Scalable data harmonization for single-cell image-based profiling with CytoTable

Authors:

Dave Bunten (https://orcid.org/0000-0001-6041-3665)[1],
Jenna Tomkinson (https://orcid.org/0000-0003-2676-5813)[1],
Erik Serrano (https://orcid.org/0000-0003-4188-9080)[1],
Michael J. Lippincott (https://orcid.org/0000-0002-8637-1448)[1],
Kenneth I. Brewer (https://orcid.org/0000-0003-4030-0254)[2],
Vince Rubinetti (https://orcid.org/0000-0002-4655-3773)[1],
Faisal Alquaddoomi (https://orcid.org/0000-0003-4297-8747)[1],
Gregory P. Way (https://orcid.org/0000-0002-0503-9348)[1]

Affiliations:

[1]University of Colorado Anschutz Medical Campus
[2]Seqera Labs S.L.

Abstract

High-content imaging (HCI) involves the automated acquisition and quantitative analysis of cell phenotypes from microscopy images. HCI has become a vital tool in biomedical research, supporting studies to characterize disease mechanisms and to identify new therapeutic agents. These studies often rely on high-throughput screening, which can involve thousands of chemical or genetic perturbations that produce terabytes of microscopy data. To extract meaningful biological insights, this data must be processed into quantitative features through a technique known as image-based profiling. A major analytical bottleneck is curating the high-dimensional, single-cell data derived from image analysis tools such as CellProfiler[1,2], DeepProfiler[3], Revvity Harmony, and Molecular Devices IN Carta.[4] These datasets suffer from inconsistent data schemas, inefficient file formats (e.g., CSV, TSV, SQLite), and undocumented ontological relationships (integrations between data objects), which present significant challenges in data processing and standardization. These challenges reduce reproducibility and slow progress in downstream applications. To solve these issues, we introduce CytoTable[5], a software package for harmonizing single-cell image-based profiling. CytoTable enables modular, portable, and cross-language data integration through a software stack comprising: (a) intermediate data curation harmonizing across multiple image analysis tools using Apache Arrow[6], (b) serialized Arrow-compatible storage using Apache Parquet[7], (c) high-performance SQL-based queries using DuckDB[8], (d) a scalable execution engine using Parsl[9] with Pythonic MapReduce[10] design patterns, and (e) a Python-based execution runtime. By combining these technologies, CytoTable increases efficiency, accessibility, and interoperability of single-cell image-based profiling workflows. In essence, CytoTable is a robust, reproducible, and scalable engine that harmonizes single-cell readouts from multiple image analysis tools, preparing for feature integration with software in the Cytomining ecosystem such as Pycytominer.[11]

Introduction

Image-based profiling is a critical part of bioinformatics processing for high-content imaging (HCI) experiments. HCI uses a pipeline of software tools: each tool reads and processes the output of the one before it, then passes its results on to the next stage (**Fig. 1A**).[12] These data processing pipelines start with wet lab assays and microscopy imaging devices that produce microscopy images, which are then analyzed with image analysis tools to produce structured, high-content morphology feature measurements. Scientists then use the feature data to find patterns and test hypotheses on specific biological processes and the impact of perturbations on cell phenotypes. Recent review articles have covered these applications comprehensively.[13,14] Critical software tools and workflows for executing this complex pipeline are designed and used by bioimage analysts[15], research software engineers[16], and others with similar roles.

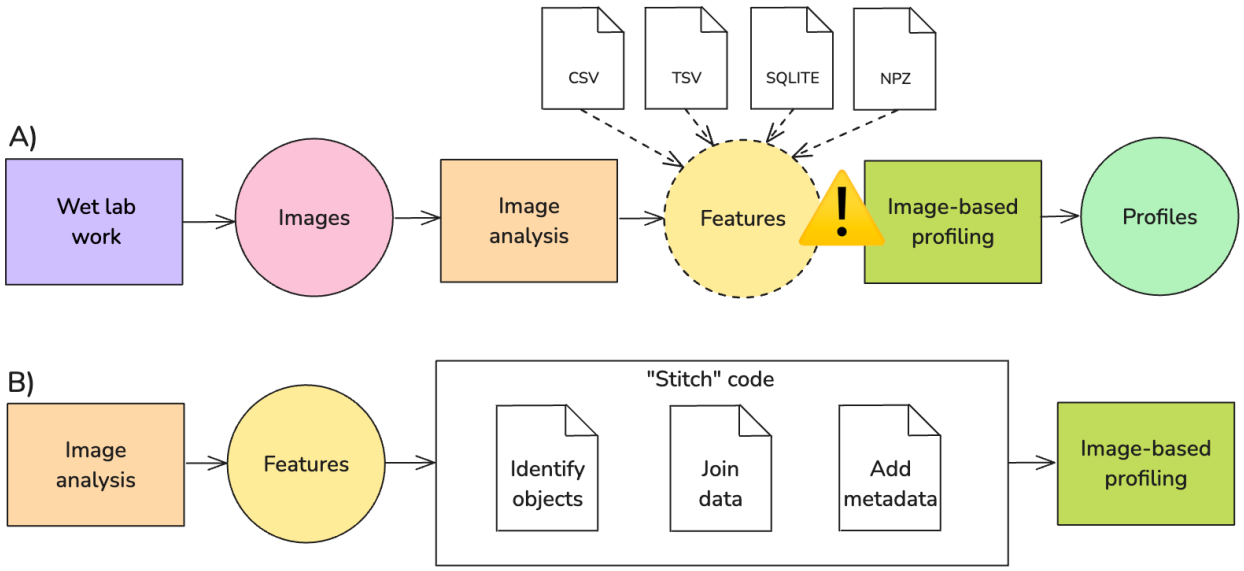

***Figure 1**. The standard data pipeline or bioinformatics workflow(s) to analyze high-content imaging (HCI) data.* **(A)** The HCI pipeline starts with a hypothesis and wet lab work that results in the collection of microscopy images. Next, image analysis converts the images into morphology features, which then progress through an image-based profiling pipeline to generate processed profiles for downstream hypothesis testing and biological discovery. Image analysis software produces features that are stored in myriad file formats from several different image analysis tools. The output features from different image analysis tools are inconsistent, posing several challenges. For example, challenges include different naming conventions, numbers of files, and different data values. **(B)** To solve these challenges, bioimage analysts "stitch" code (rewrite existing and bring together previous code iterations) to integrate data prior to image-based profiling steps, which slows progress and may lead to reproducibility concerns.

