Single-cell data analysis in the browser

Aaron Lun[1] and Jayaram Kancherla[1]

[1]Genentech, Inc. South San Francisco, CA

March 2, 2022

Abstract

We present kana, a web application for interactive single-cell RNA-seq (scRNA-seq) data analysis in the browser. Like, literally, in the browser: kana leverages web technologies such as WebAssembly to efficiently perform the relevant computations on the user's machine, avoiding the need to provision and maintain a backend service. The application provides a streamlined one-click workflow for all steps in a typical scRNA-seq analysis, starting from a count matrix and finishing with marker detection. Results are presented in an intuitive web interface for further exploration and iterative analysis. Testing on public datasets shows that kana can analyze over 100,000 cells within 5 minutes on a typical laptop.

1 Introduction

Modern web browsers are sophisticated pieces of software responsible for rendering, scripting, networking, data storage and more. New web technologies such as WebAssembly [20] have greatly enhanced browsers' capabilities for intensive computation, providing new opportunities to repurpose the browser as an interactive data analysis tool. The idea is to use the browser itself to perform statistical and computational analyses directly on the user's machine, i.e., "client-side compute". Those analysis results can then be rendered on a web page for convenient visualization and exploration within a familiar interface.

This paradigm of client-side compute in the browser has several advantages over the traditional model of server-side data analysis. Most obviously, we do not need a scalable backend server to perform any of the calculations, simplifying deployment and reducing costs. As the entire analysis is performed on the client machine, we avoid any data transfer - this reduces latency in the user interface and circumvents concerns over data ownership and privacy. Finally, we do not require installation of any data analysis environments like R or Python, ensuring that the analyses are accessible to audiences of varying computational skill.

Some prior work already exists to realize these client-side benefits for bioinformatics data analysis. For example, the BioJS registry hosts Javascript components for handling biological data [16], many of which focus on DNA and

protein sequence analysis. The BrowserGenome.org application performs read alignment and transcript quantification for RNA sequencing data [50], while the ubit2 application analyzes single-cell quantitative polymerase chain reaction datasets [15]; both use Javascript implementations of the relevant algorithms to run in the browser. However, these are exceptions to the rule; most bioinformatics web applications only use the browser to visualize results that were computed elsewhere, e.g., cellxgene [45], Epiviz [11], Cirrocumulus [17], Vitessce [24], HiGlass [25] and every R/Shiny application ever [10]. This state of affairs is not unexpected - after all, Javascript is not widely recognized as a language for data science, nor it is particularly efficient for such tasks.

The client-side paradigm is especially appealing for single-cell genomics due to the exploratory nature of its data analysis. Most use cases for single-cell sequencing involve identifying new cell subpopulations or states from heterogeneous biological samples [52], which requires several iterations of data analysis, visualization and interpretation. If the compute could be performed in the browser, the results could be immediately rendered on a web page for a seamless transition between analysis and interactive exploration. However, this is complicated by the size of the datasets, each of which involves tens of thousands of genes and up to millions of cells; and the paucity of browser-compatible implementations of relevant algorithms, most of which are written in R or Python.

We present kana, a web application for single-cell RNA-seq (scRNA-seq) data analysis inside the browser. kana provides a streamlined one-click workflow for all steps in a typical scRNA-seq analysis [2], starting from a count matrix and finishing with marker detection. Users can interactively explore the low-dimensional embeddings, clusterings and marker genes in an intuitive graphical interface that encourages iterative re-analysis. Once finished, users can save their analysis and results for later examination or sharing with collaborators. By using technologies like WebAssembly and web workers, we achieve high-performance compute for datasets containing hundreds of thousands of cells.

The kana application is available at `https://www.jkanche.com/kana`. Developers can set up their own deployments by following the instructions at `https://github.com/jkanche/kana`.

2 Application overview

Given a scRNA-seq dataset, kana implements a routine analysis with the steps listed below. We will not discuss the statistical and scientific rationale behind each step in much detail as this has been covered elsewhere [36].

1. We import a gene-by-cell count matrix from the user's machine, typically in the form of Matrix Market files such as those produced by the Cellranger pipeline. HDF5 files are also supported, using either the 10X HDF5 feature barcode matrix format or as H5AD files.

2. We compute common quality control (QC) metrics such as the total count, number of detected genes and proportion of mitochondrial counts. Low-

quality cells are defined as those cells with outlier values for any of these metrics; these are filtered out from subsequent steps.

3. We perform scaling normalization based on the library size to remove cell-specific biases. This is followed by a log-transformation to obtain a matrix of log-normalized expression values.

4. We fit a LOWESS trend [12] to the per-gene variances with respect to the means, computed from the log-expression values. We sort on the residuals to define a subset of highly variable genes (HVGs).

