
Supplementary information

Ultrafast and reference-free sequence discovery in single-cell data

In the format provided by the
authors and unedited

Supplementary Materials for "Ultrafast and reference-free sequence discovery in single cell data"

Daniel León-Periñán^{1,*}, Nikos Karaiskos^{1,*}, and Nikolaus Rajewsky^{1,2,3,4,5,6,7,8,9,*}

¹*Laboratory for Systems Biology of Regulatory Elements, Berlin Institute for Medical Systems Biology (BIMSB), Max-Delbrück-Centrum for Molecular Medicine in the Helmholtz Association (MDC), Hannoversche Str. 28, 10115 Berlin, Germany*

²*Charité - Universitätsmedizin, Charitéplatz 1, 10117 Berlin, Germany*

³*German Center for Cardiovascular Research (DZHK), Site Berlin, Berlin, Germany*

⁴*NeuroCure Cluster of Excellence, Berlin, Germany*

⁵*German Cancer Consortium (DKTK)*

⁶*National Center for Tumor Diseases (NCT), Site Berlin, Berlin, Germany*

⁷*ImmunoPreCept Cluster of Excellence, Berlin, Germany*

⁸*Einstein Center for Early Disease Interception, Berlin, Germany*

⁹*Lead contact*

** Correspondence: daniel.leonperinan@mdc-berlin.de; nikolaos.karaiskos@mdc-berlin.de; rajewsky@mdc-berlin.de*

Contents

A	Supplementary Methods	2
A.1	Overview	2
A.2	Index Construction	3
A.3	Querying	6
B	Supplementary Discussion	10
B.0.1	Discussion	10
B.1	Limitations	10
B.2	Challenges and Outlook	12
C	Supplementary Notes	13
C.1	Benchmarking existing k-mer indexing methods	13
C.2	Probe Design and Query Parameter Selection	14
C.3	Perspectives to improving the accuracy of k-mer methods	15
C.4	Parameter choices for sequence-based cell representations	16
D	Supplementary Figures	17
E	Supplementary Tables	23
	Supplementary References	30

Chapter A

Supplementary Methods

A.1 Overview

Malva implements an inverted index for the single-cell colored k -mer indexing problem. Given a collection of sequencing libraries $\mathcal{R} = \{R_1, \dots, R_S\}$ where each library R_i contains reads from n_i cells, the goal is to support queries that return, for any k -mer x , the set of cells $\text{Cells}(x) = \{(i, j) \mid x \text{ appears in cell } c_{ij}\}$. This problem differs from bulk-sample k -mer indexing in that the label space grows from $O(S)$ samples to $O(\sum_i n_i)$ cells, typically 10^6 to 10^{10} unique labels, rendering traditional color-aggregative approaches impractical.

Definitions

We define the following entities that are used and tracked throughout the indexing process:

- **Chunk:** A self-contained index unit comprising multiple samples. Each chunk is stored as an independent file and can be queried in parallel.
- **Sample:** A sequencing dataset (e.g., one 10x Chromium run) containing reads from multiple cells. Each sample has associated sample-level metadata (e.g., tissue type, donor ID, experimental condition).
- **Cell:** An individual cell within a sample, identified by its barcode. Each cell has associated cell-level metadata (e.g., cell type annotation, cluster assignment).

These entities are linked to an external metadata database via composite identifiers, enabling queries to return not only matching cells but also their biological annotations.

Data Structure

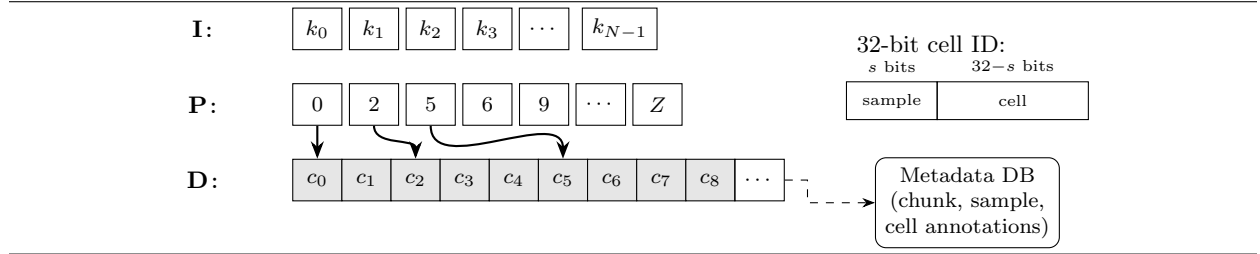
Each chunk contains an inverted index stored in compressed sparse row (CSR) format comprising three arrays (Supplementary Graph 1):

- **Sequence array I (indices):** N unique k -mers in lexicographically sorted order
- **Pointer array P (indptr):** For each k -mer at position i , $\mathbf{P}[i]$ stores the start offset into the location array; $\mathbf{P}[i + 1] - \mathbf{P}[i]$ gives the number of cells containing k -mer i
- **Location array D (data):** Z total cell identifiers (as 32-bit composites), grouped by k -mer

This structure supports queries in $O(\log N + L_x)$ time where $L_x = |\text{Cells}(x)|$.

This structure can be stored compactly by partitioning k -mers by their ℓ -base prefix (default $\ell = 12$, yielding 4^ℓ buckets). Within each bucket, suffixes and cell counts are delta-encoded as variable-length integers, and cell identifier lists are likewise delta-encoded per k -mer. A prefix lookup table maps each prefix value to the offset and size of its bucket on disk, realising the same logical CSR structure while enabling $O(1)$ first-level access and compact on-disk representation.

Graph 1 Index data structure. The three arrays form a CSR-format inverted index. Each cell identifier in the data array is a 32-bit composite encoding sample and cell IDs, see Section A.1. The chunk ID and all annotations (sample-level and cell-level metadata) are stored in an external database, accessed via the composite identifier.



Cell Identifier Encoding

Cell identifiers stored in the data array are 32-bit integers partitioned into two components: sample ID (upper s bits) and cell ID (lower $32-s$ bits). The partitioning determines the maximum number of samples per chunk and the number of cells per sample. For example, allocating 8 bits for sample ID and 24 bits for cell ID permits up to 256 samples per chunk with approximately 16 million cells each. Alternative allocations (e.g., 9 bits for sample, 23 bits for cell) allow 512 samples per chunk with approximately 8 million cells each. The choice depends on the expected cell counts per dataset. Chunks are kept relatively large (many samples) to improve query efficiency by reducing the number of separate index files that must be accessed. The chunk ID itself is stored as external metadata alongside the index file, not within the 32-bit cell identifier.

A.2 Index Construction

Construction proceeds in three phases: (1) streaming extraction of k -mers from FASTQ files into temporary chunks, (2) merging chunks within each sample, and (3) merging samples into multi-sample chunks.

K-mer Encoding and Sampling

Each k -mer is encoded as a 64-bit unsigned integer using the standard 2-bit nucleotide representation ($A \mapsto 00$, $C \mapsto 01$, $G \mapsto 10$, $T \mapsto 11$), supporting $k \leq 32$. From each read of length ℓ , we extract k -mers at non-overlapping positions $\{0, k, 2k, \dots\}$ plus a final k -mer at position $\ell - k$ to ensure complete coverage. The stride between consecutive k -mers is equal to k (i.e., non-overlapping). This sampling reduces storage by factor $\sim k$ while guaranteeing that for any query window of length $\geq 2k$, at least one indexed k -mer will be found. k -mers containing ambiguous nucleotides (N) or consisting entirely of a single nucleotide (homopolymers) are excluded from indexing. Chunk-level counts of k -mers can be computed from offsets. Indexing is associative, it does not preserve counts per label.

Barcode Handling

Cell barcodes are extracted from technology-specific read positions and validated against the manufacturer’s whitelist. Barcodes are encoded as 64-bit integers using the same 2-bit nucleotide representation as k -mers, allowing efficient storage and comparison. The barcode-to-cell mapping is maintained in an unsorted map data structure that associates each encoded barcode with its integer cell ID (or -1, if it does not exist) within the sample. This structure supports the $O(1)$ lookup during read processing. During the final merge phase, these local cell IDs are combined with the sample ID to form the 32-bit composite identifier described above.

Streaming Chunk Construction

FASTQ files are processed in a streaming fashion. For each read, the barcode is validated and k -mers are extracted. Pairs of (k -mer, cell-id) are accumulated in an in-memory buffer until reaching threshold T (default 10^8 pairs), at which point the buffer is flushed to disk as a temporary chunk via Algorithm 1.