Image-based profiling data formats and structures are myriad; there is little cohesion on file type, column names, or structure. These data also tend to be "wide", entailing many thousands of columns, which makes for the high-content nature of the approach. Comma-delimited spreadsheets (e.g., CSV, TSV, etc.) or other text-based data are commonplace and ill-suited to handle large data needs. For example, these formats do not, by default, define explicit data

types (e.g., string, float, etc.), which can lead to discrepancies during processing and larger resource requirements to infer data types on-the-fly. Comma-delimited files are also difficult to compress in comparison to database formats (which are optimized to store data), and they are prone to errors such as misquoting or manually-typed errors, which can be nearly impossible to detect. Other formats, such as SQLite, provide explicit relationships between tables as an advantage, but persist an ambiguous typing system that can result in, for example, strings in floating point columns (leading to similar data inference challenges).

A persistent and under-addressed challenge in image-based profiling is the lack of robust data integration pathways between different image analysis feature extractors (e.g., CellProfiler, DeepProfiler) and data processing engines (e.g., Pycytominer) (**Fig. 1B**). While feature extractor software extracts thousands of features from images, they require additional processing to become interoperable with downstream profiling. Specifically, these extractors do not provide standardized data or interfaces to support the transformation of features into structures compatible with image-based profiling workflows. As a result, researchers are frequently left to implement custom, *ad hoc* “stitch” code which, for example, borrows and rewrites existing code to join multiple CSVs, harmonize metadata, or reconcile segmentation identifiers in order to take full advantage of image-based profiling software. This process is not only time-consuming and error-prone, but also inhibits reproducibility and scalability across experiments and laboratories by consuming valuable research time. Addressing this integration bottleneck is critical for enabling more automated, scalable, and reproducible HCI pipelines.

Methods

CytoTable for feature integration

The increasing complexity and scale of HCI datasets demand computational tools that support scalable, interoperable, and reproducible analyses. Effective data integration is essential for ensuring biological insight and reproducibility across studies.[17] We developed CytoTable to bridge this gap as a scalable and reproducible “feature integrator” across many different image-based feature extractors and data formats (**Fig. 2A**). CytoTable facilitates the transformation of raw image-based feature outputs into harmonized, bioinformatics-ready data by resolving complex multi-table relationships and enabling high-performance, single-cell data integration for downstream profiling (**Fig. 2B**).

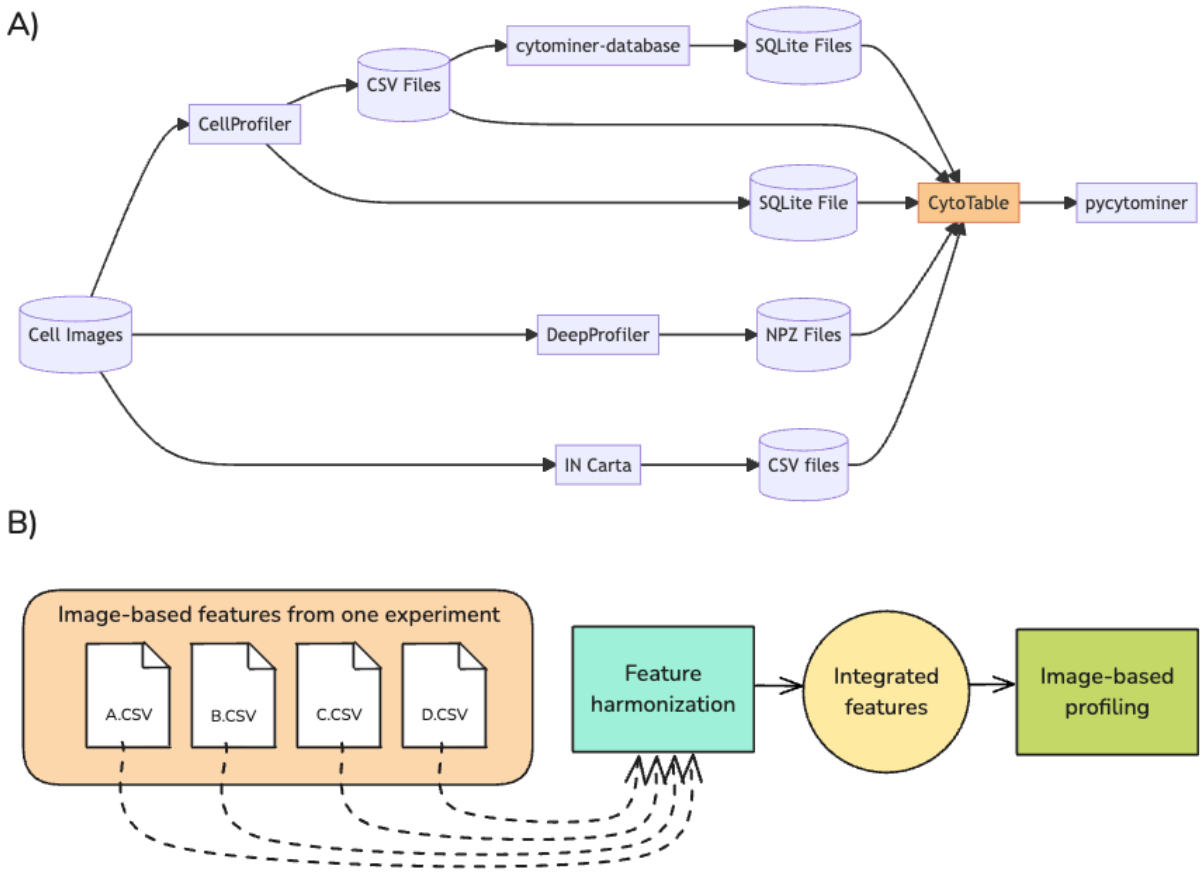

Figure 2. *CytoTable solves the important problems of feature harmonization for image-based profiling.* **(A)** Cytotable harmonizes a range of data formats commonly used in high-content imaging (HCI) workflows. These data sources originate from different image analysis software tools widely adopted in the HCI community. **(B)** Through harmonization, CytoTable outputs file formats that integrate into downstream image-based profiling bioinformatics software.

CytoTable embeds its feature-integration step inside well-established data-processing and data-mining frameworks such as CRISP-DM.[18] By translating the heterogeneous outputs of individual feature extractors into a single, schema-driven table before any downstream analysis, it removes the tool-specific quirks that usually splinter HCI workflows and limit cross-study reuse. This standardized layer turns CytoTable from a mere Extract, Transform, and Load (ETL) utility into a reproducible, modular backbone on which image-based profiling pipelines can scale confidently and interoperably (**Extended Note 1**).