5. We perform principal components analysis (PCA) on the log-expression matrix with the subset of HVGs. This yields a few top PCs that capture the heterogeneity of the data in a compressed and denoised form.

6. We apply clustering techniques on the top PCs to generate discrete sub-populations. By default, this uses multi-level (i.e., "louvain") community detection on a shared nearest neighbor (SNN) graph where each cell is a node and edges connect neighboring cells. We also support k-means clustering with some user-specified choice of the number of clusters.

7. Each cluster is characterized through differential expression analyses to detect its marker genes. Specifically, we perform a series of pairwise comparisons between clusters and summarize the effect sizes into a ranking of potential marker genes for each cluster.

8. We perform further dimensionality reduction on the top PCs to obtain two-dimensional embeddings for visualization. This includes the usual t-distributed stochastic neighbor embedding (t-SNE) and uniform manifold approximation and projection (UMAP) [54, 44].

At each step, users can easily customize key parameters (Figure 1). For example, we can adjust the QC thresholds, the number of HVGs and top PCs, the granularity of the clustering, and more. Iterative refinements to the parameters are encouraged in kana, as the application tracks dependencies between steps to enable efficient re-analysis. Specifically, when parameters are modified for any step, all subsequent steps are automatically re-executed to propagate the change to downstream results. Conversely, kana does not rerun any steps upstream of the change to avoid unnecessary recomputation and reduce latency.

Once each step of the analysis is complete, kana visualizes its results in a multi-panel layout (Figure 2). One panel contains a scatter plot for the low-dimensional embeddings, where each cell is a point that is colored by cluster identity or gene expression. Another panel contains a table of marker statistics for a selected cluster, where potential marker genes are ranked and filtered according to the magnitude of upregulation over other clusters. We also provide a gallery to visualize miscellaneous details such as the distribution of QC metrics.

Finally, users can export the analysis parameters and results for later inspection. The exported analysis can be quickly reloaded in a new browser session, allowing users to view the results without repeating the computation.

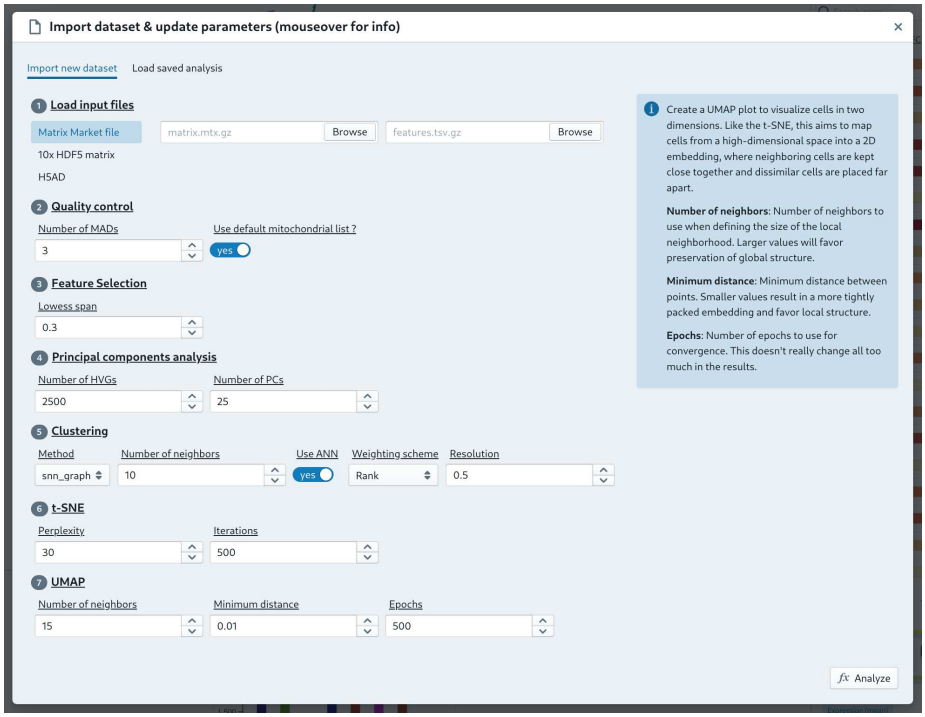

Figure 1: Screenshot showing the analysis configuration panel in the kana application. Clicking "Analyze" will perform the entire analysis.