Algorithm 1 Chunk Flush: Buffer to CSR Arrays

Require: Buffer B of (k -mer, cell-id) pairs

Ensure: Arrays $\mathbf{I}$, $\mathbf{P}$, $\mathbf{D}$ written to disk

```
1: Sort  $B$  lexicographically by ( $k$ -mer, cell-id)
2:  $\mathbf{I} \leftarrow []$ ;  $\mathbf{P} \leftarrow [0]$ ;  $\mathbf{D} \leftarrow []$  ▷ Initialize empty arrays
3: ( $\text{prev\_k}$ ,  $\text{prev\_c}$ )  $\leftarrow$  ( $\text{null}$ ,  $\text{null}$ )
4: for each  $(x, c)$  in  $B$  do
5:   if  $x \neq \text{prev\_k}$  then ▷ New  $k$ -mer
6:      $\mathbf{I}.\text{append}(x)$ 
7:      $\mathbf{P}.\text{append}(|\mathbf{D}|)$  ▷ Record current position in  $\mathbf{D}$ 
8:      $\text{prev\_c} \leftarrow \text{null}$ 
9:   end if
10:  if  $c \neq \text{prev\_c}$  then ▷ Deduplicate: presence/absence encoding
11:     $\mathbf{D}.\text{append}(c)$ 
12:     $\text{prev\_c} \leftarrow c$ 
13:  end if
14:   $\text{prev\_k} \leftarrow x$ 
15: end for
16:  $\mathbf{P}.\text{append}(|\mathbf{D}|)$  ▷ Final pointer
17: Compress  $\mathbf{I}$ ,  $\mathbf{P}$ ,  $\mathbf{D}$  blockwise and write to disk
```

Arrays are compressed to enable random access, for instance, with blockwise compression with a block offset table, or by flushing each chunk per prefix bucket. In the latter, pairs sharing the same ℓ -base prefix are grouped, their suffixes and cell counts are delta-encoded as variable-length integers, and cell identifiers are delta-encoded per k -mer. A prefix lookup table records the offset and size of each bucket. When temporary chunks are stored as pre-sorted streams of (k -mer, cell-id) pairs, the subsequent merge (Section A.2) simplifies to a standard k -way merge.

Chunk Merging

After all reads are processed, temporary chunks must be merged. The merge algorithm processes k -mers in lexicographic order using proportional windows from each chunk, combining cell sets

for shared k -mers. A windowed approach (Algorithm 2) bounds memory usage by processing proportional windows from each chunk. The window size is set to 5% of each chunk’s size (capped at 10^7 entries).

Algorithm 2 Windowed Multi-Chunk Merge

Require: Chunks $C_1, \dots, C_m$, each with arrays $(\mathbf{I}_i, \mathbf{P}_i, \mathbf{D}_i)$

Ensure: Merged index $(\mathbf{I}, \mathbf{P}, \mathbf{D})$ written to output

```

1:  $\text{ptr}[i] \leftarrow 0$  for all  $i$  ▷ Current position in each chunk
2:  $\text{win}[i] \leftarrow \min(0.05 \cdot |\mathbf{I}_i|, 10^7)$  for all  $i$  ▷ Window sizes
3: while any chunk has unprocessed entries do
4: ▷ Find maximum  $k$ -mer we can safely process this iteration
5:    $k_{\max} \leftarrow \infty$ 
6:   for each chunk  $C_i$  with  $\text{ptr}[i] < |\mathbf{I}_i|$  do
7:      $\text{end} \leftarrow \min(\text{ptr}[i] + \text{win}[i], |\mathbf{I}_i| - 1)$ 
8:      $k_{\max} \leftarrow \min(k_{\max}, \mathbf{I}_i[\text{end}])$ 
9:   end for
10:
11: ▷ Collect all entries up to  $k_{\max}$  from all chunks
12:    $E \leftarrow \emptyset$ 
13:   for each chunk  $C_i$  do
14:     while  $\text{ptr}[i] < |\mathbf{I}_i|$  and  $\mathbf{I}_i[\text{ptr}[i]] \leq k_{\max}$  do
15:        $x \leftarrow \mathbf{I}_i[\text{ptr}[i]]$ 
16:        $\text{cells} \leftarrow \mathbf{D}_i[\mathbf{P}_i[\text{ptr}[i]] : \mathbf{P}_i[\text{ptr}[i] + 1]]$ 
17:        $E \leftarrow E \cup \{(x, \text{cells})\}$ 
18:        $\text{ptr}[i] \leftarrow \text{ptr}[i] + 1$ 
19:     end while
20:   end for
21:
22: ▷ Merge entries sharing the same  $k$ -mer and write
23:   for each unique  $k$ -mer  $x$  in  $E$  do
24:      $\text{merged} \leftarrow \bigcup \{\text{cells} : (x, \text{cells}) \in E\}$ 
25:     Write  $(x, \text{sorted}(\text{merged}))$  to output arrays  $\mathbf{I}, \mathbf{P}, \mathbf{D}$ 
26:   end for
27: end while

```

The key insight is that by taking the *minimum* of the window-end k -mers across all chunks, we guarantee that all occurrences of any k -mer $\leq k_{\max}$ have been seen, enabling correct merging without loading entire chunks into memory.

When chunks are stored as pre-sorted streams of $(k\text{-mer}, \text{cell-id})$ pairs, the merge simplifies to a standard streaming k -way merge: at each step the globally smallest pair is selected across all chunk heads, pairs are accumulated by prefix, and each completed prefix bucket is flushed to disk. The windowed approach of Algorithm 2 remains applicable when merging indexed chunks (e.g., during multi-sample chunk construction).

When merging samples into multi-sample chunks, cell IDs are remapped to incorporate the sample offset within the chunk. The composite identifier then allows downstream queries to recover the chunk ID, sample ID, and cell ID, which serve as keys into the external metadata database containing sample-level annotations (tissue, donor, condition) and cell-level annotations (cell type, cluster).

Hierarchical Lookup

For indices where the sequence array exceeds available memory, we construct a hierarchical page structure. The sequence array can be partitioned into pages of P k -mers; a second-level array samples every P -th element; a third level samples every P -th element from the second; and so on until the root level contains fewer than P elements. During lookup, navigation proceeds from root to leaf: at each level, binary search identifies which child page contains the target k -mer, narrowing the search space by factor P per level. For a sequence array of N k -mers, lookup requires $O(\log_P N)$ page accesses, typically 3–4 for billion-scale indices. Realisations of this hierarchical lookup include B⁺-trees, hash-based sharding, and minimal perfect hash functions.

This can be implemented efficiently through prefix-based partitioning: each k -mer is split into an ℓ -base prefix and a $(k-\ell)$ -base suffix, and the prefix serves as a direct index into a lookup table of 4^ℓ entries (always resident in memory). This provides $O(1)$ first-level access, after which a binary search within the bucket’s sorted suffix array locates the target in $O(\log(N/4^\ell))$ comparisons, typically fewer than 20 for billion-scale indices with $\ell = 12$. The prefix table and suffix arrays are memory-mapped; cell data blocks are read on demand from disk.

A.3 Querying

At query time, all chunks are accessed in parallel. For each chunk, the query sequence is decomposed into k -mers and matched against the chunk’s index. Results from all chunks are aggregated, and the composite cell identifiers are used to retrieve metadata from the external database.

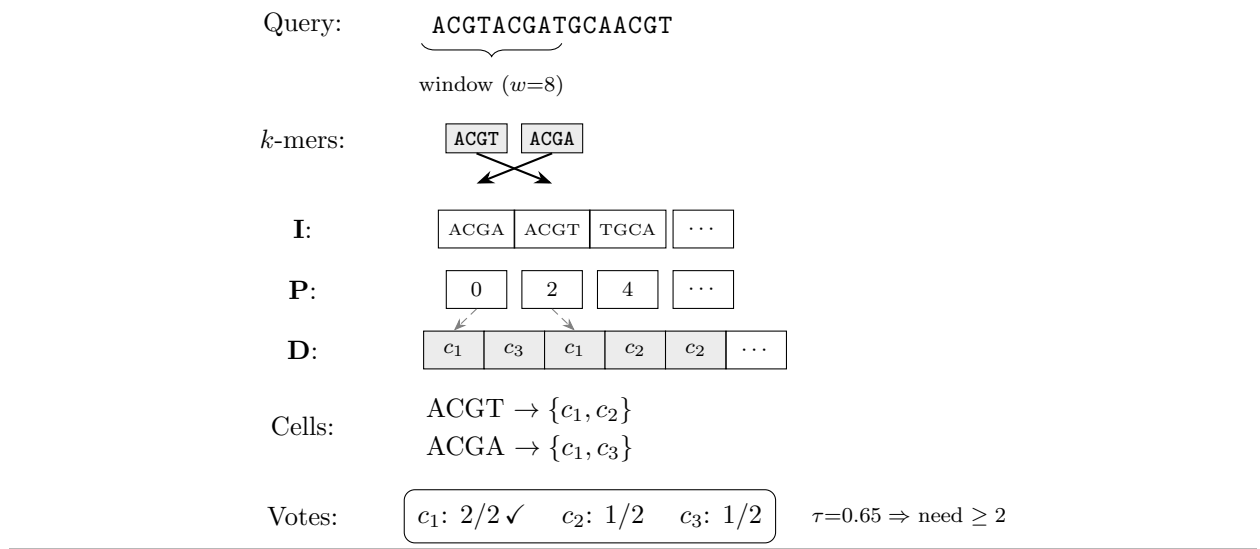
Window-Based Matching

The query algorithm (Algorithm 3 and Algorithm 4 and Supplementary Graph 2) evaluates sequence similarity using a windowed voting scheme. For a query sequence Q , we slide a window of size w (default 64 bp) with stride 1 (i.e., overlapping windows). Within each window, we extract non-overlapping k -mers at positions $\{0, k, 2k, \dots\}$ and look up their cell sets. A cell matches the window if at least fraction τ (default 0.65) of the window’s k -mers are found in that cell. The final score for cell c is the count of windows where c achieved a match. The use of overlapping windows with stride 1 ensures sensitivity to matches at any position within the query, while the non-overlapping k -mer extraction within each window controls computational cost.

Users may specify query-time filters on k -mer abundance. The parameter `count_at_least` (default 0) excludes k -mers appearing in fewer than this many cells, filtering likely sequencing errors. The parameter `count_at_most` (default 100,000) excludes k -mers appearing in more than this many cells in a chunk, filtering repetitive or ubiquitously expressed sequences that might dominate runtime and be uninformative. These filters are applied during the batch lookup phase (Algorithm 3 and 4); no abundance filtering is applied during index construction.