CytoTable components

Inspired by modular data engineering principles and the Composable Data Management System Manifesto[19], we designed CytoTable with a robust framework for managing varied image analysis output files from HCI experiments. By leveraging a unified data representation, optimized query execution, and seamless cross-language interoperability, CytoTable enables researchers to efficiently process and analyze large-scale HCI data.

This conceptual model is grounded in long-term, sustainable software practices[20,21] designed to promote maintainability alongside adaptability to avoid software collapse[22]. To meet this evolving landscape, we developed using modular, interoperable components that reflect a commitment to modular and emergent design principles, inspired by the work of the Software Gardening Almanack.[23,24] The approach not only increases the longevity of CytoTable but also leaves a durable foundation for future software tools and evolving standards in the image-based profiling community.

Cross-platform in-memory data types with Apache Arrow

At its core, CytoTable employs Apache Arrow[6] tables through PyArrow (the Pythonic API for Apache Arrow) for an intermediate in-memory representation, ensuring efficient columnar data access and minimizing the time it takes to prepare data within software procedures (**Fig. 3A**). PyArrow tables within CytoTable handle all single-cell features to optimize processing performance and enforce strict typing expectations. Apache Arrow is an open-source, columnar in-memory data format designed to optimize data processing and interoperability across multiple programming languages. This structured approach allows users and maintainers to seamlessly transition between in-memory operations and long-term storage, facilitating reproducible workflows in image-based profiling.

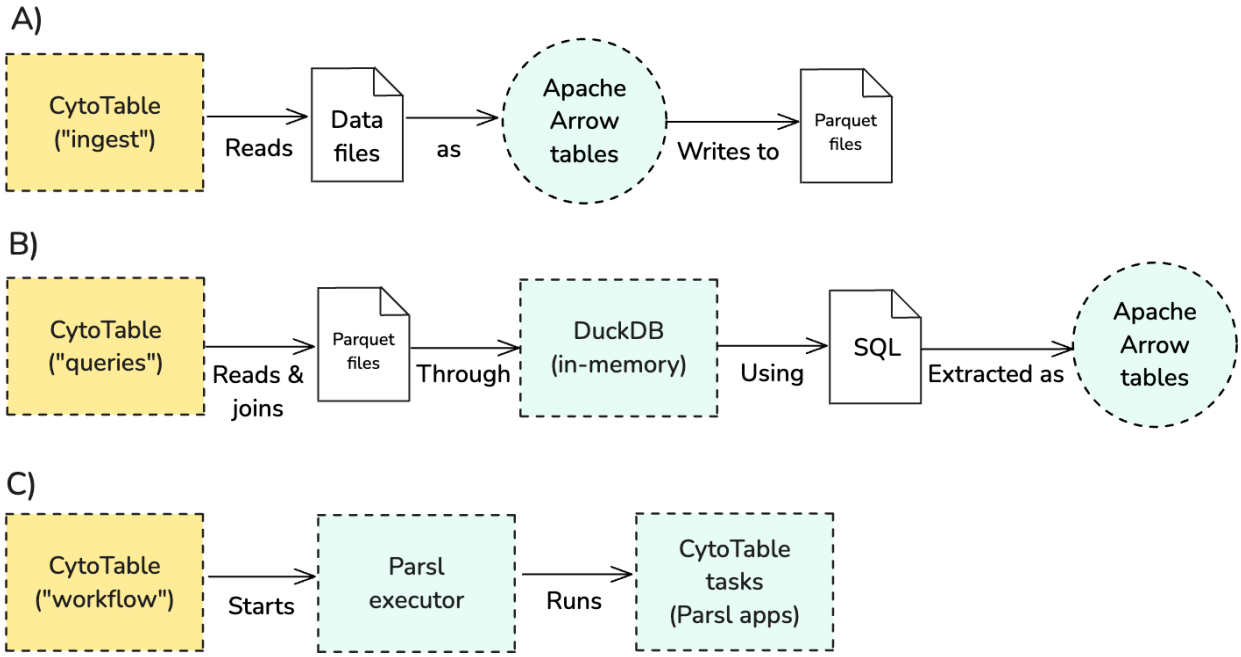

***Figure 3**. Core CytoTable processes*. (**A**) CytoTable ingests serialized data from files as Apache Arrow tables using PyArrow. CytoTable then serializes the data files into Parquet files, retaining strong data type relationships for Arrow integration. (**B**) CytoTable reads and performs data joins through an in-memory DuckDB database as a query engine using SQL to extract Apache Arrow tables. **(C)** CytoTable uses Parsl to orchestrate parallelizable tasks as a data workflow.

Consistent and performant serialization through Parquet data storage

CytoTable leverages Apache Parquet[25] as its primary format for persistent data storage, ensuring efficient handling of large-scale single-cell HCI datasets. All intermediate and finalized feature data integrated by CytoTable are exported to Parquet. By adopting Parquet's columnar storage model, CytoTable significantly reduces disk I/O and improves query performance, particularly for analytical workloads that require scanning and aggregating large amounts of structured data. Its columnar layout is particularly advantageous for analytical queries, as it allows scanning only the necessary columns instead of entire rows, significantly improving query performance. Due to these advantages, Parquet is the foundation of some of the largest data infrastructures in the world, including those used by Google BigQuery[26], Amazon Athena[27], Snowflake[28], and Apache Spark[20]. Parquet's compression and encoding capabilities further enhance storage efficiency, making CytoTable well-suited for workflows that involve iterative analysis and large-scale data sharing across computational environments.

Portable high-performance OLAP with DuckDB and SQL

To enhance cross-platform analytical performance, CytoTable integrates DuckDB[8], an embedded analytical database designed for high-speed, Structured Query Language (SQL) based queries on datasets (**Fig. 3B**). SQL, a domain-specific language introduced in 1974[30] for managing data in relational database systems, remains one of the most widely adopted standards for querying and analyzing structured data. Its enduring relevance, portability, and declarative nature made it a natural choice for integration within CytoTable. DuckDB is an embedded analytical database designed for online analytical processing (OLAP), offering high-speed execution of complex queries on structured data. Unlike traditional transactional databases, DuckDB is optimized for read-heavy workloads, leveraging vectorized query execution and automatic optimizations to efficiently handle large datasets. Its ability to execute SQL queries directly within local environments, without requiring a separate database server, makes it an ideal tool for scalable data analysis workflows. CytoTable performs all extraction and join work on single-cell feature data through DuckDB SQL, enabling users to perform complex data transformations and aggregations with minimal computational overhead, scaling even to large-scale datasets (**Extended Note 2**).

