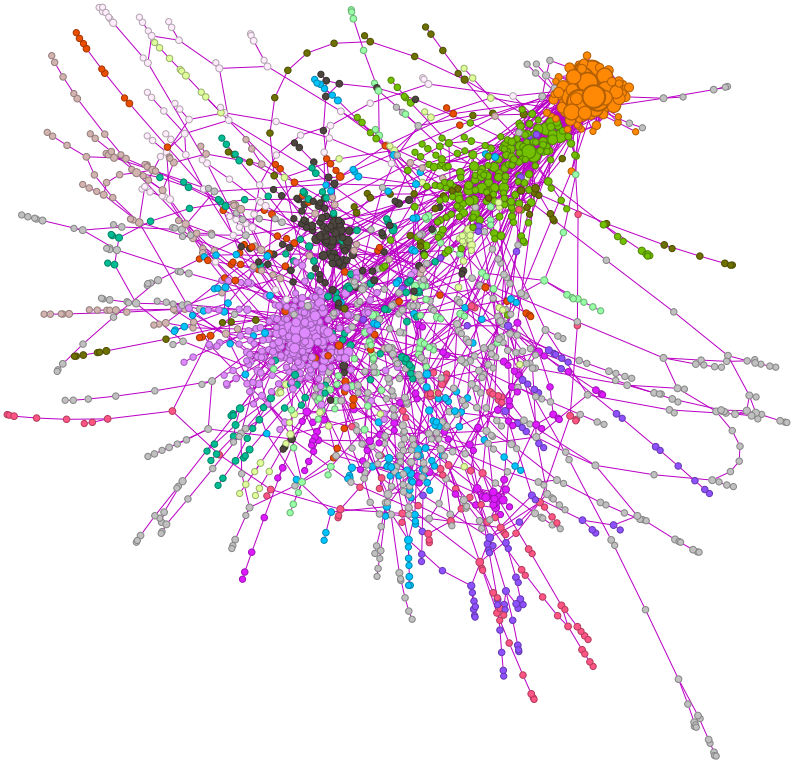
Reproducible scRNA-seq Analysis Pipeline (PBMC)



Simo Inkala

February 2026

https://github.com/inkasimo/scrnaseq-pbmc-workflow

inkasimo@gmail.com

Contents

1 Overview

This project implements a reproducible, containerized single-cell RNA-seq (scRNA-seq) analysis workflow using Snakemake, Docker, and a Python-based CLI wrapper. The pipeline is demonstrated on publicly available 10x Genomics PBMC datasets (donors 1–4) and covers the full path from raw FASTQs to downstream statistical modeling and network analysis.

The primary objective is to demonstrate disciplined workflow engineering: explicit execution structure, strict dependency control, restart-safe modular design, and transparent donor-aware statistical modeling. Reproducibility is treated as a first-class constraint through containerization, version-locked R environments, and digest-pinned runtime images.

While the biological results are internally coherent and consistent with expected PBMC structure, the emphasis of this project is methodological rather than discovery-driven. The repository is therefore intended as a technical portfolio artifact showcasing the design and implementation of a complete, auditable, and reproducible scRNA-seq analysis system.

1.1 Execution environment

Hardware:	
CPU	Intel Core Ultra 9 185H (16 cores, 22 threads)
RAM	32 GB
Storage	NVMe SSD: 1 TB (OS) + NVMe SSD: 2 TB (data)

All computations were performed on a local workstation running Windows 11 with WSL2 and Docker Desktop. The pipeline was executed entirely inside Docker containers to ensure environment isolation and reproducibility across hosts.

Due to the memory requirements of STAR genome indexing, the default WSL2 configuration was insufficient and resulted in out-of-memory (OOM) failures. The WSL2 memory limit was therefore explicitly increased to 32 GB. A reference configuration file is provided at `docs/wslconfig.example` to document the required settings and facilitate reproducible execution on similar systems.