The vote count for each cell is capped at n_k (the number of k -mers per window) to prevent repeated k -mer values within a window from inflating scores. A locality decay mechanism (line 31) penalizes cells whose k -mer hits are scattered across the window rather than contiguous: if a cell’s most recent contributing k -mer is not the current one, its vote count is decremented by one. In practice, with typical parameters ($w = 128$, $k = 24$, $\tau = 0.65$), this has a minor effect since most windows contain only $\lfloor w/k \rfloor = 5$ k -mers.

Graph 2 Query example with $k=4$, $w=8$. The query window yields two k -mers (ACGT, ACGA). Each is located in **I** via binary search (or hashing, or hierarchical lookup); pointers in **P** give the cell range in **D**. Cell c_1 contains both k -mers ($2/2 \geq \tau$) and receives one window vote. Cells c_2 and c_3 contain only one k -mer each ($1/2 < \tau$) and receive no vote.



Query Batching and Index Access Pattern

For efficiency, all k -mers across all query windows are collected, deduplicated, and sorted lexicographically before lookup. Because the sequence array is sorted, this ensures sequential access through the index, maximizing cache locality and minimizing disk seeks.

The access pattern proceeds as follows:

1. Collect all unique k -mers from all query windows
2. Sort k -mers lexicographically
3. For each k -mer in sorted order:
 - (a) Locate the k -mer in the sequence array
 - (b) Read pointer entries to determine cell range
 - (c) Read corresponding cell identifier list
4. Redistribute results to originating windows for scoring

For indices exceeding available memory, compressed array blocks are accessed via memory-mapped file I/O. The operating system's virtual memory subsystem handles paging of index blocks on demand, avoiding explicit memory management while maintaining efficient access patterns. In the server implementation, decompressed blocks are retained in an LRU cache (capacity 2×10^6 blocks) to avoid repeated decoding. With sequential access patterns, empirical cache hit rates exceed 90%.

Metadata Integration

Query results return composite cell identifiers. These are decomposed into (chunk ID, sample ID, cell ID) tuples, which serve as keys into a pre-indexed external database. This database stores:

Algorithm 3 Window-Based Query Scoring (Phase 1)

Require: Query sequence Q , index $\mathcal{I}$, window size w , k -mer size k , threshold τ

Require: Abundance filters: $f_{\min}$ (`count_at_least`), $f_{\max}$ (`count_at_most`)

Ensure: Map from cell ID to pseudocount score

```
1:
2:                                     ▷ Phase 1: Batch  $k$ -mer lookup (all windows at once)
3:  $K_{\text{all}} \leftarrow \emptyset$ 
4: for  $p \leftarrow 0$  to  $|Q| - w$  step 1 do
5:    $W \leftarrow Q[p : p + w]$ 
6:   for  $i \leftarrow 0$  to  $\lfloor w/k \rfloor - 1$  do
7:      $x \leftarrow W[i \cdot k : (i + 1) \cdot k]$ 
8:     if  $x$  contains no N then  $K_{\text{all}} \leftarrow K_{\text{all}} \cup \{x\}$ 
9:     end if
10:  end for
11: end for
12:  $\mathbf{R} \leftarrow \text{BATCHLOOKUP}(\mathcal{I}, K_{\text{all}}, f_{\min}, f_{\max})$            ▷  $k$ -mer  $\rightarrow$  cell set, filtered by abundance
```

- **Sample-level metadata:** for example, tissue type, donor ID, experimental condition, sequencing platform, publication source
- **Cell-level metadata:** for example, cell type annotation, cluster assignment, quality metrics

The separation of sequence index from metadata database allows efficient updates to annotations without rebuilding the k -mer index.

Supplementary Table 3 summarizes key parameters and their default values.

Algorithm 4 Window-Based Query Scoring (Phase 2)

Require: Query sequence Q , index $\mathcal{I}$, window size w , k -mer size k , threshold τ

Require: Abundance filters: $f_{\min}$ (`count_at_least`), $f_{\max}$ (`count_at_most`)

Ensure: Map from cell ID to pseudocount score

```
1:                                     ▷ Phase 2: Per-window scoring
2: scores  $\leftarrow \{\}$                                      ▷ Cell  $\rightarrow$  count of matching windows
3:  $n_k \leftarrow \lfloor w/k \rfloor$                                      ▷ Number of  $k$ -mers per window
4: for each unique window  $W$  in  $\{Q[p:p+w] : p = 0, \dots, |Q|-w\}$  do
5:    $K \leftarrow (x_0, x_1, \dots, x_{n_k-1})$                  ▷ Non-overlapping  $k$ -mers from  $W$ , in order
6:   votes  $\leftarrow \{\}$ ; last  $\leftarrow \{\}$                  ▷ Per-cell vote count and last  $k$ -mer index
7:   for  $j \leftarrow 0$  to  $|K| - 1$  do
8:     if  $K[j] \notin \mathbf{R}$  then continue
9:     end if
10:    for each  $c \in \mathbf{R}[K[j]]$  do
11:      votes $[c] \leftarrow \min(\text{votes}[c] + 1, n_k)$            ▷ Cap at  $n_k$ 
12:      last $[c] \leftarrow j$ 
13:    end for
14:    if  $j + 1 < n_k$  then continue
15:    end if                                     ▷ Defer scoring until window complete
16:                                     ▷ Score cells and apply locality decay
17:    for each cell  $c$  with votes $[c] > 0$  do
18:      if votes $[c] \geq \lceil \tau \cdot n_k \rceil$  then
19:        scores $[c] \leftarrow \text{scores}[c] + 1$ 
20:      end if
21:      if last $[c] < j$  and votes $[c] > 0$  then
22:        votes $[c] \leftarrow \text{votes}[c] - 1$                  ▷ Decay stale votes
23:      end if
24:    end for
25:  end for
26:  Reset votes, last for cells touched in this window
27: end for
28: return scores
```

Chapter B

Supplementary Discussion

B.0.1 Discussion

RNA diversity shapes cell identity, yet much of this information has remained buried in single-cell sequence archives. Malva makes sequences directly searchable across millions of cells, shifting analysis from predefined gene tables to the underlying sequence space. This enables queries that were previously impractical, and provides a foundation for large-scale re-analysis of public data. Malva is designed as a complementary discovery layer. It does not replace state-of-the-art reference-based methods, but extends them by enabling sequence-level searches that would otherwise require downloading and reprocessing petabytes of raw data. The motivation for sequence-level search is to make single-cell analysis less dependent on predefined references, annotations and preprocessing choices. By querying nucleotide sequences directly, Malva can recover splice junctions, unannotated transcripts, variants, synthetic constructs and other sequence features that may be missed by gene-centric workflows. This changes not only what can be searched, but also how public single-cell data can be used. Instead of treating archives as static processed matrices, Malva makes them available as a live source of molecular evidence. This is particularly relevant for AI-assisted analysis. In many current settings, neural networks learn from biological data during training but cannot easily return to the underlying experiments when making an inference. Malva provides a route to connect such models back to observed sequences in real cells. In this sense, it follows the logic of neurosymbolic and retrieval-augmented systems, where flexible statistical models are coupled to explicit external knowledge or data sources [4, 8]. For single-cell biology, this means that models and agents could formulate sequence queries, retrieve cellular evidence, and refine their analysis against public data. Malva could therefore support both interactive exploration by researchers and automated workflows in which machine reasoning remains grounded in experimentally observed molecules.

B.1 Limitations

Under our current implementation, sequence queries with fewer than 24 nucleotides are not supported. The algorithm behind Malva supports any k-mer size, but we chose 24 as it’s within a commonly used range (21-31). Furthermore, like any exact k-mer system, Malva prioritizes specificity, which can reduce sensitivity when sequences differ due to errors, RNA editing, or unannotated variants. Alternatively, seed constructs such as *syncmers* and *strobemers* can improve robustness to mutations, e.g., weighted minimizers can reduce spurious hits in repetitive regions [3, 11]; graph-aware representations, including compacted or colored de Bruijn graphs and variation graphs can capture allelic paths more comprehensively and reduce reference bias [6, 7, 5]. However, these approaches were originally developed for datasets with a modest number of samples, each with many

reads. They would need to be reformulated to be practical for SC/ST data, where an index spans up to billions of labels, each with barely thousands of observations.

There are also data-related limitations. For example, the Malva Index consists mainly of droplet-based data (e.g. 10x Chromium), which are strongly 3 prime-biased, constraining recovery of full-length isoforms. Inclusion of more samples from full-length technologies (e.g. Smart-seq2) will further expand the utility of sequence search, as demonstrated with the *Tabula muris* example (Main Text, **Fig. 3c**).

Regarding quantification, Malva currently reports pseudocounts rather than absolute molecule counts. Specifically, Malva is not UMI-aware, and operates directly on k-mer occurrences. Our saturation-based normalization (Main Text, **Extended Data Fig. 4**) addresses systematic offsets but does not account for gene- or sequence-specific amplification arising from, e.g., non-uniform PCR efficiency. When such biases are a concern, users may consider Malva’s binary presence/absence mode, an approach that has proven effective for certain single-cell analyses [10]. More suitable solutions for quantitative accuracy shall be explored (**Sup. Note C.3**). Additionally, during preprocessing, Malva does not perform barcode error correction, as this adds complexity with marginal read recovery at atlas scale [9]. Finally, Malva does not explicitly remove doublets: these methods are numerous, case-specific, and would require substantial manual curation across thousands of studies. Notably, established platforms such as CELLxGENE (to this date) also do not perform doublet filtering [1]. As with *pseudocounts*, users seeking definitive cell-type assignments should apply specific downstream analyses to the retrieved data. Overall, these design choices reflect Malva’s role as a discovery tool: it localizes cell barcodes containing specific sequences, enabling users to apply rigorous downstream filtering and analysis appropriate to their goals.