Scalable task-based parallel workflow execution with Parsl

CytoTable leverages Parsl[9], a parallel programming library for Python, to implement MapReduce-style workflows and distributed data processing. Parsl enables researchers to define modular, task-based execution pipelines that can scale seamlessly from local machines to high-performance computing (HPC) clusters (**Fig. 3C**). By abstracting task execution into a flexible dependency graph, Parsl facilitates efficient parallel execution of data transformations, distributing SQL queries, filtering operations, and other computational tasks across multiple processing units. This data-driven execution model is rooted in early dataflow semantics introduced by Dennis and Van Horn (1966), where tasks are triggered by the availability of their inputs rather than by a centralized control.[31] This structure supports both multiprocessing and multithreading by decoupling task scheduling from the limitations of sequential programming.

It is important to understand how Python interacts with CPU cores and threads to realize these dataflow benefits on today's hardware. Modern computation relies on CPU cores, which are independent compute engines housed within each CPU package. Threads are an operating-system abstraction that the scheduler maps onto those cores. In CPython versions below 3.13, every process contains a Global Interpreter Lock (GIL), a mutex that permits only one thread at a time to execute Python byte-code within that process. CPython versions 3.13 and above enable optional and eventual non-default GIL.[32] As a result, common and current CPython CPU-bound workloads may achieve true parallelism only with multiprocessing, which launches multiple processes (each with its own GIL) that the operating system (OS) can run concurrently on different cores. Multithreading creates several threads inside a single process; although the OS may place them on separate cores, they still contend for the same GIL, so the approach mainly benefits I/O-bound tasks or native libraries that release the lock. CytoTable supports both multiprocessing and multithreading, orchestrating single-cell data integration through Parsl's execution engines.

Accessible, extensible, and interoperable Python interface

CytoTable provides feature integration capabilities to image-based profiling researchers by providing well-documented application programming interfaces (APIs) which can be readily installed as a Python package (e.g., "pip install cytotable"). Once installed, the Python package is importable into Python modules or scripts to flexibly implement feature integration into image-based profiling pipelines where it makes sense. CyoTable takes advantage of features afforded by Python packaging frameworks for automated testing, quality assurance, deployment, document rendering, and other tasks to promote long-term project sustainability in one of the most used programming languages.

CytoTable is highly customizable to accommodate and harmonize multiple input sources

To accommodate different data schemas and conventions, CytoTable supports both manually specified configuration arguments and readymade presets, which are included at installation. The configuration arguments enable flexibility when it comes to data ingestion, processing, and exports. Configuration presets (e.g., "cellprofiler_csv" or "in-carta") allow users to apply prepared configuration arguments for CytoTable in common data format expectations without handcrafting specific SQL queries or other customizations (**Table 1**). When presets do not suffice, users can override specific options directly via arguments (allowing for partial use of presets). Each preset is versioned with the image analysis software, and they are included in software tests to ensure deterministic and accurate results. A special parameter, "joins", within these presets helps document the SQL-based relationship between tables, which are extracted from image analysis software. These presets are helpful in using CytoTable for known data sources or adjusting to meet new configurations.

Software	Software output format	CytoTable preset
CellProfiler	CSV	cellprofiler_csv
CellProfiler	SQLite	cellprofiler_sqlite
DeepProfiler	NPZ	deepprofiler
IN Carta	CSV	in-carta
cytominer-database	SQLite	cell-health-cellprofiler-to-cytominer-database

Table 1. *CytoTable provides quick configuration options through the use of "presets" which are supplied at runtime.* Presets allow users to take advantage of common data formats output by image analysis software to save time and reach data output sooner. Presets may be completely or partially overridden with individual CytoTable argument values for customized implementations as needed.

CytoTable supports image-based profiling outputs from CellProfiler, which is an open-source image analysis software.[1,2] It is widely used for cell segmentation, image quality control, and morphology feature extraction. It is often used in high-throughput screening, which generates many terabytes of data and demands high compute costs.[33,34] CellProfiler produces image analysis outputs in many different forms, including CSV and SQLite data (**Table 1**). CellProfiler exports in CSV or SQLite formats, based respectively on the *ExportToSpreadsheet* or *ExportToDatabase* modules. CytoTable accommodates either output format with the existing presets: "cellprofiler_csv" and "cellprofiler_sqlite". Additionally, cytominer-database is a deprecated tool that cleans and processes CellProfiler output CSV files. We created a preset specially designed for this data called: "cell-health-cellprofiler-to-cytominer-database". Because of its deprecation, we expect that this preset will only support legacy CellProfiler output data which scientists may want to reanalyze.

CytoTable also supports image-based profiling outputs from DeepProfiler, which is an open-source deep learning pipeline developed by the Cytomining community to generate morphological profiles directly from images using pre-trained deep learning models.[3] DeepProfiler outputs single-cell feature matrices in .npz format, which is a compressed NumPy archive suitable for efficient handling of high-dimensional data arrays. CytoTable provides the "deepprofiler" preset to harmonize these data.

Lastly, CytoTable supports image-based profiling outputs from IN Carta, which is a commercial image analysis software platform developed by Molecular Devices[4] to provide tools for cell segmentation, tracking, and feature extraction. IN Carta exports processed image-based features as CSV files. These files contain per-object or per-image measurements suitable for morphological profiling and downstream analytics. CytoTable provides the "in-carta" preset for use with this format. Importantly, customizing presets is possible through overrides and

customization. We expect that as other image analysis feature data formats are developed that we (or other community members) can quickly develop new customized presets.

CytoTable benchmarking with other platforms and tools

We benchmarked join operations and memory usage between CytoTable and Pycytominer on realistic single-cell profiling tasks. Pycytominer is a bioinformatics software for processing image-based profiling data, and it contains a utility class called SingleCells. This class, which is much more limited in scope than CytoTable, harmonizes CellProfiler output data prior to bioinformatics processing. We ran each benchmark six times per file type to quantify performance fluctuations.[35] We benchmarked using a Linux desktop computer with 16 CPU cores and 64 GB of RAM. Because of limited functionality in Pycytominer, benchmarking comparisons between Pycytominer and CytoTable could only be directly performed using SQLite datasets. Therefore, our CSV dataset benchmarking used the Pandas[36] Python library rather than Pycytominer directly. All CytoTable comparisons involved two different Parsl execution engines: HighThroughputExecutor and ThreadPoolExecutor, labeled “multiprocess” and “multithread” respectively. We used Parsl’s MonitoringHub[37] to profile memory for the HighThroughputExecutor because it involves specialized process spawning, which otherwise may result in inaccurate memory profiles. We performed all other memory profiling using psutil[38], a commonly used and widely accepted tool within the Python community. We strove to provide an honest, best effort benchmarking of these technologies. [39]