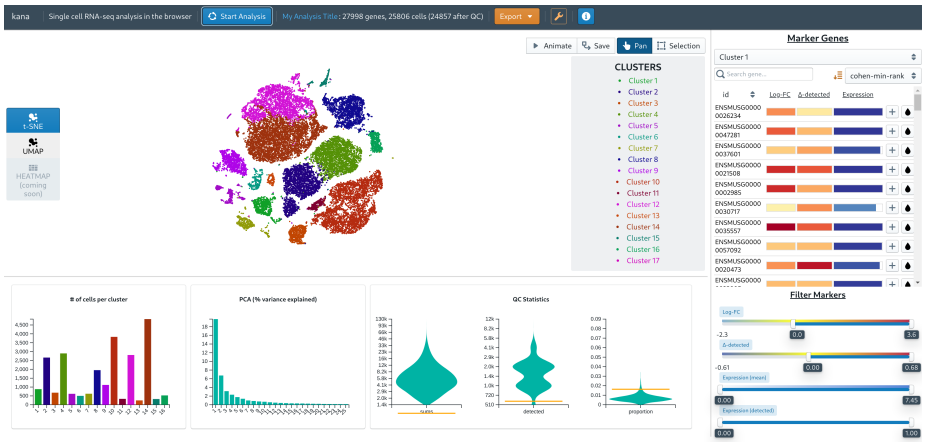

Figure 2: Screenshot showing the multi-panel layout for results in the kana application. The top-left panel is used for the low-dimensional embeddings, the right panel contains the marker table for a selected cluster, and the bottom-left panel contains a gallery of miscellaneous plots.

3 Client-side computation

3.1 Efficient compute with WebAssembly

WebAssembly (Wasm) [20] is a binary instruction format that provides a web-executable compilation target for languages like C/C++, Go and Rust. It aims to provide near-native performance for computationally intensive tasks, running alongside and complementing Javascript in web applications. The use of Wasm allows us to convert the browser into a compute engine by integrating existing scientific libraries for bioinformatics data analysis - see the biowasm project [1] for previous efforts in this direction. For kana, we collected or created C++ implementations of the algorithms required for each analysis step:

- The tatami library [38] provides an abstract interface to different matrix classes, based on similar ideas in the beachmat package [37]. In addition to the usual dense and sparse representations, tatami also supports the delayed operations implemented in the DelayedArray package [19]. This allows the creation of QC-filtered and log-normalized matrices without any duplication of expression data.

- The CppWeightedLowess library [35] contains a C++ port of the weightedLowess function from the limma package [49]. This function is, in turn, based on the LOWESS algorithm [12] implemented in R's lowess function, modified with the ability to consider weights in the span calculations.

- The CppIrlba library [29] library contains a C++ port of the IRLBA

algorithm [7] to efficiently obtain the top PCs from an input matrix. This is based on the C code in the irlba R package [8] with some refactoring to eliminate R-specific dependencies. In particular, we now rely on the Eigen library [18] for matrix operations.

- The CppKmeans library [30] implements the Hartigan-Wong [22] and Lloyd algorithms [27] for k-means clustering. The Hartigan-Wong implementation was translated from the Fortran code used by R's kmeans function. We also provide several initialization methods including kmeans++ [55] and PCA partitioning [53].

- The knncolle library [34] wraps several nearest neighbor detection algorithms in a consistent interface. This includes exact methods like vantage point tree search [56] as well as approximate methods like Annoy [9]. It is based on the BiocNeighbors package from Bioconductor [28].

- The libscran library [31] implements high-level methods for scRNA-seq data analysis, ranging from quality control to clustering. The code here originates from the scran, scuttle and scater packages [42, 43] bundled together into a single C++ library for convenience. This library relies on the igraph C library [13] for community detection from the SNN graph.

- The qdtsne library [32] contains a C++ implementation of the Barnes-Hut t-SNE algorithm [54]. This is mostly a refactored version of the code in the Rtsne package [26]. Some additional optimizations have been applied to improve scalability.

- The umappp library [33] contains a C++ implementation of the UMAP algorithm [44]. This is derived from code in the uwot R package [46].

We created the scran.js library [41] to provide Javascript bindings to these C++ libraries. Specifically, we compiled our C++ code to Wasm using the Emscripten toolchain [57], allowing kana to perform scRNA-seq-related calculations from Javascript by calling scran.js functions. The same library can also be used in other web applications via a standard NPM installation, or outside the browser if a suitable Wasm runtime environment is available.

To evaluate the efficiency of our Wasm strategy, we compared a kana analysis in the browser to that of a native executable compiled from the same C++ libraries [39]. We analyzed several public scRNA-seq datasets (Table 1) using the default kana parameters for both approaches, i.e., QC filtering to 3 MADs from the median; PCA on the top 2500 HVGs to obtain the top 25 PCs; SNN graph construction with 10 neighbors and multi-level community detection at a resolution of 0.5; t-SNE with a perplexity of 30; UMAP with 15 neighbors and a minimum distance of 0.01; and 8 threads for all parallel sections (i.e., web workers for kana, see below). We collected timings on an Intel Core i7-8850H CPU (2.60GHz, 6 cores, 32 GB memory) running Manjaro Linux. For convenience, we ran the kana timings in batch using Puppeteer [4] to control a headless Chrome browser (HeadlessChrome/98.0.4758.0).