In building Malva Index, we faced major challenges in public data accessibility and standardization. For instance, Malva currently ingests data from protocols that provide cell barcode whitelists (e.g., 10x Chromium). Although these technologies account for the vast majority of publicly available data, we plan to extend support to additional technologies by integrating preprocessing tools [13, 12, 2] to convert their barcode designs into a format compatible with Malva’s indexing pipeline. Moreover, ST data are particularly problematic: while sequencing reads are deposited in standard formats, the spatial barcode-to-coordinate mappings are often missing, stored in separate databases, or provided in exotic formats. This fragmentation makes systematic processing of spatial data (in particular, high-resolution) nearly impossible, calling for new standards for data depositing.

We also note potential concerns inherent to storing and querying human sequencing data. By querying germline variants from population genetic studies, we found strong correlations between Malva results and known population frequencies, suggesting that individual genotypes could be inferred from transcriptomic data. This risk has been noted broadly on raw bulk genome and transcriptome sequence data, and more recently even in aggregated gene count matrices [14]. Therefore, querying strategies should address these concerns similarly to how current practices address the storage of raw sequences. The ST datasets included in Malva Index stem from sequencing-based technologies. Given the growing popularity of imaging-based ST methods (e.g. 10X Xenium, Nanostring CosMx Spatial Molecular Imager, Vizgen MERSCOPE), it is compelling to ingest this data modality too. Where sequences of capture probes are available, imaging based counts could be converted to sequence counts, with the resulting sequence space being, however, limited to that of the probe sequence space.

B.2 Challenges and Outlook

Malva holds the potential to extend beyond transcriptomics while preserving its modularity and scalability. We envision that any assay generating sequence-barcoded information could be ingested: scATAC-seq for chromatin accessibility; immune-repertoire profiling (TCR/BCR); DNA sequencing of somatic variation; and any other multi-modal measurements. In proteomics, DNA-barcoded antibodies (spatial and single-cell) can be indexed, and peptide sequences from mass spectrometry can be treated as queryable strings across samples. Similarly, RNA modifications (e.g., m6A, pseudouridine) detected by direct RNA sequencing produce sequence-anchored signals that can be indexed. Extending Malva to these modalities would require addressing challenges in encoding, ambiguity handling, and downstream resolution; this is a non-trivial but promising line for future work. This also extends to long-read single-cell sequencing. Malva has no fundamental restriction on read length during indexing; the index construction treats reads as sequences of k-mers regardless of length. For long-read protocols based on 10x or similar barcoding (PacBio, ONT), the primary limitation is the higher per-base error rate, which reduces the fraction of R1 reads yielding exact barcode matches. We recommend addressing this with a barcode correction preprocessing step, as is standard practice for such data. During query, the windowed voting parameters (in particular, sliding window size and match threshold) can be adjusted to accommodate elevated error rates (**Sup. Note C.2**). As datasets expand in modalities and biological diversity, such an unifying, modular search architecture will become essential.

Chapter C

Supplementary Notes

C.1 Benchmarking existing k-mer indexing methods

K-mer search tools differ primarily in how they store the association between k-mers and their labels. Color-table designs store k-mer-to-label associations as compressed structures whose width depends on S (total labels): for example, MetaGraph uses an $N \times S$ annotation matrix compressed via RowDiff/Multi-BRWT; Mantis maps k-mers to deduplicated color classes represented as S -wide bit-vectors compressed via MST encoding. REINDEER stores an $m \times S$ count-vector matrix where querying a k-mer retrieves the full S -dimensional count vector regardless of sparsity. Inverted-index designs (Fulgor, Themisto, and Malva) store per-k-mer or per-unitig posting lists containing only the labels in which each k-mer appears, with query cost proportional to the posting list length L_k . Fulgor exploits unitig monochromaticity in the compacted de Bruijn graph to store one posting list per unitig rather than per k-mer, and further compresses by deduplicating shared label sets, which is highly effective when many unitigs have identical label sets (the typical case in bulk collections). Themisto uses a different color encoding but similarly stores compressed label sets per k-mer. Approximate membership structures (SBT, BIGSI, COBS) maintain one Bloom filter per label, with storage and query cost that grow directly with the number of labels.

In bulk collections (S in the hundreds to thousands), k-mer sharing across samples is high, annotation matrices are dense, and compression strategies exploiting color-class redundancy and unitig monochromaticity provide large gains. In single-cell data (S in the tens of millions), expression is sparse: a typical k-mer appears in a few hundred cells out of millions, color classes are rarely shared, and the annotation matrix is extremely sparse. In this regime, elaborate compression provides diminishing returns while S -dependent construction and storage costs become dominant, as reflected in our empirical comparison (**Sup. Table 1**).

We validated these claims empirically, by benchmarking current k-mer indexing approaches (MetaGraph, Fulgor, REINDEER, and BLAST for completeness) when applied to single-cell and spatial transcriptomics data, in comparison to Malva (**Extended Data Fig. 2**). Dataset selection and benchmarking details are described in Methods.

Across sequencing depths, all methods had longer indexing times compared to Malva (between 3 to 25-fold increase). All k-mer methods compressed the 24GB raw sequences (final index sizes ranging 3.1-19GB), with Malva producing the most compact representation (3.1GB).

We benchmarked query time (1 kb to 1 Mb of query sequences) using a Visium mouse brain dataset (Main Text, **Extended Data Fig. 2d**). When operating in-memory, all methods yielded comparable throughput (5-20 seconds for 1 Mb queries), but widely different memory requirements ($\sim$ 3-6GB for Malva/Fulgor vs $\sim$ 50GB for MetaGraph). As a hypothetical, in-memory atlas-scale

index would require very high memory capacity, we focused on disk-based performance. Malva’s disk-based architecture trades query latency (~ 60 s on-disk vs 12 s in-memory) for reduced memory usage (~ 400 MB vs ~ 5 GB). We note that while REINDEER supports on-disk querying, its indexing time and the need to split reads coming from different cell barcodes into different files rendered it impractical at very large single-cell-atlas scale; thus, we excluded it from the query time benchmark. Additionally, Malva generates (by default) coverage-like profiles that other methods can generate only with substantial latency penalties. Malva’s index merging capability allows incorporating new datasets using $<10\%$ of the CPU time and $<2\%$ of the memory required for complete reindexing, which becomes essential for maintaining current databases as repositories grow. Other methods either lack straightforward merging or require partial reconstruction of the graph and annotation when adding samples, since new data can change unitig boundaries and color-class assignments.

C.2 Probe Design and Query Parameter Selection

Malva’s sequence-based query performance depends on two user-specified parameters: the sliding window size (w) and the match threshold (τ). Because Malva indexes non-overlapping k -mers at positions $\{0, k, 2k, \dots, |S| - k\}$ rather than all possible k -mers over a string S , careful parameter selection is essential to ensure that queries capture the intended biological signal. We offer general guidelines in **Sup. Table 2**, that we further explain below:

For a window of size w with k -mer length $k = 24$, the number of indexed k -mers per window is $n_k = \lfloor w/k \rfloor$. A cell matches a window if at least $\lceil \tau \cdot n_k \rceil$ of these k -mers are present. This creates a fundamental constraint: queries shorter than $2k = 48$ nucleotides contain at most one indexed k -mer, reducing tolerance for sequencing errors or sampling effects.

For transcript-level quantification, the goal is to detect reads originating from anywhere along the transcript body while tolerating sequencing errors and biological variation such as SNPs or RNA editing. We recommend window sizes of $64 \leq w < 120$ nucleotides with $0.5 < \tau \leq 0.65$. These window sizes approximate typical read lengths in short-read single-cell RNA sequencing experiments. A window of $w = 64$ contains two-three indexed k -mers, while $w = 96$ contains four. With $\tau = 0.5$, a window matches if at least half of its k -mers are found, tolerating situations where one k -mer spans a sequencing error. For typical Illumina error rates ($\sim 0.1\%$), the probability of one or more errors in a 64 bp window is low, making this threshold a robust default.

Detecting specific splice junctions requires special consideration because the biological signal is localized to a short region centered on the junction. The non-overlapping index structure can cause the junction itself to fall between indexed positions. Consider a 48 bp probe centered on an exon-exon junction: if queried with $w = 48$, the two indexed k -mers fall entirely within the flanking exons, and neither spans the junction itself. The probe would then detect cells expressing either exon rather than specifically the junction. The solution is to use a probe size smaller than $2k$, and window size $w = k$, forcing that one k -mer spanning the junction will be found. Thus, for junction, variant and mutation detection, we recommend probes of $|S| < 2k$ (e.g., $|S| < 48$) nucleotides centered on the junction, with $w = 24$ and $\tau = 1.0$. Thus, each window contains only one indexed k -mer, and as the window slides, some positions will place this k -mer directly over the junction.