Results

CytoTable comparison benchmarks

As image-based profiling workflows grow in scale and complexity, efficient data handling becomes increasingly important. We developed CytoTable in response to practical limitations encountered in earlier workflows built with Pycytominer’s[11] SingleCells.merge_single_cells method, which relies heavily on in-memory operations using Pandas[36] DataFrames. While Pandas provides a flexible interface for data manipulation, its performance can degrade significantly when handling large-scale datasets, particularly during join operations, where multiple tables must be merged based on shared keys. These joins often require entire tables to be loaded and sometimes duplicated into memory, resulting in substantial resource consumption and potential scalability bottlenecks. A primary motivation for CytoTable was to reduce these memory requirements by leveraging more efficient data representations and execution strategies, enabling scalable data processing without compromising performance. We also developed CytoTable as a harmonization tool, which could accept multiple different image-analysis outputs. Pycytominer does not have this functionality, and we therefore wanted CytoTable to support newcomers (beyond CellProfiler users) into the Cytomining image-based profiling community.

CytoTable consistently minimizes memory consumption while maintaining competitive speed. When processing large CSVs, it uses several gigabytes less peak RAM than Pandas (**Fig. 5A**), though Pandas still completes a few seconds faster (**Fig. 5B**). For SQLite inputs, CytoTable again requires far less memory than Pycytominer (**Fig. 5C**) and scales linearly, whereas

Pycytominer's run time rises exponentially once file size exceeds 230 MB (**Fig. 5D**). Within CytoTable itself, the default multiprocessing mode delivers the smallest memory footprint, while multithreading offers the shortest wall-clock time. Overall, Pandas is preferable for small CSV files, and Pycytominer is preferable for small SQLite files, but CytoTable scales more gracefully as datasets grow, especially for SQLite archives, providing a better speed-versus-memory trade-off. Ongoing work aims to further accelerate CSV parsing without sacrificing CytoTable's memory efficiency.

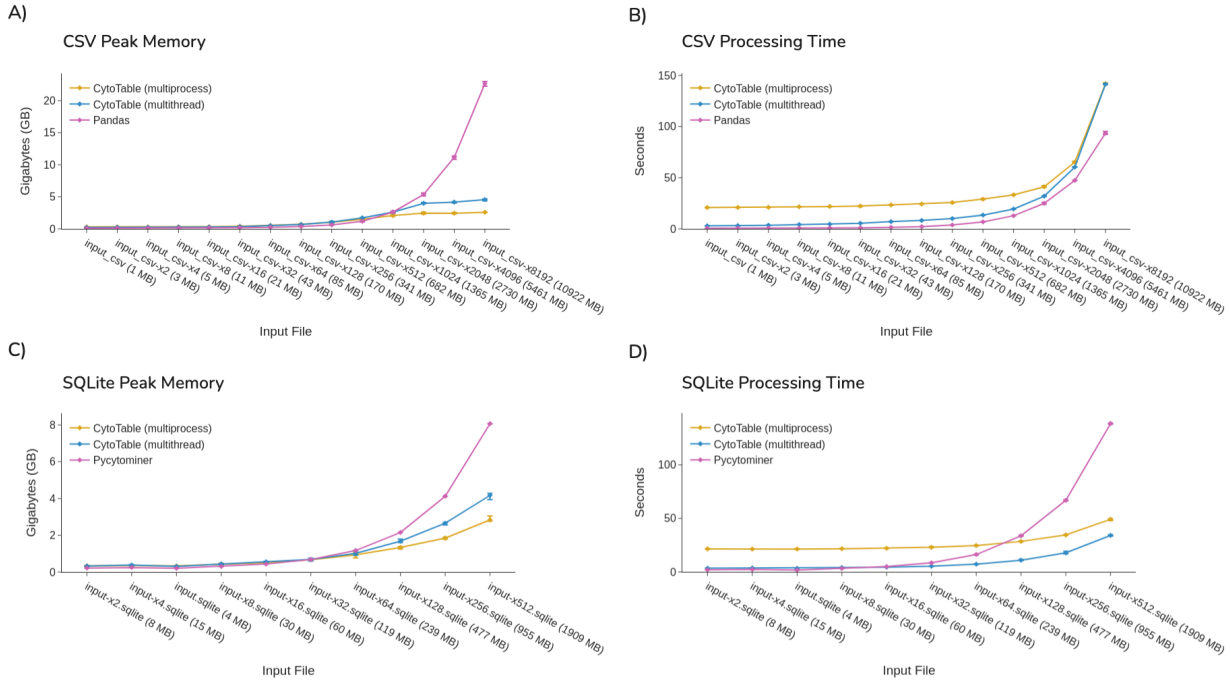

***Figure 5**. Benchmarking CytoTable and Pycytominer processing with increasingly large input files*. **(A)** CytoTable outperforms Pandas for CSV-based memory performance. **(B)** CytoTable outperforms Pandas in processing time for CSVs. **(C)** CytoTable outperforms Pycytominer for SQLite memory performance. **(D)**. CytoTable outperforms Pycytominer in processing time for SQLite files.

We chose to develop CytoTable using Apache Arrow, striking a balance between performance and flexibility. While tools like Pandas offer a user-friendly API and are widely adopted in data science, they can be inefficient when scaling to large datasets due to their reliance on dense in-memory structures. PyArrow serves as a common foundation across these ecosystems, offering a language-agnostic, columnar memory format that minimizes serialization costs and supports zero-copy data exchange. CytoTable adopts PyArrow Tables as its internal data representation to enable interoperability and standardize data types between file format and in-memory use. We observed that Apache Arrow tables through PyArrow or Polars[40] (another Arrow-compatible Python package) use roughly half the memory of Pandas DataFrames (**Fig. 6A**).