Table 1: Collection of scRNA-seq datasets used for testing.

Study	Species	Tissue	Technology	Number of cells
Zeisel *et al.* (2015) [58]	Mouse	Brain	STRT-seq	3005
Paul *et al.* (2015) [48]	Mouse	Bone marrow	MARS-seq	10368
Bach *et al.* (2017) [5]	Mouse	Mammary gland	10x Genomics	25806
Ernst *et al.* (2019) [14]	Mouse	Spermatogonia	10x Genomics	68937
Bacher *et al.* (2020) [6]	Human	T cells	10x Genomics	104417
Zilionis *et al.* (2019) [59]	Human	Lung	10x Genomics	173954

Table 2: Analysis run times for several datasets with kana or a native executable. Values are reported in seconds with standard errors from 3 runs.

Dataset	Number of cells	kana	Native
Zeisel	3005	7.00 ± 0.10	5.60 ± 0.05
Paul	10368	17.59 ± 0.20	13.52 ± 0.38
Bach	25806	54.96 ± 1.13	43.33 ± 0.39
Ernst	68937	157.15 ± 7.39	114.67 ± 1.86
Bacher	104417	228.02 ± 2.85	170.32 ± 1.34
Zilionis	173954	272.265 ± 4.22	183.77 ± 2.46

Our results indicate that kana analyses took approximately 25-50% longer to run compared to the native executable (Table 2). This is consistent with other benchmarking results [23] where the performance gap is attributed to Wasm's design constraints and the overhead of the browser's Wasm runtime environment. Our native executable was also created with a different compiler toolchain (GCC, instead of LLVM for the Wasm binary), where the same nominal optimization level (O3) may have different effects. These results suggest that some work may still be required to completely fulfill Wasm's promise of "near-native execution". Nonetheless, the current performance is largely satisfactory for kana, and will likely improve over time as browser implementations evolve along with our understanding of the relevant optimizations.

3.2 Parallelization with web workers

Web workers provide a simple mechanism for parallelization inside the browser. In kana, a web worker is used to run the compute-intensive analysis in its own thread so that the application's main thread is free to respond to user interaction. We also use web workers to parallelize the operations within some analysis steps by compiling our C++ code with PThreads support; this instructs Emscripten to implement POSIX threads as web workers for transparent parallelization across genes or cells during calculation of QC metrics, nearest neighbor

detection and marker scoring. Finally, we parallelize across steps by manually creating a separate web worker to execute the t-SNE, UMAP and clustering steps, which are independent of each other and can run concurrently.

A minor complication with PThreads is that we need to enable cross-origin isolation on the kana site. Briefly, this is a security measure that prevents inappropriate data access from malicious scripts in the same browsing context. Once cross-origin isolated, the site is permitted to use a SharedArrayBuffer for copy-free transfer of data between web workers. To achieve this, we need to serve the kana assets with the appropriate cross-origin headers – specifically, the embedder and opener policies. However, this may not always be possible, e.g., when hosting on institutional sites or public sites such as GitHub Pages. Instead, we use a service worker to cache and re-serve kana with the correct headers, ensuring that it can be easily deployed in a range of hosting environments.

As a matter of etiquette, we limit kana to two-thirds of the available threads at maximum utilization. The intention is to leave enough resources available for other activities to pass the time while waiting for the analysis to finish.

3.3 Creating layered sparse matrices

To reduce memory usage for large single-cell count matrices, we use a "layered matrix" approach that splits the input matrix by row into 3 sparse submatrices. The first, second and third submatrices contain data for genes where all non-zero counts can fit into 8-bit, 16-bit and 32-bit unsigned integers, respectively. This improves memory efficiency for datasets with many cells but low sequencing coverage; most counts can be represented as 8-bit integers to save space, leaving a few high-abundance genes to be stored in the submatrices with larger integer types. A similar strategy is applied to row indices for non-zero elements in a sparse matrix, where most datasets have fewer than 60,000 genes and can be accommodated with 16-bit integers. This gives a theoretical usage of 3 bytes per non-zero element, e.g., a dataset with 30,000 genes and 100,000 cells at 5% density requires around 500 MB of memory in this data structure.

The layered matrix representation is implemented through the delayed binding mechanism in tatami. Specifically, we create the individual sparse submatrices and then create an abstract represention of the full matrix where the submatrices are combined by row. This preserves the memory-efficient representation while presenting an interface that mimics that of a single matrix. The layered representation can then be seamlessly used with all C++ code compatible with the tatami interface. (Note that the genes are permuted from their supplied order, which requires some extra attention in downstream analyses.)

