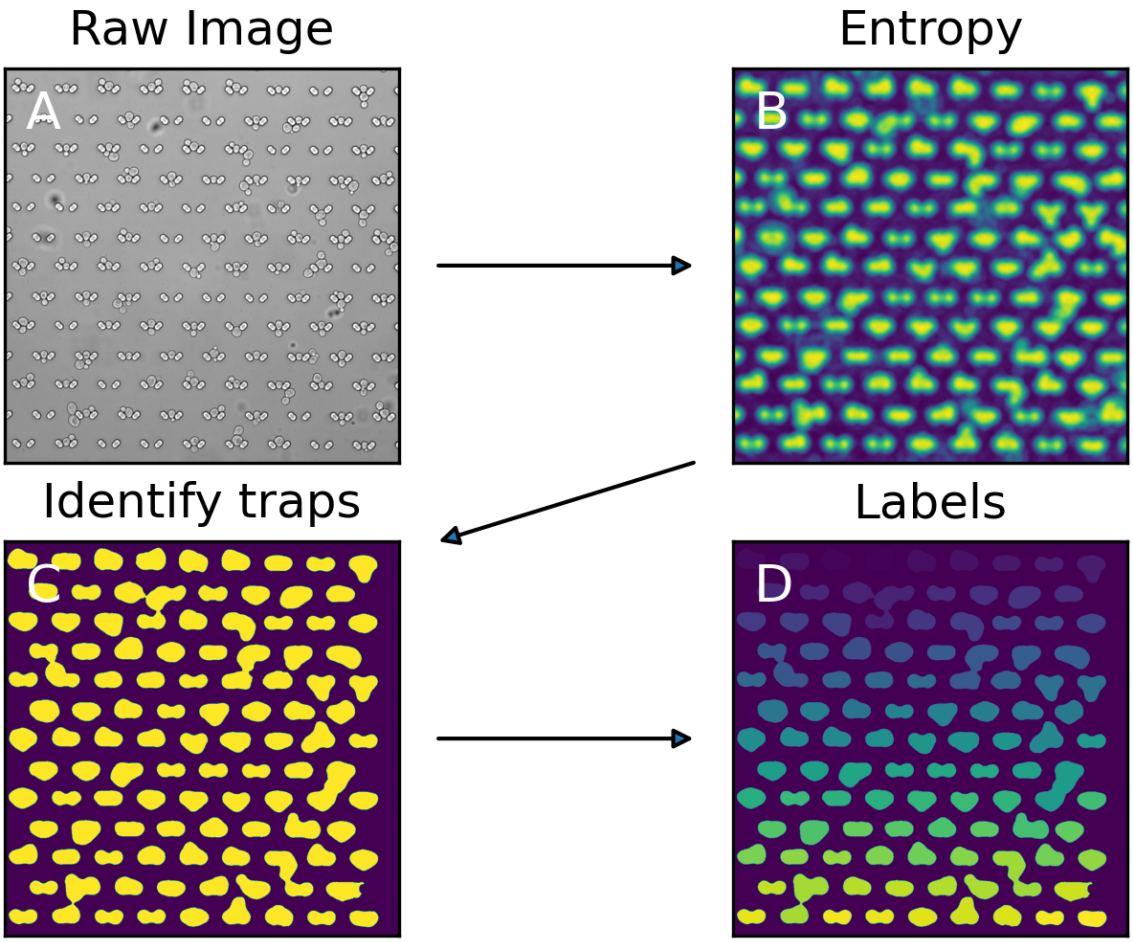

Figure 4.3: **For the first half of our tiling algorithm, we select candidates based on which we will produce a template to find a repeating pattern within our image.** For time and resource-efficient segmentation we must first identify the regions of interest where cells may become fixed and thus simplifying time-lapse imaging. The general concept is that we have a repeating pattern that we need to find - tile pillars, in the case of these ALCATRAS devices (A). We do not make any assumptions on the number, shape or cardinal orientation of these patterns. Our only assumption is that the background is homogeneous and spaces the repeating pattern. We apply a filter that calculates entropy for a group of adjacent pixels (B) - the amount of pixels we compare is proportional to total image size. Regions within the image where foreground and background meet, contain more information than regions with solely background. A cell trapped between pillars generates higher entropy values as well. Then, an Otsu filter helps discretising the entropy to set the boundaries of regions. Then we discard regions that touch the edges of the image. After this first cleaning step we obtain C: The pixels above and below the Otsu threshold are yellow and black respectively. We then assign labels to each individual region. A region is a group of consecutive pixels separated from other groups by at least one pixel in all directions (D). Colours indicate independent regions, the labels allow picking specific tiles and discard them from analysis.

the area corresponding to the union of tiles and cells as masks with pixels of value one, while the background becomes zero (See Figure 4.3C).

It is not uncommon for cells to stick to each other during the process of trapping them between pillars. To prevent these cases from affecting our tiling accuracy, we remove from the analysis tiles that do not conform arbitrary limits on tile sizes, by default somewhere between 20 and 80% the length of the framing tile. We perform this validation in two steps: First, we clean the tiles that touch the edges, as they are at risk of being incomplete (Figure 4.3C). Performing this validation step ensures that our regions contain at least one complete tile, except for the fringe case where there is a problem affecting the regularity of tiles, such as polymer bonding problems in some regions of the image.

Before the second validation step, we must label all individual regions of interest, to characterise them based on features that are not their location (Figure 4.3). After the Otsu filter, applying the trap size threshold and cleaning the borders we have the guarantee that we generate independent non-contiguous patches. This enables the selecting specific patches based on their size or location, like for instance, selecting only the patches that smaller than a certain area.

As a second validation step, we filter out objects that do not conform to the expected tile size: In Figure 4.4A we removed regions that are too big to contain a single tile or too small to be one. The former case is a common occurrence, generally produced by clusters of cells stuck between independent tiles, obstructing the background that separates these tiles; thus, after applying the entropy filter, they become a continuous non-background entity. The latter case is much less common, but can happen when tiles suffer bonding issues (e.g. tiles with only one pillar) or an object or cell stuck between tiles, but that does not touch them, so it forms a small independent cluster. By default, we select regions that cover more than 20% the tile size and less than 80%. The user define tile-size, and if necessary they can also adjust the tile-to-tile ratio.

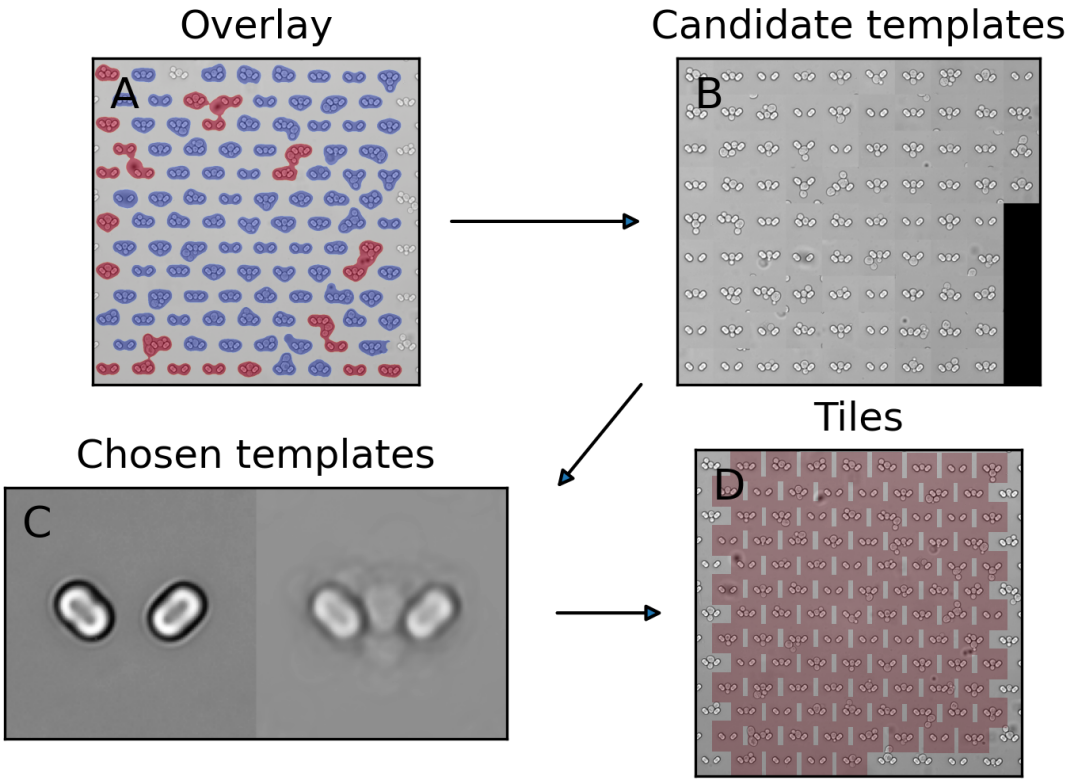

Figure 4.4: **We validate tiles to ensure we get a homogeneous set of candidates.** **A** We clean the image using a size and location criteria. By calculation major axis lengths we remove regions that are too small or too big to include a single tile. We also identify tiles that are too close to the edges and might be an incomplete tile. We then accurately predict valid tile locations. The blue overlay covers the template candidates, while the red overlay covers the rejected ones based on their major axis length and/or their proximity to the edges of an image. **B** We discarded the unsuitable tiles during the previous tile validation step, or if they directly touched the edge of the image. We extract the predicted centre of our template and fetch the region from the raw image. By calculating the centroid of the selected regions we get the approximate centre of the tile. **C** We generate two templates we aim to fit onto the original image to identify tiles. We pick the template candidate with the smallest minor axis length, (the left half of panel C) for it is more likely to contain no cell and will easily fit all tiles, occupied or not. The algorithm also averages over all valid cells, resulting in the pixel-average of all our template candidates. (panel C, right sub-panel). **D** At last, we fit both of these templates and pick the set with more tiles to obtain the final location of tiles.

We use two methods that produce one template tile each (Figure 4.4): For the first method we pick a single tile whose minor axis length (the straight line that crosses an object from one side to the opposite) is the smallest. We obtain the second template by averaging over all validated tiles; from now on I will refer to these as the minor axis and the average methods, respectively. Lastly, we fit both templates to the original image and retain the set with the largest number of tiles.

The minor axis method, as it name indicates, calculates the minor axis length of all above-threshold regions hunting for a cell-less tile. The logic is that empty tiles will generate a concavity (after entropy filtering) whereas tiles with one or more cells will have a higher convexity. This concavity-based approach works best when morphological variance between pillars is minimal - which may not be the case with manually-generated microfluidic devices. One instance in which a physical limitation affects data analysis occurs when a trap is not properly bonded - glued to the base slide of our devices (Crane et al., 2014). If it is identified as a smaller trap - but not too small - it may be picked up by the minor axis method and the template that results from the process will not be a full trap, but half one.

The averaging method's logic rests on the idea that on average most validated regions will contain similarly-looking tiles with cells. Thus averaging over them, we will result in the pixel-per-pixel average of all validated traps. It works best in images with trap heterogeneity - when there is some heterogeneity between traps, intentional or not - or overcrowded position. This method covers those fringe cases in which the minor axis method may fail.

After acquiring both templates, one as an combination of tiles and the other one as a single one, we use a peak-identifying algorithm to recover the x and y axis location of tiles in the original image. We first explore rotation and scaling hyper-parameters to guarantee the best possible fit. Rotation corrects for cases when a template is not properly oriented and the scaling of the template's and image pixel size are different, or their tiles' dimensions are different. These operations add support to devices with more than a single tile size and makes it possible for the user to provide their own

tile template; while ALIBY's design aims for automation, some researchers may prefer to use their own templates if these are available, so we maintain that as an option.

Once we have picked our rotation and scaling values, we calculate the peaks, or location in the image at which our (potentially) rotated and scaled template fits best, while filtering out any tile location that does not conform with what we consider a valid tile, such as candidate locations that are too close to the borders or to each other.

We still test multiple 90-degree rotations and pick the best fit; though unnecessary for auto-templating, it facilitates using custom templates. These operations take microseconds and are only performed once per position, so they do not have a noticeable impact on processing speed. We rotate the original image based on the best fit for rotation.

To increase generalisation, we re-scale the tile template multiple times, for user-provided templates to fit devices with traps of varying shape or size. We expect this to also improve tile identification for microfluidic devices or position containing tiles of more than one size. We again select the scale that returns the largest number of tiles.

The final - and most crucial - step of this templating algorithm is finding the local maxima in the image using the adjusted template and image. We exclude peaks that are one-third of a tile near the border of the image and within 70% of a tile of each other. Besides that, we leave the rest to a standard peak-finding function provided by *sklearn* (Pedregosa et al., 2011).

If no template yields more than 30 tile locations, we rerun the algorithm skipping the step of down-scaling before applying the entropy filter. We use the set of tiles locations that contains the largest number of tiles from all the pairs of template-generating methods and down-scaling values.

Given the lack of manual human curation, we use quantitative metrics to spot tiling errors. Regardless of the type of tile, in our device designs tiles lay at regular

distances from each other in at least one of the two cardinal axes, for instance, in the panel A of Figure 4.3 tiles are regularly spaced across both axes. This expected regularity can be reliably quantified.

There is a caveat to consider when using this metric of quality. It is important to note that position acquired in a tilted angle (i.e., rows of tiles are not horizontal nor vertical). While the tile identification algorithm will still find tiles, their horizontal and vertical regularity will decrease the further tiles alignment is from the image.

For quality assurance, we also check the distance between tile centres in both x and y axes. We add one at the location of tile centres in a matrix of zeros with m_x rows and m_y columns where m_x and m_y are the maximum centre in each corresponding axis. Then, for each axis we obtain peaks and calculate the distances between consecutive peaks. This metric works for microfluidic devices that are equally spaced in the x and x-axis It is not affected if offsets are variable along one axis, as long as the distance between tile centres is regular.

TILER TRACKS DRIFTS OVER TIME

Entire position are vulnerable to stage drifts so we need to account for that. It is the *tiler*'s job to provide the drift-adjusted tiles for any given time point. This requires the combination of tile identification and following the stage drift, which we call tile registration.

Registration is simpler than tiling. We compare contiguous images and find the x and y-axis movement it requires to provide the best fit. The strong similarity between any two consecutive time points facilitates fitting these sequences.

Our *tiler* calculates drift on a time point basis (we feed it one image at a time, instead of all the time lapse frames at the same time) to remain compatible with live experiment pipelines that may be introduced in the future. It agrees with the overarching design of the pipeline of processing images on a time point basis. It also helps to track the number of processed time points, which is useful to continue interrupted experiments.

I define tiles as uninformative if their edges extend beyond the frame by more than a third of either their length or width. If such a case happens, "NaN's" (invalid values) are provided instead, which will be passed downstream the pipeline. For indexing purposes we have to assume that the tile number remains constant over the course of an experiment; this is, our my experience, the most efficient way to deal with tiles drifting away from the image.

4.3.2 Segmentation and tracking

We use a previously-published cell segmentation and lineage prediction software at the heart of ALIBY, the Birth Annotator for Budding Yeast (BABY). For our purposes, its job is to produce cell outlines, track cell identities and predict the lineage of cells.

ALIBY's modular design allows for a drop-in replacement, as long as inputs and outputs (i.e., interfaces, in engineering terms) are consistent. As long as they use a *tiler* instance to fetch images and return cell edgemasks, labels and, optionally, lineages, we can just replace the existing BABY software with a different one. If this new network does not provide lineages, the pipeline will still work normally.

We use a modified version of BABY for compatibility and improved version control, which arose from the original work (Pietsch et al., 2022).

We include within ALIBY a *process* that wraps BABY's core, and an associated *parameters* class that converts it to the *process-parameters* convention to achieve consistency between all pipeline steps. Thus, we can avoid manually specifying BABY's parameters, which would disagree with the rest of the pipeline.

4.3.3 Extraction

Extraction converts image tiles and cell masks into a set of metrics, usually one metric per mask. The core methods of the "Extractor" class expect us to provide both tile images and masks. To acquire tile images it uses a *tiler* class to handle their acquisition, for masks it either receives them directly or fetches them from the

hdf5 results file; if we are running segmentation and extraction concurrently we pass the masks directly, but if we are running extraction after segmentation we obtain them from the results file.

The *extractor* performs parameter validation by selecting the channels it receives as parameters and an order based on the available ones for a given image. If its parameters indicate a channel that is absent in the loaded dataset, it warns the user, but ignores the channel and processes the remaining branches of the extraction tree as usual.

We extract information from masks and images in two ways, "simple" and "composite": Simple extraction uses a single stack of images and n cell masks to produce either one or n quantities (Figure 4.5); n numbers when the metric is cell-specific and one number when it is tile-specific (or, more accurately, tile-specific). Composite extraction occurs when we convert two or more images into a new one and, using a set of cell masks, apply simple extraction to this new artificial image (Figure 4.6). One instance in which this is useful is when combining fluorescence signals from multiple channels; it can also be used to normalise the values of one channel using another.

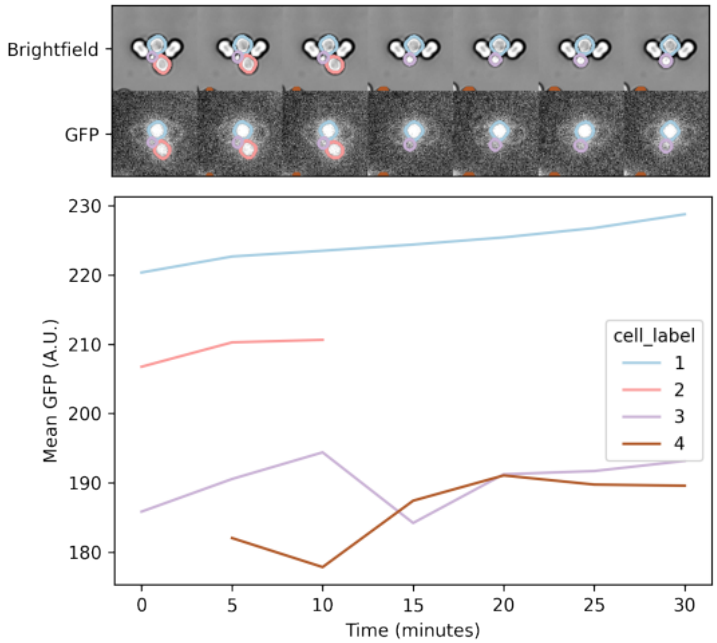

Figure 4.5: **Simple extraction combines information of masks and labels with images to produce metrics for individual cells over the time points during which those cells are present.** The top images show bright field and Green Fluorescent Protein (GFP) images. I obtained outlines from the bright field images using a segmentation model and then calculated the mean in the maximal projection fluorescence image over the pixels inside each outline. For clarity, I plot a single tile over time, but this operation occurs for all identified tiles in an image.

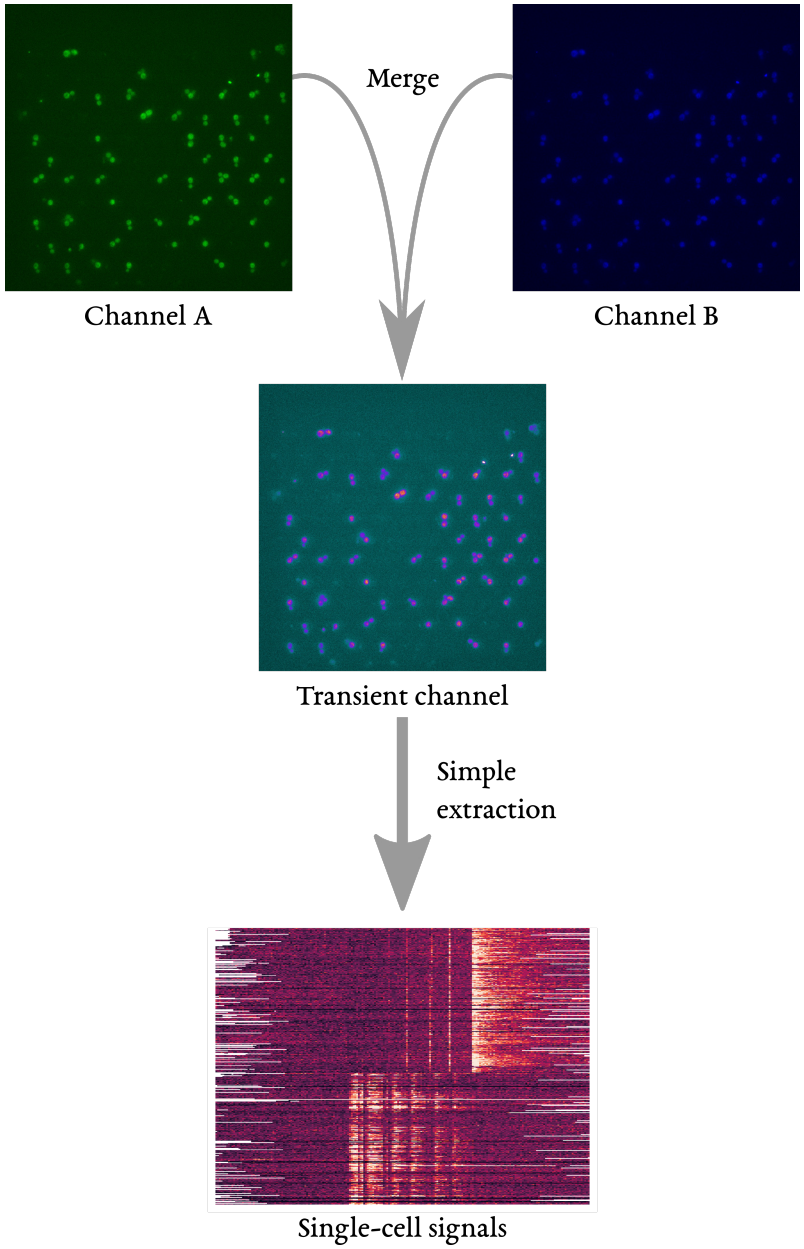

Figure 4.6: **For compound extraction we merge two channels through a mathematical operation such as addition or division, creating a new transient channel to which we apply simple extraction.** Only the numerical results - not the transient image(s) - end up in the long-term storage file.

Tree-based extraction optimises network and processing resources

Given that we apply multiple operations on the dataset, the core extraction follows a 3-level tree structure to avoid loading data more than once and facilitate extending functionality (Figure 4.3.3). Channels are the root level, so that the position are only fetched once.

The second level is a reduction algorithm, it indicates how will we collapse our image stack of different z-sections into a uni-dimensional image, when applicable. z-sections are necessary to image cells regardless of drifts across the third dimension, and also to capture intracellular components even if they are out of focus. I reduce this additional dimension selecting the maximum pixel value because the maximum fluorescence of a protein aggregate may be in a different z-plane. While some metrics can be performed on 3-dimensional images, it is more expensive computationally and results are practically the same (data not shown).

The third and last level is the metric, which indicates the specific mask-image operation we will perform over a tile.

The extraction tree is a compact way to optimise execution. To extract all datasets from all single cells, the *extractor* object transverses the tree and produces a matrix of cells versus time for each leaf node. Extraction always follows the channel, z-reduction and metric convention.

I define a branch as a combination of one element for each level in this tree structure, from root to leaves. As an example, let us define a given branch with nodes GFP in the channel level, *max projection* as reduction and *mean* as metric. In this case ALIBY would fetch the GFP channel, then calculate the maximum value for each pixel across all z-stacks and finally calculates the mean of the resulting 2-D image. We apply this procedure using each cell outline and the tile image.

At the root of the tree we find the channel level. This level instructs the tiler from which channel it is to fetch tiles. There is one special case: for metrics that do not need a channel, quantification arises instead from the cell mask. These metrics,

such as volume and area, all share the same second level "general", for whom the reduction algorithm is *None*.

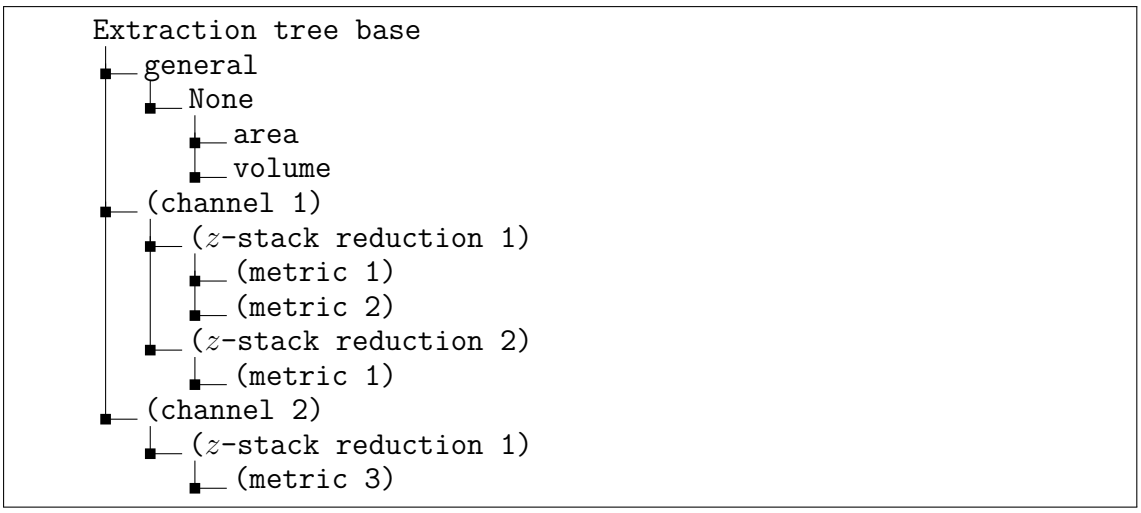

Figure 4.7: **Extraction tree structure.** Given that it is the main result of segmentation, we assume that the extraction tree will always contain some mask-derived data, such as area or volume. This also works as a way to easily know when cells are present or not. Not all branches of an extraction tree would work for this purpose because some metrics can lead to mathematically undefined numbers, whereas the sum of pixels in a mask of at least one pixel will always result in a natural number, thus will never be undefine and is the most reliable method to account for the presence of a cell.

The second level indicates the reduction algorithm used to collapse the z-stacks into a single flat image. There are multiple ways to reduce these stacks, by default we use a maximum projection (i.e., the pixel with the highest value across all stacks). Other straightforward alternatives are calculating the mean value or always selecting the stack in the middle.

There are multiple ways z-stack reduction can improve. More advanced and potentially useful options are calculating entropy or signal-to-noise ratio and selecting the desired layer. An extra level of complexity can result from applying this same approach to regions of the images, but the parameterisation of such approach would be a challenge.

The third level comprises multiple cell-level or tile-level atomic operations. We define atomic operations as functions that merge a tile and one or more cell masks to produce a number; most of the time they produce one number per mask. In the cases in which the metric yields a single output per tile, such as the median of a

tile's background, the resulting data frame is not single cell indexed, but single-tile. These are not informative of cell dynamics, but they are technical controls and can complement other metrics in additional processing downstream in the pipeline.

ALIBY provides sensible defaults for standard experiments

At some point during the data processing it is necessary to convert images into quantities to cluster and/or compare cells. In a previous subsection, I defined as extraction the process of translating fluorescence images and masks-label sets for individual cells into quantities for their comparison and analysis. There are many ways in which such transformation can occur, and that is where the concept of extraction metrics comes in.

I define extraction metrics as a set of functions that take at least one mask as an input, obtained during segmentation and mix it with images of additional channels to quantify intracellular phenomena, such as protein intensity, metrics for every segmented cell. In most atomic extraction operations, we select the pixels corresponding to mask location and apply numerical metrics on them, such as mean or standard deviation of pixel values inside a cell.

Given that the step of loading functions has practically no computational cost, we load all available extraction functions in an automated fashion, but only call them when requested within the extraction parameters. This is to maintain consistency of dynamic method loading across ALIBY's levels.

I define "general" metrics as the ones arising from masks. To extract these we do not need to download any image, the pre-existing masks should suffice. For the interest of developers, it is worth nothing that these mask-only functions are extensible, thus we can develop any other function that works on an object represented as a Boolean matrix. I chose this representation for masks due to its computational efficiency (by using less memory) and speed (by delegating masking operations to C code and thus highly efficient).

The simplest single cell operations are the ones that only require a mask. These produce properties from the predicted cell outline. The default ones are: total cell area, which is just the number of mask pixels, it indicates the area of the flat mask. For more accurate predictions we approximate the volume by assuming that the cell is an ellipse, the mask is the median plane of the ellipsoid and there is rotational symmetry around the major axis.

We obtain the mean and median of pixels inside a cell, which give an overview of the relative amount of fluorophore (and thus tagged protein) inside the cell. For protein aggregation and localisation inside cells we use basic and compound metrics to identity when our protein of interest is aggregating inside the cell: "max5", the mean of the five pixels with the highest intensity; its area-normalised counterpart, "max2p5pc", which is the mean of the top 2.5% pixels.

The last default metrics for protein localisation result from a combination of simple ones. They are the protein localisation metrics divided by the cell's median pixel intensity (Cai et al., 2008). This is, in our experience, the most informative metric from the localisation ones, as it performs a normalisation for the protein within the cell. It fares better than the top five pixels or 2.5% when looking at cells with high variability in fluorophore abundance or if fluorescence is not homogeneous within a position. We store them all because there might have been unforeseen cases where one metric performs better than another. Table 4.1 shows all default extraction metrics and the justification for them being defaults.