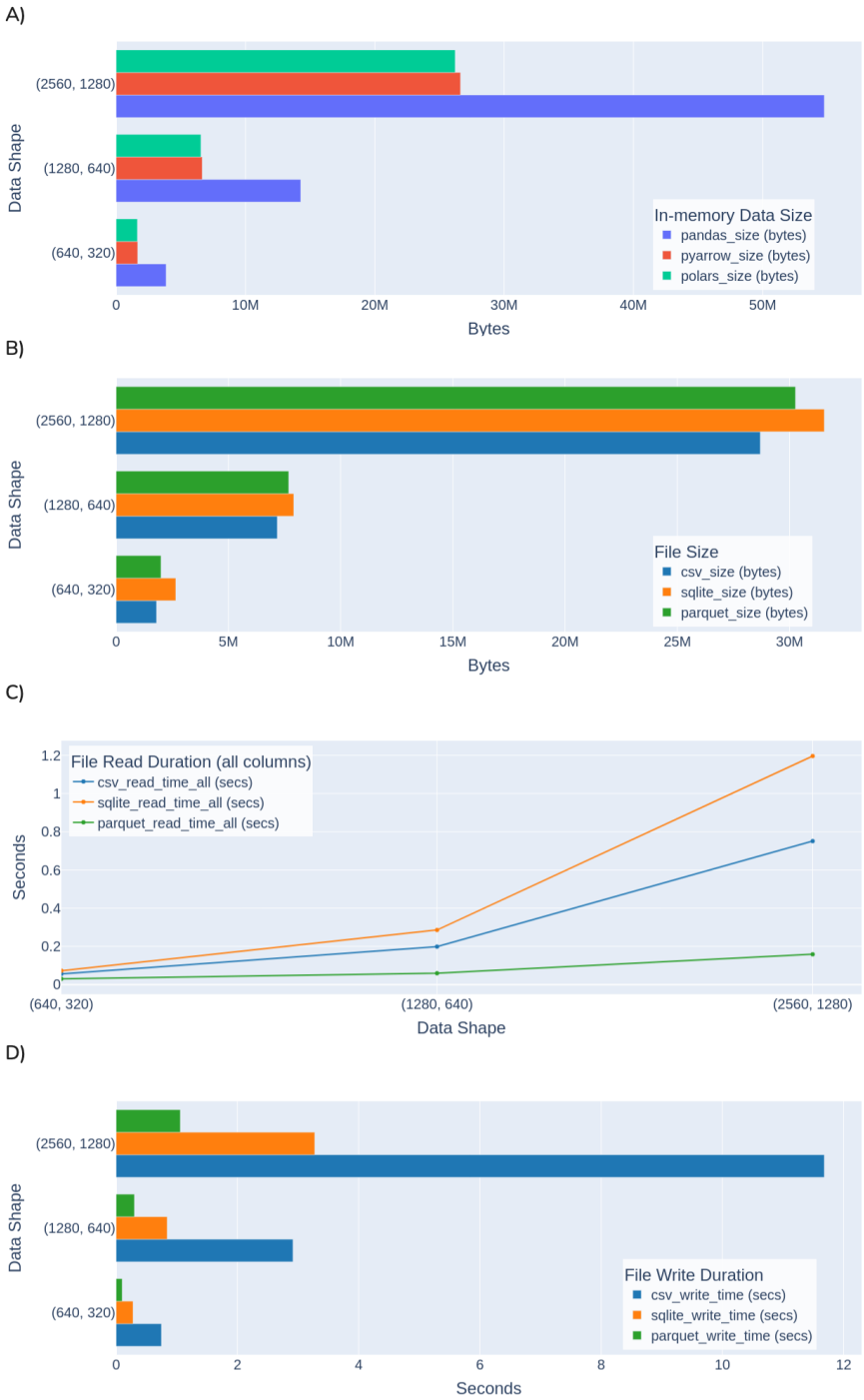

Figure 6. *Benchmarking storage and file format performance.* (**A**) Loading data through PyArrow tables uses almost half as much memory as Pandas DataFrames. We also compare Polars DataFrames which leverage Arrow constructs to help contextualize how Arrow uses memory in different packages. (**B**) Parquet is sometimes as large or larger than CSV or SQLite files for data storage. (**C**) Parquet requires lower read times as data scales when compared to CSV and SQLite file format types. (**D**) Parquet takes substantially less time to write Parquet files when compared to SQLite or CSV files.

Choosing the appropriate file format is critical for working with high-throughput image-based profiling data. CSV, though commonly used, lacks built-in support for types, compression, or metadata, which limits its suitability for large-scale, structured data. SQLite improves on this by enabling queryable, relational storage in a single file, but it is not optimized for analytical workloads and has limited parallel I/O capabilities. In contrast, CytoTable uses Apache Parquet for its persistent storage format. Parquet's columnar layout and native compression make it highly efficient for storing and retrieving subsets of large datasets, especially in scenarios involving filtering or aggregation. Its compatibility with modern data systems and ability to handle high-throughput data access patterns make it a natural choice for image-based profiling pipelines. We observed that Parquet may have roughly the same data storage size properties as SQLite or CSV files (**Fig. 6B**) but outperforms both formats when it comes to read (**Fig. 6C**) and write time durations (**Fig. 6D**).

Applied use cases of CytoTable

To demonstrate CytoTable's practical utility across diverse experimental contexts, we highlight several applied use cases where it has already been integrated into real-world HCI workflows. These examples span both small- and large-scale datasets, showcasing CytoTable's ability to standardize image-analysis outputs, support quality control, and streamline downstream analyses to enable reproducible, scalable data integration across a range of biological questions and assay designs.

Pyroptosis morphology map

Building on similar single-cell profiling workflows, "*A morphology and secretome map of pyroptosis*"[41] employed CytoTable to process data from over eight million peripheral blood mononuclear cells (PBMCs) subjected to various treatments to induce specific forms of cell death. Here, CytoTable was integrated into an image-based profiling pipeline alongside CellProfiler, facilitating the extraction and harmonization of nearly 3,000 features per cell. The formatted data enabled multimodal analyses linking morphology to inflammatory secretome responses, providing insights into mechanisms of programmed cell death.

Pediatric cancer assay optimization

In a related effort focused on optimizing experimental conditions for the Cell Painting assay applied across pediatric cancer cell lines, CytoTable was used to process morphological data.[42] Following segmentation and feature extraction by CellProfiler, CytoTable structured the single-cell profiles to support downstream processing with coSMicQC[43] and Pycytominer[11]. This formatting step helped evaluate how factors like seeding density and assay timing influenced segmentation quality and morphological consistency, contributing to improved reproducibility in large-scale profiling workflows.

NF1 Schwann Cell Project

In the study "*High-content microscopy and machine learning characterize a cell morphology signature of NF1 genotype in Schwann cells*,"[44] researchers applied a modified Cell Painting

assay to profile single-cell morphology across different *NF1* genotypes. CytoTable was used to curate and format the high-dimensional segmentation outputs, enabling downstream integration with tools such as Pycytominer and coSMicQC. The resulting dataset of over 20,000 single-cell profiles supported quality control and statistical analyses, ultimately revealing subtle morphological signatures associated with *NF1* genotype.