3.4 Examining memory usage

As an additional evaluation, we recorded the memory usage of the kana analyses (Table 3). For kana, we used the final size of the Wasm heap as a convenient proxy for the peak memory used by the application. This omits some memory usage on the Javascript side but should capture the most memory-intensive data

Table 3: Memory usage of kana or a native executable when analyzing datasets from Table 1. Values are reported in megabytes with standard errors from 3 runs. The number of non-zero elements for each dataset is also shown in millions.

Dataset	Cells	Non-zeros	kana	Native
Zeisel	3005	11.3	148.31 ± 0	158.20 ± 0.05
Paul	10368	11.9	224.87 ± 0	306.26 ± 0.06
Bach	25806	61.2	761.12 ± 0	965.17 ± 0.03
Ernst	68937	163.1	1981.87 ± 0	2512.08 ± 0.06
Bacher	104417	143.0	2165.94 ± 0	2437.75 ± 1.34
Zilionis	173954	53.7	2480.81 ± 0	2600.66 ± 0.04

structures, particularly the count matrix. For comparison, we also recorded the maximum resident set size after each run of the native executable. In all cases, memory usage with Wasm was comparable to that of the native executable and maintained below 3 gigabytes, which is reasonable for a client device.

Importantly, kana's memory usage lies below Wasm's hard limit of 4 gigabytes of addressable memory. This limit is a consequence of the use of 32-bit pointers in the current Wasm specification, and exceeding this limit will cause allocation errors and application failure. Future Wasm releases may support 64-bit pointers [51] or multiple memories [47]; if these proposals are accepted, kana will be able to process datasets beyond the 4 GB limit.

4 User interface concepts

4.1 Interlinked graphics

Different panels of the kana application (Figure 2) can share information with each other to facilitate interactive exploration of the dataset and analysis results. For example, we can color the embeddings according to the expression of a gene selected from the marker table. Upon selection, kana retrieves the log-normalized expression values for that gene from the sparse matrix and passes this data to the scatter plot for coloring. Users can also adjust the color gradient to improve contrast and highlight differences at particular ranges of expression.

A more complex example involves the detection of new marker genes for a custom selection of cells. Users can create a custom selection by brushing on regions of interest in the embedding panel. If this selection is saved, kana will perform a differential expression analysis to detect upregulation inside the selection compared to all other cells. Statistics for each gene are then shown in the marker table for examination. Each custom selection and its statistics are treated as part of the analysis state and are saved during export.

Inspired by the gallery in Cirrocumulus, users can save specific views of the embeddings for later perusal. This is useful for simultaneously viewing multiple

plots colored by different marker genes to characterize complex populations.

4.2 Progressive rendering

Each result is immediately rendered on the interface once the corresponding analysis step is complete. For example, the distribution of QC metrics appear once the QC step is finished, followed by the plot of the percentage of variance explained once the PCA is done, and so on. This improves application performance by avoiding the rendering bottleneck that would otherwise occur if all plots were drawn at once. It also serves as a visual progress indicator and improves the user experience by showing meaningful content as soon as possible.

The marker table is a more subtle example of progressive rendering. Only the genes in the current view of the table are rendered, with the remaining visual elements being dynamically created as the user scrolls up or down to see other genes in the ranking. This ensures that we do not waste time rendering tens of thousands of rows when only the top few are likely to be viewed.

Given that kana is computing the embeddings in the background, we can actively monitor the change in the coordinates for each cell across t-SNE iterations or UMAP epochs. We do so by extracting the coordinates at regular intervals and rendering them on the embedding panel with WebGL, effectively creating an animation of the embedding as it is refined over time. This is mostly present for entertainment value but may provide some educational insights into the process of creating the embeddings (e.g., early exaggeration for t-SNE).

4.3 Exporting and reloading

Users can choose to export their analyses to the browser's cache via its in-built IndexedDB database system. This dumps the analysis state (i.e., parameters and results) into a Gzip-compressed JSON inside the cache, along with the input data files. Users can then reload their analysis in a new browser session without needing to keep track of any files. Moreover, kana will attempt to deduplicate input files in the cache based on the file name, size and MD5 checksum. This reduces disk usage when storing multiple analysis states for the same dataset.

Alternatively, users may export the analysis state to a file that is downloaded to their machine. This creates a single binary file containing the Gzip-compressed analysis state and the embedded input files. Supplying this file to kana will then restore the previous analysis in the same manner as using the browser cache. The benefit of this approach is that files exported by one user can be reloaded by other users on different machines, allowing users to easily share their analyses with each other by simply transferring the relevant files. In fact, computationally savvy users can even parse the kana-exported file in other programming frameworks (e.g., R, Python) for custom processing.