No previous studies test z-stack reduction methods, but from previous works in our group z-seems to be a sensible choice (Granados et al., 2018a). It is likely to vary depending on the image acquisition settings and even the fluorophore abundance and *reporter*-specific properties. Some metrics seem to have a higher signal-to-noise ratio depending on these features, so we strive to provide a standard set of metrics and make a decision based on the results.

Table 4.1: List of default extraction metrics. An extraction metrics is a function that takes a mask and a tile and returns a quantity. It is is a formalisation of the way we trasform whole images into data points. The columns are a short name (or identifier, which can be used to find their implementation in the code base), long name (a human readable name) and its purpose. Characterisation metrics characterise cell growth and shape, intensity metrics quantify amounts of intracellular probes, localisation metrics are used to measure inhomogeneity of the probe and technical metrics are used to quantify artefacts and to generate more complex metrics that correct for them.

Short name	Long Name	Purpose
area	Cell mask area	Characterisation
volume	Volume (derived from flat mask)	Characterisation
eccentricity	Eccentricity	Characterisation
mean	Mean	Intensity
median	Median	Intensity
max5	Mean of Maximum 5 pixels	Localisation
max2p5pc	Mean of top 2.5% pixels	Localisation
imBackground	Image background	Technical

Other complex operations: background subtraction and multi-channel processing

Not all imaging needs are satisfied by quantifying cell information from raw data images. Correction of imaging artefacts and multi-channel processing are important software features that increase the reliability of the final results. Both these processing steps are performed within extraction because afterwards there is no access to images nor tiles (by design).

Besides the direct processing of imagess, ALIBY performs artefact correction through background subtraction. To correct for background noise, we calculate the median value of all pixel not belonging to any cell. This number is subtracted from every pixel in the original channel and the result is stored as in a different channel branch. Negative values are passed as-is and dealt with downstream.

The reasoning behind storing both corrected and uncorrected extractions is based on the empirical knowledge that in some experimental setups, background subtraction can be a hindrance instead of a benefit. This way scientists accessing the data can

decide which data (with or without subtraction) is more appropiate, on a case-per-case basis.

Some fluorescence signals require multiple images to work, such as ratiometric probes (e.g., pHluorin, a GFP-derived fluorophore whose intensity at different wavelengths correlates with pH) (Mahon, 2011). We can merge channels to create new ones or mix the data of multiple fluorophores when using multiple fluorescent probes. Channels are combined before extracting the data and adding the results as a new branch of extraction trees (for example, the one seen in the top half of Figure 4.6).

ALIBY processes these new "channels" exactly like any other channel. It applies the same reduction operations and extraction metrics. Applying this intermediate background subtraction step produces two channels in total: The original "raw" channel and the background subtraction channel.

4.3.4 Postprocessing

Postprocessing is a *step* composed of operations that only work on other data, not images. Once the image analysis section of the pipeline finishes,the computational bottleneck transitions from network transfer speeds or neural network operations into raw computing power. This step is faster than segmentation and extraction. Faster than the former because the convolutions performed by the neural network are the most expensive process in the entire pipeline.

Processes are independent from each other, yet we also have the option to automatically load them within the pipeline or call them independently as a function. Having these alternatives is advantageous: the former method of using them facilitates automation while the latter eases the development and individual testing of processes. Their independence also makes these methods reusable for analysing data from other sources, such as plate reader experiments, or in an entirely independently-developed pipeline. The only assumption of processes is that they work on a matrix-like dataset. Rows are individual cells and columns are time points.

Postprocesses can also be interpreted (and, in practice, used) as functions. I strongly suggest against treating them as mere functions though, as it forgoes the reproducibility earned by having an associated *parameters* class. The *postprocessABC* is a base class and extends the *process* class. It is a base class because it provides a skeleton of methods and attributes that multiple other classes can inherited and implement. It extends the process class because it inherits methods and attributes from it, and provides additional functionality.

The way *postprocessABC* extends the standard *process* by adding an "*as function*" method -a function that belongs exclusively to a class -that returns the equivalent function, to which we can pass arguments. This trade-off is acceptable for data exploration, development and testing, but I highly discourage its use for actual analysis; it is easy to forget that we modified some data and in doing so reproducibility is lost.

Lineage relationships within our microscopy experiments help extract insightful datasets

The distinguishing characteristic of our data when compared to other microscopy time-lapses is the mother-daughter relationship between cells. Previous work indicates that most of the biomass growth for a given mother-bud blob occurs in the bud (Hartwell & Unger, 1977). There are multiple methods to quantify growth, some based on the change in volume or biomass produced, others based on the number of births (Soifer et al., 2016) Our objective is to not only to quantify the signal and growth of single cells, but to use those signals and combine them with data of mothers and daughters.

While essential for multiple growth-related metrics, we should aim to differentiate when this lineage-based filtering is essential and when it is not. For instance, to study single cell responses to environmental cues using transcription factors that localise into the nucleus, lineage information may not be relevant. Oppositely, we

require lineage information to study cell cycle processes and their interaction to internal signalling.

Two *postprocessing* steps polish the data

Prepost are *postprocesses* executed before the basic *postprocessing* routines. These are in charge of correcting errors caused the limitations of our deep and machine learning tools, as well as selecting our *de facto* group of cells to study. Information derived from prepost ensures the consistency of cells for a given set of parameters. We can apply the merge and selection processes to any dataset with columns as time points in a consecutive manner, but they do not change the data indices. Different datasets will arise from using a variety of selection criteria, but a well-defined selection of cells maximises the comparability between multiple subsets of cells.

In order to maximise the number of long tracks we retrieve from a given position, we must merge tracklets before picking them, that is, filtering-out tracklets that do not follow our criteria of choice, generally presence of a cell during more than 80% of the experiment. We can salvage tracklets cut in half if we merge them before applying picking processes. If we apply pick before merging, it is unlikely that the latter will have any effect, for picking cleans most of the cells, among which we often find accidentally-split tracklets.

These processes do not modify the data directly, but write their results in a separate section. As explained at the beginning, we never modify data written into memory unless re-running the pipeline from a specific stage. All this is to maintain data reproducibility as much as possible.

1. *Merger* identifies and fixes tracking errors using time-based information

   Despite its high accuracy, our segmentation and tracking algorithms may commit mistakes that we aim correct once image processing has ended. For instance, cell pivoting can cause a tracking error, resulting in two tracklets that must be merged (refer to the Tracking chapter for an example). During *post-*

processing we can exploit the inter-time relationships in our data that we have overlooked thus far. We use cell mask volume to identify pairs of tracklets that are likely to be the same cell, but were not originally recognised as such. For a given tracklet, we predict its volume at the next time point and if we find another one that starts in that specific time point and finds itself within a tolerance range of 10% around the prediction of the former tracklet, we assume they are the same cell. When handling tracklet merges we limit the information we keep trace of to the merge events. To preserve the integrity of the raw data and maintain reproducibility, we only record the cell identities of merged tracklets.

We first filter-out the cells present for less than five time points, because we use a filter that requires at least five continuous time points to smooth the signal, and we look for growing cells. We also apply a threshold for cells that are growing, even if slightly, on average. We do not aim to identify whether cells are growing or not, instead we remove tracklets with negative trajectories, which in most cases (the transition period during osmotic shock being an exception) are erroneous.

To do so we look at the first and last time points in which a cell was present. We then select all pair of tracklets whose last and first values are in contiguous time points, and calculate the difference between all available pairs of contiguous tracklets. Afterwards we apply a Savitzky-Golay filter and a linear fit to the first tracklet from every pair. Using this fit we predict the volume of that cell in the next time point, and if there is a cell that appeared for the first time during that next point and its volume is within ten percent of the predicted volume, we assume those two are the same track and merge them. We handle the fringe cases, namely when two cells are within the region of acceptance and their first appearance occurs at that specific time point, by choosing the cell with the area that is closest to the predicted one.

Merging tracklets can only happen once, for after we merge all tracklets that meet the aforementioned set of requirements, more cannot arise. The linear predictions of merged tracks do not change because we ignore cells present for less than five time points and the linear regression does not go further back than that. In other words, tracklet merging does not affect the trajectory of their linear prediction at the edges.

When calculating the sequence of merge operations, we must implement it backwards in time to avoid generating tracklet inconsistencies. I define orphan tracklets as being able to merge to a second one, but the latter was previously incorporated into a third tracklet. By performing all tracklets backwards in time, we remove the possibility of this orphaning happening, and given that we solve tracklet-identity conflicts beforehand, there is no possibility of making an orphan of a tracklet.

2. *Picker* filters-out uninformative cells

To select the cells present with lineage information, we need to remove additional objects that are not trapped cells or their buds. The source of these can vary from debris identified as cells to actual dead cells that are of no use to our analyses. We then use growth rate as a complementary criterion to remove buds predicted by BABY's lineage assignment algorithm. Using growth rate to filter-out debris has caveats, as growth interruption can be an expected outcome of experiments.

In addition to the housekeeping function of removing noisy tracklets, picker can also select cells based on their signals. These signals are generally (but not necessarily) bright field derived-ones. The basic selection method is tracklets present for a given percentage of an experiment. This is similar but slightly different to tracklets present for a given number of time points. The first criteria is more inclusive and thus the default, as it is possible (and not uncommon) that the tracklet between first and last appearance is not continuous.

Picker can select cells based on their average expression of a fluorophore, or the nuclear localisation of it, provided we feed it the adequate signal. Another useful picking method is selecting the top ten cells - or even a fraction, such as twenty percent, for that matter - of a position, based on the signal we feed it. It can even use a combination of multiple signals, allowing for more flexible and active interactions with the data.

It can also select cells based on lineage information. This is particularly useful when trying to select mothers and their associated daughters over the course of an experiment. One example of the mother and its daughters are visible in Figure 4.8, on the bottom left panel.

Similar to *merger*, picking can iteratively using multiple criteria, until there are no more cells to pick. The main difference between the two of them is that *picker* can apply multiple intersections (selection of cells) and unions (combination of multiple sets of cells), while *merger* applies the same operation with no way back. While multiple consecutive merge events do not make any difference, multiple applications of picker are common, as it is part of its design and is considerably more variable.

When comparing features that do not share the exact same set of cells, we retain the intersection. The number of cells we retain decreases if we are reducing the amount of valid values as we process it. For example, applying a smoothing average reduces our data size by $n - 1$ where n is our smoothing window, assuming all internal values (values between the edge) are valid. The upside is that the data is more highly curated as we include more processes, assuming those processes are sensible. Given that we register the entire processing pipeline as metadata, can always trace-back in processing and recover every single step.

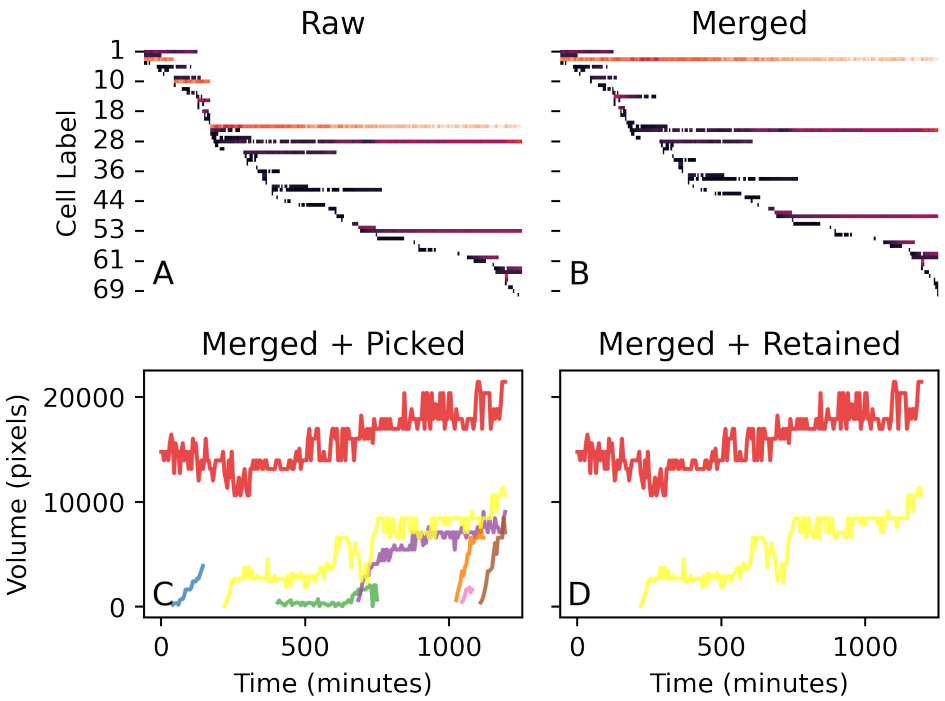

Figure 4.8: **We preprocess volume tracklets to obtain cleaner results.** We show the same tile in all panels, with different filters applied. Panel A is the raw data, exactly as BABY produces it. Panel B is the result of applying the merge process, a tracklet that was split in three in **A** comes together into one. We then applied a lineage-picking *process* to obtain panel C, where each tracklet is an individual cell. Alternatively, we can also use a threshold for retained cells - the ones that are present for more than eighty percent of the experiment - which results in panel D. We omit specifying cell labels in the bottom panels for the sake of simplicity.

I set standardised processes, yet *postprocessing* is composable

For consistency and automated tests, all these processes follow the same *process-parameters* architecture. This facilitates automated testing of all this features using simulated data.

Detrending and smoothing operations, such as **savgol** (Savitzky-Golay filter) or **gpsignal** (Gaussian processes with smoothing and estimation of change rate) can help discern between biological or technical noise and signal. The former process is a fast smoothing method, while the latter is a more complex process that not only smooths but also provides statistical metrics and rate of change at every time point.

gpsignal requires exploring the hyper-parameter space and thus, while generally better than **savgol**, it is not done unless the experiment passes quality and results controls, as it increases processing time by about fifty percent.

The general-purpose signals are useful for both bright field and fluorescence-derived signals. With **dsignal** we calculate the change in values on a tracklet over consecutive time points. To dampen big changes in the result we can smooth the signal before hand using a sliding window. This smoothing is independent of other detrending or smoothing operations.

Using **findpeaks** (peak-finding algorithm) we identify the highest and/or lowest time points of a signal. Depending on the type of signal this can mean different things: If we are looking at a mean fluorescence signal with a fluorophore-tagged protein, for instance, it indicates the time points at which cells had the highest volume of protein in their vicinity. Alternatively, if we are looking at *msn2*, a transcription factor that pulses - moves stochastically into the nucleus (Dalal et al., 2014) - under certain conditions, peaks will indicate the moments at which the cell is relaying a general stress message.

We also developed **bud metric**, which uses lineage to merge all daughter tracklets for a given mother into one - generally non-continuous - tracklet. If more than one daughter is present at any given time point, the most recent one takes priority, because that is the bud that might still share cytosolic content with the mother. This means that we follow the new bud's phenotype until it divides and another one appears. This metric is usually discontinuous because there is often a period between cytokinesis - after which the daughter cell is usually flushed away - and identification of the new bud as an independent entity. It can use either birth events or cytokinesis predictions to determine at which time point we stop accounting for the bud's signal, as it and its mother stop sharing cytosol and the bud becomes independent.

Combining multiple *process*/es/ we can extract additional bits of information. For example, using **dsignal** twice - analogous to a second derivative - and then **findpeaks** on **max5_ med** signal we can find the point in time at which the rate

of protein aggregation reached its peak for any given cell. Alternatively, we can also use **dsignal** once on a cell volume signal and then **bud metric** to obtain the growth rate (measured as rate of change in volume) to recover the bud's volume of all mothers any given time point.

Not all *postprocess*/es/ run by default. Also, there are additional set to use by default. For the sake of brevity, those are further explored within ALIBY's documentation (https://aliby.readthedocs.io/en/latest/). Table 4.1 summarises the standard processes ALIBY uses when run with default parameters.

Table 4.2: Table with default processes. First column shows abbreviated name.

Name	Full Name	Purpose
dsignal	Signal change over time	Calculate changes in a given signal over time to visualise cell's response rates to environmental changes.
findpeaks	Identification of local peaks	Identify the time at which intracellular signalling occurred.
budding	BABY's raw budding event prediction	Generate the first time point at which a bud was visible. It is related to birth events.
savgol	Smoothing using Savitsky-Golay filter	Smooth a metric over time to estimate the general trend.
bud metric	Metric of bud for a given mother	Get a metric of the current bud for a given mother.

4.4 Discussion

4.4.1 Design policies

Focus on standardisation and reproducibility

One of my main goals is ensuring analyses are reproducible and have a sensible set of defaults. Each analysis result contains all parameters required to recreate itself. Additionally, the metadata contains the software version and, if they exist, comments made by the performer of the experiment. I also include the identifier

from our public repository. All this makes individual files completely independent of each other avoiding cross-dependency issues.

We can resume experiments whose processing suffered an interruption, as the resulting file contains all necessary information on how it was originally processed. This approach also allows for side-by-side comparison between independent technical repeats of a given experiment.

1. My *process-parameters* schema ensures consistency at multiple scales

   I define a *process* as an abstract class that requires a *parameters* instance as input, validates its values and processes any incoming data following the values instructions. In regards to specific implementation, a strict requirement is that any *process* must contain an abstract *run* method, taking data as an input and, sometimes, additional arguments. This method is an abstraction that encapsulates every action of a given *process*. In other words, it is the only way a *process* alter any data. The roles of processes are data and parameters validation, to then operate on the data. There are multiple advantages of following this structure, but the biggest one is that every action undertaken by *process* is traceable by its associated *parameters*, enabling us to go back and replicate a processing pipeline from an existing one.

   A *parameters* class contains information its associated *process* requires to analyse data. It is akin to a set of instructions that the process understands and interprets. *parameters* can be interchangeable with dictionaries or *yaml* files. These are usually used for configurations, because they are human-legible *yaml* files are a superset of the more common *json* files, so any *yaml* file is convertible to *json*. We can generate bespoke scripts for both, but *yaml* files are easier to edit, as they use new lines and indentation to determine hierarchies and relationships between atomic data, whereas *json* depends on brackets for different hierarchy levels.

   A *process* class interprets and executes the instructions contained in its associated *parameters*'s class. It first validates the parameters received and there

are three possible outcomes: If these parameters work without a hitch, the process will run with no warnings; if there are inconsistencies, but they are solvable or non-essential for its basic functionality, the process will still run but with warnings; lastly, if the parameters contain a critical inconsistency, that it cannot ignore, the *process* will interrupt the pipeline.

This *process-parameters* abstraction is an intentional design choice. It eases the development of more complex use cases, such as fetching additional information from a multitude of possible sources (e.g. a remote server or a file) before processing data. It enforces a stringent structure to make processes resistant to code or data structure evolution. The tradeoff is increased overhead, but this could be solved by a wrapper that converts standard functions into *process-parameters* pairs, increasing ALIBY's extensibility by simplifying the addition of new processes.

Processes generally relay the writing routines to an additional class, *writer*. This allows for type-specific adjustments depending on the shape of *process*'s results. The Writer deals with how to reshape this data to save it into the *hdf5* file (Figure 4.9).

I chose to make *process* classes independent from their *writer* and thus the *process-parameters* pairs are difficult to modify by design. This design decision aims to preserve data consistency and reproducibility.

Balance of extensibility and automation

Even if my main design goal for ALIBY is analysing long-term microscopy videos, it works on shorter use cases equally. Only one caveat exists for short experiments, and it is that performance of lineage assignment depends on multiple time points to work, thus it may underperform for snapshots or short time-lapses.

Data processing must also be compatible with multiple data sources if we aim to appeal to the scientific community; though originally designed to work on trap-based setups, I now works on other kinds of tiling. Thus I developed three main

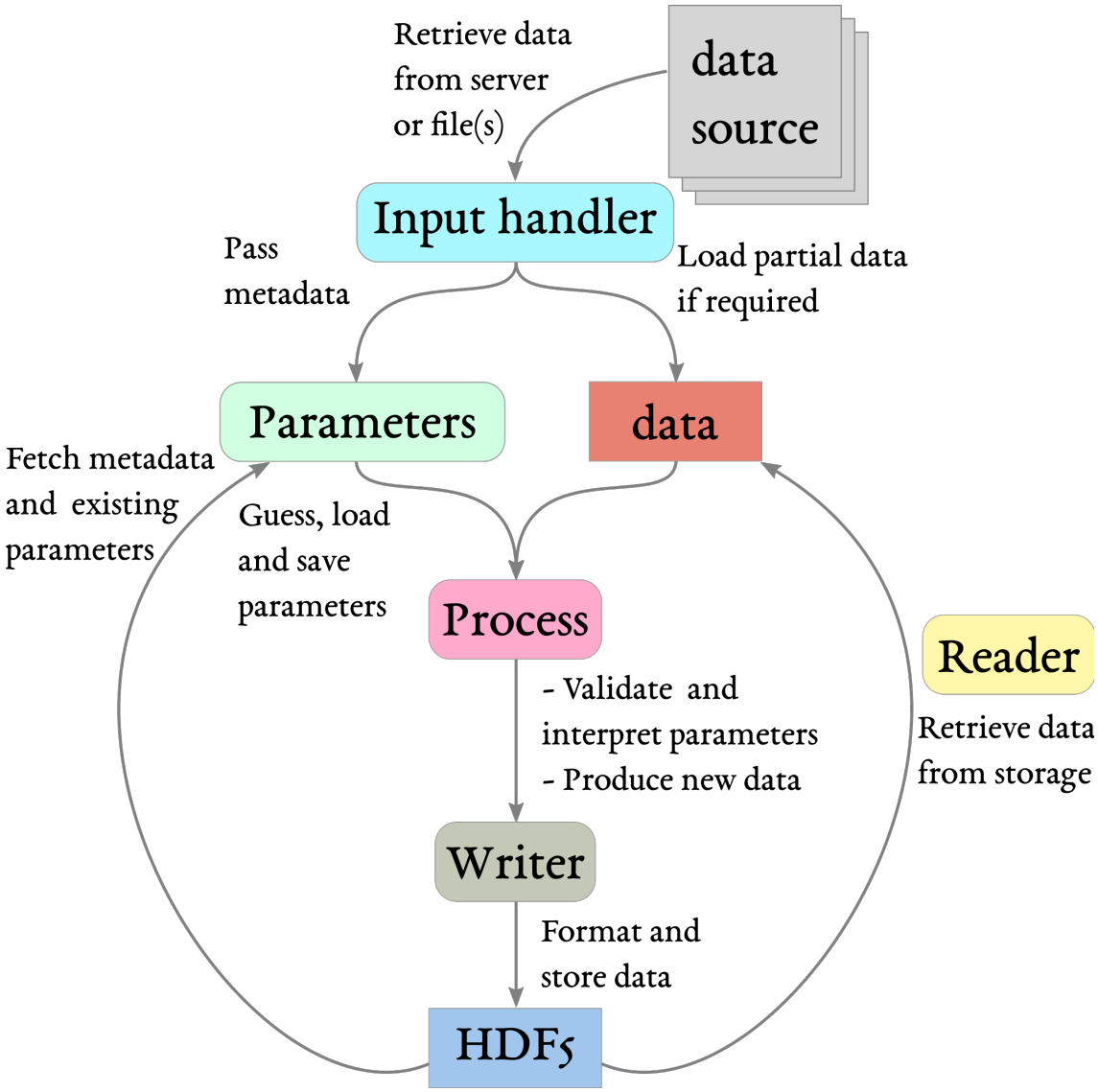

Figure 4.9: **The components of ALIBY follow a schema consistent at all over the code base.** The core of this design pattern are a *process-parameters* pair. All components that operate on data follow the same principle: We load a process using a set of *parameters*, and then feed it data that is handed to a *writer* that stores it in a long-term storage file. Depending on the overarching pipeline, we may then use this stored data to feed additional *process-parameters* couples and further the analyses.

data formats to take as an input: From raw local files, folders or remote servers; any of them must ideally include basic metadata. There is no standardisation for bioimage formats, although that is changing recently thanks to modern microscopy formats, such as OME-TIFF - an image format that combines TIFF images and XML metadata. ALIBY translates our custom metadata into a simple hash-like structure, which we use when placing metadata with results.

As a design decision, I include default values for every pipeline step. This lowers the barrier for new users and simplifies automated tests, the disadvantage is inaccurate results if the data is very different from ours. Default parameters aim to be a general guess and should be overridden as familiarity with the software increases. I thus offer out-of-the-box functionality and flexibility to explore parameters based on the experimental setup and biological context.

I assume input images arrive as consecutive time points to remain compatible to future live-processing setups. Our storage structures are dynamic in both cells and duration to cope with the uncertainty about the duration of our experiments. Not making *a priori* assumptions like this facilitates the incorporation of feedbacks for live-processing, because it can "pause" the software while the microscope is imaging.

Large code bases reach a level of complexity at which the development challenge becomes complexity management, not solving the task itself. With that in mind, we group tools based on whether they at the top level of processing (so-called pipeline *step*/s/, in charge of complicated groups of tasks, and interact with other abstract components) or low-level (solve a single well-defined task, and directly interact with the data), such as Input/Output (I/O) or networking. In addition to these groupings other technical factors require accommodation. For instance, the networking module requires compilation in its host computer, slowing down automated testing. We then placed the networking components at the top level, so it is only installed when strictly necessary.

ALIBY's modularity enables developers to tweak or replace specific components without propagating unintended consequences to other sections. Once familiar with the structure, multiple developers can work on independent submodules without obstructing each other. Automatically-generated documentation and automated testing further enhances accessibility.

Regarding automatically generated reports for every processed experiment, I incorporated metrics to assess their apparent performance. The synthesis of experimental results into a reduced set of useful values provides a way to easily identify high from low-quality experiments. Once low quality experiments are spotted, removing them becomes justifiable, maintaining overall database hygiene and sustainability.

Automatic loading of functions

Given that ALIBY aims to use a standard set of *steps*, these will behave with consistent I/O, we avoid calling these by name and opt instead to automate their import: instead of explicitly referencing indificual functions, they are retrieved sfrom a specific location within ALIBY. Automatically loading functions and *process*es forces developers to comply with fixed designs (e.g., all *process*es requiring a *run* method). If a developer wants to add a new function, they need only to write it in a documented location and assign it a new name. ALIBY automatically loads it when its name is in the parameters of the orchestrator (i.e., *extractor* or *postprocessor*). These design choices result in the best of both worlds: all implemented functions are available to load, but the wrapper of those functions will only load and use them when appropriately provided via a *parameters* instance.

Expanding on the concept of automatic loading, I abstracted all repetitive tasks with consistent interfaces (I/O); repeating code manually makes the entire pipeline difficult to modify and harder to maintain, ALIBY instead accesses existing functions from a specific location within *postprocessor*. A validation step may at this step be necessary, for example, when matching a channel specified within the extraction *parameters* to channels available in the input images. Extraction functions, that

transform one or more masks and an image to single values, exemplify a consistent interface. All metrics share input and output structures: they receive a cell mask and - optionally - an image, and output a single value. Addition of new metrics becomes trivial.

Orchestration and organisation

Once explained all individual components we are in need of a component that takes manages all other components, we define this entity as *pipeline*. It coordinates all other elements of the workflow until Postprocessing (inclusive). It has additional functions: Writing the metadata onto files, handling the decisions how to follow when dealing with existing experiments, as there are multiple courses of action: We can overwrite or not the entirety or a subset of *step*. *pipeline* is thus the main factor involved in processing reproducibility.

1. Writers store data into a long-term storage results file A *writer* is a class in charge of recording the results from steps. As a general rule, we use one per Step to further modularise them. Matches the components to their respective *writer*s in an automated (and iterative) manner, thus facilitating the addition of more steps if necessary. They move information from a processing pipeline into long-term storage, namely the *hdf5* file. They all deal with reformatting and reshaping data to optimise reading speed and reduce file sizes as much as possible.

    - *DynamicWriter*: This is the base from which most *writer* are written, computationally speaking. Writing an abstraction for our general way to write data makes inherited implementations easier to design and maintain. It facilitates the creation of new writers to add steps to the pipeline, for we only need to specify the format. Maintenance remains true as long as we avoid using deep nested inheritance (more than three levels of inheritance) or multiple inheritance (a new class inheriting from multiple other classes).

It does not account for new methods of fetching data, for that is not any *writer*'s job.

- Tiling: This *step* produces the simplest result among all pipeline steps, for this reason we did not create an independent *writer* but manually record its results into the *hdf5* file. It writes two outputs: initial position and drifts over time. The former results from tiling
- Segmentation *Writer* (*BABYWriter*): BABY produces multiple values: cell labels, lineage and outlines, and its *writer* must handle each of those data structures in a different way. We represent lineage as pairs of cell labels, thus they can be easily stored as standard cell labels with an additional column for the labels. While cell outlines are smaller in size than cell masks - as they can be stored as sparse matrices.
- The *writer* for *extractor* and *postprocessor* (named *writer* within the code base): Both *step* produce the same type of output: one or multiple matrices where individual cells (or in some particular cases, tiles) are rows and time points are columns, we can then have them share the same *writer*.
- *StateWriter*: I developed one last *writer* to keep track of BABY's outputs that are not passed onto files, as well as to define a place to flag experiments for curation and debugging. It keeps tabs on data that is not saved elsewhere in files, namely the status of BABY, for it tracks information of past time points as it processes new ones.