2 Repository Organization

The repository is organized to enforce a clear separation between execution logic,configuration, containerized dependencies, and generated data. This structure follows best practices for reproducible computational workflows and explicitly distinguishes between upstream engineering steps and downstream analytical stages.

- `containers/`
  Docker build context defining a fully self-contained execution environment. This includes

all system dependencies, bioinformatics tools, and a fully specified R environment restored via `renv`. The archived image is intended to be the primary execution environment for reproducible runs.

- `workflow/`
  Snakemake workflow definition encoding the full directed acyclic graph (DAG).This includes rules for FASTQ acquisition or validation, quality control, optional read trimming, reference preparation, STARsolo alignment, and downstream analysis stages.

- `config/`
  User-editable configuration files (notably `config.yaml`) controlling dataset definitions, resource usage, algorithmic parameters, and execution toggles (e.g. downloading FASTQs vs. validating existing data, enabling trimming, network analysis settings).

- `resources/`
  Static, versioned resources bundled with the workflow to ensure reproducibility. This includes the 10x Genomics barcode whitelist and MSigDB gene set collections (Hallmark and C7), avoiding reliance on unstable external URLs.

- `data/` Input data and reference assets generated or downloaded at runtime. This directory contains raw FASTQs, reference genomes and annotations, STAR indices, and (if enabled) trimmed FASTQs.

- `results/` All pipeline outputs and logs. This includes QC reports, STARsolo outputs, downstream Seurat objects, differential expression results, network analyses, and detailed execution logs. Outputs are intentionally excluded from version control due to size.

- `docs/`
  Project documentation, including the user manual, this technical report, developer notes, and a small set of representative execution artifacts under `docs/example_outputs/` to demonstrate successful pipeline execution.

- `scripts/`
  R scripts implementing downstream analyses. These include Seurat-based QC and annotation, pseudobulk differential expression and equivalence testing, enrichment analysis, and co-expression network construction.

- `run_analysis.py`
  A Python-based CLI wrapper that orchestrates Snakemake execution inside Docker. The wrapper enforces section-based execution, provides explicit control over which parts of the pipeline are run, and prevents accidental full executions.

3 R Environment and Reproducibility Strategy

3.1 Overview

All R-based analyses in this workflow are executed inside a Docker container with a fully specified and reproducible R environment.

Dependency management is handled using `renv`, while containerization ensures isolation from host-specific configurations. This design minimizes environment drift and allows consistent re-execution across machines and over time.

3.2 R Version and Package Management

The container runtime is pinned to a fixed R base image (`rocker/r-ver:4.3.3`). R package dependencies are declared in a version-locked `renv.lock` file, and package resolution is pinned to a CRAN snapshot via the environment variable:
`RENV_CONFIG_REPOS_OVERRIDE=https://packagemanager.posit.co/cran/2024-01-01`.

The lockfile captures:

- The R version used (R 4.3.3)
- The CRAN snapshot date (`https://packagemanager.posit.co/cran/2024-01-01`)
- The exact versions and sources of all R packages required for the analysis

During the Docker image build, the full R environment is restored non-interactively using `renv::restore()`. The restored library is installed into a fixed path inside the container (`/opt/renv/library`) via `RENV_PATHS_LIBRARY`, and no R package installation or snapshotting occurs at runtime.

Bioconductor resolution is explicitly pinned during the image build (`Bioconductor 3.18`) to ensure consistent installation of Bioconductor packages.

3.3 Vendoring of Seurat

The core analysis depends on Seurat [1] and SeuratObject [2], which are fast-evolving packages and common sources of reproducibility issues due to upstream repository changes or availability.

To mitigate this risk, specific versions are vendored directly into the repository:

- `renv/vendor/seurat-v4.4.0.tar.gz`
- `renv/vendor/seurat-object-v4.1.4.tar.gz`

During Docker image build, the vendored sources are copied into the container (`COPY renv/vendor/ /work/renv/vendor/`) and installed by `renv::restore()` as local package sources recorded in `renv.lock`.
As a result:

- Docker builds do not rely on GitHub availability or branch state for these packages
- Seurat versions are decoupled from upstream CRAN/Bioconductor changes
- The exact Seurat source code used in the analysis is preserved with the project

4 Docker Integration

The Docker image is built from `rocker/r-ver:4.3.3`, providing a fixed R runtime on a Debian-compatible base. System-level dependencies required for R packages and external tools (e.g. STAR, FastQC) are installed explicitly during the image build.