5 Discussion

Single-cell data analysis on the client is not a particularly novel concept. In many cases, the dataset already lives on the user's machine, so this is a natural place to perform the analysis with the tool of choice (typically R or Python). kana's key innovation lies in its use of modern web technologies to perform the analysis directly in the browser. This eliminates the difficulties of software installation and makes the analysis accessible to a non-programming audience. At the same time, we retain all the benefits of client-side operations. Specifically, our availability only depends on the client device, not on a backend server; there is no latency from transferring data or results across a network; ownership of the dataset clearly remains with the user, avoiding issues with data privacy and associated regulations; and compute time is cheap, if not effectively free.

Client-side compute has interesting scalability characteristics compared to a traditional backend approach. Most obviously, we are constrained by the computational resources available on the client machine, which can vary from modest (e.g., most laptops) to extremely basic (e.g., mobile devices). This limits the size of any single dataset that can be analyzed by a particular client. However, in other respects, client-side compute is more scalable than backend compute; the former automatically distributes analyses of many datasets across any number of machines at no cost and with no configuration. This is especially relevant for web applications like kana where the maintainers would otherwise be responsible for provisioning backend computing resources. The cloud might be infinitely scalable but - unfortunately - our bank accounts are not.

That said, how do we deal with large datasets? This is not straightforward with the web technologies currently used in kana - our available memory is limited by the Wasm specification and our CPU utilization is (somewhat artificially) capped, so even if a powerful client device was available, kana would not be able to fully exploit its capabilities. Fortunately, our use of C++ means that we are not limited to computation in the browser. We can easily provide wrappers to the same underlying libraries in any client-side framework, e.g., as a command-line tool or as an extension to existing data science ecosystems. For example, the scran.chan package [40] allows users to repeat the kana calculations in an R session, where there are no restrictions on memory usage or thread utilization. The same approach can be used for other languages that support a foreign function interface like Python or Julia. Indeed, one could use the wrapped C++ libraries to run large analyses on a sufficiently provisioned backend, export the analysis state in a kana-compatible file format, and then serve the exported state to client machines for exploration in the browser.

Future work will involve extending kana to handle other common tasks in scRNA-seq data analysis, namely batch correction [21] and cell type assignment [3]. We will also factor out kana's graphical elements into custom Web Components for re-use across multiple applications. Of greatest interest, though, will be the continued evolution of web technologies - this includes further optimizations to the Wasm specification and its implementations, as well as upcoming standards like WebGPU to leverage graphics hardware for general computa-

tion. By incorporating these advances, applications like kana can extend the capabilities of the browser for compute-intensive bioinformatics in the client.

6 Software availability

Here we summarize the key software repositories used in this paper. For brevity, we will omit the various C++ libraries that have been listed previously.

- The kana application is deployed at `https://www.jkanche.com/kana` with source code at `https://github.com/jkanche/kana`.
- The scran.js library is available at `https://npmjs.com/package/scran.js` with source code at `https://github.com/jkanche/scran.js`.
- The scran.chan R package based on the same C++ libraries is available at `https://github.com/LTLA/scran.chan`.
- The command-line interface based on the same C++ libraries is available at `https://github.com/LTLA/scran-cli`.
- All datasets used here can be downloaded from `https://github.com/jkanche/random-test-files`.
- Benchmarking code and results are available at `https://github.com/jkanche/kana.perf`.

References

[1] Robert Aboukhalil. Biowasm, 2019. `https://github.com/biowasm/biowasm`.

[2] R. A. Amezquita, A. T. L. Lun, E. Becht, V. J. Carey, L. N. Carpp, L. Geistlinger, F. Marini, K. Rue-Albrecht, D. Risso, C. Soneson, L. Waldron, H. Pagès, M. L. Smith, W. Huber, M. Morgan, R. Gottardo, and S. C. Hicks. Orchestrating single-cell analysis with Bioconductor. *Nat Methods*, 17(2):137–145, 02 2020.

[3] Dvir Aran, Agnieszka P Looney, Leqian Liu, Esther Wu, Valerie Fong, Austin Hsu, Suzanna Chak, Ram P Naikawadi, Paul J Wolters, Adam R Abate, et al. Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage. *Nature immunology*, 20(2):163–172, 2019.

[4] The Chromium Authors. Puppeteer, 2004. `https://github.com/puppeteer/puppeteer`.

[5] Karsten Bach, Sara Pensa, Marta Grzelak, James Hadfield, David J Adams, John C Marioni, and Walid T Khaled. Differentiation dynamics of mammary epithelial cells revealed by single-cell RNA sequencing. *Nature communications*, 8(1):1–11, 2017.