Pipeline orchestrates all steps and *writers*

In addition to containing and handling all other *processes*, the *pipeline* class is a process itself itself, which provides multiple advantages. The entire *pipeline* object is exportable as a dictionary or yaml file, making it simple to share entire pipelines as a single file, improving reproducibility

Due to its wrapper-like structure, *pipeline* is not a standard *process*. It may need to replace the metadata on the results file when a subset of *step* is re-run with different

parameters, thus is not constrained to a specific *writer*. It also applies filters to the datasets to fetch and early stop operations to avoid processing clogged tiles.

Grouper accumulates multiple time series

The next challenge to solve in order to achieve high-throughput as aggregating data from independent position in a logical way. In other words, scale-up the volume of data. First all positions in an experiments must be combined, and only then we can think of combining multiple experiments. The variety in design of microfluidic devices complicates the design of a system.

Multiple position may belong to the same chamber, or multiple chambers with the same strain and/or environmental condition. Despite ALIBY storing each one of them independently, during analyses, we usually group cells that either share genotype or experience the same environmental condition. The task of grouping cells in a fully automated manner based on their position alone is challenging due to the variety in devices, experimental conditions and setups that plagues time series microscopy databases.

To accumulate multiple position, I developed multiple criteria to define groups that represent a common cell type or condition). The simplest method I designed uses the position filenames, given that one filename per position is produced; this name is manually set by the experimentalist during an experiment's setup). This solution's main advantage is how it requires minimal information; its main downside is that it relies on consistent naming criteria. It also makes assumptions on the name structures, but provides in exchange an understandable group name. In the alternative approach that I designed, metadata may be used when groups were defined manually, irrespective of position names.

Combining both approaches, that is, using position names and manually-assigned grouping, I consider is the best approach. No algorithm can organise data without manual curation, or without assumptions on filenames or positions to use. My approach to accumulate records provides the best return-on-investment by assuming that the filenames determines its group unless metadata about groups is explicitly

provided. The goal of all this grouping is to reduce the burden on the analyst's workflow by providing a sensible organisation of the records.

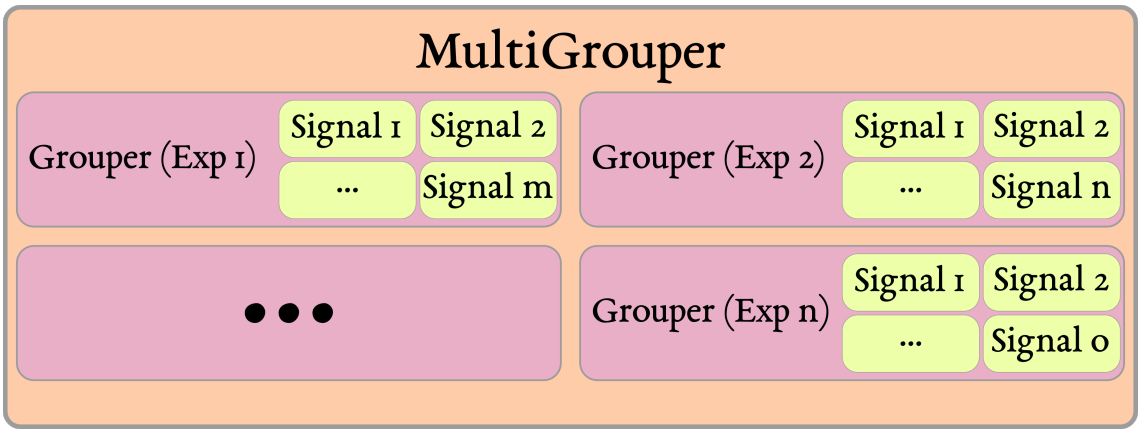

Figure 4.10: **Structure of *multigrouper*. It contains multiple *signal* to accumulate over multiple position and/or experiments.** To encapsulate an entire project into a structure that follows the same conventions as *signal* and (single-experiment) *grouper* we can use *multigrouper*, which accumulates and organise a set of experiments.

Multigrouper accumulates data of positions from multiple experiments. It embodies the final goal of ALIBY, as it allows comparing multiple independent experiments and, even more importantly, to compare records from independent biological replicates, reducing the impact of day-to-day variation. Multiple *grouper* instances comprise it, providing both fine-grained control of individual positions and access to all positions together.

Given that *signal* returns records whereby time (in minutes) is in the y-axis, concatenating records with a variety of imaging intervals intervals will lead a botched combined record that contains gaps between. The solution for these situations relies on combining multiple time points into bins, but I suggest against this approach, as it may decrease time resolution in experiments with frequent imaging. Regardless, downstream processing can remove these gaps when required, to further process or visualise the data.

As a convenience, *groupers* also provide tools to glance over general features of an experiment or multiple experiments (Figure 4.10). These tools help to decide which signals are we to use and which we will remove from any further analysis. This is of

particular importance when analysing non-homogeneous experiments and positions or experiments that we might want removed from the analysis at some point.

For most users, both mono and multi-experiment *groupers* are the main tool to retrieve data for exploratory as well as automated analyses.

Reporter helps identifying useful experiments

Reporter is activated at the end of an ALIBY pipeline. It aims to provide a consistent and condensed summary of any experiment. It uses an experiment matrix: A data frame where rows are cells and columns are single cell features obtained from reducing time series into one value per cell; *reporter* uses this input to produce plots as requested by the user. The standard report has the following structure:

First, the **quality assurance** section shows the technical quality of an experiment, tiling performance, the number of correctly processed positions and, if applicable, at which point did clogging occur. Combining all of these answers the question: "Can I extract valid data from this experiment?" at a glance.

- **Number of tiles**: Shows the number of tiles segmented for each position. It identifies erroneously-segmented positions, which usually deviate from a given average number of tiles.
- **Tiling entropy**: Shows the regularity x and y-axis distance of tiles for each *group*. A *Group* with higher error bars than others indicates lesser regularity, implying that tiles were not identified uniformly across at least one axis.
- **Processed time points**: position-wise indicator of the last time point processed for all positions. It helps discard the possibility of missing data if a time series plot contains uneven number of time points for any given *group*.
- **Time series of tile-normalised cell number**: It shows the average number of cells per tile over the experiment time-course. It is most useful when troubleshooting a cells clogging over time and, oppositely, flushed-away cells upon media switches.

- **Distribution of tile occupancy**: Helps identify groups with fewer cells. It helps identifying when positions of a given strain or condition are either empty or overcrowded at the start of the experiment.
- **Distribution of cell volumes at start, middle and end of experiment**: Identify cell strains not growing. It also helps get an idea of the size - and thus age - of our cells at the start of an experiment. This may indicate whether two experiments are comparable, as ageing increases stress responses (Mouton et al., 2020). As an added bonus, it also shows how comparable are the size/age distributions of our cell *groups*.

Once quality assurance has been performed, we calculate multiple growth features; we combine data from a common stage (i.e., environmental condition), for instance, if an experiment follows a stage sequence feast-famine-feast, we group all time points belonging to the first stage, all timepoints belonging to the second one, and so on. The different growth calculations are:

- Retained cells' Δvol indicates mother growth rates, for they are generally the ones staying for most of the experiment.
- Bud cells' Δvol shows the maximum change in volume for all bud, that is, what is the maximum rate of biomass production for a given mother and bud, when looking the volume change of all buds. This returns is a single value per mother.
- Absolute budding events indicates total sum of buddings that arose in a given stage or the entire experiment.
- Average time between birth events limits the calculation of birth speed to the time points between identified birth events, yielding a more accurate picture of completed cell cycle than average births.

Fluorescence signals are reported both in absolute values and normalised over individual cells' average. We use both methods because they convey distinct sets of information: Raw values are a proxy for relative protein concentration. If strains in the same experiment contain different proteins tagged, the protein concentration

variability can lead to widely variable protein concentrations between cell groups. If we intend to compare these dynamics of signals with different basal value it is much more useful to normalise them by each cells' expression.

There are two simple metrics that offer, in my experience, the most information from single cell fluorescence. The first one is *max5 by median*, the Protein aggregation metric, highest five pixels inside a cell divided by the value of the median pixel inside the same cell. It is based on previous work on transcription factor localisation (Cai et al., 2008), but dividing the median instead of the mean. The second and more conventional metric is the median pixel value. It works as a proxy of protein concentration. Further metrics will be explored in Chapter 5.

Custom optimisations and features

I implemented multiple algorithms that I consider important to highlight. As far as I know, these are the approaches that make the pipeline as efficient as computationally possible. Some details of know-how and implementations are also added, for they might be useful to the reader if trying to understand or reuse internal components of ALIBY.

We split experiments into position because the *hdf5* limits us to a single writing operation at the same time for a given file. A limitation of this file type is its restriction on simultaneous write access. To process multiple images at the same time we have to use different files, specifically one per position. While this forces us to aggregate data the based on filenames, it compensates this through flexibility on how to organise, select and reuse subsets of data.

Positions are the minimal unit composing datasets. We can remove the ones with technical issues, and - with access to the original image database - replicate the processing of an entire experiment from a single position. This is because each *hdf5* file contains a copy of the metadata for the entire *pipeline*.

We can then filter-out position that we want to skip, or just manually delete them if they are beyond salvation.

Threading is the feature that makes ALIBY a faster implementation than previous software that tried to solve similar issues. One of the limitation of the original software is the single-thread limitation imposed by MATLAB. While it could be partially offset by using threading on the python side of things as shown by (Pietsch et al., 2022), full parallelisation provides much faster results.

- **High-speed indexing:** One of the main challenges is keeping low-level or commonly repeated operations fast. *numpy* broadcasting, that is, optimised array-to-array operations, makes it highly efficient. This becomes particularly relevant when accumulating signals from multiple positions and experiments, for post-processing events may be applicable. If these are, the acquisition and processing of cell signals would be slow and unresponsive without *numpy* optimisations.

- **Context-based networking:** For parallelisation, thus faster processing speeds, we require opening multiple independent connections to our image server. If these connections are not closed, the server can freeze. When opening these connections in a non-context manner they may hang upon encountering an error, that is, they may not close the connection properly. We use context managers, which make sure that connections are only temporarily open and close if an error happens. Using a context manager forces network connections to the server to be temporary, avoiding OMERO due to freezing, otherwise likely to happen upon network connectivity problems.

- **State saving:** ALIBY records the state of an experiment analysis at each time point. When running in computers without a dedicated GPU or slow connection there is a risk of time-outs. The only *step* of the pipeline that is time-dependent is BABY's tracking, as it goes backwards in time to correct for acquisition errors. We thus save this state at every time point to continue where we stopped if needed.

1. Extraction development tips

ALIBY attempts to guess the set of parameters using the experiment's metadata. For example, if the experiment's metadata contains a pHluorin channel, then, given that this probe requires two fluorescence intensities to estimate pH, it will calculate ratio of channels with these fluorescence and produce a new "artificial" channel to extract data, as previously shown in Figure 4.6. Defining a set of sensible defaults improves inter-experiments comparability, and eases the processing of additional technical replicates or experiments with similar goals.

ALIBY contains an extraction metric that uses convolutions - operations of weighted matrices that operate on groups of contiguous pixels that combines their collective information - to predict nuclear localisation. It performs better than most other metrics we have tested (Pietsch et al., 2022), but it is slower than most simpler-yet-functional metrics. Due to our prioritisation of speed I excluded it from the defaults.

Whenever possible, we use vectorisation, which is a technical computing optimisation implemented via the *numpy* Python library, to perform atomic extraction operations as fast as possible. Other functions that collapse multiple dimension into one, such as median, which requires sorting pixels, slow down processing speed. By delegating numerical operations to *numpy* vectorised functions we ensure the highest possible processing speed achievable using the Python programming language.

The implementation of new and custom extraction functions has minimal requirements. The only restriction is acceptance of two matrices: a Boolean matrix containing the cell mask and a non-Boolean matrix as the microscopy image. For instance, a custom function implemented was clustering the pixels of fluorescence within a cell and returning the mean of the group with higher values.

4.4.2 Features

Our automated processing provides better scaling than the alternatives

To my knowledge, no existing software can perform automated end-to-end analysis to the degree of ALIBY.

The software from other labs that most resembles ours requires manual selection of tiles (Aspert et al., 2022), and focuses on birth prediction performance, leaving its segmentation capabilities as secondary. Our ALIBY + BABY combination, on the other hand, provides automated tiling and access to not only birth events, but also cell outlines and lineage predictions for tiles with multiple mother cells. Focusing on birth rates neglects the other way to measure growth, namely change in cell size.

We thus must compare it with a previous software developed by the Swain Lab to solve the same problem. Data run on a desktop computer with a GPU and a 10-core processor. Its computational speed is twice as fast as the previous MATLAB equivalent. processes most datasets in between five and eight hours, as opposed to between ten and fifteen. Runs in Python, a widespread open-source language, instead of MATLAB.

ALIBY provides faster access to partial results, does not require manual identification of tiles and provides a report automatically. First positions are accessible in half-an-hour, as opposed to until the experiment finishes. After finding tiling parameters for a single position, experiments don't require human intervention, the previous pipeline required manually curating tiles. ALIBY provides a report summarising the experiment.

We are able to see volume oscillations in mother cells within our results, which to my knowledge are not reported in literature. Despite a thorough investigation on where could these come from, we could not find evidence of imaging artifacts. After directly looking at tens of cell time-lapses, I came to the conclusion that these volume oscillations, while minor when compared to cell volume, are consistent and

thus I presume it a phenomenon arising from either the experimental setup or a biological feature.

Our new tiling algorithm enables novel experimental design

Trap-tiling robustness determines viable device designs

One of the main features of this algorithm is its self-templating mechanism. We are not fitting a manually-curated template with a predefined direction, instead we divide the image into background and foreground, produce a template from the foreground and fitting it to the entire position. Its agnosticism about trap vertical or horizontal orientation offers a multitude of advantages.

This self-templating tiling technique enables experimentation with microfluidic devices. In our experience, trap-based microscopy tiling requires images are within a 90-degree rotation from the template. While it is possible to rotate them, anything different from 90, 180 or 270 degrees introduces requires artificial smoothing and thus introduces additional noise.

Our new approach also facilitates the development of microfluidic devices that contain chambers (a chamber contains multiple positions) with distinct tile designs. While this is a fringe case at the time of writing this thesis, it will become more important as users bring a wider range of organisms under the lens of a microfluidics approach and need to test different tile designs. The constant evolution of the design of these devices requires an algorithm that is flexible enough to follow suit.

The limited number of assumptions in this tiling algorithm does not increase as the design of microfluidic devices evolves, as it follows the same prerogatives most of these trap-based devices take for granted: Individual cells, or groups of cells, must be physically unconstrained, and colony growth is uncommon due to a constant flow of media flushing away daughter cells. The ability to generalise of this tiling algorithm lies in that simple assumption, for as long as we have regions having background between them, a consistent tile structure and more background than foreground, the algorithm should be able to select the pattern and find it all over the image.

I aim to leverage existing and future tools

If we swap BABY with other segmentation software, such as *Delta* or *Cellstar* (O'Connor et al., 2022; Versari et al., 2017), ALIBY can - in theory - work for other setups. BABY specialises in setups where the flow of media flushes away daughter cells, and will perform better in these cases than in others, such as fixing cells on pads. In agarose pad setups, cells accumulate in a single tile and calculating edges and radii using a neural network becomes more computationally expensive, decreasing the benefits provided by the optimisations in ALIBY, but still being usable for many of its other features, such as reproducibility.

4.4.3 Noteworthy design components

The modular design favours repurposing of tools

Components, specially the lower-level tools such as networking and I/O interfaces, serve for multiple purposes. The networking classes serve both tiler and a dataset exploration tool, argo. *tiler*, which, among other things, fetches regions of interest for both segmentation and extraction, is also a component of our time-lapse visualisation tool.

The *cells* class serve as base of multiple other items. *Extraction* depends on it when fetching outlines from local files, the set of visualisation tools ImageViewer use both *tiler* and *cells* to display tiles and their outlines. Which I expect will serve as a base to a Graphical User Interface (GUI) in the future.

Consistent, fast and widespread cell indexing

Indexing optimisation is essential due to the hierarchical nature of our data and uncertainty in for how long cells will be present. Our experimental setup enforces a strict order of: positions, tiles, cell labels and time points. While we define position and tiles at the start of the experiment and they do not change throughout, we cannot say the same for cell labels. As cells grow and produce new buds, they tend

to drift away due to the constant media, cell labelling must keep up with these events.

I defined our standard structure to store time series as a 2-dimensional array in which rows are unique cells and columns are time points. We identify unique cells using by keeping our hierarchical structure in the form of an independent matrix (these are named MultiIndices within the Python data science parlance). Overall, this structure translates to a data frame where time points are the y-axis and the multiple indices in the x-axis can indicate individual cells, and we can both access the time-lapse in its entirety and specific cells with ease.

As a general rule, Python is a slow language when compared to most other data science languages, thus performance is reliant on additional tools that generally delegate numerical tasks to other languages. To access data from multiple flattened arrays (each holding information on tile id, time and cell label), we use two high-efficiency methods, which one we use depends on the amount of values we need. If we want a single value, we use a third-party module (`py_find_1st`) that is faster than *numpy*. Linear array is the most compact storage structure, and its indexing is by far the most frequent operation within ALIBY, it thus requires the fastest methods available.

The second method is called Boolean masking, and it is used to fetch multiple values, for instance, all cells at a given time or all frames during which a specific cell is present. In practice, Boolean masking is comparing n multiple arrays (each array contains tile, time point or cell label indices) against m numerical values (e.g., two numbers indicating tile, and time point) to request, for instance, cells present during the last five frames and located in the first tile. The usage of these two strategies prevents indexing from becoming a bottleneck, as using this indexing drastically increases processing speed, bypassing the inherent limitations of multi-level indices.

I separate processes from Input/Output operations

Our back-end components are a set of tools we use to interact with our local and remote data structures. Pipeline users may not be aware of their existence, but they are the stepping stone for building additional tools, both user and developer-focused. Extended tools use at least one back-end tool under the hood.

We use independent *writer* and Reader classes for our different stages or processing. Figure 4.11 shows an overview of Readers and *writer*, and their associated processing stage. Tiling does not use a reader because its contents are much simpler than the others.

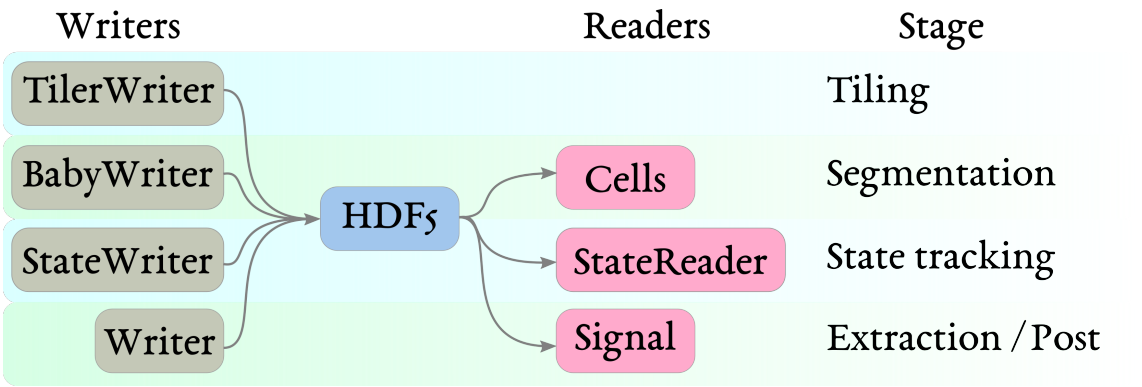

Figure 4.11: ALIBY interfaces and their relationships.* Once we pass data to a writer it applies formatting and stores it in a given address

Note that we actively limited the plotting and data wrangling wrappers as much as possible. The logic behind this decision follows two principles: First, ALIBY's classes return data in such a format . The purpose of plotting functions that we do provide is automating experiment reports.

1. *Readers* move data from a results file to the user or *pipeline*.

   I define *readers* as a set of tools used to retrieve data from long-term *hdf5* storage files. They then format the data as needed and pass directly to the user or sent to another class for further processing or aggregation. These classes are seldom accessed by the normal users, for they only return a small section

of entire experiments. They do enable consistent and reproducible access to results from ALIBY pipelines.

2. *Cells* provide single cell identifiers, outlines and lineage

   The class *cells* is our bridge to access stored data to identify a specific cell or set of cells. It loads the output of BABY: cell labels, outlines and linage information. It only depends on local information, thus it does not need access to network tools. It is only directly used during processing time when a BABY instance is not available, thus when we are re-extracting an experiment. It is analogous to BABY*writer*.

3. *Signal* loads time series data and may apply essential *postprocesses* when applicable.

   signal's purpose is fetching and formatting the stored data into single cell data matrices. This tool also applies simple postprocessing filters before passing the modified data to the user or calling function. Restricting postprocessing modifications to a *reader* ensures data integrity all over. The caveat of this approach is data processing speed, ALIBY mitigates delays this might cause by using high-efficiency operations reprocessing the data.

   This class is also in charge of converting time point data into a consistent unit (minutes, by default), if the corresponding metadata is available. This conversion makes time-lapses more informative, and comparing time series side-by-side easier. This class is the *reader* analogous to *extractor*'s and *postprocessor*'s *writer*.

   For developers, *signal* provides multiple ways to access the datasets. These modify the selection of cells and whether or not we apply additional modifications to the tracklets. There are multiple criteria to select cells: By default we get retained cells, that is, cells that are present for 80% of the experiment, and applies merge operations; *signal* can also fetch cells labelled as mothers,

daughters or both; additionally, it can skip altogether sub-selecting cells and only apply merge operations.

signal is a low level class, meaning that it interacts directly with the underlying data storage structure. It also means that it is a building block for other tools that fetch data from multiple resources. General users don't come in contact with this class, however it is at the core of all dataset-fetching tools. For developers, it an essential tool due to its access to raw data, key requirement for faster debugging and testing.

4. *StateReader* can recover the last circumstances of an experiment, both if interrupted or finished.

   StateReader, analogously to StateWriter, is a safeguard to avoid having to segment experiments from scratch when encountering any contingency during processing. It enables restarting an interrupted experiment at any point, be it an accidental interruption (e.g. resulting from a network error) or an intentional one (e.g. a user not processing all time points at once).

   In addition to the last processed experiment, *StateReader* also stores flags on the conditions under which the last processing pipeline finished. For instance, if there is a network connection it indicates that it was a network issue, as opposed as a planned end to processing or a fully-completed processing pipeline.

We reinforce existing standards by following OMERO bindings to access remote data

We use OMERO (Li et al., 2016) as our main data management tool. It is a software to store and manage large datasets of microscopy data, thus solving most of the initial challenges we face when dealing with large amounts of data. It provides a programmatic interface to automate our analysis pipelines, as well as a web user interface to manually check individual experiments. It is easy to set up, open source

and uses standardised formats. To our convenience, OMERO can handle images, metadata and even analytics out of the box.

We define an interface as an object (or abstraction) that handles all server connections and requests. Interfaces are independent of the image-processing classes, for they are a convenient method for communicating with OMERO servers and multiple independent tools can use them with goals beyond data processing. We use the *omero-py* module to retrieve data from OMERO servers, a tool that enables us to obtain images from OMERO servers over the network.

Under the assumption that by fetching images from remote servers we put added use context managers to avoid leaving orphan processes that can make our server hang. It works under the assumption that the network may fail when transmitting over the network. Via Python context management we ensure ALIBY will not leave connections hanging, which in turn put servers at risk of becoming unresponsive.

For our lab's setup we store *a* position as multidimensional images arranging dimensions in the following order: Number of time points, number of channels, and x, y and z-axis. If different channels have an unequal number of stacks, such as bright field having five stacks but GFP three, channels with fewer z-stacks range from stack zero to stack $n_{max} - n_{channel}$ where n is the number of stacks. This convention ensures a valid image is present in the first stack for any channel.

Alongside the images, we store experimental metadata on OMERO. We obtain experiment information and the metadata files and use them to automatically process our results by generating parallel pipelines to process position independently from each other. Due to the lack of a standardised metadata format in live-microscopy, we use our own, but the user can provide ALIBY with additional information using the OME-TIFF format, as well as pass this information directly within the Data structure The results file for a single position has the following tree structure, where the text on the right is the metadata contained at that given level.

Extensibility was put to test for this thesis

In addition to the main *pipeline*, I have developed tools that are useful for the researcher.

I developed argo as a programmatic interface to explore an OMERO database and, forward them to automated analyses using aliby through bespoke scripts. Its main purpose is streamlining the process for advanced users, making it simple to browse datasets interactively. It depends on OMERO web servers having correctly annotated dates and tags.

ImageViewer is a programmatic interface to access tiles multidimensional images and their associated masks, it can be used to display cells and their associated outlines using info from a results *hdf5* file and access to the raw images, be it local or remote. For instance, ImageViewer produced the top panel of Figure 4.5 in an automated manner. Using additional methods of *cells* class made finding examples of tiles with n number of cells during t time points easy and straightforward.

4.4.4 Development roadmap

Multiple features or elements of ALIBY are extensible and new features can integrate and improve both its efficiency and usability. Its development is tightly linked to the advances in acquisition software and hardware, as well as the new microfluidic devices in use and biological questions posed.

To increase the potential user base, support for additional metadata is necessary. Due to the lack of standardisation in the field I assumed that the data would have a certain shape and also provided a way to indicate this in the arguments. There is an trade-off between providing flexibility and maintaining support for multiple metadata structures.

On a technical note, using non-uniform rational B-splines (NURBS) for cell outline representation would improve fidelity over the current splines that BABY calculates, as shown by (Grove et al., 2011) in the biomedical field. Our current method for

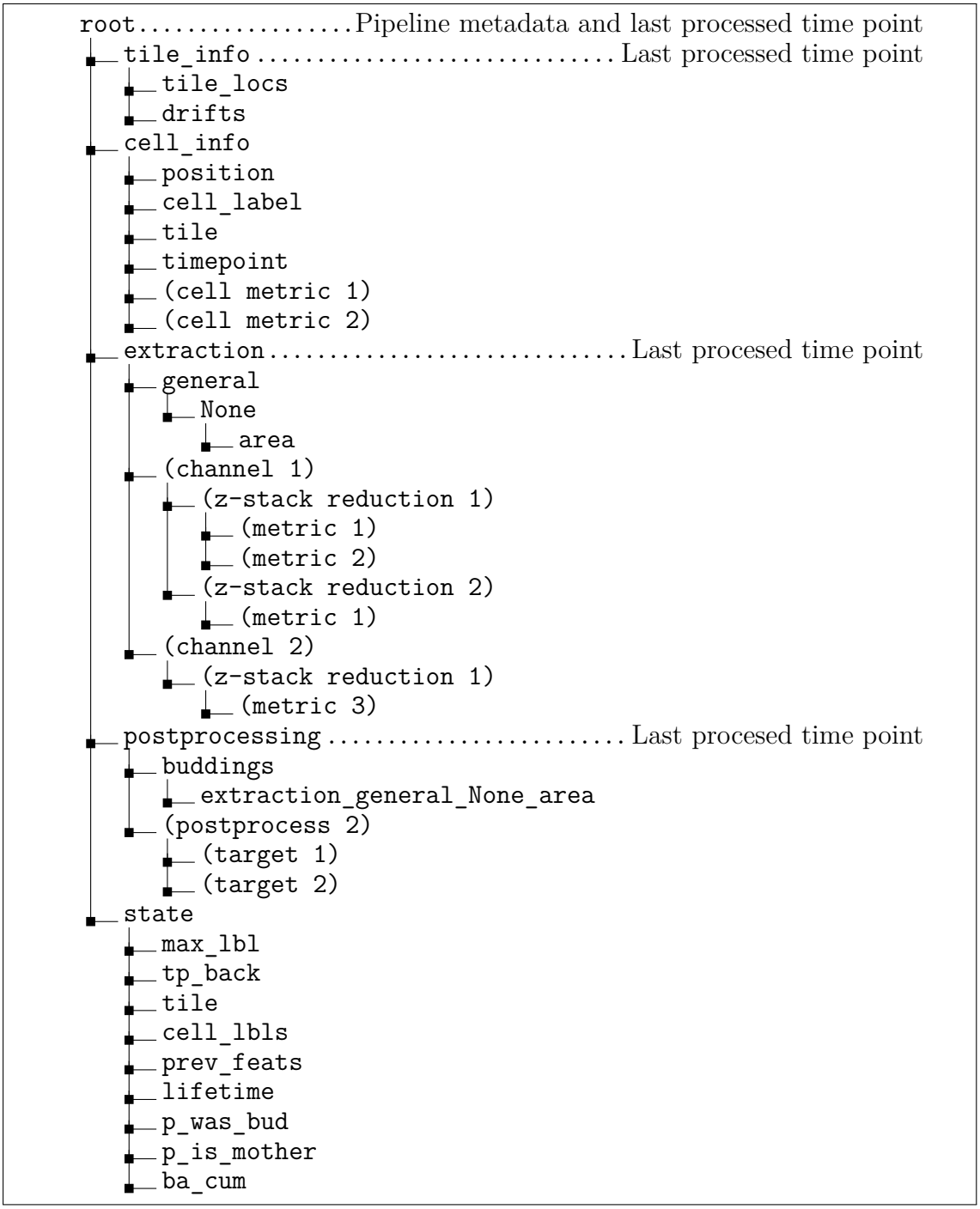

Figure 4.12: **Structure of results *hdf5* file**. If the group includes metadata is shown as a comment on the right-side.

defining a cell outlines makes assumptions on the number of nodes and shape of a cell. Generalising using Bezier curves would potentially increase its portability to describe outlines of cells with a wide variety of shapes, making ALIBY more useful for researches who do not focus on yeast or yeast-like cells.