Note: Index construction is agnostic to read length, therefore analysis of long-read-based is straightforward, but requires some special considerations. Primarily, long-read data usually has a higher base-calling error rate (typically 1 to 5% for older ONT/PacBio chemistries/basecalling). This affects both barcode matching during indexing and k -mer recovery during query. For indexing, we recommend applying a barcode error correction tool prior to running Malva, so that barcodes

with single-nucleotide errors are rescued rather than discarded. For querying, the sliding window size w should be set to approximately the expected read length or a representative fragment thereof, and the match threshold should be lowered to account for the expected error rate. For example, with an error rate of e per base, the probability that a k -mer of length k is error-free is approximately $(1 - e)^k$; at $e = 0.01$ and $k = 24$ this is approximately 0.79, so roughly 20% of k -mers will be affected. Setting $\tau = 0.5$ with $w = 120$ (yielding $n_k = 5$ k -mers per window) tolerates up to two corrupted k -mers per window, which is sufficient for error rates up to approximately 5%. At higher error rates, users may consider reducing k or increasing w to maintain sensitivity.

We systematically evaluated across k -mer sizes (8–24bp) and probe lengths (8–44 bp) to confirm that default indexing and querying parameters ($k = 24$, probe length $\approx 2 \cdot k$ bp) lie within the optimal detection window (**Sup. Fig. 2**). However, these might be adjusted by query type. For merely “detection” tasks, precision increases with both k -mer size and probe length, as longer k -mers reduce spurious matches. For “discriminative” tasks, i.e., to distinguish between different versions of a transcript where a mutant allele must be distinguished from its wild-type counterpart, precision can decrease at longer probe lengths with smaller k -mer sizes. This occurs because longer probes may contain k -mers that no longer span the variant position, allowing detection of the wild-type sequence. When the wild-type is abundantly expressed (e.g., mitochondrial transcripts), this inflates false positives. For such tasks, probe length should remain short enough that at least one k -mer spans the variant (i.e., probe length $< 2k$ centered on the variant), or larger k -mer sizes should be used.

In summary, for a window with n_k indexed k -mers and threshold τ , the maximum number of tolerated errors equals $n_k - \lceil \tau n_k \rceil$. At $w = 64$ ($n_k = 3$) with $\tau = 0.5$, one error is tolerated; at $w = 96$ ($n_k = 4$) with $\tau = 0.65$, one error is tolerated; with $\tau = 1.0$, no errors are tolerated regardless of window size. Additionally, users should also consider that most datasets in Malva Index derive from 3'-biased protocols, so probes targeting 5' regions of transcripts will have reduced sensitivity due to lower coverage in the underlying data.

C.3 Perspectives to improving the accuracy of k -mer methods

Malva’s current implementation reports pseudocounts rather than UMI-corrected molecule counts, representing a fundamental limitation for precise quantitative analysis. UMI information could theoretically be incorporated by indexing composite cell-UMI labels, but this would expand the label space from millions of cells to billions of combinations, creating substantial challenges for index merging and storage. Our saturation-based correction for PCR bias (Main Text, **Extended Data Fig. 4**) provides a preliminary solution to pseudocount inflation, yet more sophisticated normalization approaches could account for sequence-specific biases and technology-dependent artifacts that affect different transcript regions differently.

The current implementation employs masking of repetitive k -mers (via Dustmasker) to minimize false positives, inevitably reducing sensitivity for sequences containing such elements. Future, more advanced approaches could dynamically adjust k -mer weights based on genomic frequency or employ context-dependent scoring schemes. Similarly, our sliding window approach with fixed thresholds (Main Text, **Extended Data Fig. 2c**) represents a conservative strategy; adaptive windowing or probabilistic scoring models could better balance sensitivity and specificity across diverse sequence types while maintaining computational efficiency.

C.4 Parameter choices for sequence-based cell representations

The reference-free analysis relies on a bucketing strategy that groups k-mers based on their w-mer ($w < k$) composition upon MinHash signatures. We note a relationship between w-mer length and bucket size with the preservation of cellular relationships (Main Text, **Extended Data Fig. 10c**).

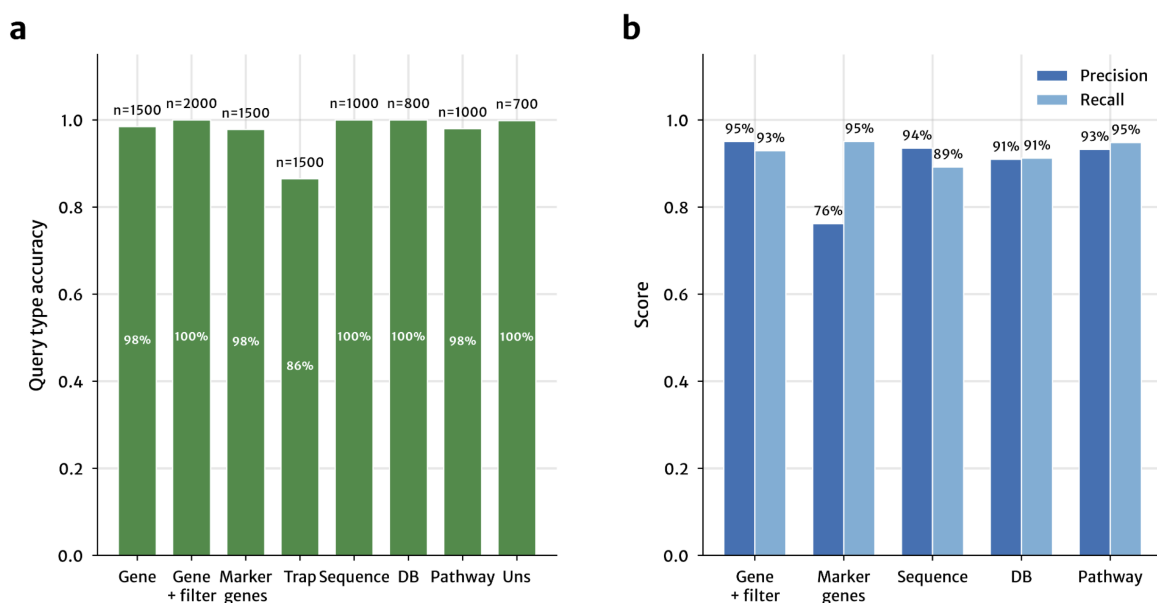
We measured the Latent Space Agreement (LSA) metric indicating local structure preservation on the human tumor Open-ST dataset (Methods) as it contains very diverse transcriptomic states and microbial signal. LSA dropped significantly for w-mer sizes $w < 5$ across all bucket sizes. We hypothesize that most sequences shorter than the 5 bp threshold lack sufficient specificity and therefore lack *biological information* (at least for the purpose of distinguishing species or cell types/states). For instance, most transcription factor binding sites require at least ~ 6 bp, and the genetic code itself operates on triplet units. Below this threshold, w-mers become dominated by low-complexity patterns (e.g., AT-rich or GC-rich stretches) that reflect compositional bias rather than functional sequence relationships.

We observed the highest LSAs around $w=7-9$. Indeed, 7-9 nucleotide patterns are known to capture meaningful biological signals including splice site recognition sequences (typically 6-9 nucleotides), ribosome binding sites, and short regulatory motifs. The information content of k-mers around this length are sufficiently complex to distinguish functional from random sequences, as demonstrated by other methods that, e.g., distinguish coding from noncoding sequences via hexamer usage bias [15].

The agreement scores plateau beyond $w=12$, reflecting diminishing returns as longer w-mers become increasingly sparse. While 16-18 nucleotide patterns maintain high LSAs, they offer no practical advantage over shorter w-mers and require larger bucket numbers to avoid information loss due to uniform collapse of the hash function.

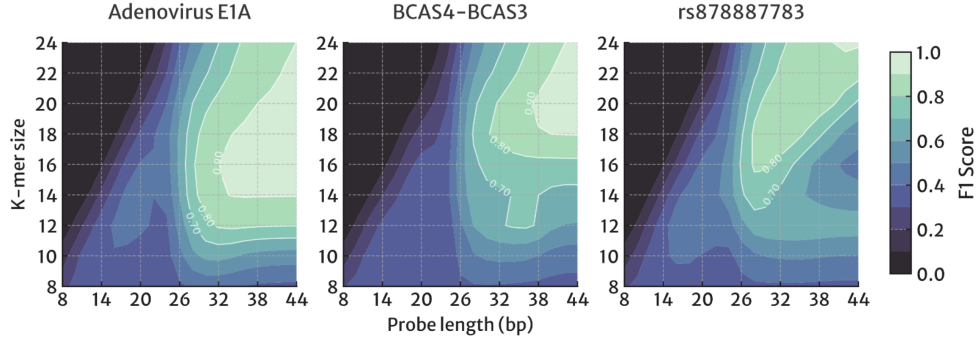
Chapter D

Supplementary Figures

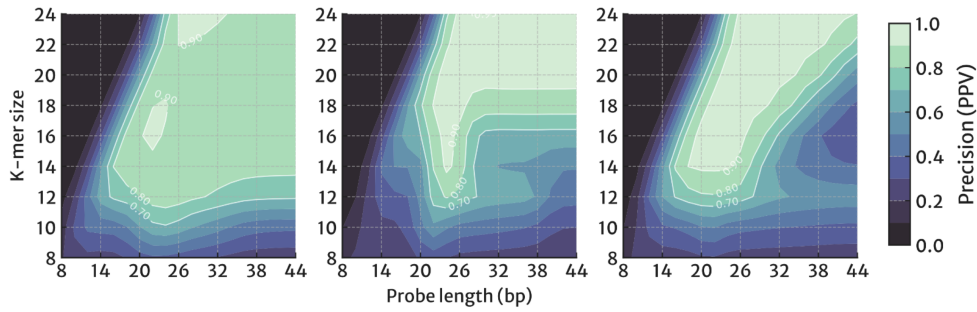


Supplementary Figure 1: **Validation of natural language query interface accuracy.** (a) Query type classification accuracy across eight categories (n per category shown above bars). The model achieves high accuracy in routing queries to the correct processing pathway, with lowest performance on over-inference trap queries designed to test inappropriate filter addition. (b) Feature extraction performance for query types requiring entity recognition. Precision and recall are shown for gene name extraction, marker gene identification, sequence parsing, database identifier resolution, and pathway mapping. The model maintains high precision across categories, critical for avoiding false constraints on search results. Uns: unsupported queries correctly rejected.