The image also configures explicit `renv` paths to avoid host-dependent behavior:

- `RENV_PATHS_LIBRARY=/opt/renv/library`
- `RENV_PATHS_CACHE=/opt/renv/cache`
- `RENV_CONFIG_CACHE_SYMLINKS=FALSE`

This ensures that both the installed packages and the renv cache are fully contained within the image filesystem and are not affected by host-side caches or symlinks.

All R packages are restored during image build using `renv::restore()`. The Docker build does not modify the lockfile; dependency changes must be made explicitly by updating `renv.lock` and committing it to version control.

4.1 Docker Image Distribution

A pre-built Docker image corresponding to the archived workflow environment is published to GitHub Container Registry (GHCR). The versioned release can be pulled using:

```
docker pull ghcr.io/inkasimo/scrnaseq-pbmc-workflow:v1.0.2
```

For strict archival reproducibility, the digest-pinned image corresponding exactly to the environment used to generate the archived results can be pulled using:

```
docker pull ghcr.io/inkasimo/scrnaseq-pbmc-workflow@sha256:
80354b76e76405636c43e73902236e0399d26978a214227afbafa46fc0555bb8
```

Local image builds (`docker build`) are supported for development purposes but are not guaranteed to be bit-identical to the archived image due to potential upstream system library changes.

4.2 Separation of Concerns

- Host system: used primarily to edit code and commit changes. Dependency updates are captured by regenerating and committing `renv.lock`. In practice, lockfile updates are performed by launching an interactive R session inside the Docker container used for development, then committing the resulting lockfile on the host.
- Docker image (execution): consumes the lockfile in a read-only fashion during image build and runtime, providing a stable and reproducible execution environment.
- IDE tooling: automatic IDE-driven package installation is disabled to prevent contamination of the project dependency set.

Result

This setup ensures:

- A pinned and auditable R environment (R version, CRAN snapshot, and package sources)
- Long-term stability of critical dependencies (notably Seurat via vendoring)
- Clear provenance of all software components
- Safe integration with Snakemake and container-based execution

In summary, the combination of `renv`, vendored critical packages, and Docker provides a robust and auditable foundation for reproducible single-cell RNA-seq analysis.

4.3 Windows / WSL / Docker Desktop

This workflow runs inside Docker and bind-mounts the repository into the container. On Windows, Docker Desktop must have access to the repository path; otherwise the container will not be able to read files under `/work`.

If the following type of error is encountered:

```
Snakefile "workflow/Snakefile" not found
```

Docker Desktop file-sharing settings should be checked for the drive containing the repository. Bind-mount restrictions are common on managed or corporate machines; in such cases, running the workflow on Linux or macOS avoids these limitations.

5 Execution Model

The pipeline is orchestrated using Snakemake and executed inside Docker containers to ensure reproducibility and avoid host-specific environment drift.

The workflow is implemented as a `workflow/Snakefile` (the authoritative DAG and rule logic) and an optional Python wrapper (`run_analysis.py`) that provides a safer, section-based CLI for running the workflow inside the Docker image.

Conceptually, users select a "section" (e.g. QC, alignment, downstream), the wrapper maps that intent to explicit Snakemake targets (mostly `*.done` sentinel files), and Snakemake computes and executes the minimal set of rules required to produce those targets.

5.1 Snakemake workflow (`workflow/Snakefile`)

The Snakefile encodes the pipeline as a directed acyclic graph (DAG) of rules, where each rule declares explicit `input`, `output`, and execution logic (shell or Python `run` blocks).