While I designed ALIBY with live-processing of experiments in mind, it does not currently support its implementation, for it needs connectivity with acquisition software. This is another important direction of development.

We do not make assumptions on yeast tile designs, but we do assume that there is a regular repeating pattern present in multiple places over a given position. In some images where we track single cell colonies there is no guarantee that our tiling method will find all regions of interest. This is easily solvable using an additional tiler only based on entropy, skipping the usage of templates.

We currently ignore most of the z-stack information by using standard z-stack projections. More advanced algorithms that analyse information on the different stacks to decide which one(s) to use are likely to improve our signal-to-noise ratio.

Another assumption of ALIBY is the homogeneity of time intervals. Improving our signal-to-noise ratio entails sacrificing imaging frequency. Alternatively we can use variable time-intervals, increasing it in information-rich time points - around environmental changes - and reducing at all other times.

ALIBY does not include any database functionality, but just using plain folders is a versatile method for the current amount of data. The development of a results-only database would greatly facilitate data aggregation and project-wide data comparisons.

Developing a way to transform any function into the *process-parameters* schema would be useful to lower the barrier of entry for new developers.

An interface and GUI are important for any microscopy-based project. The most direct way to add this is using existing software and integrating it to our needs. I think the *napari* project has at the time of writing has community support and may integrate with ALIBY without much effort.

We have yet to integrate a framework for quantification of uncertainty and noise propagation. This would help identify the source of noises and help improve experimental methodologies.

4.5 Conclusions

Non-trivial software development turned out to be challenging for reasons that I did not expect: The goal migrates from solving a problem towards solving a multitude while keeping complexity under control; additionally, it is seldom clear what is the right approach to model biological data as a computational data structure. I do not think there is one solution for all, it thus ends up being as a design decision based on the trade-offs existing available tools provide.

Design decisions must be made when developing any software. Being both the user and developer of scientific software makes deciding which features and design components are better for biological discoveries. Leveraging the modern software ecosystem enables acquiring novel data, but it requires the constant research of new tools and -most times -the development of new ones.

As a minor side note, I do not think Python is the best language to do this type of work, but the momentum it has at the moment in which this thesis written makes it the only choice if we aim for others to actually benefit from this work. Due to the big-gets-bigger phenomenon experienced in everything software-related, it is likely to still be the case for many years to come.

Whilst experiments must still be performed, delegating the analysis to others limits the agency of the experimentalist on the conclusions obtained from a given experiment. The biggest challenge that arises from this is to train biologists in data science and computational scientists in biology. Communicating tools and designs is complicated and requires a high degree of technical knowledge on computational platforms and frameworks.

Computer sciences and biology will only intertwine more in coming years. Skills and knowledge from each fields are important not only to build new tools to answer biological questions. They also allow us to reexamine the questions we can ask and how reliable the answers *actually* are.

5 Phenotyping population heterogeneity using CTP synthase aggregation

5.1 Introduction

Protein reorganisation in aggregates is a widespread phenomenon in multiple types of eukariotic cells: At least 180 proteins assemble into aggregates, and from those only nine proteins have been found to form elongated filaments (Narayanaswamy et al., 2009; Noree et al., 2010a). The function of these aggregates is not fully clear, but the physical state of these proteins may impact their activity or degradation.

Proteins that form filaments co-localise into four different assemblies: *ura7* and *ura8*, paralogs that form a complex of inactive CTP synthases; the proteins codified by the genes *gcd7*, *gcd2*, *gcd6*, *gcn3* and *sui2* aggregate together, they also comprise the translation initiation factori eIF2B; finally, *glt1* and *psa1* form one filament complex each (Noree et al., 2010a). The product of *glt1* forms the secondary pathway of glutamate biosynthesis from ammonia.

Some of these proteins form filaments energetic deprivation, in the case of the CTP synthase complex it depends on both cytosolic acidification and substrate availability (UTP). *glt1* and *psa1* also form aggregates under nutrients deprivation, but their aggregation mechanisms and function are not known (Noree et al., 2010b).

This chapter shows a quantitative analysis of filament assemblies, and whether they have a beneficial or deleterious effect on cell growth as measured by number of budding events (proliferative growth), or average volume increase over time. We use CTP synthase (CTPS) as a model protein, *Saccharomyces cerevisiae* contains two paralogs, *ura7* and *ura8*, that catalyse the conversion of UTP to CTP. These

paralogs appear to have retained their original (and shared) functionality, and are widespread across eukaryote cells.

The main reason behind the usage of this particular filament-forming protein is its existing role as a model for protein aggregation: as mentioned previously, CTP synthase is a model for aggregate assembly-driven regulation. Excessive aggregation of CTP synthase may lead to slower growth, and this aggregation reduces enzyme activity (Lynch et al., 2020).

5.2 Methods

5.2.1 Microfluidics setup

We use ALCATRAS devices developed in our lab (Crane et al., 2014). Newer devices use designs based on media bypass channels to reduce drastic changes in pressure (Lee et al., 2012). Additionally, the microfluidic devices used for this chapter the three chambers per experiment, instead of the original ALCATRAS device, as it contained just one chamber.

5.2.2 Aggregation metric

I implemented our aggregation metric (also referred to as *max5 by median*), it uses a bright point inside a cell and the median fluorescence to estimate the fraction of proteins that are localised in a small region (Cai et al., 2008). There is no standard metric to measure aggregation, although historical approaches include the usage of standard deviation as a proxy; it does not account for cell-to-cell variability in protein concentration. Additionally, a robust metric should not require parameters to fit in a small set of images; as it would become specific to a limited set of imaging conditions. We needed metric that accounts for the information provided by one or more focal points of aggregation (the brightest spot or spots inside the cell) and the cytosol. This type of approach been previously used when the goal is to quantify intracellular nuclear localisation (Granados et al., 2018b).

The *max5* metric is divided by the background-subtracted fluorescence median inside the cell, to account for variations in the concentration of the fluorophore inside the cell. Thus our aggregation metric can be expressed in Equation 5.1:

$$agg_{cell} = \frac{max5_{cell} - bg}{median_{cell} - bg} \tag{5.1}$$

Where *max5* is the top five pixels inside a cell, *median* the median value of all pixels within a cell, and *bg* is the median value of pixels outwith all cells in the vicinity - or more specifically, within a *tile* (see Chapter 2). While a more complex metric may yield slightly different results, *max5* divided my median strikes, in my experience, an adequate balance between calculation speed, interpretability and robustness. It is a very fast method, while quantifying aggregation more accurately than other simple metrics. Its validation, which uses manually curated data where aggregates are detected, is shown in Figure 5.1, in which most cells with visible aggregates result in higher values. It also includes cells that are in the edges of tiles, showing that it does not influence the result. More than one aggregate may be visible in some cells due to the sampling capturing cells at different stages of aggregation.

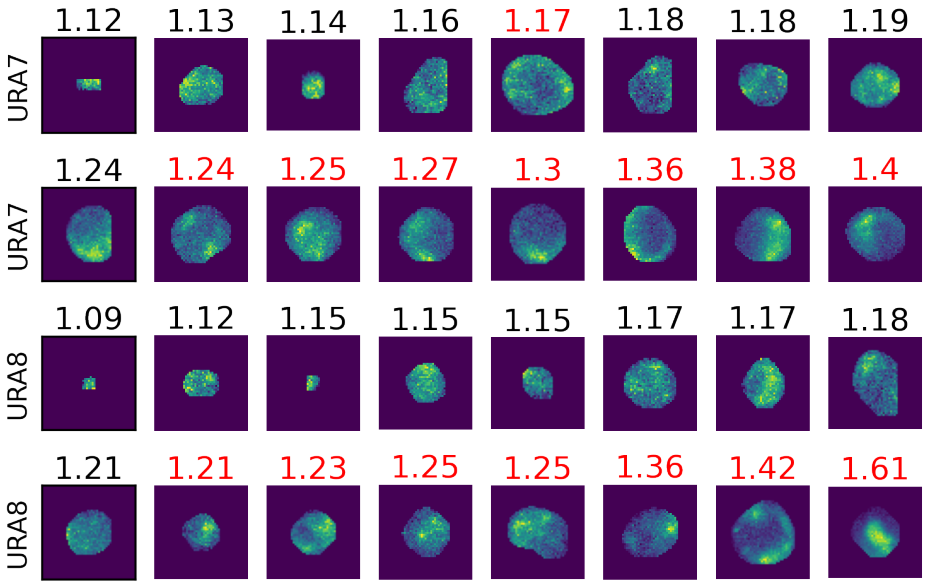

Figure 5.1: **Representative examples of cells in starvation.** Cells were randomly sampled from the last 100 minutes of famine. The background of each image is set as the lowest value inside the cell to highlight intracellular pixel intensities. the value on top of each image is the aggregation metric, and they are sorted first by group (two rows for each group) and then by this value (left to right). The colour of these titles indicates whether they were manually annotated as "cells with aggregates": Red text on top means that the cell is annotated as containing aggregates and black titles indicate that it does not. To assume aggregation it requires at least one focal point of aggregation not in the edges of the cytosol. Cells were moved to the centre of the image; some cells may appear cropped because they were located at the edge of a tile.

Based on the previous figure, the *Max5 by median* metric seems indicative enough of aggregation. For this reason, I chose it to be our only metric to quantify protein aggregation. From here onward, the *max5 by median* metric will also be referred to as the "Aggregation metric".

Savitzky-Golay filter

The Savitzky-Golay filter is a digital signal processing technique used for smoothing or differentiation of data. It operates on a moving window of data points and per-

forms a least-squares fit to estimate the underlying trend. The filter is particularly effective in reducing noise while preserving important features of the signal.

It can be expressed using the following equations:

$$\mathbf{y} = \mathbf{X} \cdot \mathbf{c} \tag{5.2}$$

Where y is the filtered output signal vector, X is the coefficients matrix, constructed from the data points within the moving window and c is the filter coefficients vector to be determined.

Polynomial basis functions are used to construct the polynomial basis function matrix. The two parameters used are the polynomial degree p and the length of the moving window N. The matrix has shape $N \times (p+1)$ where rows represent data points within the moving window and columns correspond to a polynomial basis function.

$$\mathbf{X} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^p \\ 1 & x_1 & x_1^2 & \cdots & x_1^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^p \end{bmatrix} \tag{5.3}$$

where x_i is the /i/th data point in the moving window.

The filter coefficients cc are obtained by solving the least-squares problem:

$$\mathbf{c} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y} \tag{5.4}$$

Once the coefficients c are determined, we can compute the filtered signal from Equation 5.2. Within this work, I used this filter to estimate changes in single cell volume, I used this filter is applied before calculating the changes in volume.

Single cell comparison at different stages

To compare filament formation inside individual cells in different (consecutive) environmental conditions I split a (feast-starve-feast) time series into three chunks, where each one is a distinct stage in the experiment. Relative aggregation is then calculated as the ratio of the average aggregation at each stage, expressed as follows:

$$\frac{feast}{starve_i}, i \in 1, 2 \tag{5.5}$$

In other words, I average the aggregation of each cell in all three stages and then divide the average aggregation during feast by each of the aggregations during starvation.

5.2.3 Experimental information

Table 5.1: **List of environmental conditions to which we subjected cells across all our experiments.**

Media	Carbon source	Notes
Synthetic Complete	2% Glucose	0.5% Bovine Serum Albumin added to coat the microfluidic device
Synthetic Complete	None	0.5% Bovine Serum Albumin added to coat the microfluidic device

Table 5.2: **List of *Saccharomyces cerevisiae* strains used in this chapter.**

Main name	Strain	Reporter	Notes
ura7	BY4741	ura7-GFP	Strain from (Ibstedt et al., 2014)
ura8	BY4741	ura8-GFP	Strain from (Ibstedt et al., 2014)
ura8H360A	BY4741	ura8-GFP	Strain from (Hansen et al., 2021)
ura8H360R	BY4741	ura8-GFP	Strain from (Hansen et al., 2021)
pHluorin/ura8	BY4741	pHluorin, ura8-GFP	Produced for this and other projects.

1. Image processing pipeline

   We used our image-analysis suite ALIBY ("Swain Lab / Aliby / Aliby," n.d.), which takes advantage of existing AI-based segmentation tools ((Pietsch et al.,

2022), specifically ALIBY's commit 31cb5a4, PyPI version 0.1.65). Parameter and setup files are provided on public repository of this thesis ("Alán Muñoz / Thesis," n.d.), alongside all the figure-producing scripts.

2. Signal processing

   Signals were processed using ALIBY's standardised *Grouper* + *Chainer* live batch-processing tool and then rearranged for plotting via ALIBY's kymograph utilities. All associated code is in the repository hosting this thesis. Due to their size, the raw time lapses are available upon request.

3. Hosting

   All software developed for this thesis is hosted at the public *GitLab* instance. Data is hosted on a public server hosted provided by the University of Edinburgh. For cell tracking, training data and scripts are hosted on *Kaggle*'s public repository.

5.3 Results

5.3.1 Time lapse microfluidics recover filament formation dynamics for both single cell and populations

The first approach to understand the volume of data we are dealing with is combining all cells and time points into a single figure. An overview of the environmental dynamics we used for this chapter is shown in Figure 5.2. Panel A indicates the glucose concentration in an experiment with a symetric single starvation. Panel B shows asymmetric dual starvation, which will be used later on.

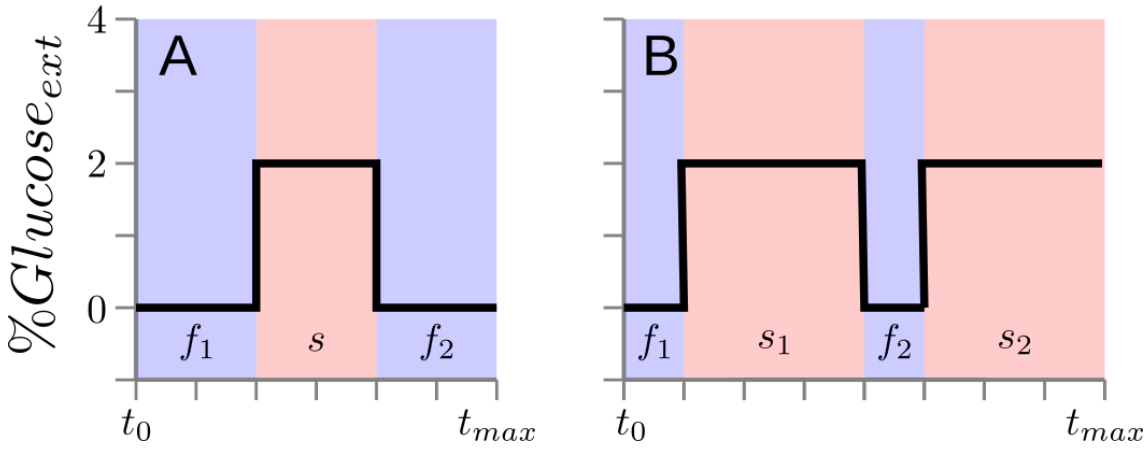

Figure 5.2: **Environmental dynamics of the experiments of this chapter, as measured by external glucose concentration over time.** We control the environment using pressure control devices, inducing carbon source stress by removing glucose from the media. Starvation stages (s_x) are shown in a red background and feast stages (f_x) have a blue background.

Figure 5.3 shows the results of our standard experimental design. We grow the wild-type *Saccharomyces cerevisiae* cells whose CTP synthase genes *ura7* and *ura8* were tagged with Green Fluorescent Protein (GFP) in ALCATRAS devices (Crane et al., 2014) that trap single cells. I divide the experiment different into stages (where I use stage to mean time interval in an experiment with changing environments, during which media is constant): Starting with constant flow of 2% glucose Synthetic Complete (SC) media (feast), at $t = 300$ (minutes) we switch to SC media free of carbon-source (famine), triggering filament assembly in both populations (Figure 5.3A). At $t = 600$ glucose returns to the growing chambers, aggregation is reversed and cells resume growth - they cannot duplicate their genome and divide when these filament-forming proteins assembled into aggregates (Hansen et al., 2021).

To understand if each gene displays unique behaviour, we calculate the mean over time for each strain. Using a graphical representation of single-cell measurements where each individual cell is a row, with changing values over time (kymograph). The imaging interval is five-minutes, and also the resolution at which we can quantify aggregation and demonstrate that most cells react to starvation, even if marginally

so (Figure 5.3A), with varying response times across the population. Interestingly, aggregates form in a group of cells almost immediately, and reaches the plateau within the first few minutes (black rectangle at the lower middle section in the kymograph), whereas most cells commit to their filament assembly later during starvation. Once filaments are formed, they do not dissolve unless environmental conditions change, but some short bursts of aggregation are visible in cells that do not commit as clearly during starvation.

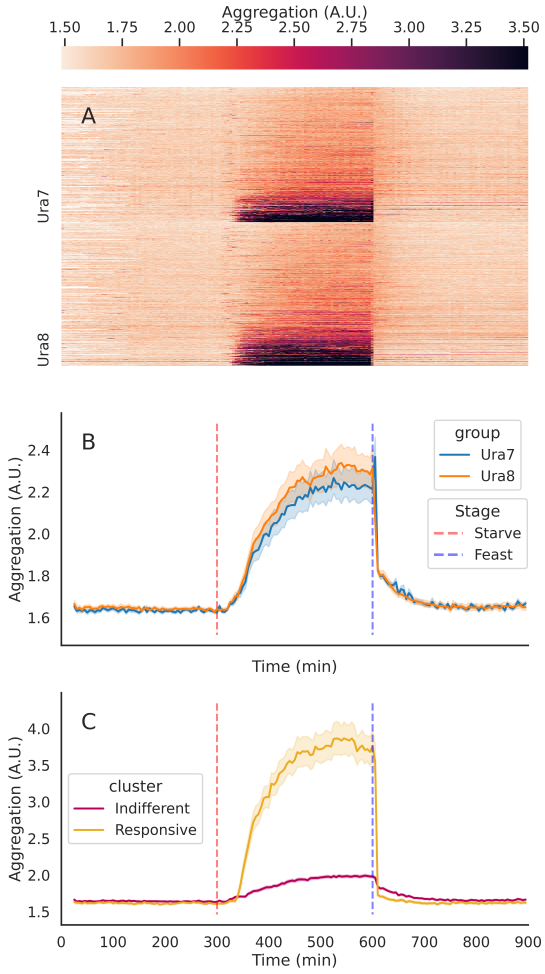

Figure 5.3: **The paralogs *ura7* and *ura8* aggregate upon glucose removal and disaggregate upon glucose re-addition in an asymmetric manner. A.** We imaged 1087 cells set in place using an ALCATRAS device and analysed individual cell traces using ALIBY (Chapter 2). Dashed vertical lines indicate the time at which media changed from SC + 2% glucose to SC with no carbon source (red) and back (blue). Shaded areas are the 95% confidence intervals as calculated by the *seaborn* Python package, further explained in the Methods section. **B.** kymograph of single cell traces vertically sorted by total aggregation over the entire time lapse. Each cell is represented as a row; each column represents an individual time-point at which cells were imaged. The imaging interval is 300 seconds. **C.** By clustering cells based on their likelihood to contain filaments the population into two. By using these two groups we define indifferent and responsive cells. This experiment was repeated twice, with similar result, with similar results.

Our first results seem to agree with previous studies. The average behaviour of filament assembly of both paralogs is similar and reaches a similar value during late starvation ($500 < t < 600$ minutes). This behaviour coincides with previous reports of isoforms co-localising (Franzmann & Alberti, 2019). Yet, according to *in vitro* negative stain imaging from *Hansen et al.*, the isoforms show slight differences in assembly dynamics from each other (Hansen et al., 2021).

5.3.2 Assembly of CTP synthase filaments is reversible and asymmetric

To reduce the amount of data and understand how cell individually react to different environments, we compact all data-points during each stage. We aim to compare the feast (period with presence of glucose) that bracket a famine (a starvation period) to characterise and compare the response of individual cells in sequential environments. We ask whether cells revert to their "basal" state (i.e., when cells are happily growing) and how fast they move into the next basal equilibrium point (the signal at a value that does not significantly change), assuming they do so. To do this we reduce all the values of our three consecutive experimental stages - feast, famine and feast - to their average value for each cell.

To inquire about the reversibility and timing distributions within and between groups, previously reported as a bimodal distribution of filament assembly (Hansen et al., 2021; Noree et al., 2010a, 2014), I cluster cells in two groups based on their protein aggregation profile. In Figure 5.4A I compare the aggregation metric between feasts, scaled by the famine that splits them. My definition of "basal aggregation" is based on the average protein aggregation value during feast stages. In formal terms, and borrowing the nomenclature from Figure 5.2A relative aggregation is expressed as:

$$R(x) = \frac{\text{MEAN}(f_x)}{\text{MEAN}(s)} \qquad x \in 1, 2 \tag{5.6}$$

Thus the axes in Figure 5.4A are $r(1)$ in the x-axis and $r(2)$ for the y-axis. Most cells' pre-starvation protein aggregation directly correlates to their post-starvation aggregation. This shows two things: The first result from this is that filament formation is fully reversible to basal levels. Secondly, the level of basal protein aggregation does not visibly change due to a single starvation period.

From this data, I can estimate the time it takes cells to transition between cytosolic states (i.e., with CTP synthase aggregated or dissolved), as shown in Figure 5.4B. This was observable in Figure 5.3C, where filament dissolution appears more sudden for responsive cells (red cluster), I wanted to analyse the assembly and dissolution rates group-wise. Figure 5.4B reveals that for most cells, assembly of CTP synthase filaments takes longer dissolution, as suggested by the presence most data points under the dashed diagonal that denotes equal aggregation and dissolution rates. Furthermore, responsive cells show a narrower distribution for aggregate dissolution, confirming the abruptness of their transition out of pseudo-hibernation.

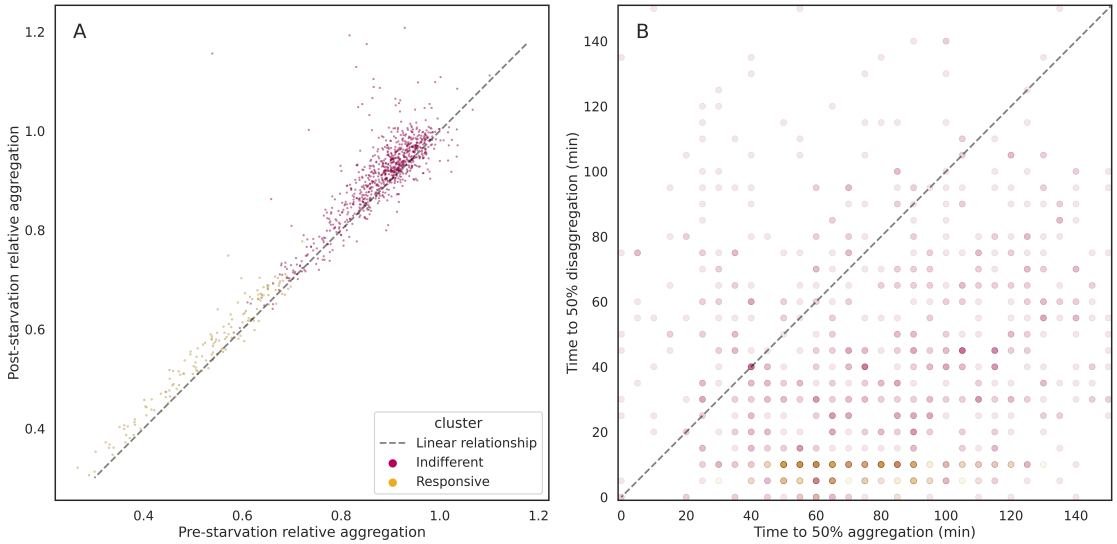

Figure 5.4: **Assembly of CTP synthase is reversible and asymmetric. A.** Protein aggregation before and after the famine (before t=300 minutes and after t=600 minutes) stage relative to mean aggregation during famine ($300 < t < 600$ minutes). Cells above the diagonal reduce their basal aggregation while the ones under the diagonal increase it. Cells along the diagonal go back to similar basal aggregation levels. **B.** Time it takes for each cell to surpass 50% its own time-average aggregation during transition from feast to famine (x-axis) and to resume growth from famine to feast (y-axis). Aggregation takes longer than dissolution, and filaments in responsive cells dissolve faster than aggregators.

5.3.3 Mutants with altered aggregation dynamics may provide further insight

I acquired mutant strains with sequence modifications that affect CTP synthase filament assembly but maintain their enzymatic functionality (Hansen et al., 2021). Their behaviour in agarose pad experiments has been characterised in Figure 5.5 as the following: wild-type ("wt") cells form filaments upon glucose deprivation and dissolve them once glucose is back returns. The non-assembler mutant *ura8*H360A, is unable to form visible aggregates in batch culture experiments. Its CTP synthases thus appear unresponsive to environmental changes. According to previous enzyme activity assays, the catalytic activity of proteins with this mutation is higher than the wild-type version.

I define another strain as the high aggregator, a mutation in any CTP synthase changes the translation of amino acid 360 from Histidine to Arginine (simulating the protonated state of the protein). It is a strain with higher affinity to form CTP synthase and lower affinity to dissolve filaments (also referred to as *ura*8H360R). Upon assembly it is able to form aggregates just like the wild-type. In batch culture, assembly occurs at a faster rate than the wild-type. The catch is exposed upon glucose reintroduction, for its filaments do not dissolve when moved to fresh media in batch culture experiments.

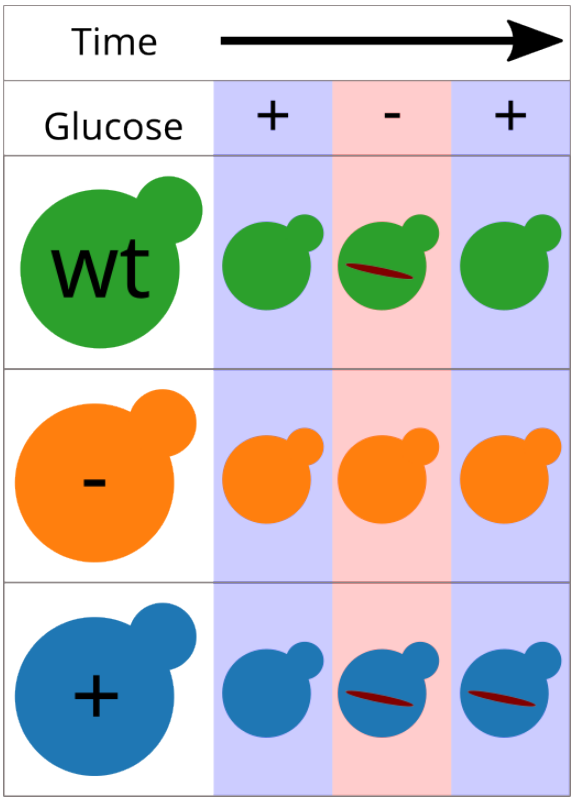

Figure 5.5: **Diagram of aggregation behaviour in the aggregation mutants based on *Hansen et al.* (Hansen et al., 2021).** The wild-type strain *ura8* (green) contains diluted CTP synthases during growth; upon glucose starvation these proteins assemble into filaments that dilute upon re-addition of glucose in the media. Cells with mutation H360A (Histidine to Alanine in the amino acid 360), which I name low aggregator, show low to no aggregation responses to changes in glucose in the media. The last strain, which I define as high aggregator, contains the mutation H360R (Histidine to Arginine in the amino acid 360) and reacts to starvation similarly to the wild-type, but once glucose is reintroduced filaments are not lost diluted. It is worth noting that mutations exist only in one of the paralogs, not both, due to technical constraints; for instance, mutating both copies has a strong deleterious effect on viability.

In Figure 5.6 I show examples for the three strains. Annotated aggregates were sampled more commonly in Wild-type and High Aggregation cells. This reinforces our previously validated metric, while also showing that some cells still aggregate in the Low Aggreagtion mutants.