Discussion

CytoTable is a core component of the Cytomining ecosystem, which is a vibrant open-source community focused on developing interoperable tools for HCI and high-throughput image-based profiling. The data outputs from CytoTable support streamlined input into downstream software such as coSMicQC[43], which performs rigorous quality control on single-cell segmentation results, and Pycytominer[11], a flexible framework for processing image-based features into formats compatible with statistical and machine learning workflows. While each of these tools offers powerful capabilities on their own, they depend on a consistent, scalable data infrastructure to effectively bridge the gap between raw image-derived measurements from image analysis tools and actionable biological insights. By delivering a standardized, memory-efficient, and SQL-compatible format for single-cell image-based profiling, CytoTable serves as a key integrative layer in this ecosystem.

As the image-based profiling field has matured through practical use and iterative development, several technical challenges and design considerations have emerged.[12] These considerations include in-memory and serialized data types, consistent floating point numeric precision capture within in-memory data types, and memory allocator effectiveness combined with data types such as Apache Arrow (**Extended Note 4**). These experiences have informed how CytoTable handles data format variability, numerical precision, and memory management across different execution environments. CytoTable solves these challenges for the image-based profiling field.

Limitations and future directions

In the spirit of *panta rhei*—that everything flows and nothing stays the same—we acknowledge that the HCI and image-based profiling landscapes are constantly shifting. New tools, formats, and workflows continue to emerge, and any software aiming to support this community must evolve alongside it. CytoTable is built with this adaptability in mind: not as a rigid framework, but as a modular system designed to meet the changing needs of a diversity of bioimage informatics tasks. Looking ahead, we outline several key areas where CytoTable will continue to grow in response to this evolving ecosystem.

Expanding input sources

While CytoTable currently provides robust support for standard image-based profiling outputs, future development will expand its compatibility with a broader range of image analysis input types through additional presets and custom configurations. Currently tailored to formats such as CellProfiler's CSV and SQLite exports, upcoming versions aim to natively support additional tools and pipelines in the image-based profiling ecosystem. This expansion will improve interoperability across profiling tools and reduce the need for custom preprocessing scripts,

making CytoTable a more flexible entry point into standardized single-cell morphology analysis workflows.

Enhancing output format options

Emerging formats such as Lance[45] and Nimble[46] (initially called “Alpha”) offer new opportunities for scalable data storage, streaming access, and versioned datasets optimized for analytical performance and cloud-native environments. Integrating these formats will provide users with greater flexibility in managing large-scale single-cell datasets, particularly in settings that demand fast I/O, partial reads, or remote data access. Lance in particular includes array-value capabilities which are a gap of Parquet and could be leveraged to great effect when it comes to features, image data (as arrays), or shape vectors (of image objects). It also is optimized for vector search which could offer benefits to profiling efforts. AnnData[47] is another potential output format option to integrate with other tooling in the scverse[48].

Integrating with CytoDataFrame

Finally, we plan a deeper integration with CytoDataFrame[49], an in-memory data abstraction for single-cell image-based profiling. We designed CytoDataFrame to couple cell measurements with associated metadata, segmentation masks, and raw images in memory, supporting interactive analysis and visualization in environments like Jupyter. Future work will explore bi-directional compatibility between CytoTable and CytoDataFrame (e.g., enabling .to_cytotable() and .from_cytotable() methods) to bridge persistent storage and live, human-in-the-loop exploration that is required for iterative development and hypothesis generation.

Resource Availability

- CytoTable is an open-source project and its source code can be viewed and downloaded from https://github.com/cytomining/cytotable .
- CytoTable’s installation and usage documentation is available at https://cytomining.github.io/CytoTable/ .
- A brief tutorial on how to use CytoTable is available at https://cytomining.github.io/CytoTable/tutorial.html .
- A notebook based example notebook showing CytoTable in use is available at: https://cytomining.github.io/CytoTable/examples/cytotable_mise_en_place.html
- The repository containing the code used to conduct benchmarking and generate results is available at https://github.com/cytomining/CytoTable-benchmarks .

Declaration of interests

KIB is an employee of Seqera Labs S.L. The authors declare no other competing interests.

Acknowledgments

JT, DB, and GPW were supported in part by Alex’s Lemonade Stand Foundation ‘A’ Award and Tap Cancer Out to GPW (Grant # 23–28306). GPW and ES were supported in part by an American Heart Association Collaborative Sciences Award (24CSA1255857) to GPW. Special

thanks goes to the following for their help in contributing to CytoTable design and development or related work.

- Way Lab (https://www.waysciencelab.com/): Cameron Mattson
- Broad Institute (https://www.broadinstitute.org/): Shantanu Singh, Beth Cimini, Sam Chen
- Yale School of Medicine (https://medicine.yale.edu/): Samir Amin
- University of Wisconsin-Madison Morgridge Institute for Research (https://morgridge.org/): Juan C. Caicedo, Nikita Moshkov
- Sanford Burnham Prebys (https://sbpdiscovery.org/): Alexandre Colas, Aashna Lamba

References

1. Carpenter, A. E. *et al.* CellProfiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biol.* **7**, R100 (2006).
2. Stirling, D. R. *et al.* CellProfiler 4: improvements in speed, utility and usability. *BMC Bioinformatics* **22**, 433 (2021).
3. Moshkov, N. *et al.* Learning representations for image-based profiling of perturbations. *Nat. Commun.* **15**, 1594 (2024).
4. In Carta. Molecular Devices.
5. Bunten, D., Alquaddoomi, F., Serrano, E. & Way, G. CytoTable. Zenodo https://doi.org/10.5281/ZENODO.14888111 (2025).
6. Apache Arrow. Apache Software Foundation.
7. Apache Parquet. Apache Software Foundation.
8. Raasveldt, M. & Mühleisen, H. DuckDB: an Embeddable Analytical Database. in *Proceedings of the 2019 International Conference on Management of Data* 1981–1984 (ACM, Amsterdam Netherlands, 2019). doi:10.1145/3299869.3320212.
9. Babuji, Y. *et al.* Parsl: Pervasive Parallel Programming in Python. in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* 25–36 (ACM, Phoenix AZ USA, 2019). doi:10.1145/3307681.3325400.
10. Dean, J. & Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM* **51**, 107–113 (2008).
11. Serrano, E. *et al.* Reproducible image-based profiling with Pycytominer. *Nat. Methods* **22**, 677–680 (2025).
12. Caicedo, J. C. *et al.* Data-analysis strategies for image-based cell profiling. *Nat. Methods* **14**, 849–863 (2017).
13. Seal, S. *et al.* Cell Painting: a decade of discovery and innovation in cellular imaging. *Nat. Methods* **22**, 254–268 (2025).
14. Way, G. P., Sailem, H., Shave, S., Kasprowicz, R. & Carragher, N. O. Evolution and impact of high content imaging. *SLAS Discov.* **28**, 292–305 (2023).
15. Cimini, B. A. *et al.* The crucial role of bioimage analysts in scientific research and publication. *J. Cell Sci.* **137**, jcs262322 (2024).
16. Cohen, J. *et al.* The Four Pillars of Research Software Engineering. *IEEE Softw.* **38**, 97–105 (2021).
17. Lapatas, V., Stefanidakis, M., Jimenez, R. C., Via, A. & Schneider, M. V. Data integration in biological research: an overview. *J. Biol. Res.-Thessalon.* **22**, 9 (2015).
18. Shearer, C. The CRISP-DM Model: The New Blueprint for Data Mining. *J. Data Warehous.* **5**, 13–22 (2000).
19. Pedreira, P. *et al.* The Composable Data Management System Manifesto. *Proc. VLDB*