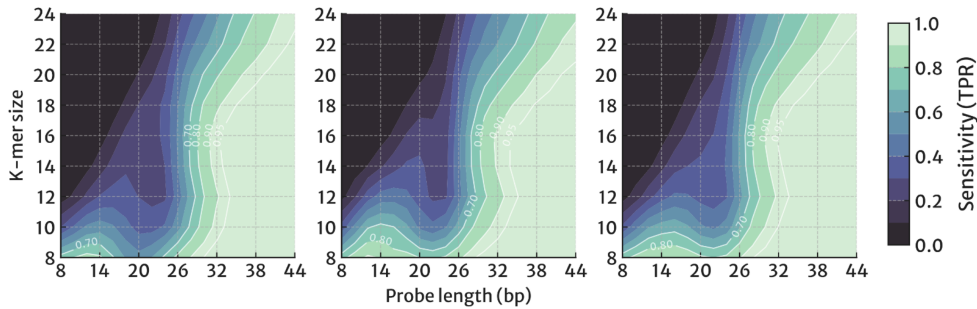
a Optimality analysis for F1-scores



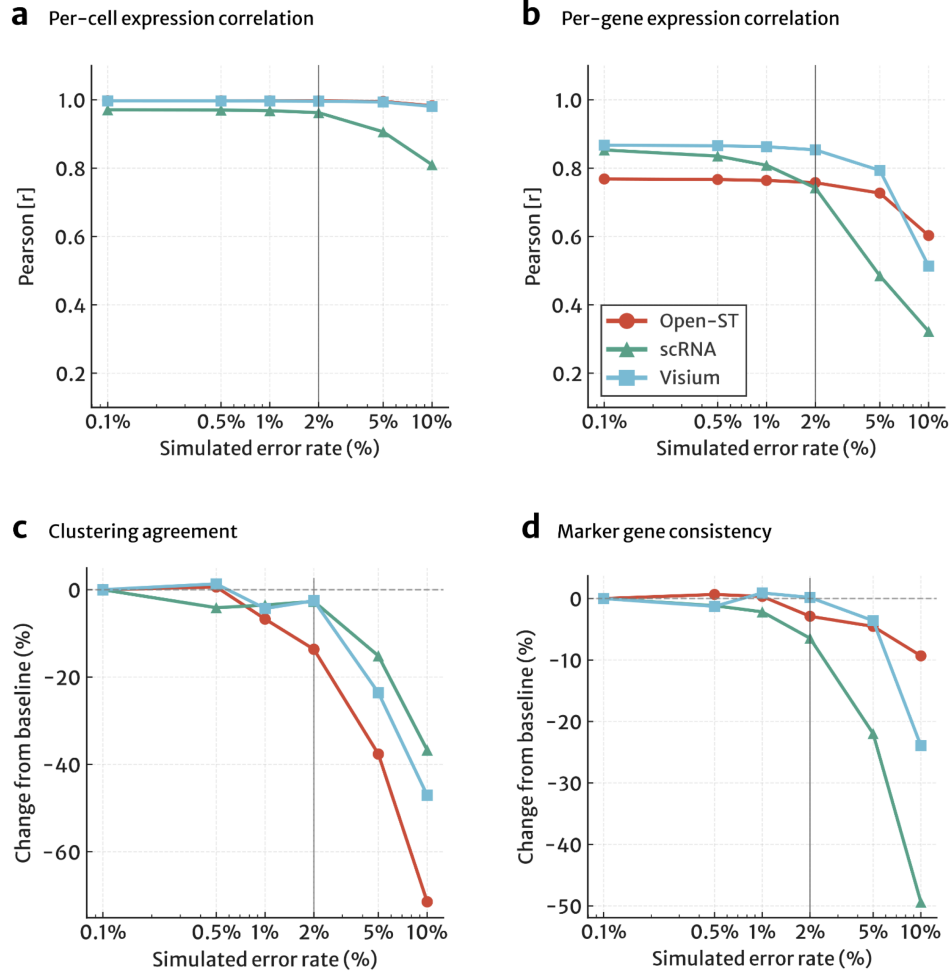
b Optimality analysis for Precision



c Optimality analysis for Sensitivity



Supplementary Figure 2: **Optimality analysis for k-mer size and probe length selection.** Contour plots showing (a) F1 score, (b) precision (Positive Predictive Value; PPV), and (c) sensitivity (True Positive Rate; TPR) as a function of k-mer size and probe length across the benchmark settings from Main Text **Extended Data Fig. 5**. Metrics were computed using Flexiplex detections as ground truth at 0% simulated error rate. White contour lines indicate iso-performance thresholds. F1 scores plateau at high values for k-mer sizes ≥ 18 and probe lengths ≥ 32 bp, supporting our default parameter choice of $k = 24$ and maximum probe length >44 bp for reliable SNP detection.

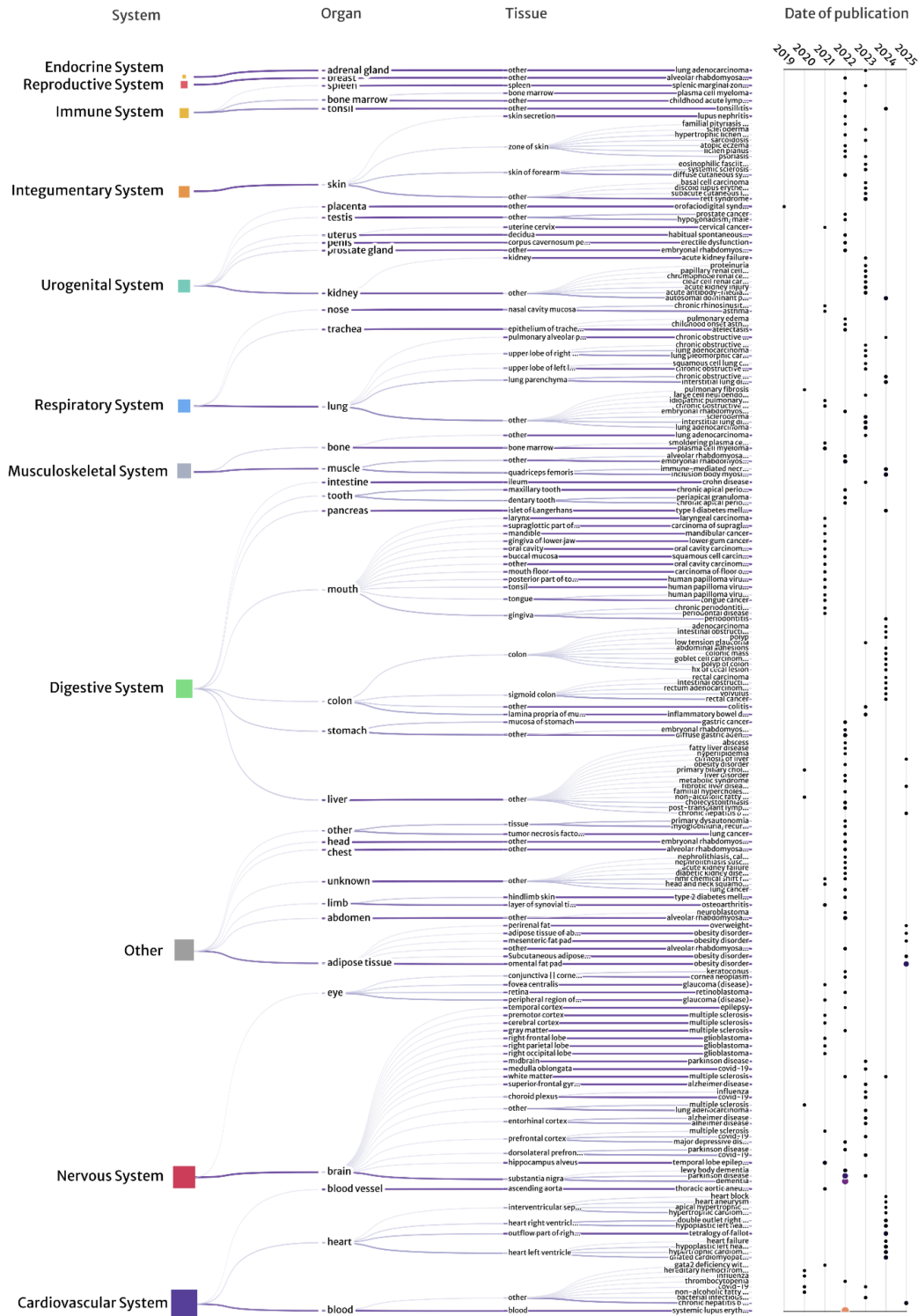


Supplementary Figure 3: **Robustness of Malva pseudoquantification to simulated sequencing errors.** (a) Per-cell Pearson correlation between Malva pseudocounts and reference counts (UMI) across simulated error rates ranging from 0.1% to 10%. (b) Per-gene Pearson correlation across the same error rates. (c) Clustering agreement measured by percent change in Normalized Mutual Information (NMI, with respect to 0% error) between Leiden clusters derived from Malva pseudocounts versus reference counts. (d) Marker gene consistency showing the percentage of differentially expressed genes with concordant fold-change direction ($|\log_2 FC| > 0.5$) between Malva and reference quantifications.