A core design choice is the use of lightweight sentinel "done" files (e.g. `fastqc.done`, `starsolo.done`, `seurat_qc.done`) as rule outputs. These serve two purposes:

- They provide a stable, human-auditable completion marker for each stage without requiring Snakemake to hash or reason over large binary outputs.
- They make the workflow resilient to partial outputs: a rule is only considered complete when its sentinel file is created after all expected artifacts have been written successfully.

Sentinels are created atomically (write `.tmp` then move/replace), which reduces the chance of leaving a misleading "complete" marker after an interrupted job.

The workflow also supports two execution branches: `raw` (default) and `trimmed` (optional). Trimming is applied only to Read 2 and, if enabled, all downstream steps automatically route to the trimmed branch while preserving the on-disk STARsolo layout (legacy mapping of `untrimmed` → `raw`).

Configuration is driven by `config/config.yaml`, with CLI overrides supported for key toggles (e.g. download FASTQs, build STAR index, enable trimming). Rules are written to be explicit about required resources (threads, reference paths, gene sets, marker sets), and logs are captured under `results/logs/` for auditability.

5.2 Python wrapper (`run_analysis.py`)

While the Snakefile can be executed directly inside Docker, most users run the pipeline via `run_analysis.py`, which wraps Snakemake in a section-based CLI.

The wrapper's main job is to translate high-level user intent into explicit Snakemake targets. Each CLI "section" corresponds to a curated list of target files (primarily `*.done` sentinels and a small number of concrete outputs such as `multiqc_report.html`). Snakemake then builds exactly what is required to satisfy those targets, and nothing else.

This approach improves usability and safety:

- **Section-based execution:** users run a named stage (`qc`, `align`, `downstream`, ...) instead of managing raw Snakemake targets.
- **Prevents accidental full runs:** the wrapper requires selecting a section, rather than implicitly executing the full `rule all`.
- **Mode toggles are explicit:** flags like `--trimmed` and `all_no_download` translate into controlled Snakemake config overrides (e.g. `trim_enabled=true`, `download_fastqs=false`).
- **Docker execution is standardized:** the wrapper always executes Snakemake inside the pinned container image, bind-mounting the repository and passing consistent CPU/thread settings.

- **Early failure checks:** the wrapper validates that Docker bind mounts can see `workflow/Snakefile`, avoiding confusing "Snakefile not found" failures caused by file-sharing restrictions on some Windows/WSL setups.

The wrapper also exposes convenience commands for discovery (`--list-sections`, `--list-donors`) and supports passing through advanced Snakemake options via `--extra` for power users.

5.3 Available wrapper sections

The full user manual documents all sections and examples; for quick reference, the wrapper exposes the following top-level sections:

- `download_data`
- `download_data_and_qc`
- `qc`
- `ref`
- `trim`
- `trim_and_qc`
- `align`
- `upstream`
- `upstream_no_download`
- `build_seurat_object_qc`
- `filter_and_normalize_seurat`
- `cluster_annotate_seurat`
- `deg_and_tost`
- `networks`
- `downstream`
- `all`
- `all_no_download`
- `unlock`

Internally, the wrapper maps these user-facing sections onto specific target sets (for example, `qc` expands to per-donor FastQC sentinels plus the aggregated MultiQC report). This design keeps the workflow modular and transparent while making execution approachable for users who do not want to interact directly with Snakemake DAG mechanics.

6 Data

Raw FASTQ files are obtained from publicly available 10x Genomics Chromium X Series datasets corresponding to the 5k Human PBMC 3′ Gene Expression (GEM-X v4) libraries for Donors 1–4. Each donor is provided as a separate FASTQ archive and downloaded directly from the 10x Genomics data portal.

The samples consist of peripheral blood mononuclear cells (PBMCs) isolated from healthy human donors (male, age 18–35), prepared by 10x Genomics and sourced from Cellular Technologies Limited. Cells were processed using the Chromium GEM-X Single Cell 3′ Reagent Kits v4 chemistry and sequenced on an Illumina NovaSeq 6000 platform at approximately 39,000 read pairs per cell.