Endow. **16**, 2679–2685 (2023).
20. Wilson, G. *et al.* Best Practices for Scientific Computing. *PLoS Biol.* **12**, e1001745 (2014).
21. Wilson, G. *et al.* Good enough practices in scientific computing. *PLOS Comput. Biol.* **13**, e1005510 (2017).
22. Hinsen, K. Dealing With Software Collapse. *Comput. Sci. Eng.* **21**, 104–108 (2019).
23. Bunten, D., Davidson, W. & Way, G. P. Growing Resilient Scientific Software Ecosystems: Introducing the Software Gardening Almanack. *Better Scientific Software (BSSw) Blog* https://bssw.io/blog_posts/growing-resilient-scientific-software-ecosystems-introducing-the-software-gardening-almanack (2025).
24. Bunten, D., Davidson, W., Alquaddoomi, F., Rubinetti, V. & Way, G. The Software Gardening Almanack. Zenodo https://doi.org/10.5281/ZENODO.14765834 (2025).
25. Apache Parquet. Apache Software Foundation.
26. Melnik, S. *et al.* Dremel: interactive analysis of web-scale datasets. *Proc. VLDB Endow.* **3**, 330–339 (2010).
27. Amazon Web Services, Inc. Amazon Athena. (2025).
28. Dageville, B. *et al.* The Snowflake Elastic Data Warehouse. in *Proceedings of the 2016 International Conference on Management of Data* 215–226 (ACM, San Francisco California USA, 2016). doi:10.1145/2882903.2903741.
29. Zaharia, M. *et al.* Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* 2 (USENIX Association, USA, 2012).
30. Chamberlin, D. D. Early History of SQL. *IEEE Ann. Hist. Comput.* **34**, 78–82 (2012).
31. Dennis, J. B. & Van Horn, E. C. Programming semantics for multiprogrammed computations. *Commun. ACM* **9**, 143–155 (1966).
32. PEP 703 – Making the Global Interpreter Lock Optional in CPython.
33. Way, G. P. *et al.* Morphology and gene expression profiling provide complementary information for mapping cell state. *Cell Syst.* **13**, 911-923.e9 (2022).
34. Chandrasekaran, S. N. *et al.* Three million images and morphological profiles of cells treated with matched chemical and genetic perturbations. *Nat. Methods* **21**, 1114–1121 (2024).
35. Bunten, D., Lippincott, M., Alquaddoomi, F. & Way, G. CytoTable-benchmarks. Zenodo https://doi.org/10.5281/ZENODO.15425830 (2025).
36. McKinney, W. Data Structures for Statistical Computing in Python. in *Proceedings of the 9th Python in Science Conference* (eds. Walt, S. van der & Millman, J.) 56–61 (2010). doi:10.25080/Majora-92bf1922-00a.
37. Monitoring. *Parsl User Guide documentation* https://parsl.readthedocs.io/en/stable/userguide/advanced/monitoring.html.
38. psutil. https://github.com/giampaolo/psutil .
39. Raasveldt, M., Holanda, P., Gubner, T. & Mühleisen, H. Fair Benchmarking Considered Difficult: Common Pitfalls In Database Performance Testing. in *Proceedings of the Workshop on Testing Database Systems* (Association for Computing Machinery, New York, NY, USA, 2018). doi:10.1145/3209950.3209955.
40. Vink, R. & Contributors, the P. Polars: Dataframes powered by a multithreaded, vectorized query engine, written in Rust. (2025).
41. Lippincott, M. J. *et al.* A morphology and secretome map of pyroptosis. Preprint at https://doi.org/10.1101/2024.04.26.591386 (2024).
42. Tomkinson, J., Serrano, E., Curd, J., Li, W. & Way, G. pediatric_cancer_atlas_profiling. Zenodo https://doi.org/10.5281/ZENODO.15548848 (2025).
43. Tomkinson, J., Bunten, D., Rubinetti, V. & Way, G. coSMicQC. Zenodo https://doi.org/10.5281/ZENODO.14797008 (2025).
44. Tomkinson, J. *et al.* High-content microscopy and machine learning characterize a cell morphology signature of NF1 genotype in Schwann cells. Preprint at

https://doi.org/10.1101/2024.09.11.612546 (2024).
45. Pace, W. *et al.* Lance: Efficient Random Access in Columnar Storage through Adaptive Structural Encodings. Preprint at https://doi.org/10.48550/ARXIV.2504.15247 (2025).
46. Chattopadhyay, B. *et al.* Shared Foundations: Modernizing Meta's Data Lakehouse. in *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023* (www.cidrdb.org, 2023).
47. Virshup, I., Rybakov, S., Theis, F. J., Angerer, P. & Wolf, F. A. anndata: Access and store annotated datamatrices. *J. Open Source Softw.* **9**, 4371 (2024).
48. Virshup, I. *et al.* The scverse project provides a computational ecosystem for single-cell omics data analysis. *Nat. Biotechnol.* **41**, 604–606 (2023).
49. Bunten, D., Tomkinson, J. & Way, G. CytoDataFrame. Zenodo https://doi.org/10.5281/ZENODO.14797074 (2025).
50. Martinez-Plumed, F. *et al.* CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories. *IEEE Trans. Knowl. Data Eng.* **33**, 3048–3061 (2021).
51. Datatypes In SQLite. https://www.sqlite.org/datatype3.html .
52. NumPy Constants. https://numpy.org/doc/stable//reference/constants.html .