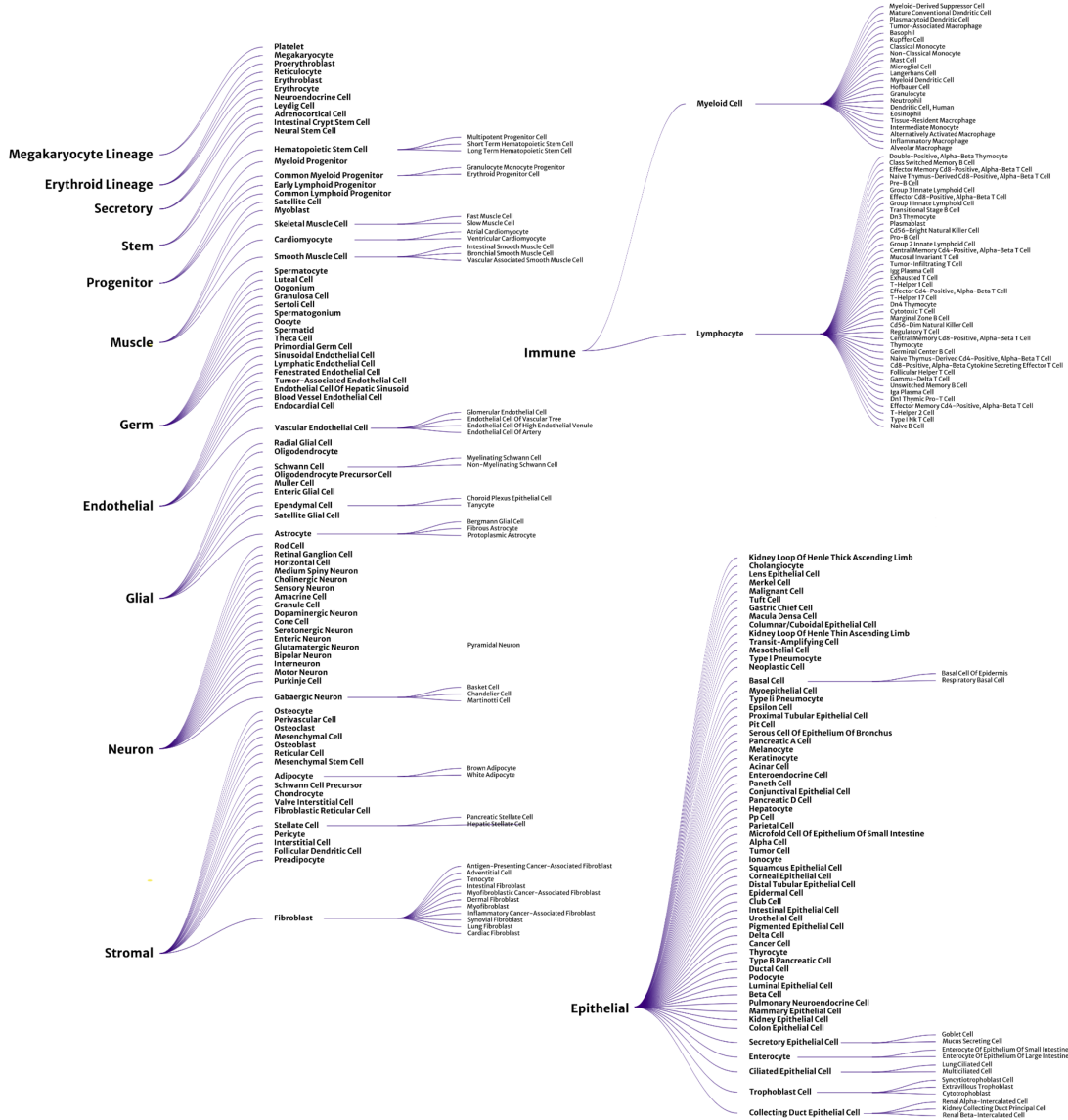
a Ontology of healthy human samples in Malva Index



b Ontology of diseased human samples in Malva Index



C Simplified tree of human cell types in Malva Index



Supplementary Figure 4: **Ontology of samples and cell types in Malva Index.** (a) Ontology of systems/organs/tissues in health and in disease (b); (c) Simplified tree of cell types. Complete database statistics are available online at <https://malva.mdc-berlin.de>.

Chapter E

Supplementary Tables

Supplementary Table 1: **Performance benchmark across methods and datasets.**

Dataset	Method	Peak Memory (GB)	CPU time (h)	Index Size (GB)	Query Time (s/Mb)	Query Memory (GB)
Visium	Malva	2.5	0.3	6.9	20	0.02
	Fulgor	8.2	1.2	5.6	5	5.6
	MetaGraph	14.1	5.6	13.4	3	49.8
1% HCA (~1.5TB)	Malva	161.0	98.7	109.0	133	0.3
	Fulgor	427.2	428.4	326.1	51	326.1

Supplementary Table 2: **Recommended query parameters for common use cases.** For each use case, we provide guidance on probe design and parameter selection. Window size (w) determines how many indexed k-mers are evaluated per sliding window; threshold (τ) specifies the fraction of k-mers that must match for a window to count as positive. All recommendations assume $k = 24$. See **Sup. Note C.2** for more details.

Use case	Probe design	Window size (w)	Threshold (τ)
Transcript/gene expression	ex- Full transcript or gene body	64-120	0.5-0.65
Splice junction (also for circRNA)	35-45 bp centered on junction	24	1.0
SNV detection	35-45 bp centered on mutation	24	1.0
3'UTR or other iso-form analysis	Region of interest	64-96	0.5-0.65

Supplementary Table 3: Default parameter values for index construction and querying.

Parameter	Symbol	Default	Description
<i>Indexing</i>			
K-mer length	k	24	Nucleotides per k -mer
Prefix length	ℓ	12	Bases for prefix-based partitioning
Buffer threshold	T	10^8	Pairs before chunk flush
Merge window	-	5%	Fraction of chunk per iteration
Window cap	-	10^7	Maximum entries per merge window
Sample ID bits	s	8	Bits for sample within chunk
<i>Querying</i>			
Query window	w	64	Nucleotides per scoring window
Window stride	-	1	Overlapping windows
Match threshold	τ	0.65	Fraction of k -mers required
Min abundance	$f_{\min}$	0	<code>count_at_least</code> filter
Max abundance	$f_{\max}$	100000	<code>count_at_most</code> filter
Cache capacity	-	2×10^6	Blocks in LRU cache

Supplementary Table 4: Software, reference resources, and data services used in this study.

Category	Tool or resource	Version	Use in this study	Reported in
K-mer indexing and querying	Malva	v1.0.0	K-mer indexing and sequence-search platform used throughout the study.	Code Availability
API client	malva_client	v0.3.4	Performs queries against the Malva index.	Code Availability
Web backend	Flask	v3.1.0	Provides the Malva Explorer backend and REST API.	Methods: The Malva Platform
Scientific computing	NumPy	v2.2.6	Supports numerical operations and array processing.	Methods: The Malva Platform
Data processing	pandas	v2.3.2	Supports dataframe processing and local metadata storage in the Malva Explorer backend.	Methods: The Malva Platform
Data processing	Polars	v1.27.1	Supports dataframe processing and local metadata storage in the Malva Explorer backend.	Methods: The Malva Platform
Database management	SQLite	v3.49.1	Provides local metadata storage for the Malva Explorer backend.	Methods: The Malva Platform
Programming language	Python	3.11	Used for the API client, workflow orchestration, and general-purpose programming.	Methods: The Malva Platform

Continued on the next page.

Table 4 continued from the previous page.

Category	Tool or resource	Version	Use in this study	Reported in
Single-cell analysis	Scanpy	v1.11.4	Used for preprocessing, annotation, clustering, trajectory analysis, and visualisation.	Methods: Cell type classification; Validation of biological signal preservation
Low-complexity masking	dustmasker	v1.0.0	Masks low-complexity k-mers to prevent artificial peaks in coverage analyses.	Methods: Sequence querying and pseudo-quantification; Sequence coverage quantification
Natural-language processing	Ollama	v0.13.4	Provides local language-model inference for the Llama 3.1 integration.	Methods: Querying
Large language model	Llama 3.1	8B, instruction-tuned	Translates natural-language requests into structured queries using retrieval-augmented generation.	Methods: Querying
Test generation	gpt-oss:120b	Not stated	Generates synthetic test cases for validation of the natural-language query interface.	Methods: Natural language query interface validation
Metadata harmonisation	text2term	v4.5.0	Maps biomedical metadata to ontology terms using semantic-similarity methods.	Methods: Metadata standardisation
Disease ontology	MONDO	v2025-09-02	Provides standardised disease terms for metadata harmonisation.	Methods: Metadata standardisation
Anatomy ontology	UBERON	v2025-12-04	Provides standardised anatomical terms and organ-to-system relationships.	Methods: Metadata standardisation
Cell-type ontology	Cell Ontology	v2025-12-17	Provides controlled cell-type terms and supports construction of a simplified curated ontology.	Methods: Cell type classification
Gene-set database	MSigDB	v2025.1.Hs	Provides curated Reactome, Gene Ontology, and other pathway gene sets for retrieval-augmented generation.	Methods: Querying

Continued on the next page.

Table 4 continued from the previous page.