Libraries were generated as paired-end, dual-indexed runs with the following configuration: 28 cycles for Read 1 (cell barcode + UMI), 10 cycles for i7 index, 10 cycles for i5 index, and 90 cycles for Read 2 (cDNA insert). Read 1 therefore encodes the 16 bp cell barcode and 12 bp UMI, while Read 2 contains the transcript-derived sequence used for alignment and quantification.

The datasets were originally processed by 10x Genomics using the `cellranger multi` pipeline, and automated cell type annotations were generated using the `human_pca_v1_beta` reference model. **However, in this project, only the raw FASTQ files are used; all alignment and downstream analyses are performed independently within the present workflow.**

Each donor dataset contains approximately 5,000–6,000 recovered cells and represents cryopreserved human PBMC suspensions. The data are licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0) license and are publicly accessible through the 10x Genomics website.

Companion datasets include the individual 5k PBMC datasets for Donors 1–4 and a 20k multiplexed PBMC dataset combining all donors; however, this workflow analyzes the donors independently.

7 Toy Demonstration Dataset and Mini Reference

To enable a fast, resource-light demonstration of the workflow, a self-contained toy dataset was constructed. The objective of the toy configuration is to allow users to execute the full upstream pipeline (QC → alignment → Seurat object construction → normalization → clustering) within approximately 5–10 minutes on a standard workstation.

7.1 FASTQ Subsampling

A reduced read set was generated from the publicly available 10x Genomics PBMC data by truncating the original FASTQ files to a fixed number of reads per mate. This preserves realistic

read structure (16 bp cell barcode + 12 bp UMI) while dramatically reducing runtime.

The toy configuration uses:

- Single donor
- 100,000 read pairs
- Untrimmed execution branch

This preserves realistic 10x geometry while keeping alignment computationally lightweight.

7.2 Mini Reference Construction

Using the full GRCh38 reference for the toy demonstration would require tens of gigabytes of RAM for index loading. To ensure broad usability, a chromosome-restricted reference was created.

Chromosome-restricted FASTA Chromosome 1 was extracted from the primary assembly:

```
samtools faidx GRCh38.primary_assembly.genome.fa chr1 \
  > GRCh38.chr1.fa
```

Chromosome-restricted GTF The annotation was filtered to retain header lines and chr1 records only:

```
awk 'BEGIN{FS="\t"} /^#/ || $1=="chr1"' \
  gencode.v45.annotation.gtf \
  > gencode.v45.chr1.gtf
```

STAR Index Generation For the current toy configuration, the STAR index is **not** precomputed or distributed. Instead, the index is generated dynamically during execution of the toy workflow:

```
STAR \
  --runMode genomeGenerate \
  --genomeDir data/ref/toy/star_index_chr1 \
  --genomeFastaFiles GRCh38.chr1.fa \
  --sjdbGTFfile gencode.v45.chr1.gtf \
  --sjdbOverhang 89
```

Building the chromosome-restricted index at runtime requires only modest memory and typically completes within 2–5 minutes on a standard workstation. This approach significantly reduces the size of the distributed toy bundle while preserving realistic alignment behavior.

7.3 Design Rationale

The toy dataset is intentionally separated from the full production configuration:

- Toy demo: chromosome-restricted reference, STAR index built during execution, fast execution.
- Full archive: complete GRCh38 + GENCODE v45 index prebuilt for production-scale analysis.

Both are distributed under the same Zenodo concept record to ensure reproducibility and long-term archival stability. This separation reflects a deliberate engineering choice: demonstration constraints are distinct from production-scale analysis requirements.

8 Quality Control

Raw FASTQ quality was assessed using FastQC v0.11.9 and summarized with MultiQC v1.33. QC results were used to evaluate the necessity of adapter trimming prior to alignment.

Across donors and lanes, base quality profiles are high and stable over the read length, and the aggregated MultiQC status table reports `PASS` for R2 *Adapter Content*, indicating no evidence of pervasive adapter contamination in the cDNA read.