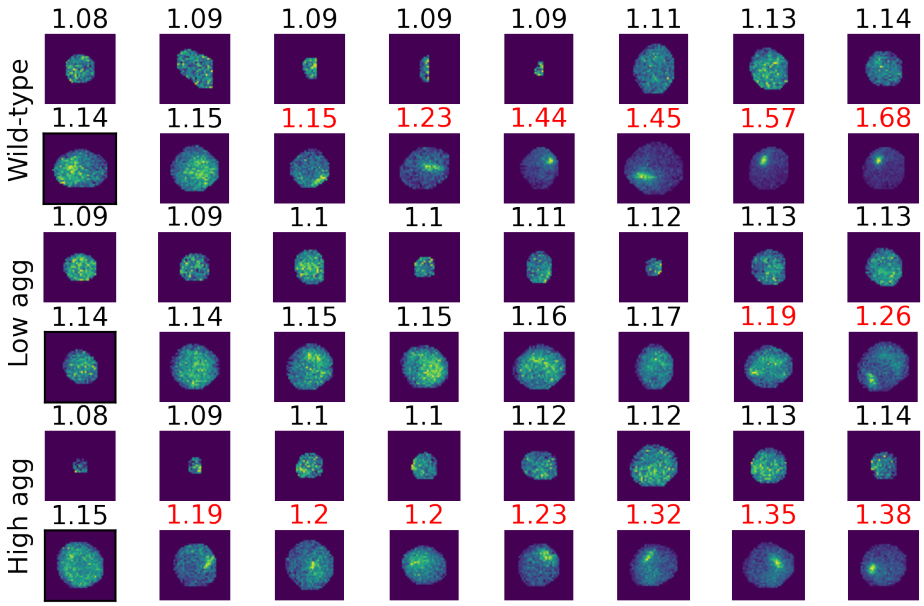

Figure 5.6: **Representative examples of cells in starvation.** Cells were randomly sampled around a feast-famine transition. The background of each image is set as the lowest value inside the cell to highlight intracellular pixel intensities. the value on top of each image is the *Max5 by median* metric, and they are sorted first by group (two rows per group) and then by this value (horizontally, top row first). The colour of these titles indicates whether they were manually annotated as "cells with aggregates". Cells were moved to the centre of the image; some cells may appear cropped because they were located at the edge of a tile.

5.3.4 Consecutive famines reveal memory-like behaviour in the wild-type

Before studying dynamical properties, I set out to characterise the distributions and responses in the simplest way possible. Figures 5.7, 5.8, 5.9 and 5.10 show

the distribution of average aggregation comparing consecutive or equivalent stages. We compare consecutive feast-to-famine transitions, and compare the average aggregation of every cell: feast-against-feast and famine-against-famine. I wondered whether cells have memory of previous starvation and changed their behaviour. I also questioned whether the distributions of mutants and the wild-type are similar to each other.

Figure 5.7 shows the basal aggregation of all strains during the two feasts. It sets the baseline to compare further stages. In panel 5.8, which compares the first feast and the following famine, the distribution of all strains expands rightward, indicating widespread filament assembly. The change is less pronounced in the low assembly mutant (Figure 5.8). Similarly, we can compare the distribution being stretched vertically between Figures 5.7 and 5.9; as before, the wild-type and the high aggregator populations show a wider response when compared to the low aggregator. It is also noteworthy that the wild-type distribution shows a higher basal response during its second feast.

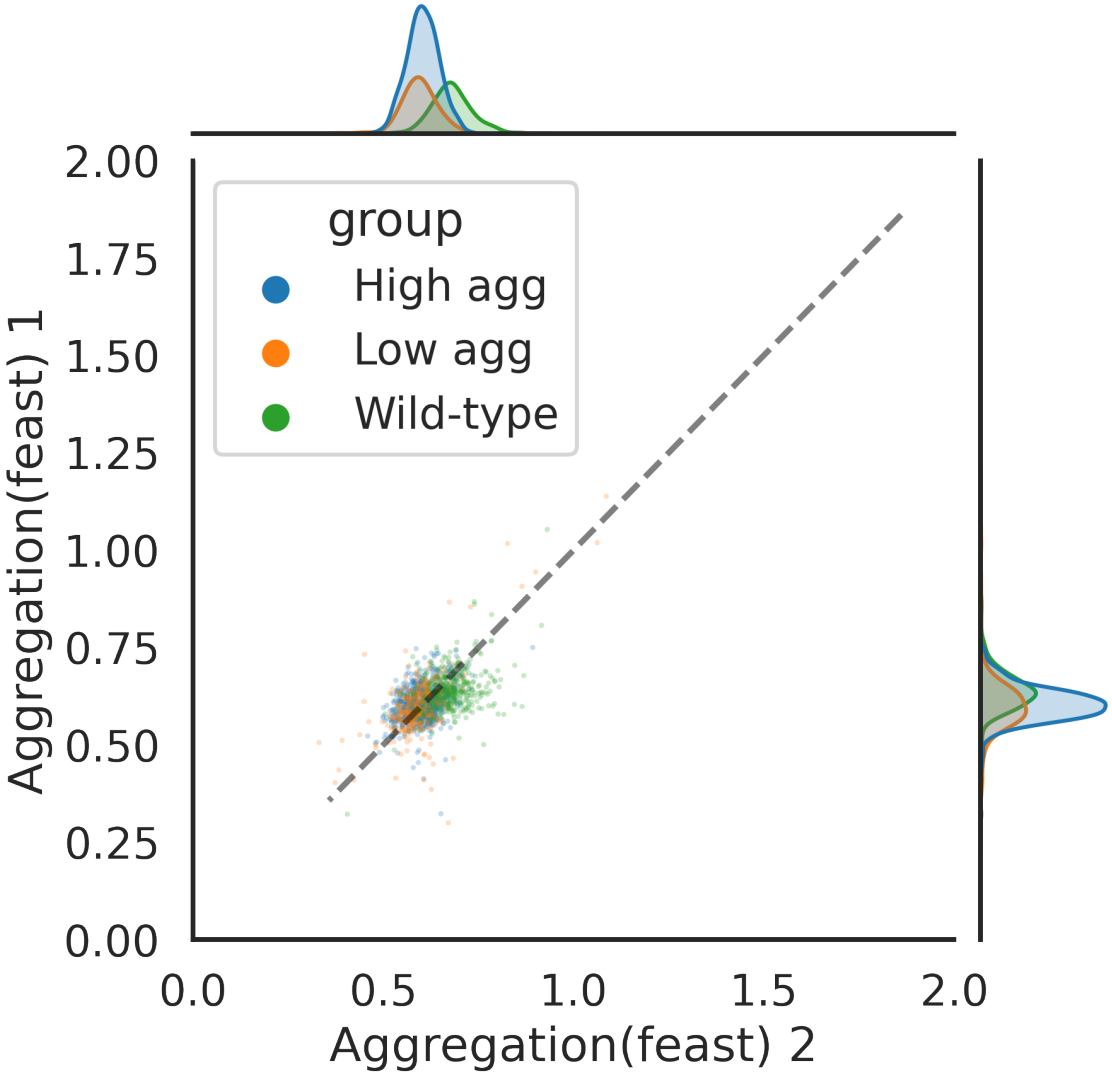

Figure 5.7: **Single cell comparisons of aggregation metrics for consecutive and equivalent stages.** Aggregation is measured as the intracellular five pixels with highest fluorescence divided by the median value. During feast periods all strains sustain low aggregation at similar levels. Media conditions are SC in 2% glucose for feast, and glucose-less SC for famine. The number of cells for each strain are as follows: 266 cells for the wild type (ura8), 296 for the low aggregation mutant (*ura8H360A*) and 714 for the high aggregator one (*ura8H360R*). These experimental conditions are shared by this and the following three figures. This experiment was repeated twice, with similar results.

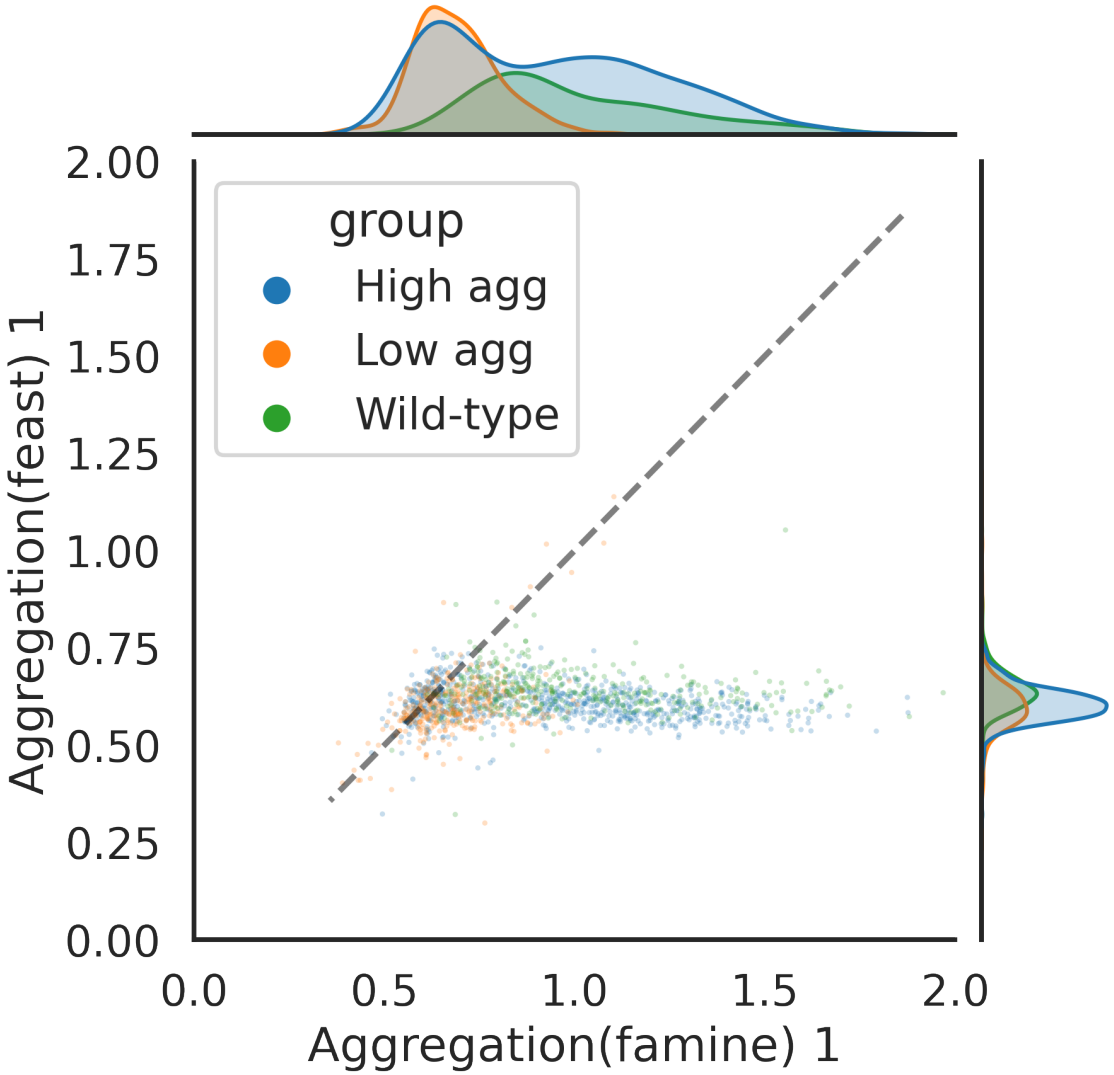

Figure 5.8: **Comparison of first feast and famine set.** The first famine period triggers widespread filament assembly, but each strain reacts with varying intensities: wild-type and high aggregators strains produce higher responses than low aggregators (*ura8H360A*).

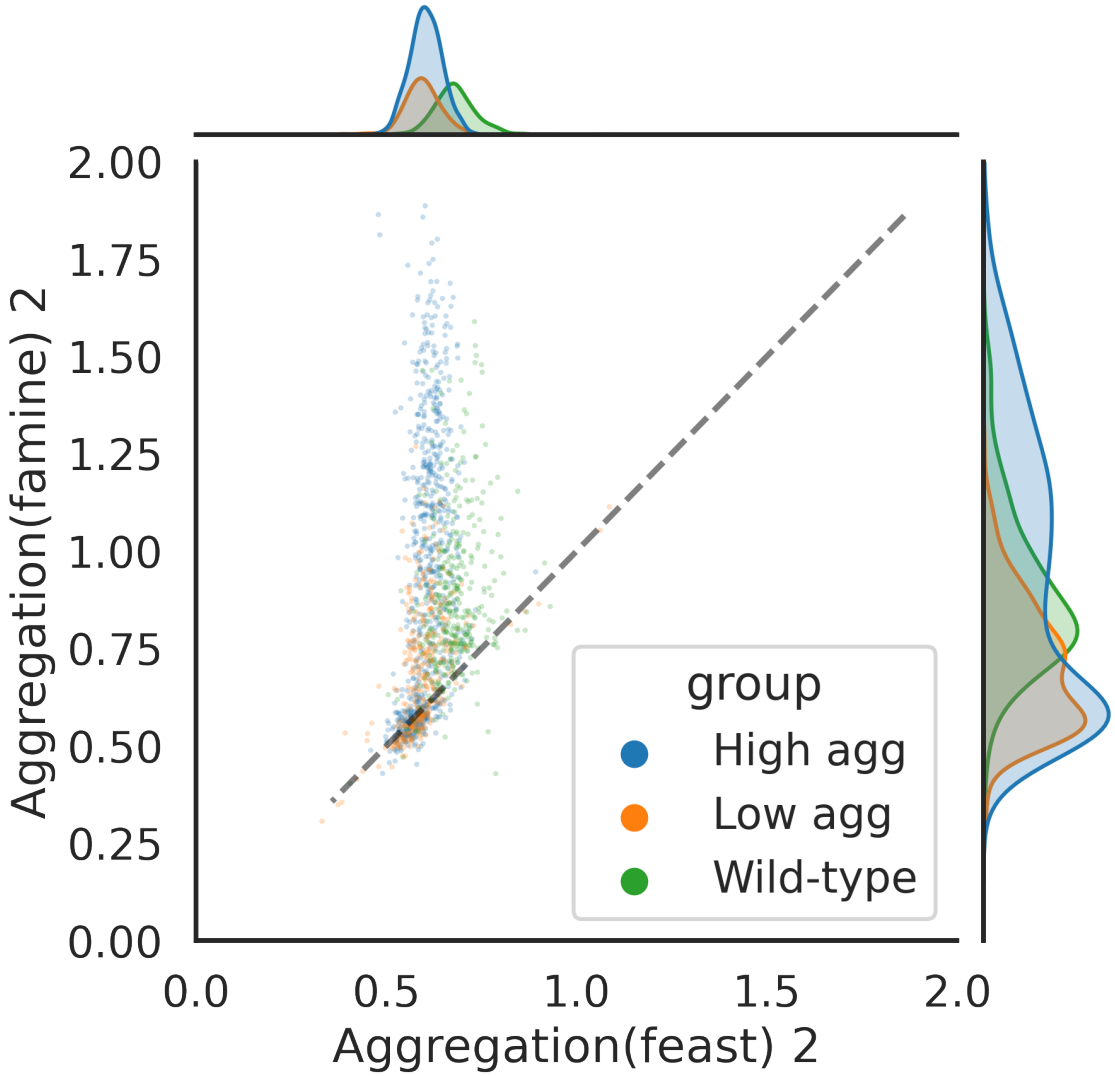

Figure 5.9: **Famine-to-feast transition comparison.** Analogous to the previous figure, this feast-to-famine transition reveals maintain their ability to aggregate, yet aggregation in the wild-type seems to falter.

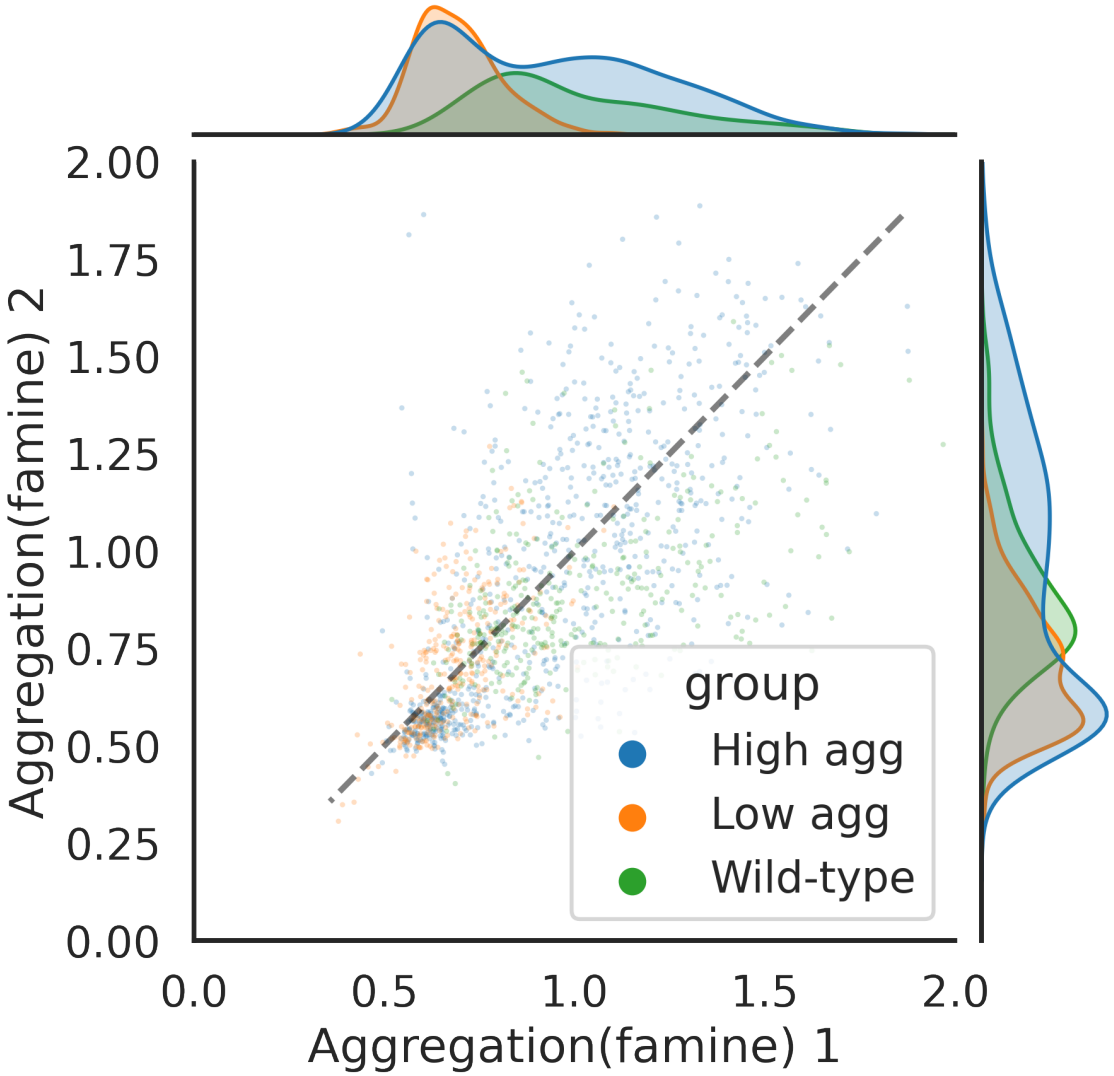

Figure 5.10: **Comparison of protein filamentation of both famines.** Comparing famine-to-famine aggregation at the first and second famine periods we can see how the high aggregator and low aggregators react alike, their slopes are near the diagonal, yet the wild-type's second famine was less pronounced than the first one, which implies either a memory mechanism, or famine is less deleterious for the wild type, reducing the probability of aggregation.

5.3.5 We generate a set of features from time lapses of both mother and daughter cells

The size and complexity of our data is both a blessing and a curse, as it demands bringing the amount of data to a manageable size for interpretation, not without its caveats. While accessible and informative, averaging single cell signals over time obscures heterogeneity, which is a unique selling point of our approach. To extract biological insight from thousands of time series, one of our goals is defining collective behaviours to explain single cell behaviour from other features sourced from that same cell (Chatfield, 2003).

As a first approach to analyse our time series - from hereon referred to as traces - is calculating their average change over time. Figure 5.11 shows an example of this applied to the volume signal. The top panel shows the family traces of the volume signal, that is, the trace for the mother (solid light blue line), together with its daughters (the remaining non-vertical non-solid lines).

As a measure of mother-bud pair fitness, we use the maximum change of a daughter's volume. After testing mother, bud and combined volume growth, it provided similar estimates to the sum of mother and bud, and its calculation is simpler. Maximum rate of volume change occurs when the cell grows at its fastest under the best possible conditions. Under three assumptions this metric can be used as a proxy for growth: First, all cells grow under identical environmental conditions (Crane et al., 2014); second, most volume growth in a mother-bud pair occurs in the latter (Allard et al., 2018); and third, cytokinesis occurs at a minimum bud volume, providing enough time for growth to be quantifiable (Kellogg, 2003; Soifer & Barkai, 2014; Turner et al., 2012). These three conditions are true for our experimental setup. Thus the maximum growth rate of the daughters should be a reliable indicator of the mothers' growth rate.

Figure 5.11 shows how we use family traces to produce time-independent features for both mother and buds, one number representing the mother cell and another one the bud. The top panel shows the standard volume trace, while the bottom panel

shows the conversion into another one. In this case the transformation shows the change in volume over time. We then express each cell in the family as its maximum volume change. At this point we need to decide how to compress this information into a single number.

The mother's maximum growth rate can be expressed as:

$$\text{GROWTH}_M = \max(V_m(t) - V_m(t - \Delta t)) \qquad t_1 < t < t_{max} \tag{5.7}$$

Where $V(t)$ is the (post-smoothing) volume of the cell at time t. Similarly, the instantaneous volume growth of a given daughter is given by:

$$\text{MAX}_{trace} = \text{MAX}(V_d(t) - V_d(t - \Delta t)) \qquad t_1 < t < t_{max} \tag{5.8}$$

And the daughter-driven growth of the mother is given by the maximum growth of any daughter (also called m5m_max_daughter):

$$\text{GROWTH}_D = \text{MAX}(\text{MAX}_d) \qquad d \in \text{daughters}(m) \tag{5.9}$$

Thus the expressions shown in Equations 5.8 and 5.7 result in one number for the mother and one for the daughter, in this case , representing the volume change in the mother and the maximum growth among all its daughters, respectively (This is obtained by selecting the maximum among all arrows in the bottom panel of Figure 5.11).

This approach to collapse individual or multiple time series into a single number - from hereon just referred to as a feature - works for traces other than volume. For instance, we can use it on the mean fluorescence of a single cell or the *max5 by median* metric. My next aim is to characterise single cells (and hence mother-bud pairs too) using multiple features.

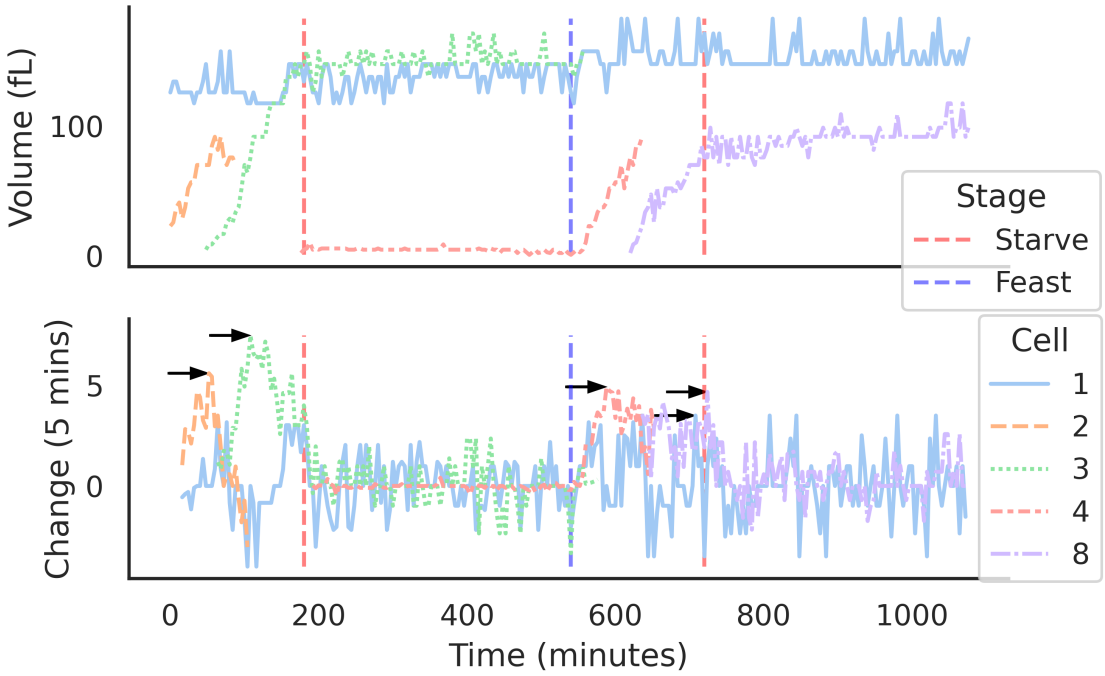

Figure 5.11: **From mother-daughter single cell traces we generate a set of features aiming to characterise their dynamics.** This example shows volume family traces. Dashed vertical lines indicate environmental transitions, red for feast-to-famine media and blue for famine-to-feast. The numbers display cell unique identifiers. Cell one is the mother of all other cells. The bottom panel shows the discrete time derivative in volume (after smoothing using a Savitzky-Golay filter). Black arrows in the bottom panel indicate the maximum volume change for each cell, which I use to calculate both mother and daughter maximum growth rates.

5.3.6 We calculate signalling and phenotype correlations by curating mother and bud features

Features produced from a common trace are usually correlated, but our aim is to find correlations between signalling (CTP synthase aggregation) and physiology (cell volume and growth). Three main clusters become visible in our correlation plot. The first cluster, occupying the top left corner, shows the cells' volume and growth features. The second cluster, in the middle, shows aggregation and change in aggregation features. The third and smallest cluster, located in the bottom right

corner, contains two features indicating number of buds and average time between buds.

The checkers-like appearance of the first two clusters arise from alternating features using mothers and daughters. While partially correlated, the fact that the features derived from fluorescence (e.g., any *max5 by median*) are not completely so implies that the overlap of mothers and daughter still permits independent dynamics. In other words, knowing the signalling status of a mother does not guarantee knowing the daughters'.

In addition to signalling-to-phenotype comparisons, we get phenotype-to-phenotype correlation; while results may not be too surprising they do increase our knowledge, characterising single cell growth. For instance, within the first cluster of correlations, the volume of cells at the beginning and end of the movie are highly correlated (first and third features); in other words, mother cells do not usually grow much faster than each other - most cells grow at a similar pace. At the other end of the spectrum, the initial volume of daughter cells is not correlated with any other feature, which is also expected - but reassuring. Signalling-to-signalling correlations tend to contain more redundancy and thus be less informative, excepting the previously-mentioned mother-vs-daughter.

The bottom right cluster in Figure 5.12 compares the average time between buds being identified, in other words, the time between budding events, and the total number of buds. It serves as yet another confirmation of the coherence between our features and provides a limit for negative correlations.

I limited the amount of features to three provenances (volume, aggregation and bud discovery) for ease of visualisation, but many more signals and features were compared, such as average fluorescence. They were either not too informative or redundant compared to the features in Figure 5.12.

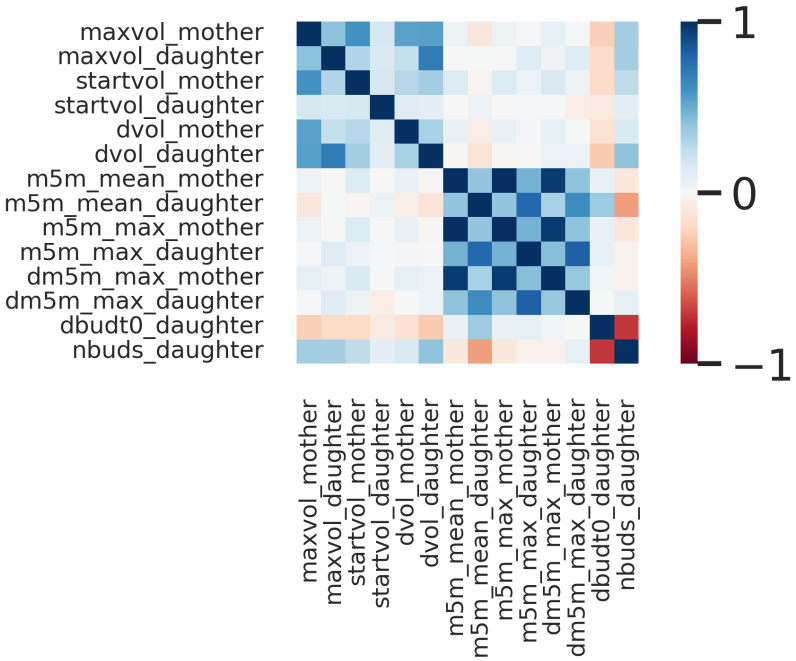

Figure 5.12: **Pair-wise Spearman (Rank) correlation between cell features.** Each feature corresponds to either a mother cell or a set of daughters except for the last two (average interval between consecutive buds and total number of buds), which depend only on daughters. The first six features describe growth and arise from volume traces (maxvol_mother, maxvol_daughters, startvol_mother, startvol_daughter, dvol_mother, dvol_daughter), the following six from describe aggregation using *max5 by median* traces (m5m_mean_mother, m5m_mean_daughter, m5m_max_mother, m5m_max_daughter, m5m_max_mother, m5m_max_daughter) and the last two from bud discovery (dbudt0_daughter and nbuds_daughter). The exact calculation of each feature is explained in the glossary.

The most notable correlation is the mean aggregation compared to the total number of buds (the last column, red square in the middle). In accordance with the bottom right corner's negative correlation between average time of bud emergence, measured as the average time it takes us to find two consecutive daughters, and number of buds, mean daughter aggregation correlates positively with the average bud spotting time. I further investigate these features to better confirm whether aggregation and growth are linked.