[6] Petra Bacher, Elisa Rosati, Daniela Esser, Gabriela Rios Martini, Carina Saggau, Esther Schiminsky, Justina Dargvainiene, Ina Schröder, Imke Wieters, Yascha Khodamoradi, et al. Low-avidity CD4+ T cell responses to SARS-CoV-2 in unexposed individuals and humans with severe COVID-19. *Immunity*, 53(6):1258–1271, 2020.

[7] James Baglama and Lothar Reichel. Augmented implicitly restarted lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing*, 27(1):19–42, 2005.

[8] Jim Baglama, Lothar Reichel, and B. W. Lewis. *irlba: Fast Truncated Singular Value Decomposition and Principal Components Analysis for Large Dense and Sparse Matrices*, 2019. R package version 2.3.3.

[9] E. Bernhardsson. Annoy, 2021. `https://github.com/spotify/annoy`.

[10] Winston Chang, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. *shiny: Web Application Framework for R*, 2021. R package version 1.7.1.

[11] Florin Chelaru, Llewellyn Smith, Naomi Goldstein, and Héctor Corrada Bravo. Epiviz: interactive visual analytics for functional genomics data. *Nature methods*, 11(9):938–940, 2014.

[12] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

[13] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[14] Christina Ernst, Nils Eling, Celia P Martinez-Jimenez, John C Marioni, and Duncan T Odom. Staged developmental mapping and X chromosome transcriptional dynamics during mouse spermatogenesis. *Nature communications*, 10(1):1–20, 2019.

[15] Jean Fan, David Fan, Kamil Slowikowski, Nils Gehlenborg, and Peter Kharchenko. UBiT2: a client-side web-application for gene expression data analysis. *bioRxiv*, page 118992, 2017.

[16] John Gómez, Leyla J García, Gustavo A Salazar, Jose Villaveces, Swanand Gore, Alexander García, Maria J Martín, Guillaume Launay, Rafael Alcántara, Noemi Del-Toro, et al. BioJS: an open source Javascript framework for biological data visualization. *Bioinformatics*, 29(8):1103–1104, 2013.

[17] Joshua Gould, Yiming Yang, and Bo Li. Cirrocumulus, 2021. `https://cirrocumulus.readthedocs.io/en/latest/`.

[18] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[19] Pagès H., Hickey P., and Lun A. T. L. *DelayedArray: A unified framework for working transparently with on-disk and -memory array-like datasets*, 2021. R package version 0.20.0.

[20] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. *SIGPLAN Not.*, 52(6):185–200, jun 2017.

[21] Laleh Haghverdi, Aaron TL Lun, Michael D Morgan, and John C Marioni. Batch effects in single-cell rna-sequencing data are corrected by matching mutual nearest neighbors. *Nature biotechnology*, 36(5):421–427, 2018.

[22] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[23] Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. Not so fast: Analyzing the performance of WebAssembly vs. native code. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 107–120, Renton, WA, July 2019. USENIX Association.

[24] Mark Keller, Chuck McCallum, Ilan Gold, Trevor Manz, Tos Chan, Sehi Lyi, Jennifer Marx, Peter Kharchenko, and Nils Gehlenborg. Vitessce, 2022. `http://vitessce.io`.

[25] Peter Kerpedjiev, Nezar Abdennur, Fritz Lekschas, Chuck McCallum, Kasper Dinkla, Hendrik Strobelt, Jacob M Luber, Scott B Ouellette, Alaleh Azhir, Nikhil Kumar, et al. Higlass: web-based visual exploration and analysis of genome interaction maps. *Genome biology*, 19(1):1–12, 2018.

[26] Jesse H. Krijthe. *Rtsne: T-Distributed Stochastic Neighbor Embedding using Barnes-Hut Implementation*, 2015. R package version 0.16.

[27] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[28] A. T. L. Lun. *BiocNeighbors: Nearest Neighbor Detection for Bioconductor Packages*, 2021. R package version 1.12.0.

[29] A. T. L. Lun. C++ library for IRLBA, 2021. `https://github.com/LTLA/CppIrlba`.

[30] A. T. L. Lun. C++ library for k-means, 2021. `https://github.com/LTLA/CppKmeans`.

[31] A. T. L. Lun. A C++ library for single-cell data analysis, 2021. `https://github.com/LTLA/libscran`.

[32] A. T. L. Lun. C++ library for t-SNE, 2021. `https://github.com/LTLA/qdtsne`.

[33] A. T. L. Lun. A C++ library for UMAP, 2021. `https://github.com/LTLA/umappp`.

[34] A. T. L. Lun. Collection of KNN algorithms, 2021. `https://github.com/LTLA/knncolle`.

[35] A. T. L. Lun. Weighted LOWESS for C++, 2021. `https://github.com/LTLA/CppWeightedLowess`.