Category	Tool or resource	Version	Use in this study	Reported in
Human reference genome	hg38 (GRCh38.p13)	GRCh38.p13; packaged with GENCODE v38	Supports human gene mapping and variant analysis.	Methods: Querying; Cell type classification; Sequence-based variant detection validation; Germline variant query probe design
Mouse reference genome	mm10 (GRCm39)	GRCm39.113	Supports mouse gene mapping.	Methods: Querying; Validation of biological signal preservation
Human reference genome	GRCh37	GRCh37: HG1287	Provides the human reference used with phase 3 data from the 1000 Genomes Project.	Methods: Germline variant query probe design
Transcript annotation	GENCODE	v38 human; v19 for 1000 Genomes variants; GRCm39.113 mouse	Provides gene models for transcript mapping, isoform analysis, and back-splice junction detection.	Methods: Querying; Cell type classification; Germline variant query probe design; Isoform usage
Benchmark indexer	MetaGraph	v0.4.1, RowDiff / Multi-BRWT	Competing k-mer indexing tool used in benchmark experiments.	Methods: Benchmarking
Benchmark indexer	REINDEER	v1.4.7-1-g0812ad128	Competing indexing tool used in benchmarks; each barcode is treated as a separate input file.	Methods: Benchmarking
Benchmark indexer	Fulgor	v4.0.0	Competing indexing and query tool evaluated using the meta-diff variant.	Methods: Benchmarking
Index preprocessing	bcalm2	v2.3.0	Constructs de Bruijn graphs as preprocessing for REINDEER.	Methods: Benchmarking
Sequence alignment	BLAST+	v2.16.0	Counts k-mers before MetaGraph indexing and provides MEGABLAST-based benchmark comparisons.	Methods: Benchmarking

Continued on the next page.

Table 4 continued from the previous page.

Category	Tool or resource	Version	Use in this study	Reported in
Pseudoalignment	kallisto-bustools	v0.28.2	Provides alignment and quantification for benchmark comparisons.	Methods: Benchmarking
Workflow management	Snakemake	v7.32.4	Orchestrates benchmark workflows and records resource usage.	Methods: Benchmarking
Sequence alignment	STAR	v2.7.11b	Aligns reads to reference genomes for ground-truth variant calling and coverage visualisation.	Methods: Benchmarking; Validation of biological signal preservation; Sequence-based variant detection validation; Isoform usage
Sequence alignment	minimap2	v2.1.1-r341	Aligns assembled contigs to host genomes and transcriptomes.	Methods: Cell clustering and marker sequence discovery
Sequence assembly	SPAdes	v4.2.0	Performs RNA-mode <i>de novo</i> assembly of differential k-mers into contigs.	Methods: Cell clustering and marker sequence discovery
Sequence classification	Kraken2	v2.1.5	Classifies contigs taxonomically using a RefSeq PlusPF database containing bacterial, archaeal, viral, and human sequences.	Methods: Cell clustering and marker sequence discovery
Barcode processing	flexiplex	v1.02.3	Detects barcodes and searches sequences in FASTQ files.	Methods: Barcode preprocessing; Sequence-based variant detection validation
Sequencing repository	Sequence Read Archive	Data frozen 11 May 2025	Provides raw sequencing data for indexing and acquisition through SRA prefetch.	Methods: Data sources, crawling and availability
Single-cell data portal	Human Cell Atlas Data Portal	Data frozen 11 May 2025	Provides the primary dataset registry and metadata feeds.	Methods: Data sources, crawling and availability
Sequence repository	GEO	Not stated	Provides metadata feeds and datasets for automated ingestion.	Methods: Data sources, crawling and availability

Continued on the next page.

Table 4 continued from the previous page.

Category	Tool or resource	Version	Use in this study	Reported in
Genomics database	CNGBdb	Not stated	Provides datasets for ingestion into the platform.	Methods: Data sources, crawling and availability
Data portal	10x Genomics Data Portal	Not stated	Provides benchmark datasets, including PBMC and Visium mouse-brain data.	Methods: Data sources, crawling and availability; Validation of biological signal preservation
Data collection	sra-tools	v3.4.1	Acquires raw sequencing data using SRA prefetch.	Methods: Data sources, crawling and availability
Data collection	bamtofastq	v1.4.1	Converts archived BAM files to FASTQ format.	Methods: Data sources, crawling and availability
Data collection	dnaio	v1.2.4	Parses FASTQ files for barcode extraction and read processing.	Methods: Data sources, crawling and availability
Polyadenylation database	polyASite	v3.0	Provides known polyadenylation sites for validation of alternative polyadenylation detection.	Methods: Benchmarking polyadenylation site detection against polyASite
Somatic-mutation database	scTML	As of 20 December 2025	Provides uniformly processed single-cell RNA-seq datasets with matched variant calls.	Methods: Large-scale somatic mutation detection and benchmark against scTML
Population genetics	1000 Genomes Project	Phase 3	Provides biallelic SNPs with minor-allele frequency of at least 1% for germline variant probes.	Methods: Germline variant query probe design
Reference sequences	RefSeq	Release 231	Provides viral genome sequences for exogenous-sequence screening.	Methods: Large-scale arbitrary nucleotide sequence search
Workflow orchestration	malvacrawl	v1.0.0	Internal service for dataset registration, metadata extraction, and data ingestion.	Methods: Data sources, crawling and availability

Continued on the next page.

Table 4 continued from the previous page.

Category	Tool or resource	Version	Use in this study	Reported in
Clustering algorithm	leidenalg	v0.10.2	Performs cell clustering for marker-based scoring and k-mer bucket analysis.	Methods: Cell type classification; Cell clustering and marker sequence discovery
Trajectory analysis	Diffusion pseudo-time	Included in Scanpy v1.11.4	Orders cells along inferred biological trajectories.	Methods: Validation of biological signal preservation

Supplementary References

- [1] Shibla and Abdulla, Brian Aevertmann, Pedro Assis, Seve Badajoz, Sidney M Bell, Emanuele Bezzi, Batuhan Cakir, Jim Chaffer, Signe Chambers, J Michael Cherry, Tiffany Chi, Jennifer Chien, Leah Dorman, Pablo Garcia-Nieto, Nayib Gloria, Mim Hastie, Daniel Hegeman, Jason Hilton, Timmy Huang, Amanda Infeld, Ana-Maria Istrate, Ivana Jelic, Kuni Katsuya, Yang Joon Kim, Karen Liang, Mike Lin, Maximilian Lombardo, Bailey Marshall, Bruce Martin, Fran McDade, Colin Megill, Nikhil Patel, Alexander Predeus, Brian Raymor, Behnam Robatmili, Dave Rogers, Erica Rutherford, Dana Sadgat, Andrew Shin, Corinn Small, Trent Smith, Prathap Sridharan, Alexander Tarashansky, Norbert Tavares, Harley Thomas, Andrew Tolopko, Meghan Urisko, Joyce Yan, Garabet Yeretssian, Jennifer Zamanian, Arathi Mani, Jonah Cool, and Ambrose Carr. Cz cellxgene discover: a single-cell data platform for scalable exploration, analysis and modeling of aggregated data. *Nucleic Acids Research*, 53(D1):D886–D900, November 2024.
- [2] Oliver Cheng, Min Hao Ling, Changqing Wang, Shuyi Wu, Matthew E Ritchie, Jonathan Göke, Noorul Amin, and Nadia M Davidson. Flexiplex: a versatile demultiplexer and search tool for omics data. *Bioinformatics*, 40(3), February 2024.
- [3] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, February 2021.
- [4] Artur d’Avila Garcez and Luis C. Lamb. Neurosymbolic ai: The 3rd wave, 2020.
- [5] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, August 2018.
- [6] Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. *Genome Biology*, 21(1), September 2020.
- [7] Jamshed Khan, Marek Kokot, Sebastian Deorowicz, and Rob Patro. Scalable, ultra-fast, and low-memory construction of compacted de bruijn graphs with cuttlefish 2. *Genome Biology*, 23(1), September 2022.
- [8] Gary Marcus. The next decade in ai: Four steps towards robust artificial intelligence, 2020.
- [9] Páll Melsted, A. Sina Boeshaghi, Lauren Liu, Fan Gao, Lambda Lu, Kyung Hoi Min, Eduardo da Veiga Beltrame, Kristján Eldjárn Hjörleifsson, Jase Gehring, and Lior Pachter. Modular, efficient and constant-memory single-cell rna-seq preprocessing. *Nature Biotechnology*, 39(7):813–818, April 2021.

- [10] Peng Qiu. Embracing the dropouts in single-cell rna-seq analysis. *Nature Communications*, 11(1), March 2020.
- [11] Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome Research*, 31(11):2080–2094, October 2021.
- [12] Jakob Schuster, Kathleen Zeglinski, Lucinda Xiao, Olivia Voulgaris, Sarahi Mendoza Rivera, Stephin J. Vervoort, Matthew E. Ritchie, Quentin Gouil, and Michael B. Clark. Powerful read processing with `matchbox`. November 2025.
- [13] Delaney K Sullivan and Lior Pachter. Flexible parsing, interpretation, and editing of technical sequences with `splitcode`. *Bioinformatics*, 40(6), June 2024.
- [14] Conor R. Walker, Xiaoting Li, Manav Chakravarthy, William Lounsbery-Scaife, Yoolim A. Choi, Ritambhara Singh, and Gamze Gürsoy. Private information leakage from single-cell count matrices. *Cell*, 187(23):6537–6549.e10, November 2024.
- [15] Ligu Wang, Hyun Jung Park, Surendra Dasari, Shengqin Wang, Jean-Pierre Kocher, and Wei Li. Cpat: Coding-potential assessment tool using an alignment-free logistic regression model. *Nucleic Acids Research*, 41(6):e74–e74, January 2013.