5.3.7 If sorted by daughter average aggregation, proliferation trends negatively correlate

As a first approach, we show in Figure 5.13 both signals after sorting them by the average aggregation of daughters. Due to inherent accumulation of biological and technical noise, we use a Savitzky-Golay filter to observe the general trend of budding. This results in an overall negative trend, where the smooth line reaches ~6 buds for cells whose daughters show minimal aggregation and ~2.6 for mothers of responsive daughters.

This approach confirms the correlation spotted in Figure 5.12, but it does not provide information on the population distribution of our mutant strains. Neither does it account for the mothers' aggregation, can provide useful information. The distributions of both mother's aggregation and the growing mother-bud pairs are useful to understand the factors involved in their decision-making.

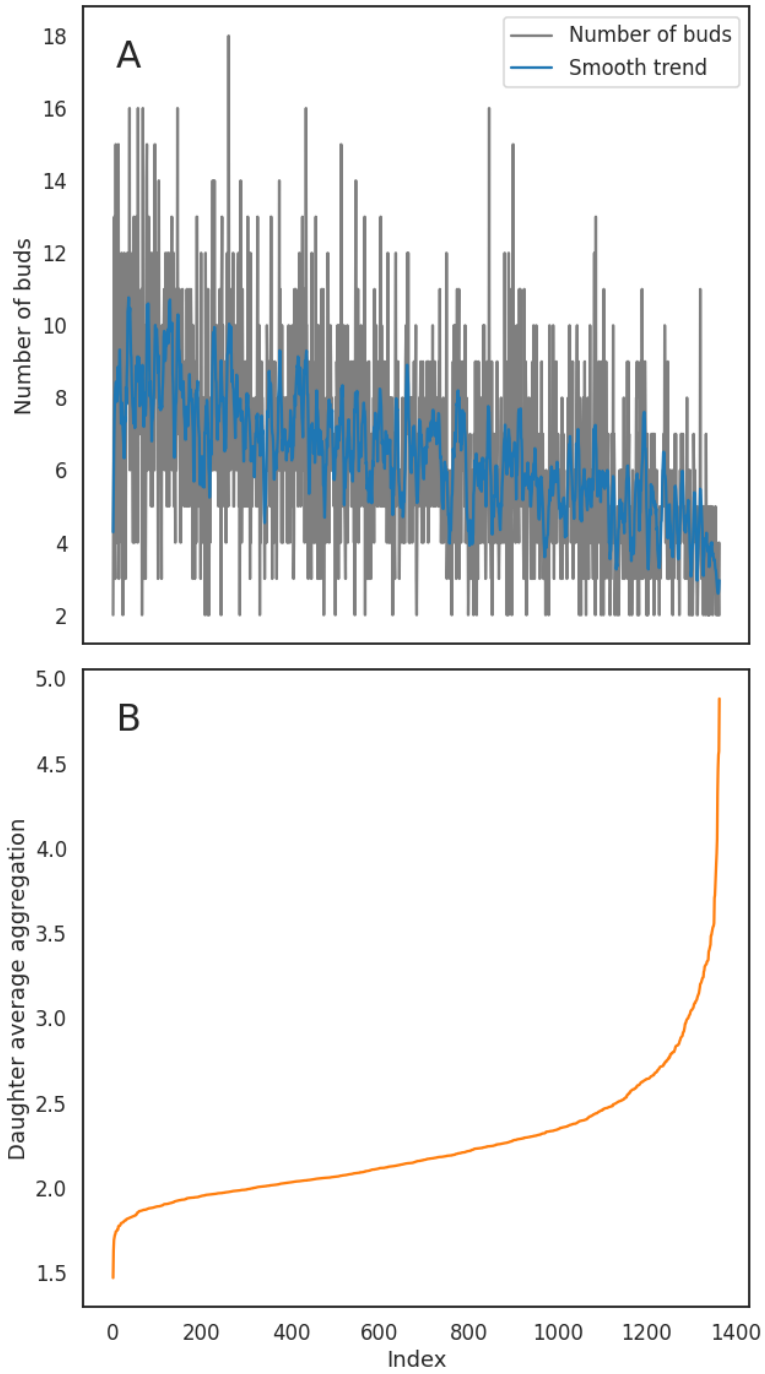

Figure 5.13: **Aggregation in daughter cells correlates negatively to the trend in number of buds.** The top panel shows single cell buddings for each mother cell. Cells are sorted by their daughters' average aggregation. The blue line over the grey signal is the smoothed trend. The bottom panel shows average aggregation used to sort the top panel.

5.3.8 Starvation-driven synchronisation is visible through budding events

Figures 5.15 shows a graphical representation of proliferative growth (i.e., growth measured as bud events instead of change in volume). Feast and famine cycles are visible as white and black stripes, respectively. The white points are time points at which buds were spotted for the first time (i.e., bud spotting).

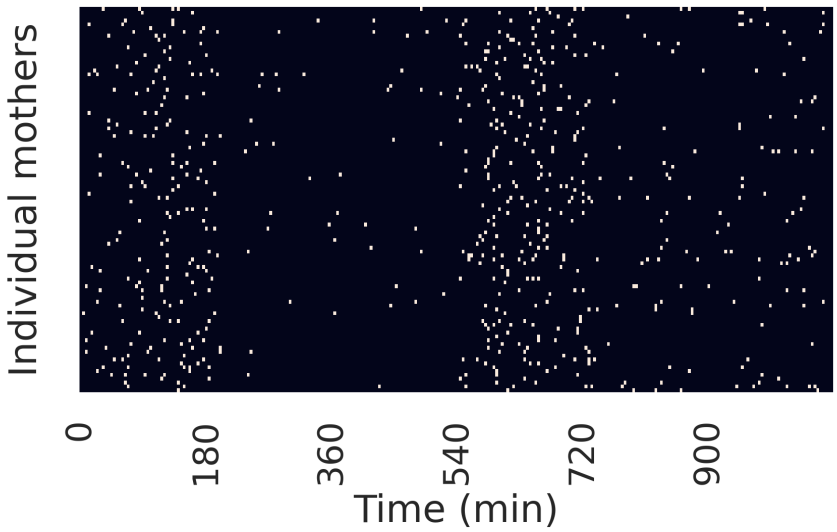

Figure 5.14: **Sample kymograph for budding events.** Each row represents an individual mother cell, and every column is a time-point. In this sample there are 100, approximately 10% the total number of mothers used for this subsection of the chapter.

In Figure 5.15 cells are still growing in volume during famine, albeit at a slower rate. Waves of birth events are visible during the second famine (between minutes 540 and 720). I predict the peaks to be a consequence an increasing number of cells reaching the size and resource thresholds necessary to start division while they are starving. They are unable to divide only due to lack of energy, but they are prepared in all other regards.

I hypothesise that the daughter cells grew beyond the regulatory size threshold during the first famine stage. Minor features of the time-dependent signal can be explained as data processing artefacts. The increasing birth numbers at the start

can arise from two sources, one biological and one technical. On the biology side, cells are likely undergoing a lag phase after loading them into the device, as they were growing in batch culture experiments, despite it having the same media and temperature. From the technical angle, the observation of births increasing over the first 200 minutes can also be explained by some cells not being yet trapped into their respective traps until later during the experiment.

Even if we select mother cells present for more than eighty percent of the experiment, not all these cells are present at the very start. The threshold was chosen to guarantee their presence for at least the last minutes of famine ($t_0 < 216$). Some cells might not become trapped until after a few frames. For this reason, the effective number of cells in our processed data may increase over the first 200 minutes. From thereon this number remains fixed until the last 200 minutes of the experiment, when it starts decreasing. Regardless of those considerations, the impact of some cells being absent at the edges has minimal effect on our analysis.

In the top panel of Figure 5.15, the valley that shapes the second wave at around the 700-minute mark is more pronounced than the first one. This hints at a delay in the second division following a famine. Alternatively, it could be a delay for passing START. The main question is why do many cells that divide during the first slump do not on the second one.

Although many white data points are visible outside the white stripes (the feast stages), they do so at a much lower rate than within these stages. They are generally false positives produced from tracking errors - that is why peaks tend to occur around time points of environmental transition. This is not always the case though, as a small - yet non-negligible - fraction of those events are true new buds.

Manually analysing these cases would consume too much time, but I consider it a venture worth pursuing if done semi-automatically through classification algorithms that can differentiate noise from real growth, such as *catch22* (Lubba et al., 2019). Cells that can change their internal state into being able to grow under no glucose are

most likely producing energy, from amino-acids. The reason behind this transition may be quantifiable, but would be tangential to this thesis.

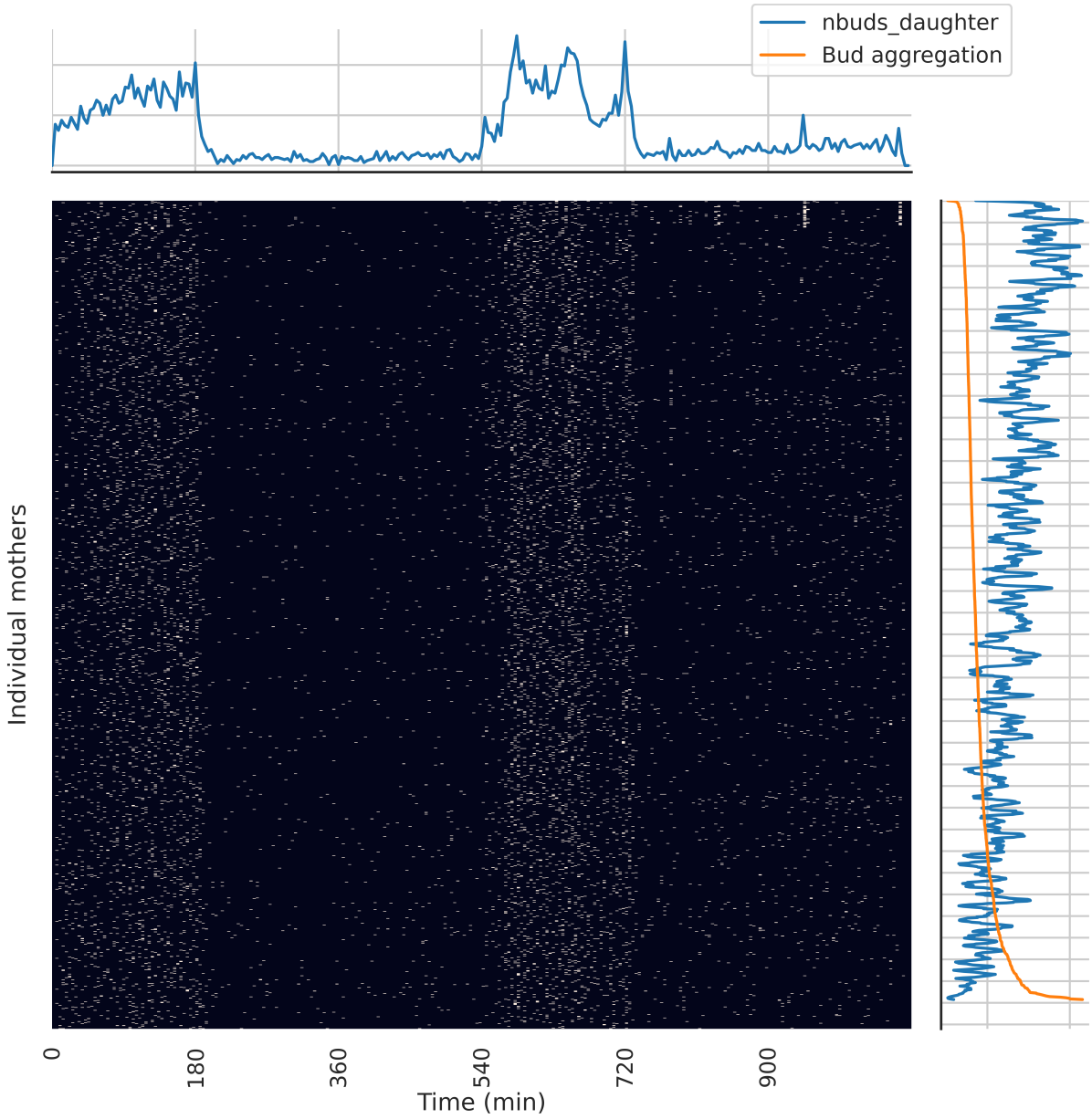

Figure 5.15: **Using a Joint Plot, bud spotting events illustrates how starvation synchronises populations and subsequent birth waves.** The central section shows birth events for mothers as a heatmap. Each row represents a mother and each column a time-point. White points in the heatmap indicate that a bud was spotted for the first time. The marginal plots, located on top and right, show the total birth events of each cell (right panel) and time-point (top panel). The top panel shows the raw sum of bud spotting events at each time-point. The right panel shows the number of buds spotted for each mother, smoothed using a Savitzky-Golay filter to display the general trend. The orange line in the panel on the right indicates protein aggregation, thus showing the order in which mothers are sorted.

5.3.9 Proteins reveals the phenotypes inside a genetically-identical population

Due to the fast aggregate disolution during feast periods across all three strains, I expected the famine stages to provide more information about the cell signalling phenotype. I assume any non-parametric clustering algorithm, such as k-means, can use these information-rich intervals to split our sea of individual time series into groups.

I assume two possible responses to each famine period (aggregation or indifference), though, as mentioned before, indifference means low reactivity, I assign one of four groups, $++$, $+-$, $-+$, or $--$ to each sequence by assuming each time-point as a feature and *k-means* clustering looking to assign four groups. Due to limitations of clustering algorithms on using missing data - a common challenge, in machine-learning - I only apply this to cells that were present during the entirety of the last three stages (the second feast and its preceding and posterior famines).

Based on their protein assembly behaviour, I assign names to each of these clusters based on two factors: Their initial responsiveness and their aggregation trend between famine stages. The first symbol in the nomenclature arises from absolute protein aggregation intensity, $+$ for high-aggregators and $-$ for indifferent cells. The second symbol indicates the behaviour of a cell during the second famine relative to the first one. $+$ for the increasing cluster and $-$ for the decreasing one. To draw a comparison between this nomenclature and the previous one from our wild-type clusters: "Indifferent" cells are equivalent to $--$ and a fraction of $-$, whereas "Responsive" cells are $+-$, $++$ and the remaining fraction of $-+$.

The filament assembly properties of all four clusters are shown in Figure 5.16 both as a function of time and as fractions of each strain's population. In panel A, the average aggregation profiles are shown surrounded vertically by their confidence intervals. In panel B all cells of each strain are split into fractions based on how many cells of each cluster they have.

As a side-note, The variability at the first minutes of the first famine ($200 < t < 300$) is not acquisition noise. Aggregation for those time points were visually confirmed for more than a hundred cells. Furthermore, no focus loss was observed in bright-field channels in those time points. Their origin could be due to signalling derived from osmotic shock, pulses after media change have been observed in multiple experiments, or, more likely, actual environmental variability produced by technical issues with the microfluidic system. It is worth noting that the second famine stage shows a smoother trend. The experiment was replicated and the results were similar, but with less data points, that is why I chose this one to display.

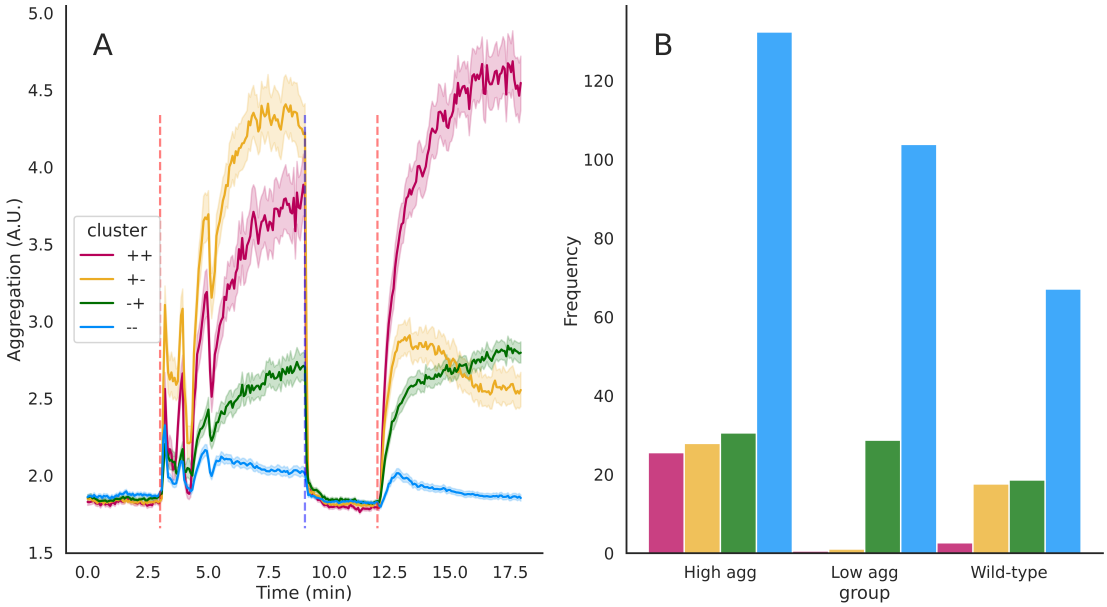

Figure 5.16: **We cluster our cells into four groups based on their fluorescence over time across different environments. A.** Average filament assembly of groups over time. Clusters are shown in different colours, and shade is 95% confidence interval calculated via bootstrapping. **B**. Stack plot of strains and their population composition according to aggregation profiles. *ura8* is the wild-type, *ura8*H360A the low aggregation strain and *ura8*H360R the high aggregator. For simplicity, I named each clusters at two consecutive positions, either plus (+) or minus (−) at each of the two positions. The first symbol is based on the intensity of the reaction of a cluster's average. The second symbol is based on the relative behaviour of that group compared to the first one. For example, +− has a strong first reaction to glucose deprivation the first time around, but this response is reduced during the second one.

Figure 5.17 validates the results seen in the previous figure. The clusters that show high protein aggregation in the lineplot tend to have more and clearer aggregates. These were randomly sampled, and may not be representative of the population heterogeneity, but offer a glimpse at what is going on inside cells. A clearer overview of the values grouped by cluster is visible in 5.18. Cells were moved to the centre of the image; it is unlikely to find cells with edges cropped because they were selected from individuals that persisted for most of the experiment, so they were likely at the centre of a tile. The catplot shows that cells with high aggregation metric values are almost always annotated as cells containing aggregates.

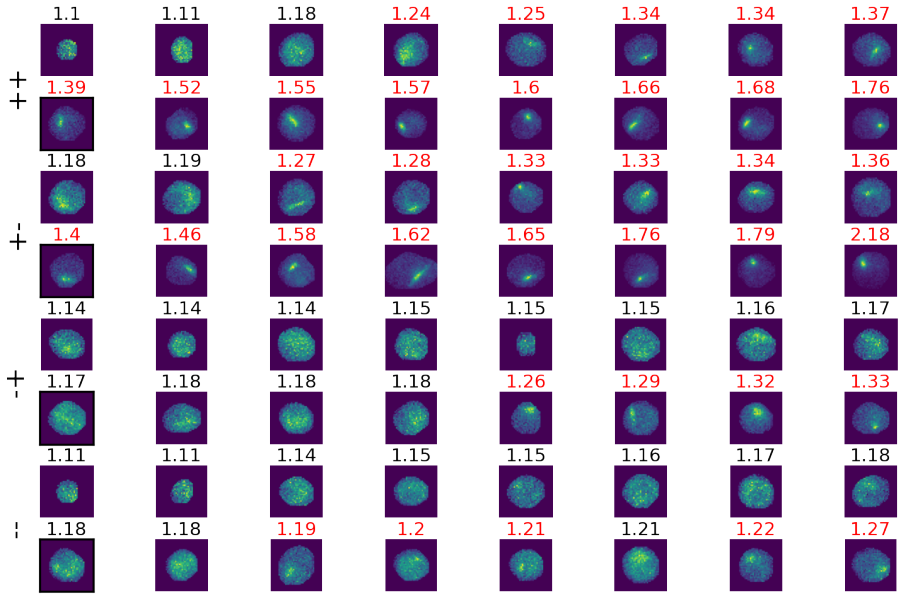

Figure 5.17: **Representative cells sampled from each cluster.** From the previous figure, I randomly sampled 16 cells for each cluster from the last hour of the first famine period. The text on top of each cell is its *Max5 by median* value. The cells are grouped by cluster (++, +−, −+ and −−), two rows each. They are sorted horizontally by the value on the top, on a row-per-row basis. Cells containing aggregates are are beneath red text.

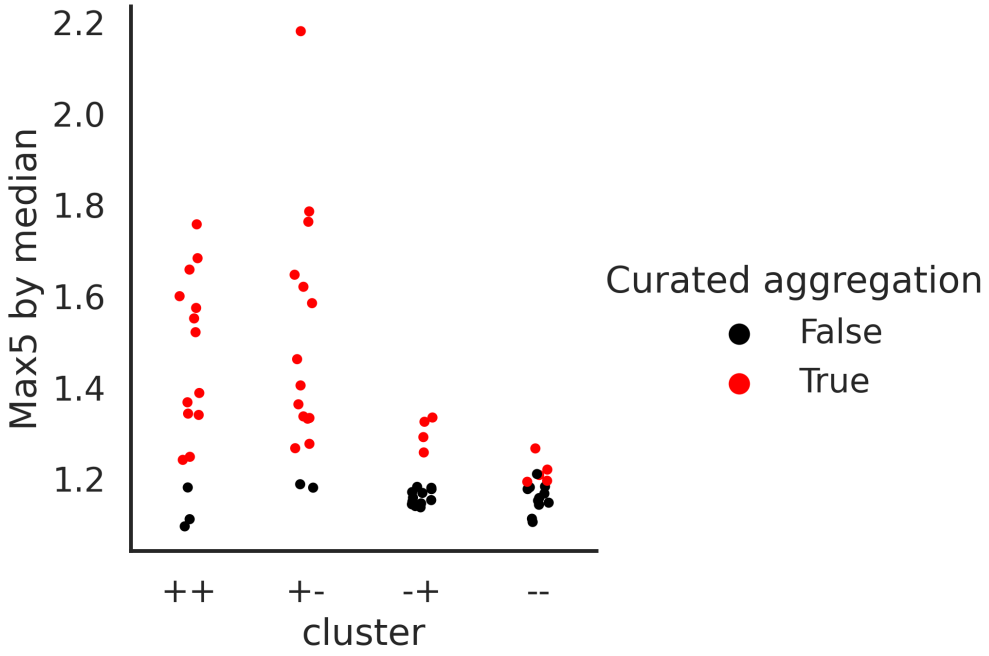

Figure 5.18: **The *Max5 by median* metric follows annotated aggregation.** Each column is a cluster of cells, and each point is a cell distributed vertically based on its *Max5 by median* metric. Red points were manually annotated as cells with aggregates, and black points are cells with no visible protein aggregation.

5.3.10 Indifferent cells restart growth faster

Figure 5.19 shows the correlation between daughter average aggregation and the number of buds with cells grouped by aggregation profile (the mother's aggregation entire time series). In both daughter aggregation and number of buds $--$ cells have a distinctive behaviour. The distribution of the daughter's mean aggregation (top plot) for the $-+$ group spreads across the region between $--$ and $++$ and $+-$. These last two show a longer tail, in all likelihood this is similar to their mothers, on whom the grouping was based.

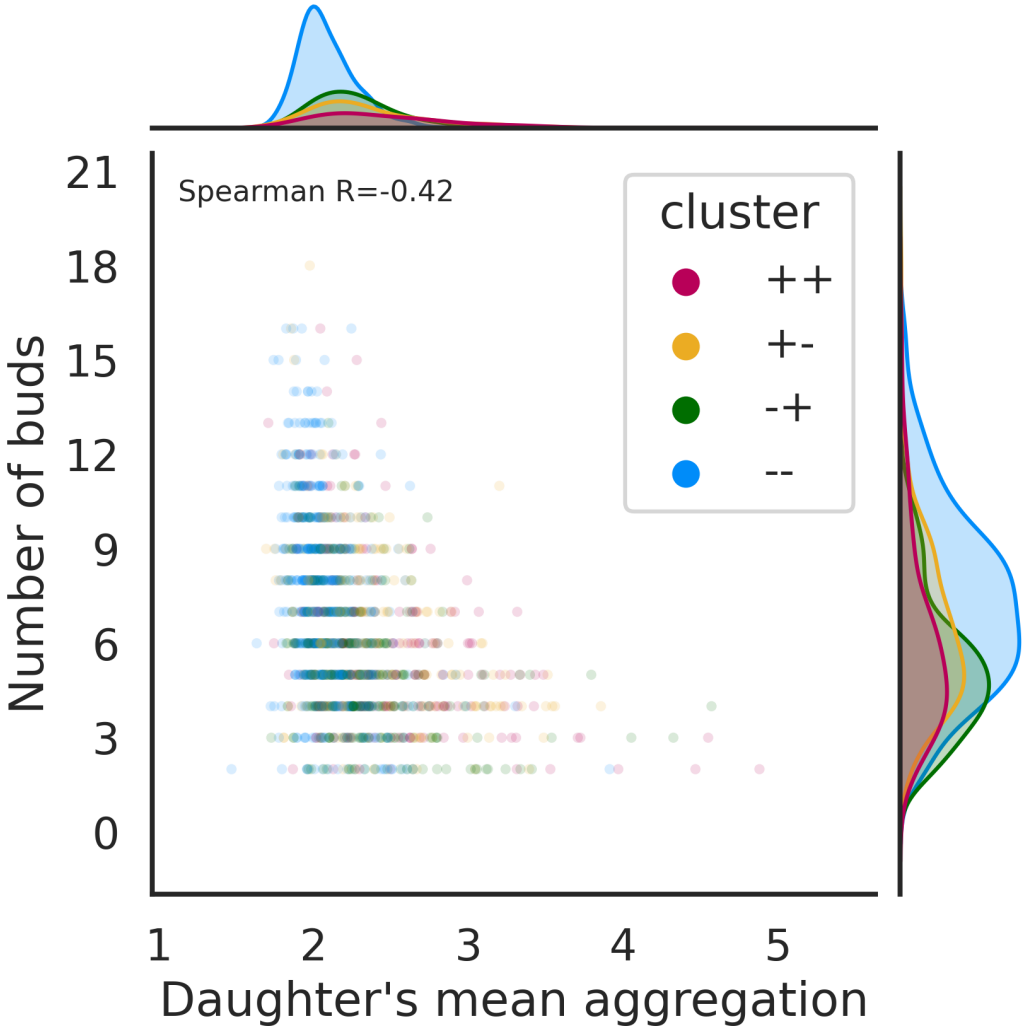

Figure 5.19: **The average aggregation of daughters correlates with the number of daughters produced by their mother.** Cells grouped in the consistently-insensitive group (−−) produce a distribution with lower averages and spread of budding events than the other aggregation profiles. The Spearman Rank correlation between these two variables across all cells is -0.38.

5.3.11 Mother and bud assembly metrics correlate

Figure 5.20 displays phenotypic features linking cytosolic properties of both mother and bud cytosol . Most mothers have higher aggregation indices than their daughters. This differential is even more pronounced for highly-responsive clusters (++ and +−).

The distribution of maximum aggregation for highly-responsive clusters (yellow and red) have a similarly broad range of responses, in contrast to the distributions of daughters with low response (blue and green histogram in the right panel in Figure 5.20) The top panel shows that non-responsive mothers have a bimodal distribution whose second peak (less pronounced, with more aggregation) coincides with the peak for the "-+" strain.

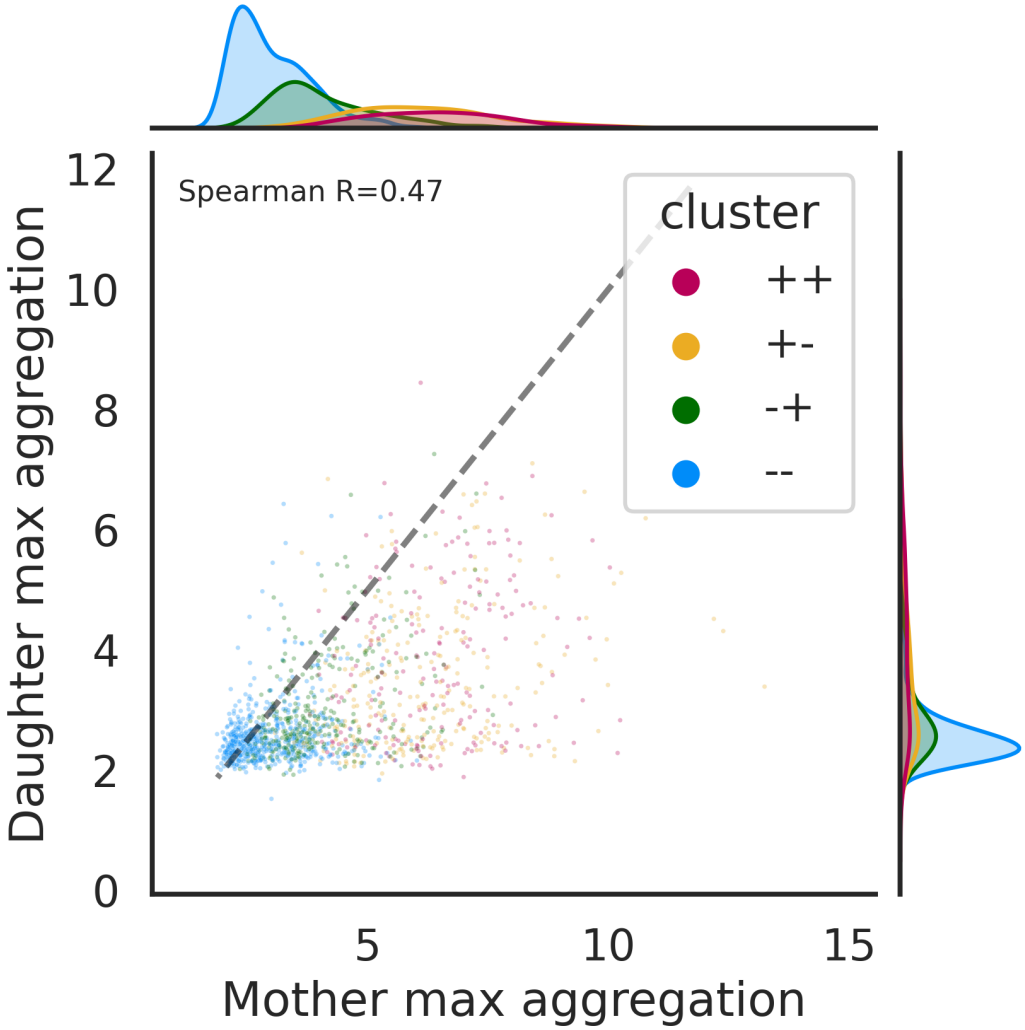

Figure 5.20: **Correlation plot comparing maximum aggregation of mothers versus daughters.** The marginal plots represent the distributions of each metric: the panel on top shows the mothers' aggregation distribution and the panel on the right shows the daughters'. Colours indicate the clusters. The dotted diagonal shows x=y. Spearman Rank correlation is shown in the central plot.