[36] A. T. L. Lun, R. A. Amezquita, R. Gottardo, and S. C. Hicks. Orchestrating single-cell analysis with Bioconductor, 2020. `https://bioconductor.org/books/release/OSCA/`.

[37] A. T. L. Lun, H. Pagès, and M. L. Smith. beachmat: A Bioconductor C++ API for accessing high-throughput biological data from a variety of R matrix types. *PLoS Comput Biol*, 14(5):e1006135, 05 2018.

[38] Aaron Lun. A C++ API for all sorts of matrices, 2021. `https://github.com/LTLA/tatami`.

[39] Aaron Lun. CLI for single-cell analyses, 2021. `https://github.com/LTLA/scran-cli`.

[40] Aaron Lun. A slimmed-down version of scran, 2021. `https://github.com/LTLA/scran.chan`.

[41] Aaron Lun and Jayaram Kancherla. Single cell RNA-seq analysis in Javascript, 2021. `https://github.com/jkanche/scran.js`.

[42] Aaron T. L. Lun, Davis J. McCarthy, and John C. Marioni. A step-by-step workflow for low-level analysis of single-cell rna-seq data with bioconductor. *F1000Res.*, 5:2122, 2016.

[43] Davis J. McCarthy, Kieran R. Campbell, Aaron T. L. Lun, and Quin F. Willis. Scater: pre-processing, quality control, normalisation and visualisation of single-cell RNA-seq data in R. *Bioinformatics*, 33:1179–1186, 2017.

[44] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. UMAP: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018.

[45] Colin Megill, Bruce Martin, Charlotte Weaver, Sidney Bell, Lia Prins, Seve Badajoz, Brian McCandless, Angela Oliveira Pisco, Marcus Kinsella, Fiona Griffin, Justin Kiggins, Genevieve Haliburton, Arathi Mani, Matthew Weiden, Madison Dunitz, Maximilian Lombardo, Timmy Huang, Trent Smith, Signe Chambers, Jeremy Freeman, Jonah Cool, and Ambrose Carr. cellxgene: a performant, scalable exploration platform for high dimensional sparse matrices. *bioRxiv*, 2021.

[46] James Melville. *uwot: The Uniform Manifold Approximation and Projection (UMAP) Method for Dimensionality Reduction*. R package version 0.1.10.9000.

[47] Ömer Sinan Agacan and Andreas Rossberg. Multi memory proposal for webassembly, 2021. `https://github.com/WebAssembly/multi-memory`.

[48] Franziska Paul, Ya'ara Arkin, Amir Giladi, Diego Adhemar Jaitin, Ephraim Kenigsberg, Hadas Keren-Shaul, Deborah Winter, David Lara-Astiaso, Meital Gury, Assaf Weiner, et al. Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell*, 163(7):1663–1677, 2015.

[49] Matthew E Ritchie, Belinda Phipson, Di Wu, Yifang Hu, Charity W Law, Wei Shi, and Gordon K Smyth. limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47, 2015.

[50] Jonathan L Schmid-Burgk and Veit Hornung. BrowserGenome.org: web-based RNA-seq data analysis and visualization. *Nature Methods*, 12(11):1001–1001, 2015.

[51] Ben Smith, Wouter van Oortmerssen, and Andreas Rossberg. Memory64 proposal for webassembly, 2021. `https://github.com/WebAssembly/memory64`.

[52] Oliver Stegle, Sarah A Teichmann, and John C Marioni. Computational and analytical challenges in single-cell transcriptomics. *Nature Reviews Genetics*, 16(3):133–145, 2015.

[53] Ting Su and Jennifer G Dy. In search of deterministic methods for initializing k-means and gaussian mixture clustering. *Intelligent Data Analysis*, 11(4):319–338, 2007.

[54] Laurens van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014.

[55] Sergei Vassilvitskii and David Arthur. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2006.

[56] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, page 311–321, USA, 1993. Society for Industrial and Applied Mathematics.

[57] Alon Zakai. Emscripten: an llvm-to-javascript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–312, 2011.

[58] Amit Zeisel, Ana B Muñoz-Manchado, Simone Codeluppi, Peter Lönnerberg, Gioele La Manno, Anna Juréus, Sueli Marques, Hermany Munguba, Liqun He, Christer Betsholtz, et al. Cell types in the mouse cortex and hippocampus revealed by single-cell rna-seq. *Science*, 347(6226):1138–1142, 2015.

[59] Rapolas Zilionis, Camilla Engblom, Christina Pfirschke, Virginia Savova, David Zemmour, Hatice D Saatcioglu, Indira Krishnan, Giorgia Maroni, Claire V Meyerovitz, Clara M Kerwin, et al. Single-cell transcriptomics of human and mouse lung cancers reveals conserved myeloid populations across individuals and species. *Immunity*, 50(5):1317–1334, 2019.