5.3.12 The genotype of a strain does not define the resulting phenotype: "clonal" populations are composed of multiple phenotypes

Figure 5.21 shows the correlation between aggregation of mothers and daughters, but this time grouped by strain (i.e., genotype). The wild-type spreads between the two mutants, of which the high aggregator is confirmed to account for most of the high-responders.

The distributions of wild-type and high aggregator strains show a longer tail, reiterating that cells that contain clear protein aggregates are non-existent in the low assembly mutant, shown by most cells having aggregation index lower than 4. It is also interesting that the high aggregator and low aggregator strains share a peak at the lower end of aggregation (top panel, leftmost peaks).

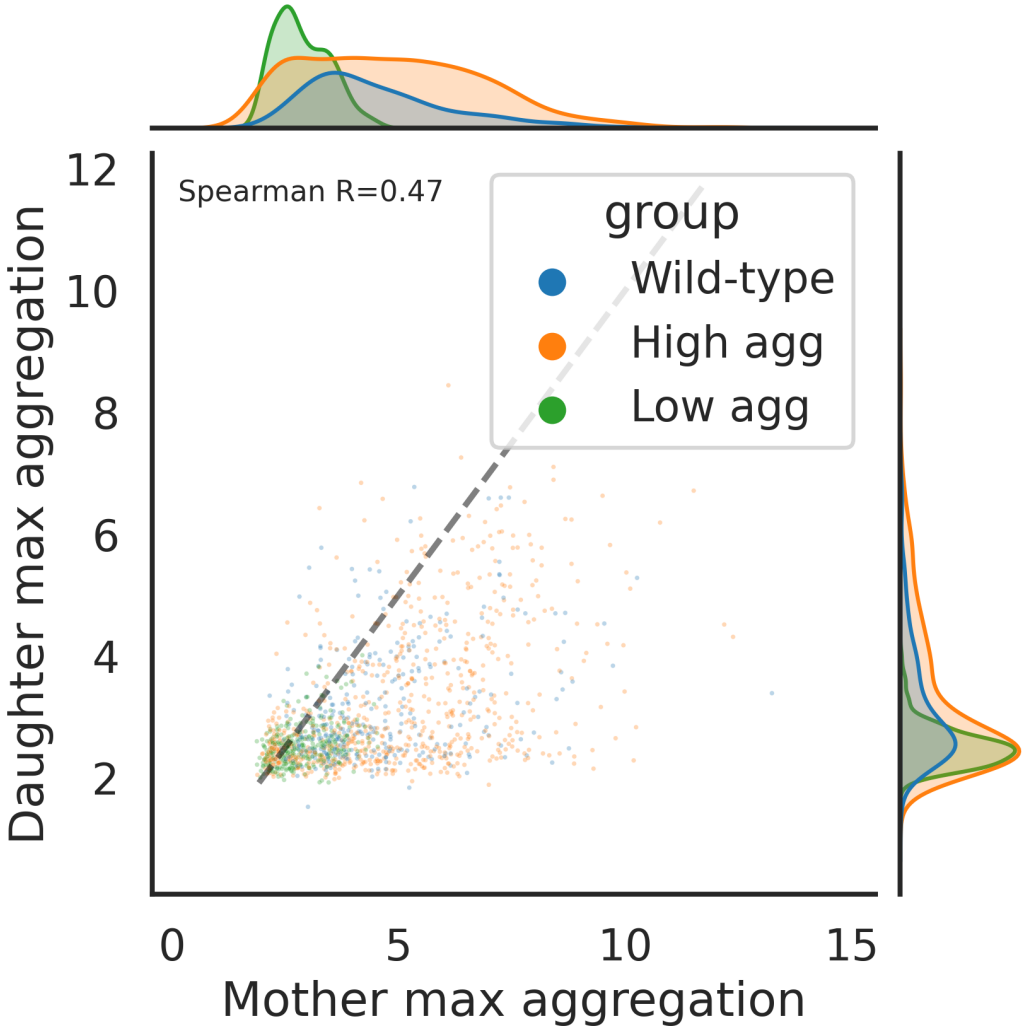

Figure 5.21: **Mother and daughter maximum aggregation grouped by genotype.** Marginal panels show the distribution of each axis, mothers' maximum aggregation at the top and daughters' on the right. The dotted diagonal line represents x=y. Spearman Rank correlation is calculated for all cells together.

Our first results seem to agree with previous studies. The average behaviour of filament assembly of both paralogs is similar and reaches a similar value during late starvation ($500 < t < 600$ minutes). This behaviour coincides with previous reports of isoforms co-localising (Franzmann & Alberti, 2019). Yet, according to *in vitro* negative stain imaging from *Hansen et al.*, the isoforms show slight differences in assembly dynamics from each other (Hansen et al., 2021).

5.4 Discussion

The size and complexity of our data is both a blessing and a curse, as it demands bringing the amount of data to a manageable size for interpretation, not without its caveats. While accessible and informative, averaging single cell signals over time obscures heterogeneity, which is a unique selling point of our approach. To extract biological insight from thousands of time series, one of our goals is defining collective behaviours to explain single cell behaviour from other features sourced from that same cell (Chatfield, 2003).

5.4.1 Some cells are predisposed to form filaments

In published literature, when cells are stressed for long periods of time, most cells form CTP synthase filaments (Noree et al., 2014). This hints at cytosolic aggregates as either consequence of or response against harsh environmental conditions, though triggered by energetic constraints and nutrient availability. It is known now that assembly of CTP synthase rods and, although transient, foci, is determined by the cytosolic pH of a cell and the abundance of its substrate (UTP).

If starvation lasts long enough, it is likely that after a first wave of "predisposed" - perhaps old and/or "unhealthy" - cells transitioning into this state, comes a slow and steady stream of cells switching to the aggregated state. Long-term experiments with low acquisition frequency would be necessary to prove this conjecture, thus, the currently technically challenging limitation to prove this theory is a technical one.

The main issue is phototoxicity in cells that are already stressed from lack of glucose, although I presume such an experiment is viable with existing infrastructure.

5.4.2 Only the Wild-type adjusts its response upon recurrent stress

Overall, data from sequential feast-famine cycles indicates that the wild-type behaves differently between consecutive feasts-famine periods, while strains with abnormal filament assembly do not display memory-like regulation. When comparing both famine periods the wild-type changes its behaviour. It shares a similar distribution to the high aggregator during the first famine, but aggregation is reduced during the second one. The cells that formed aggregates with high intensity (intracellular pixels with a low median and high maxima) did not aggregate as strongly during the second starvation period.

Although our results show similar dynamic as previous reports (Hansen et al., 2021) if we compare thoroughly, we also find one dynamic that is strikingly different. Both dissolution and aggregation are observable for all strains, though at varying intensities for each strain. This discrepancy may be explained by the experimental setup. In the previous report they transfer cells between media, and only look for assemblies dissolution for up until one hour, in an unspecified - likely low - number of cells. Another factor that may be causing this discrepancy is that *Hansen et al.* move some cells from one media to a different one, whereas we keep the cells in place and change the media they are under. Thus, these cells never sense high densities and - as far as we know - cannot communicate with each other when stuck in different regions of our microfluidic device.

5.4.3 Data mining of microscopy time lapses

The first results revealed the mother and bud growth patterns. Buds (or daughters) grow at a faster pace than mother cells during their early development, which confirms what decades of yeast research has shown, supporting the validity of our entire methodology. Feast-to-famine transitions abruptly stop growth the first time, but

on the second iteration, the buds' volume change remains mostly positive. Additionally, it is apparent that mother cells do increase in volume, albeit at a slower rate than their daughters. This is an unexpected discovery in light of yeast literature indicating that the volume of mother yeast cells remains largely fixed.

As a measure of mother-bud pair fitness, we use the maximum change of a daughter's volume. After testing mother, bud and combined volume growth, it provided similar estimates to the sum of mother and bud, and its calculation is simpler. Maximum rate of volume change occurs when the cell grows at its fastest under the best possible conditions. Under three assumptions this metric can be used as a proxy for growth: First, all cells grow under identical environmental conditions (Crane et al., 2014); second, most volume growth in a mother-bud pair occurs in the latter (Allard et al., 2018); and third, cytokinesis occurs at a minimum bud volume, providing enough time for growth to be quantifiable (Kellogg, 2003; Soifer & Barkai, 2014; Turner et al., 2012). These three conditions are true for our experimental setup. Thus the maximum growth rate of the daughters should be a reliable indicator of the mothers' growth rate.

Using this approach to mine correlations, I showed that after producing high volumes of data it is viable to analyse it all in an automated manner. This approach led to the discovery of a negative correlation between the filament formation inside a bud and the number of buds its mother has. Further research is required to better understand the mechanisms that produce this and other correlations.

5.4.4 Clustering aggregation signals reveals the phenotypic composition of mutants

Using an agnostic clustering algorithm as a tool to simplify the groupings and define these clusters as aggregation profiles. This allowed us to classify cells not based on their genotype, but on their phenotypic trait. In this case the phenotypic trait was its collection of responses to external stimuli.

The high aggregator and low aggregator mutants showed bimodal distributions, while the wild-type does not. After clustering, the low-end of these distributions was grouped together, while the other two belonged in separate clusters. This suggests that changing the likelihood of aggregation to occur switches the populations to other stable states, different to the one in the wild-type.

Different clusters show unique filament assembly dynamics, and yet the composition of every subpopulation (i.e., strain) is an amalgam of multiple clusters. It is important to study population structures determine the probabilities of how an average cell may respond to environmental cues.

5.4.5 Buds surpass regulatory size threshold during famine

There are bursts of bud events after a famine. These may be the consequence of an increasing number of cells reaching the size and resource thresholds necessary to start division while they are starving. They are unable to divide only due to lack of energy, but they are prepared in all other regards (Barber et al., 2020).

I hypothesise that buds grew beyond the regulatory size threshold during the first famine stage. Minor features of the time-dependent signal can be explained as data processing artefacts. The increasing birth numbers at the start can arise from two sources, one biological and one technical. On the biology side, cells are likely undergoing a lag phase after loading them into the device, as they were growing in batch culture experiments, despite it having the same media and temperature. From the technical angle, the observation of births increasing over the first 200 minutes can also be explained by some cells not being yet trapped into their respective traps until later during the experiment.

The availability of cells at the start of the experiment may influence births events on the first time points of the experiment. Even if we select mother cells present for more than eighty percent of the experiment, not all these cells are present at the very start. The threshold was chosen to guarantee their presence for at least the last minutes of famine. Some cells might not become trapped until after a

few frames. For this reason, the effective number of cells in our processed data will always increase during the first time points. From thereon this number remains fixed until the last 200 minutes of the experiment, when it starts decreasing. Regardless of those considerations, the impact of some cells being absent at the edges has minimal effect on our analysis.

In some rare cases, cells restart growth before glucose is back in the media, this has been seen by other groups (Jacquel et al., 2020). This has also been observed when analysing the data for this thesis, regardless, manually analysing these cases impossible in practice, but I consider it a venture worth pursuing if done semi-automatically through classification algorithms that can differentiate noise from real growth, such as *catch22* (Lubba et al., 2019). Cells that can change their internal state into being able to grow under no glucose are most likely producing energy, from amino-acids. The reason behind this transition may be quantifiable, but would be tangential to this thesis.

5.4.6 Aggregates may not just be a response to stress, but a consequence

This high correlation between protein aggregation indices of mothers and daughters is expected as well, due to the shared cytosol. What was surprising to discover that their relationship is not that strong, as one would expect if the cytosolic contents are shared and proteins diffused freely between mother and bud.. Alternatively, this outcome may be a result of a time span in which the daughter is independent, assuming the physiological state of the cytosol drifts away from the mother's after cytokinesis.

Mother-bud pairs whose buds contain protein aggregates were likely unable to maintain their pH and changed "strategy", going into an even more acidic state, which in turned triggered the aggregates. This could be related to dynamics shown where a subpopulation of cells is eventually driven towards death when unable to transition

into respiration (Jacquel et al., 2020). Our stress is not as harsh as theirs, so our cells do not die, but those cells are the ones that fail to grow as fast.

6 Summary and general discussion

6.1 Chapters overview

In Chapter 2, I demonstrated how my tracking system performs highly and is robust thanks to the main parameter being pixel size. All other parameters are implicitly embedded in the classifying models. It does not make explicit assumptions about the outline of cells. In its current state, it is not suitable for cells with high motility, such as the ones shown in (Ulicna et al., 2021). Nevertheless, it is a general algorithm that does not make assumptions about the data source, requiring only cell labels. It performs a single task as best as the training data provides, and while heuristics might improve its performance in specific cases, I chose to leave that to further elements down the pipeline to make it portable enough to migrate it to other systems.

Using a joint segmentation and tracking benchmark is necessary to contextualise the accuracy of the whole pipeline, but it is beyond the scope of this thesis, and is further explored in (Pietsch et al., 2022).

In Chapter 3 I explained my design of a pipeline for automated microscopy time lapse analysis. I included the main challenges that arise when dealing with high-throughput datasets and which approaches I found to be the most adequate. It aims to show software architecture design and as a general map for developers.

In Chapter 4 I presented a review of available tools that can help develop sustainable scientific software. I covered a range of goals, from advanced text editing to automatic package publishing. Many of the tools offered were instrumental in the development of ALIBY, and have been tested by multiple people over the course of two years.

In Chapter 5 I showed a quantitative leap compared to previous aggregation studies: we are able to analyse five to tenfold more cells. This uncovers the population heterogeneity that may lie hidden by low cell counts. We also obtain a more robust average behaviour to feed mathematical models trying to explain protein aggregation dynamics. Combined with lineage predictions this technique opens a window on the complexities in yeast cell populations.

6.2 Experimental conditions must be accounted for interpretation of results

We also demonstrated why experimental context matters by showing regulatory differences between batch culture experiments and microfluidics. One explanation for this mismatch is the environmental conditions of those setups. In batch growth intracellular pH is acidified not only due to signalling but also because of by-products of growth; their metabolic and regulatory state might be different to that of cells that only suffered glucose deprivation while the other conditions stayed the same. While both experiments move cells to a carbon-deprived media from a carbon-rich one, their cells were likely experiencing more acidic conditions due to fermentative growth "waste" accumulating (Orij et al., 2011), while the cells in our experiments were starving in the same pH as at the start of the experiment.

Using minimal media should stop volume increases during starvation periods. All of our experiments were performed in Synthetic Complete media, which cells may use to grow even in absence of a carbon source - although, as seen in the data presented, as well as previous data from us and other groups, the vast majority cells are unable to switch into that mode of growth from fermentation within the time span of our famine stages (Jacquel et al., 2020).

The widespread aggregation indicates that starvation-driven acidification occurs to varying degrees across the populations. I think that the intensity of their reaction is correlated with age, as we have seen bigger cells having a higher probability to

assemble filaments. This agrees with previous observations of older cells having a more acidic cytosolic pH and a more basic vacuolar pH.

Another surprise is that we did not confirm any kind of fitness advantage in forming filaments, which is one of the hypotheses surrounding the phenomena (Franzmann & Alberti, 2019). On the contrary, excessive filament assembly impairs both proliferation and biomass production. This implies that it is a consequence of cells not being able to maintain their pH homeostasis - "healthy" cells do not need filaments. After all, pH and nutrients homeostasis, ageing and stress responses are all intertwined (Gutierrez et al., 2022; O' Neill et al., 2020).

6.3 ALIBY as a general microscopy analysis pipeline

On a more general note, transforming multidimensional images into data that is insightful to biology is not a challenge to be underestimated. The amount of coordination required between multiple components need more abstraction than what sets of scripts can provide. All the more so when our long-term goal is to generalise for multiple experimental setups.

I defined single cell growth and aggregation features, and distinguishing important and unimportant features is where the bottleneck lies now. In the future we may want to let a deep-learning algorithm define and learn these from signals with different time intervals and environmental conditions. In my opinion, neural networks should reach the point where they can find the best predictors from intracellular signals. This advance will enable automated data mining of cell behaviour and signalling at the cost of some interpretability. LSTMs (Yu et al., 2019) and recurrent transformer networks (Hutchins et al., 2022) are prime candidates to serve as a base, the former for tracklets, Transformers because of they are unaware of information context, as daughters of a mother are independent entities that may be analysed all within the same context.

The future of ALIBY - like most scientific software - depends on its extensibility and how it evolves to fullfill the everchanging needs of scientists. From my experience

exploring existing software tools for Chapter 4, most open-source software that is associated to publications is abandoned unless a community adopts it; few software can survive the test of time without collaboration of more than one research group. I have, to the best of my abilities and situational constraints, followed good practices and generalise to make ALIBY useful to the – at the time of writing, small – yeast microscopy community.

ALIBY is functional but incomplete, for software is seldom finished. More documentation, tutorials and guides, most times neglected components of sustainable software, demand writing. Additionally, it lacks features I consider important. For instance, access to multiple segmentation, tracking and lineage models; integrating work from existing published models would make it much more versatile and attractive for biologists of multiple branches, beyond the realm of computational biology. Nothing is stopping generalising to other organisms, we already used it on other yeast species, namely *Cryptococcus neoformans* and *Candida glabrata*. Our current lineage assignment inside BABY is the only component making yeast-specific, as it uses mother-daughter asymmetry to predict mothers and buds. A roadmap is necessary to delineate the software features that could increase community adoption.

6.4 Biology is dynamic and heterogeneous, and our analytical approaches should account for that

Due to the intrusive nature of most -omic methods, molecular biology studies usually use snapshots of population averages. The mechanistic link between genotypes, metabolism regulatory signals and visible phenotypes cannot be elucidated from snapshots. By tagging proteins that produce inspectable phenotypes, such as protein concentration, aggregation, localisation and organelle morphologies. Performing these analyses on hundreds or thousands of cells allows us to observe for the first time how population structures change over time at the single cell level, linking a single genotype to a phenotypic traits distribution.

Dynamic data also provides direct cause and effect links. We can understand how the decisions made by one cell in the past determine its fitness relative to the population. This in turn provides insight on which are the main drives changes in population averages and distributions over time.

Recent advances in microscopy, imaging automation and image processing tools have made image acquisition and environmental control more accessible. Coupled with microfluidics and high liquid control precision we can regulate the environment in which cells grow. We characterised cell behaviours before, during and after stresses, finding consistent patterns. Dynamic environments inherently provide more information by forcing cells to react, while static ones usually provide information on cell cycle and ageing. Even so, few experiments have been performed to characterise cells in constant environments; a set of experiments to compare growth and proliferation in multiple conditions, such as an array of carbon sources, would be an important milestone for single cell studies.

Due to a lack of existing methods to analyse this type of data, I developed a approach that uses correlations and clustering. My goal was to create a computational framework to mine relationships between intracellular signalling and growth. Its level of abstraction enables portability to cater to different biological questions and signals, not only protein aggregation. The adoption of high throughput microscopy analysis by the broader biology community depends on how accessible and robust it is. By accessibility I mean how easy it is for an inexperienced user to generate and analyse their results – this includes accessibility to adequate imaging facilities. By robustness I mean how stable those results are for a wide variety of experimental setups. If ALIBY and the subsequent analysis pipeline become flexible enough to leverage existing segmentation and tracking tools, it promises to become an important tool for single-cell phenotyping.

6.5 Are biology-driven heuristics actually better?

The marriage between image analysis and time series is only possible nowadays thanks to the latest developments of computational tools. Computational analysis of biological images as a field is far from new. Signal processing is a field practically as old as physics themselves. Historical approaches to image analysis always tried to adjust physical models to produce descriptive models of biology. The development of novel architectures in the field of deep learning led me to ponder about the benefits of assuming biological relationships when building predictive models.

At some point in the near future, the community of bioimage analysts must weight the advantages and disadvantages of introducing domain-specific heuristics (in our case knowledge of biology to solve a particular question) as opposed to a "naive" machine or deep learning approach (e.g., not constraining the models using contextual biological insight). If biological scientists (or scientists-developers) codify prior knowledge like, for instance, known regulatory networks or expected growth rates for a specific carbon source, this may lead to faster short-term returns, but could as easily constraint a model or system, forcing the next person in line to start from scratch.

Biological preconceptions are a two-edged sword; scientists working in multidisciplinary fields must be careful when using the leverage that is knowing biology. If we distract ourselves too much on preconceptions, we run the risk of an entity brute-forcing their way using unimaginable amounts of computing power, such as massive deep learning models such as *AlphaFold* did to the field of protein structure (Ruff & Pappu, 2021). After working on collection of projects presented in this thesis, I think we should limit biologically-inspired heuristics to two sections: Data structures and biophysical models. An adequate data structure that, for example, describes the relationships between different species of a strain will provide the scalability necessary to aggregate data entries over the years. Biophysical models provide the interpretability that deep learning models seem to be lacking.

As a final thought, I think that overengineering algorithms and tools to fit too many biological assumptions makes them rigid, and though it seems like a good short-term solution, it will likely lead to those becoming useless over time. Frameworks and tools that evolve and improve while accumulating new curated data will become the dominant components in its encompassing scientific field.

7 Bibliography

Alán Muñoz / Thesis. (n.d.). In *Gitlab*. Retrieved October 26, 2022, from https://gitlab.com/afer/phd-thesis

Allard, C. A. H., Decker, F., Weiner, O. D., Toettcher, J. E., & Graziano, B. R. (2018). A size-invariant bud-duration timer enables robustness in yeast cell size control. *Plos One*, *13*(12), e0209301. https://doi.org/10.1371/journal.pone.0209301

Aspert, T., Hentsch, D., & Charvin, G. (2022). DetecDiv, a generalist deep-learning platform for automated cell division tracking and survival analysis. *Elife*, *11*, e79519. https://doi.org/10.7554/eLife.79519

Bakker, E., Swain, P. S., & Crane, M. M. (2018). Morphologically constrained and data informed cell segmentation of budding yeast. *Bioinformatics*, *34*(1), 88–96. https://doi.org/10.1093/bioinformatics/btx550

Barber, F., Amir, A., & Murray, A. W. (2020). Cell-size regulation in budding yeast does not depend on linear accumulation of Whi5. *Proceedings of the National Academy of Sciences*, *117*(25), 14243–14250. https://doi.org/10.1073/pnas.2001255117

Bheda, P., Aguilar-Gómez, D., Kukhtevich, I., Becker, J., Charvin, G., Kirmizis, A., & Schneider, R. (2020). Microfluidics for single-cell lineage tracking over time to characterize transmission of phenotypes in Saccharomyces cerevisiae. *Star Protocols*, *1*(3), 100228. https://doi.org/10.1016/j.xpro.2020.100228

Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A Quick Introduction to Version Control with Git and GitHub. *Plos Computational Biology*, *12*(1), e1004668. https://doi.org/10.1371/journal.pcbi.1004668

Bowen, D., & Ungar, L. (2020). *Generalized SHAP: Generating multiple types of explanations in machine learning* (arXiv:2006.07155). arXiv. https://doi.org/10.48550/arXiv.2006.07155

Bowsher, C. G., & Swain, P. S. (2014). Environmental sensing, information transfer, and cellular decision-making. *Current Opinion in Biotechnology*, *28*, 149–155. https://doi.org/10.1016/j.copbio.2014.04.010

Cai, L., Dalal, C. K., & Elowitz, M. B. (2008). Frequency-modulated nuclear localization bursts coordinate gene regulation. *Nature*, *455*(7212), 485–490. https://doi.org/10/d85x2n

Cellpose: A generalist algorithm for cellular segmentation | Nature Methods. (n.d.). Retrieved August 25, 2022, from https://www.nature.com/articles/s41592-020-01018-x

Chatfield, C. (2003). *The analysis of time series: An introduction*. Chapman and hall/CRC.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Chivers, I., & Sleightholme, J. (2015). An introduction to algorithms and the big O notation. In *Introduction to programming with Fortran* (pp. 359–364). Springer.

Conda/conda. (2022). conda. https://github.com/conda/conda

Crane, M. M., Clark, I. B. N., Bakker, E., Smith, S., & Swain, P. S. (2014). A Microfluidic System for Studying Ageing and Dynamic Single-Cell Responses in Budding Yeast. *Plos One*, *9*(6), e100042. https://doi.org/10.1371/journal.pone.0100042

Dalal, C. K., Cai, L., Lin, Y., Rahbar, K., & Elowitz, M. B. (2014). Pulsatile Dynamics in the Yeast Proteome. *Current Biology*, *24*(18), 2189–2194. https://doi.org/10.1016/j.cub.2014.07.076

Dendorfer, P., Rezatofighi, H., Milan, A., Shi, J., Cremers, D., Reid, I., Roth, S., Schindler, K., & Leal-Taixé, L. (2020). *MOT20: A benchmark for multi object tracking in crowded scenes* (arXiv:2003.09003). arXiv. https://doi.org/10.48550/arXiv.2003.09003

Dietler, N., Minder, M., Gligorovski, V., Economou, A. M., Joly, D. A. H. L., Sadeghi, A., Chan, C. H. M., Kozi
'nski, M., Weigert, M., Bitbol, A.-F., & Rahi, S. J. (2020). A convolutional neural network segments yeast microscopy images with high accuracy. *Nature Communications*, *11*(1), 5723. https://doi.org/10.1038/s41467-020-19557-4

Doom Emacs. (n.d.). In *Github*. Retrieved August 24, 2022, from https://github.com/doomemacs

Edelstein, A. D., Tsuchida, M. A., Amodaj, N., Pinkard, H., Vale, R. D., & Stuurman, N. (2014). Advanced methods of microscope control using μmanager software. *Journal of Biological Methods*, *1*(2), e10. https://doi.org/10.14440/jbm.2014.36

Elowitz, M. B., Levine, A. J., Siggia, E. D., & Swain, P. S. (2002). Stochastic Gene Expression in a Single Cell. *Science*, *297*(5584), 1183–1186. https://doi.org/10.1126/science.1070919

Emami, N., Sedaei, Z., & Ferdousi, R. (2020). Computerized cell tracking: Current methods, tools and challenges. *Visual Informatics*. https://doi.org/10.1016/j.visinf.2020.11.003

Falconnet, D., Niemistö, A., J. Taylor, R., Ricicova, M., Galitski, T., Shmulevich, I., & L. Hansen, C. (2011). High-throughput tracking of single yeast cells in a microfluidic imaging matrix. *Lab on a Chip*, *11*(3), 466–473. https://doi.org/10.1039/C0LC00228C

Fazeli, E., Roy, N. H., Follain, G., Laine, R. F., von Chamier, L., Hänninen, P. E., Eriksson, J. E., Tinevez, J.-Y., & Jacquemet, G. (2020). Automated cell tracking using StarDist and TrackMate. *F1000research*, *9*, 1279. https://doi.org/10.12688/f1000research.27019.1

Fortunato, L., & Galassi, M. (2021). The case for free and open source software in research and scholarship. *Philosophical Transactions of the Royal Society a: Mathematical, Physical and Engineering Sciences*, *379*(2197), 20200079. https://doi.org/10.1098/rsta.2020.0079

Franzmann, T. M., & Alberti, S. (2019). Protein Phase Separation as a Stress Survival Strategy. *Cold Spring Harbor Perspectives in Biology*, *11*(6), a034058. https://doi.org/10.1101/cshperspect.a034058

GitHub - python/mypy: Optional static typing for Python. (n.d.). Retrieved August 24, 2022, from https://github.com/python/mypy

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkačik, G., & Swain, P. S. (2018a). Distributed and dynamic intracellular organization of extracellular information. *Proceedings of the National Academy of Sciences*, *115*(23), 6088–6093. https://doi.org/10/gdprk9

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkačik, G., & Swain, P. S. (2018b). Distributed and dynamic intracellular organization of extracellular information. *Proceedings of the National Academy of Sciences*, *115*(23), 6088–6093. https://doi.org/10.1073/pnas.1716659115

Grove, O., Rajab, K., Piegl, L. A., & Lai-Yuen, S. (2011). From CT to NURBS: Contour Fitting with B-spline Curves. *Computer-Aided Design and Applications*, *8*(1), 3–21. https://doi.org/10.3722/cadaps.2011.3-21

Gundersen, O. E. (2021). The fundamental principles of reproducibility. *Philosophical Transactions of the Royal Society a: Mathematical, Physical and Engineering Sciences*, *379*(2197), 20200210. https://doi.org/10.1098/rsta.2020.0210

Guo, D., Pei, Y., Zheng, K., Yu, H., Lu, Y., & Wang, S. (2020). Degraded Image Semantic Segmentation With Dense-Gram Networks. *Ieee Transactions on Image Processing*, *29*, 782–795. https://doi.org/10.1109/TIP.2019.2936111

Gutierrez, J. I., Brittingham, G. P., Karadeniz, Y., Tran, K. D., Dutta, A., Holehouse, A. S., Peterson, C. L., & Holt, L. J. (2022). SWI/SNF senses carbon starvation with a pH-sensitive low-complexity sequence. *Elife*, *11*, e70344. https://doi.org/10.7554/eLife.70344

Hansen, J. M., Horowitz, A., Lynch, E. M., Farrell, D. P., Quispe, J., DiMaio, F., & Kollman, J. M. (2021). Cryo-EM structures of CTP synthase filaments reveal mechanism of pH-sensitive assembly during budding yeast starvation. *Elife*, *10*, e73368. https://doi.org/10.7554/eLife.73368

Hartwell, L. H., & Unger, M. W. (1977). Unequal division in Saccharomyces cerevisiae and its implications for the control of cell division. *Journal of Cell Biology*, *75*(2), 422–435. https://doi.org/10.1083/jcb.75.2.422

Hutchins, D., Schlag, I., Wu, Y., Dyer, E., & Neyshabur, B. (2022). *Block-Recurrent Transformers* (arXiv:2203.07852). arXiv. https://doi.org/10.48550/arXiv.2203.07852

Ibstedt, S., Sideri, T. C., Grant, C. M., & Tamás, M. J. (2014). Global analysis of protein aggregation in yeast during physiological conditions and arsenite stress. *Biology Open*, *3*(10), 913–923. https://doi.org/10.1242/bio.20148938

Jacquel, B., Aspert, T., Laporte, D., Sagot, I., & Charvin, G. (2020). pH fluctuations drive waves of stereotypical cellular reorganizations during entry into quiescence. *Biorxiv*, 2020.11.25.395608. https://doi.org/10.1101/2020.11.25.395608

Kapoor, V., & Carabaña, C. (2021). Cell Tracking in 3D using deep learning segmentations. *Python in Science Conference*, 154. https://doi.org/10.25080/majora-1b6fd038-02b

Kellogg, D. R. (2003). Wee1-dependent mechanisms required for coordination of cell growth and cell division. *Journal of Cell Science*, *116*(24), 4883–4890. https://doi.org/10.1242/jcs.00908

Kery, M. B., Radensky, M., Arya, M., John, B. E., & Myers, B. A. (2018). The Story in the Notebook: Exploratory Data Science using a Literate Programming

Tool. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–11. https://doi.org/10.1145/3173574.3173748

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., & others. (2016). *Jupyter Notebooks-a publishing format for reproducible computational workflows.* (Vol. 2016).

Knuth, D. E. (1984). Literate Programming. *The Computer Journal*, *27*(2), 97–111. https://doi.org/10.1093/comjnl/27.2.97

Köster, J., & Rahmann, S. (2012). Snakemake a scalable bioinformatics workflow engine. *Bioinformatics*, *28*(19), 2520–2522. https://doi.org/10.1093/bioinformatics/bts480

Lee, S. S., Vizcarra, I. A., Huberts, D. H. E. W., Lee, L. P., & Heinemann, M. (2012). Whole lifespan microscopic observation of budding yeast aging through a microfluidic dissection platform. *Proceedings of the National Academy of Sciences*, *109*(13), 4916–4920. https://doi.org/10/gfwc8g

Li, S., Besson, S., Blackburn, C., Carroll, M., Ferguson, R. K., Flynn, H., Gillen, K., Leigh, R., Lindner, D., Linkert, M., Moore, W. J., Ramalingam, B., Rozbicki, E., Rustici, G., Tarkowska, A., Walczysko, P., Williams, E., Allan, C., Burel, J.-M., … Swedlow, J. R. (2016). Metadata management for high content screening in OMERO. *Methods*, *96*, 27–32. https://doi.org/10.1016/j.ymeth.2015.10.006

Liu, Z., Jin, L., Chen, J., Fang, Q., Ablameyko, S., Yin, Z., & Xu, Y. (2021). A survey on applications of deep learning in microscopy image analysis. *Computers in Biology and Medicine*, *134*, 104523. https://doi.org/10.1016/j.compbiomed.2021.104523

Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., & Jones, N. S. (2019). Catch22: CAnonical Time-series CHaracteristics. *Data Mining and Knowledge Discovery*, *33*(6), 1821–1852. https://doi.org/10.1007/s10618-019-00647-x

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

Lynch, E. M., Kollman, J. M., & Webb, B. A. (2020). Filament formation by metabolic enzymesA new twist on regulation. *Current Opinion in Cell Biology*, *66*, 28–33. https://doi.org/10.1016/j.ceb.2020.04.006

Mahon, M. J. (2011). pHluorin2: An enhanced, ratiometric, pH-sensitive green florescent protein. *Advances in Bioscience and Biotechnology (Print)*, *2*(3), 132–137. https://doi.org/10.4236/abb.2011.23021

Mattiazzi Usaj, M., Styles, E. B., Verster, A. J., Friesen, H., Boone, C., & Andrews, B. J. (2016). High-Content Screening for Quantitative Cell Biology. *Trends in Cell Biology*, *26*(8), 598–611. https://doi.org/10.1016/j.tcb.2016.03.008

Matula, P., Maška, M., Sorokin, D. V., Matula, P., Ortiz-de-Solórzano, C., & Kozubek, M. (2015). Cell Tracking Accuracy Measurement Based on Comparison of Acyclic Oriented Graphs. *Plos One*, *10*(12), e0144959. https://doi.org/10/c6sq

Minas, G., Jenkins, D. J., Rand, D. A., & Finkenstädt, B. (2017). Inferring transcriptional logic from multiple dynamic experiments. *Bioinformatics*, *33*(21), 3437–3444. https://doi.org/10.1093/bioinformatics/btx407

Moen, E., Borba, E., Miller, G., Schwartz, M., Bannon, D., Koe, N., Camplisson, I., Kyme, D., Pavelchek, C., Price, T., Kudo, T., Pao, E., Graf, W., & Van Valen, D. (2019). *Accurate cell tracking and lineage construction in live-cell imaging experiments with deep learning* [Preprint]. Bioinformatics. https://doi.org/10.1101/803205

Mouton, S. N., Thaller, D. J., Crane, M. M., Rempel, I. L., Terpstra, O. T., Steen, A., Kaeberlein, M., Lusk, C. P., Boersma, A. J., & Veenhoff, L. M. (2020). A physicochemical perspective of aging from single-cell analysis of pH,

macromolecular and organellar crowding in yeast. *Elife*, *9*, e54707. https://doi.org/10.7554/eLife.54707

Nadkarni, A. K., McDonough, V. M., Yang, W.-L., Stukey, J. E., Ozier-Kalogeropoulos, O., & Carman, G. M. (1995). Differential Biochemical Regulation of the URA7- and URA8-encoded CTP Synthetases from Saccharomyces cerevisiae. *Journal of Biological Chemistry*, *270*(42), 24982–24988. https://doi.org/10.1074/jbc.270.42.24982

Narayanaswamy, R., Levy, M., Tsechansky, M., Stovall, G. M., O'Connell, J. D., Mirrielees, J., Ellington, A. D., & Marcotte, E. M. (2009). Widespread reorganization of metabolic enzymes into reversible assemblies upon nutrient starvation. *Proceedings of the National Academy of Sciences*, *106*(25), 10147–10152. https://doi.org/10.1073/pnas.0812771106

Noree, C., Monfort, E., Shiau, A. K., & Wilhelm, J. E. (2014). Common regulatory control of CTP synthase enzyme activity and filament formation. *Molecular Biology of the Cell*, *25*(15), 2282–2290. https://doi.org/10/f6bjh9

Noree, C., Sato, B. K., Broyer, R. M., & Wilhelm, J. E. (2010a). Identification of novel filament-forming proteins in Saccharomyces cerevisiae and Drosophila melanogaster. *The Journal of Cell Biology*, *190*(4), 541–551. https://doi.org/10/dnpxv7

Noree, C., Sato, B. K., Broyer, R. M., & Wilhelm, J. E. (2010b). Identification of novel filament-forming proteins in Saccharomyces cerevisiae and Drosophila melanogaster. *Journal of Cell Biology*, *190*(4), 541–551. https://doi.org/10/dnpxv7

Ochkov, V. F., Stevens, A., & Tikhonov, A. I. (2022). Jupyter Notebook, JupyterLab Integrated Environment for STEM Education. *2022 VI International Conference on Information Technologies in Engineering Education (Inforino)*, 1–5. https://doi.org/10.1109/Inforino53888.2022.9782924

Orij, R., Brul, S., & Smits, G. J. (2011). Intracellular pH is a tightly controlled signal in yeast. *Biochimica et Biophysica Acta (Bba) - General Subjects*, *1810*(10), 933–944. https://doi.org/10.1016/j.bbagen.2011.03.011

Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. *Ieee Transactions on Systems, Man, and Cybernetics*, *9*(1), 62–66. https://doi.org/10.1109/TSMC.1979.4310076

Ozier-Kalogeropoulos, O., Adeline, M.-T., Yang, W.-L., Carman, G. M., & Lacroute, F. (1994). Use of synthetic lethal mutants to clone and characterize a novel CTP synthetase gene in Saccharomyces cerevisiae. *Molecular and General Genetics Mgg*, *242*(4), 431–439. https://doi.org/10.1007/BF00281793

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

O' Neill, J. S., Hoyle, N. P., Robertson, J. B., Edgar, R. S., Beale, A. D., Peak-Chew, S. Y., Day, J., Costa, A. S. H., Frezza, C., & Causton, H. C. (2020). Eukaryotic cell biology is temporally coordinated to support the energetic demands of protein homeostasis. *Nature Communications*, *11*(1), 4706. https://doi.org/10.1038/s41467-020-18330-x

Padovani, F., Mairhörmann, B., Falter-Braun, P., Lengefeld, J., & Schmoller, K. M. (2022). Segmentation, tracking and cell cycle analysis of live-cell imaging data with Cell-ACDC. *Bmc Biology*, *20*(1), 174. https://doi.org/10.1186/s12915-022-01372-6

PDM. (2022). PDM. https://github.com/pdm-project/pdm

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

Petrovska, I., Nüske, E., Munder, M. C., Kulasegaran, G., Malinovska, L., Kroschwald, S., Richter, D., Fahmy, K., Gibson, K., Verbavatz, J.-M., & Alberti, S. (2014). Filament formation by metabolic enzymes is a specific adaptation to an advanced state of cellular starvation. *Elife*, *3*, e02409. https://doi.org/10.7554/eLife.02409

Pietsch, J. M. J., Muñoz, A. F., Adjavon, D.-Y. A., Farquhar, I., Clark, I. B. N., & Swain, P. S. (2022). *A label-free method to track individuals and lineages of budding cells* (p. 2022.05.11.491488). bioRxiv. https://doi.org/10.1101/2022.05.11.491488

Pinkard, H., Stuurman, N., Ivanov, I. E., Anthony, N. M., Ouyang, W., Li, B., Yang, B., Tsuchida, M. A., Chhun, B., Zhang, G., Mei, R., Anderson, M., Shepherd, D. P., Hunt-Isaak, I., Dunn, R. L., Jahr, W., Kato, S., Royer, L. A., Thiagarajah, J. R., … Waller, L. (2021). Pycro-Manager: Open-source software for customized and reproducible microscope control. *Nature Methods*, *18*(3), 226–228. https://doi.org/10.1038/s41592-021-01087-6

Poetry: Dependency Management for Python. (2022). Poetry. https://github.com/python-poetry/poetry

Ponsford, A. H., Ryan, T. A., Raimondi, A., Cocucci, E., Wycislo, S. A., Fröhlich, F., Swan, L. E., & Stagi, M. (2020). Live imaging of intra-lysosome pH in cell lines and primary neuronal culture using a novel genetically encoded biosensor. *Autophagy*, *0*(0), 1–19. https://doi.org/10/gg2cnn

Pre-commit/pre-commit. (2022). pre-commit. https://github.com/pre-commit/pre-commit

Psf/black. (2022). Python Software Foundation. https://github.com/psf/black

PyCQA/flake8. (2022). Python Code Quality Authority. https://github.com/PyCQA/flake8

PyCQA/isort. (2022). Python Code Quality Authority. https://github.com/PyCQA/isort

Pyenv for Windows. (2022). Pyenv for Windows. https://github.com/pyenv-win/pyenv-win

Pypa/twine. (2022). Python Packaging Authority. https://github.com/pypa/twine

RACINE, J. S. (2012). RSTUDIO: A PLATFORM-INDEPENDENT IDE FOR R AND SWEAVE. *Journal of Applied Econometrics*, *27*(1), 167–172. https://www.jstor.org/stable/41337225

Reilly, T. (2022). *Darglint*. https://github.com/terrencepreilly/darglint

Reinke, A., Tizabi, M. D., Sudre, C. H., Eisenmann, M., Rädsch, T., Baumgartner, M., Acion, L., Antonelli, M., Arbel, T., Bakas, S., Bankhead, P., Benis, A., Cardoso, M. J., Cheplygina, V., Christodoulou, E., Cimini, B., Collins, G. S., Farahani, K., van Ginneken, B., … Maier-Hein, L. (2022). *Common Limitations of Image Processing Metrics: A Picture Story* (arXiv:2104.05642). arXiv. https://doi.org/10.48550/arXiv.2104.05642

Ruff, K. M., & Pappu, R. V. (2021). Alphafold and implications for intrinsically disordered proteins. *Journal of Molecular Biology*, *433*(20), 167208.

Salem, D., Li, Y., Xi, P., Phenix, H., Cuperlovic-Culf, M., & Kærn, M. (2021). YeastNet: Deep-Learning-Enabled Accurate Segmentation of Budding Yeast Cells in Bright-Field Microscopy. *Applied Sciences*, *11*(6), 2692. https://doi.org/10.3390/app11062692

Schmidt, U., Weigert, M., Broaddus, C., & Myers, G. (2018). Cell Detection with Star-Convex Polygons. In A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, & G. Fichtinger (Eds.), *Medical Image Computing and Computer Assisted Intervention MICCAI 2018* (pp. 265–273). Springer International Publishing. https://doi.org/10.1007/978-3-030-00934-2_30

Schulte, E., Davison, D., Dye, T., & Dominik, C. (2012). A Multi-Language Computing Environment for Literate Programming and Reproducible Research. *Journal of Statistical Software*, *46*, 1–24. https://doi.org/10.18637/jss.v046.i03

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *Ieee Access*, *5*, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Shen, Q.-J., Kassim, H., Huang, Y., Li, H., Zhang, J., Li, G., Wang, P.-Y., Yan, J., Ye, F., & Liu, J.-L. (2016). Filamentation of Metabolic Enzymes in Saccharomyces cerevisiae. *Journal of Genetics and Genomics*, *43*(6), 393–404. https://doi.org/10.1016/j.jgg.2016.03.008

Simple Python Version Management: Pyenv. (2022). pyenv. https://github.com/pyenv/pyenv

Sofroniew, N., Lambert, T., Nunez-Iglesias, J., Evans, K., Bokota, G., Bussonnier, M., Peña-Castellanos, G., Winston, P., Yamauchi, K., Pop, D. D., alisterburt, Pam, Liu, Z., Solak, A. C., Gaifas, L., Buckley, G., Sweet, A., Lee, G., Rodríguez-Guerra, J., … Gohlke, C. (2022). *Napari/napari: 0.4.15*. Zenodo. https://doi.org/10.5281/zenodo.6344271

Soifer, I., & Barkai, N. (2014). Systematic identification of cell size regulators in budding yeast. *Molecular Systems Biology*, *10*(11), 761. https://doi.org/10.15252/msb.20145345

Soifer, I., Robert, L., & Amir, A. (2016). Single-Cell Analysis of Growth in Budding Yeast and Bacteria Reveals a Common Size Regulation Strategy. *Current Biology*, *26*(3), 356–361. https://doi.org/10.1016/j.cub.2015.11.067

Spacemacs: Emacs advanced Kit focused on Evil. (n.d.). Retrieved August 24, 2022, from https://www.spacemacs.org/

Srinath, K. (2017). Python the fastest growing programming language. *International Research Journal of Engineering and Technology*, *4*(12), 354–357.

Stallman, R. (2009). Viewpoint Why" open source" misses the point of free software. *Communications of the Acm*, *52*(6), 31–33.

Stallman, R. M. (1981). EMACS the extensible, customizable self-documenting display editor. *Acm Sigoa Newsletter*, *2*(1-2), 147–156. https://doi.org/10.1145/1159890.806466

Stringer, C., & Pachitariu, M. (2022). *Cellpose 2.0: How to train your own model* (p. 2022.04.01.486764). bioRxiv. https://doi.org/10.1101/2022.04.01.486764

Swain Lab / aliby / aliby. (n.d.). In *Gitlab*. Retrieved October 26, 2022, from https://git.ecdf.ed.ac.uk/swain-lab/aliby/aliby

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

Turner, J. J., Ewald, J. C., & Skotheim, J. M. (2012). Cell Size Control in Yeast. *Current Biology*, *22*(9), R350–R359. https://doi.org/10.1016/j.cub.2012.02.041

Ulicna, K., Vallardi, G., Charras, G., & Lowe, A. R. (2021). Automated Deep Lineage Tree Analysis Using a Bayesian Single Cell Tracking Approach. *Frontiers in Computer Science*, *3*. https://www.frontiersin.org/articles/10.3389/fcomp.2021.734559

Ulman, V., Maška, M., Magnusson, K. E. G., Ronneberger, O., Haubold, C., Harder, N., Matula, P., Matula, P., Svoboda, D., Radojevic, M., Smal, I., Rohr, K., Jaldén, J., Blau, H. M., Dzyubachyk, O., Lelieveldt, B., Xiao, P., Li, Y., Cho, S.-Y., … Ortiz-de-Solorzano, C. (2017). An objective comparison of cell-tracking algorithms. *Nature Methods*, *14*(12), 1141–1152. https://doi.org/10/gcg36d

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Versari, C., Stoma, S., Batmanov, K., Llamosi, A., Mroz, F., Kaczmarek, A., Deyell, M., Lhoussaine, C., Hersen, P., & Batt, G. (2017). Long-term tracking of budding yeast cells in brightfield microscopy: CellStar and the Eval-

uation Platform. *Journal of the Royal Society Interface*, *14*(127), 20160705. https://doi.org/10.1098/rsif.2016.0705

Visani, G., Bagli, E., Chesani, F., Poluzzi, A., & Capuzzo, D. (2022). Statistical stability indices for LIME: Obtaining reliable explanations for machine learning models. *Journal of the Operational Research Society*, *73*(1), 91–101. https://doi.org/10.1080/01605682.2020.1865846

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

Welcome Sphinx documentation. (n.d.). Retrieved August 24, 2022, from https://www.sphinx-doc.org/en/master/

Wood, N. E., & Doncic, A. (2019). A fully-automated, robust, and versatile algorithm for long-term budding yeast segmentation and tracking. *Plos One*, *14*(3), e0206395. https://doi.org/10.1371/journal.pone.0206395

Yang, W.-L., McDonough, V. M., Ozier-Kalogeropoulos, O., Adeline, M.-T., Flocco, M. T., & Carman, G. M. (2002). *Purification and Characterization of CTP Synthetase, the Product of the URA7 Gene in Saccharomyces cerevisiae* [Research-Article]. American Chemical Society. https://doi.org/10.1021/bi00201a028

Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, *31*(7), 1235–1270. https://doi.org/10.1162/neco_a_01199

This ensures that the subsequent sections are being included as root items in the bookmark structure of your PDF reader.

Glossary

Long Short-Term Memory	neural network architecture that accumulates information of recently processed images. 17
adenine diphosphate	high-energy molecule, main energy carrier in cells (in its low energy state). 21
adenine triphosphate	high-energy molecule, main energy carrier in cells. 21
filament assembly	type of protein aggregation in which the resulting polymer expands linearly towards opposite sides. 22
glutamine	amino acid used in protein biosynthesis, key player in multiple biochemical functions. 21
microfluidics	a system to manipulate small amount of fluids to have a precise control of the environment where cells grow. 7
phosphate	inorganic phosphate, component of high-energy molecules. 21
position	Frame of vision captured by a microscopy camera. 11, 12, 72, 108, 112, 122
tile	Sub-region in position containing data of interest. 11, 79, 98
trap	pair of pillars that keep cells in place during a microfluidics experiment. 12
cells	I/O interface to access information on cell identifiers and masks. 117, 120, 123
extractor	ALIBY step that uses masks and images to quantify cell features, producing signals. 71, 83, 86, 105, 107, 120

grouper tool for accumulating multiple from one or multiple experiments. 109

hdf5 Hierarchical Data Format 5; a standard for store and compress large amounts of data in a tree-like structure. 72, 83, 102, 106, 107, 112, 119, 123

max5 by median Protein aggregation metric, highest five pixels inside a cell divided by the value of the median pixel inside the same cell. 112, 132, 150, 152, 153, 177, 178, 180

max5 Protein aggregation metric, highest five pixels inside a cell. 130, 131

merger Postprocess that merges tracklets based on the likelihood of them being the same track. 97

multigrouper Grouper that accumulates data from multiple experiments. 109

parameters set of instructions structured as a dictionary or class and fed to a process for analytical reproducibility. 71, 82, 93, 98, 101–103, 105, 125

picker Postprocess that selects cells for further analysis based on criteria of time-permanence. 97

pipeline main analysis orchestrator, manages all processing steps, I/O and Metadata. 71, 106, 107, 112, 119, 123

pma1 main proton pump of the cytosolic membrane in yeast. 196, 198

postprocessor ALIBY step that modifies signals to further characterise cells. 71, 73, 105, 107, 120

postprocess process used by Postprocessor to modify cell signals (e.g., Merger, Picker, SavGol). 71, 100

process group of functions grouped as instances that operates on input data based on a fixed set of parameters. 71, 82, 93, 98, 99, 101–103, 105, 107, 125

reader family of classes that reads data from long-term storage file for visualisation or processing.. 120

reporter tool to produce a short experimental report. 90, 110

signal low level interface that processes time series data from an hdf5 file, returning a time signal. 109, 120, 121

step set of processes within the ALIBY suite that processed images or image-derived data. 92, 104, 106, 107, 113

tiler ALIBY step that uses bright field images to find places where cells are traped in-place. 71, 74, 81, 82, 117

writer family of classes that store the results data from ALIBY steps into a long-term storage file. 102, 103, 106–108, 119, 120

ura7 Wild-type yeast strain whose *ura7* CTPS protein has been tagged with GFP. 22–24, 130, 137, 139

ura8 Wild-type yeast strain whose *ura8* CTPS protein has been tagged with GFP. 22–24, 130, 137, 139, 143, 160, 170, 187

*ura8*H360A Mutant Yeast strain whose *ura8* gene has been modified to induce resistance to filament formation by replacing the amino acid 360 from Histidine to Alanine; it is also tagged with GFP. 142, 145, 146, 160, 169, 172, 173, 187

*ura8*H360R Mutant Yeast strain whose *ura8* gene has been modified to induce likelihood to filament formation by replacing the amino acid 360 from Histidine to Arginine; it is also tagged with GFP. 145, 160, 172, 187

aggregation profile	Group of cells within a population defined by clustering their aggregation over time using the k-means algorithm. 159, 162, 186, 189, 196
agora	low-level tools shared by ALIBY components. 71–73
ALCATRAS	A Long-term Culturing And TRApping System, microfluidic devices to trap yeast cells for live-cell microscopy. 7, 11, 13, 68, 72, 76, 130, 137, 139
ALIBY	Analyser of Live-cell Imaging for Budding Yeast; a software suite to automate single cell microscopy data analyses. 70, 71, 73, 74, 80, 82, 89, 92, 100, 102, 103, 105, 108, 109, 112–115, 117–120, 122, 123, 125, 139, 201, 203–205
argo	programmatic interface to explore an OMERO database. 117, 123
BABY	Birth Assignment for Budding Yeast; a cell segmentation and tracking software. 16, 17, 71–73, 82, 96, 98, 107, 113, 115, 117, 120, 123
bud	extension of mother yeast that grows in volume to become an independent cell. 16
bud spotting	First time-point at which a new bud was discovered. It is related but not equal to a birth event. 153, 156, 158, 180, 183, 185
CTP synthase	enzyme that catalyses the conversion of UTP to CTP, the rate-limiting step of CTP synthesis. 20–23, 129, 130, 137, 141–143, 151, 168–170, 178, 194
dbudt0_daughter	Average time between the discovery of consecutive buds for a given mother. 153, 180

dvol_daughter	Maximum volume growth between two consecutive time points among all daughters of a given cell across all time points (for a given mother). 153, 180
dvol_mother	Maximum volume growth between two consecutive time points of a mother cell. 153, 180
family traces	Set of traces belonging to a mother and its predicted daughters. 149, 151, 176, 177
famine	Microfluidics stage during which cells were growing in presence of necessary nutrients and glucose. 137, 140, 145, 151, 167, 172, 177, 194, 195
feast	Microfluidics stage during which cells were growing in presence of necessary nutrients and glucose. 137, 140, 144, 145, 151, 167, 171, 172, 177, 194
high aggregator	strain with higher affinity to form CTP synthase and lower affinity to dissolve filaments (also referred to as *ura*8H360R). 143, 144, 146, 170, 171, 173, 195
ImageViewer	programmatic interface to access tiles multidimensional images and their associated masks. 117, 123
kymograph	graphical representation of single-cell measurements where each individual cell is a row, with changing values over time. 136, 137, 139, 156, 183
m5m_max_daughter	Maximum aggregation change between two consecutive time points across all daughters and time points (for a given mother). 153, 180
m5m_max_daughter	Maximum value of metric ”Max 5 by median” metric among all daughter cells and time points (for a given mother). 150, 153, 176, 180
m5m_max_mother	Maximum aggregation change between two consecutive time points of a mother cell. 153, 180

m5m_max_mother	Maximum value of metric "Max 5 by median" metric in the mother cell. 153, 180
m5m_mean_daughter	Average of each daughter's highest "Max 5 by median" metric (for a given mother). 153, 180
m5m_mean_mother	Max 5 by median metric in the mother cell. 153, 180
mask	regions of interest that characterise the bounds of an object within an image. 5, 16
maxvol_daughters	Maximum volume of all daughter cells across all time points (for a given mother). 153, 180
maxvol_mother	Maximum volume of the mother cell and time points. 153, 180
nbuds_daughter	Total number of buds for a given daughter. 153, 180
object detection	task of detecting the location of object within an image. 1
OME-TIFF	microscopy image format that combines TIFF images and XML metadata.. 104
OMERO	Open Microscopy EnviROnment, software to manage, among other data, image microscopy datasets. 10, 16, 68, 113, 121, 122
pillar	structure of microfluidic devices used to trap mother yeast cells in place. 12
proliferative growth	cell growth as measured by number of budding events. 129, 156, 183
regulatory state	collection of intracellular metabolic and signalling conditions at a given moment. 18
Savitsky-Golay filter	digital filter that smooths data to increase accuracy without distorting the tendency. 18
segment	task of detecting the boundaries of an object within an image. 1

stage	time interval in an experiment with changing environments, during which media is constant. 137, 140, 144, 145, 167, 171, 172
startvol_daughter	Average volume of all daughters of a given mother at the first time-point in which they appeared. 153, 180
startvol_mother	Volume of the mother cell at the first time-point in which it appeared. 153, 180
trace	time series values for a specific pair of unique cells and metric (e.g., volume, max5, mean fluorescence). 149, 150, 176, 177
tracking link	Link between cell outlines in multiple time points. 62
tracklet	a list of numbers that represent a unique cell over the course of an experiment. 17
U-Net	semantic segmentation CNN architecture, popular for biological imaging. 5

Acronyms

CTP Cytidine Triphosphate. 21, 130

DIC Differential Interference Contrast. 5, 15

DNA Deoxyribonucleic Acid. 67

GFP Green Fluorescent Protein. 22, 86, 137

GUI Graphical User Interface. 117

I/O Input/Output. 71, 104, 105, 117, 119

LED Light-Emitting Diode. 6

PDMS Polydimethylsiloxane. 75

ROI Region Of Interest. 15, 16

SC Synthetic Complete. 137, 145, 172

TIFF Tag Image File Format. 104

TLJH The Littlest Jupyter Notebook. 34

UTP Uridine Triphosphate. 21, 129, 130, 194

XML Extensible Markup Language. 104