The primary flagged item is *Overrepresented sequences*, most notably in Donor 4 R2 files. The top sequences occur at approximately FastQC's reporting threshold and are consistent with known short-oligo / library-structure artifacts (e.g., template-switch / SMARTer-like carryover) rather than systematic adapter read-through affecting a substantial fraction of reads.

For 10x Genomics 3′ Gene Expression data, explicit trimming is typically unnecessary and can be counterproductive: standard single-cell pipelines (e.g., Cell Ranger and STARsolo) rely on preserving the expected read structure, and modest 3′ artifacts are generally handled via soft-clipping during alignment without compromising UMI counting. Therefore, trimming was omitted for the main analysis and is only enabled if strong adapter signal is observed in QC.

The workflow supports an optional trimming branch via a dedicated wrapper section (and corresponding Snakemake rules), but this branch was not activated for the reported results given the QC outcome (see Section **Read Trimming (cutadapt)**).

All QC reports are written to `results/qc/` (FastQC per-donor outputs and aggregated MultiQC reports), with execution logs stored under `results/logs/qc/`.

9 Read trimming (cutadapt)

The workflow supports an optional trimming branch implemented with `cutadapt` and exposed through dedicated wrapper sections (e.g. `trim`, `trim_and_qc`) and the `-trimmed` flag. When

activated, all downstream steps (QC, alignment, and analysis) resolve against trimmed FASTQs and write into a separate `trimmed/` results branch.

Trimming is applied to Read 2 (cDNA) only; Read 1 (cell barcode + UMI) is never sequence-modified. Read pairs are filtered jointly using `-pair-filter=any`, so pairs are discarded if R2 fails the minimum-length threshold.

9.1 Configuration

Trimming parameters are defined in `config/config.yaml` and passed into the Snakemake rule via `TRIM_R2_*` variables.

```
trim:
  enabled: false
  adapter_r2: AGATCGGAAGAG
  q_r2: 20
  minlen_r2: 20
```

In the reported trimming run, `cutadapt v4.9` was executed with `-A AGATCGGAAGAG`, `-Q 0,20`, and `-minimum-length 0:60`, i.e. an effective `minlen_r2` of 60.

9.2 Implementation details

Outputs are written lane-by-lane to temporary files and atomically published on success; the `trim.done` sentinel is created only after all lanes complete. This prevents partially written FASTQs from being consumed downstream.

9.3 Observed impact

Across all donors, the proportion of read pairs discarded as too short after trimming was low and consistent (approximately 0.3–0.4%). This indicates minimal short-insert filtering and no evidence of pervasive adapter read-through.

Given the strong baseline QC metrics and the negligible trimming impact, downstream analyses in this report use the default untrimmed branch. The trimming pathway remains available but was not required for these datasets.

10 Alignment and Quantification

10.1 Choice of aligner: STARsolo vs Cell Ranger

Alignment and UMI-aware quantification were performed using `STARsolo` (STAR v2.7.11b) rather than Cell Ranger. The primary motivation is transparency and reproducibility: STARsolo exposes all alignment and barcode-handling parameters explicitly, integrates directly into Snakemake, and runs inside the pinned Docker environment.

This avoids reliance on opaque defaults or vendor-controlled pipelines while preserving compatibility with standard 10x data structures (gene–cell matrices equivalent to Cell Ranger output).

10.2 Reference genome and annotation

The reference was built locally from the following components:

- GRCh38 primary assembly
- GENCODE v45 gene annotation
- STAR v2.7.11b
- 10x GEM-X 3′ v4 barcode whitelist (bundled in `resources/barcodes/`)

The STAR index was generated inside the container using `--sjdbOverhang 89` (R2 length 90 bp), and is not distributed with the repository due to size. Index generation requires approximately 25–30 GB RAM.

10.3 Read structure and chemistry assumptions

Libraries correspond to 10x Chromium GEM-X 3′ v4 chemistry. Read structure assumptions were made explicit in the STARsolo invocation:

- R1: 16 bp cell barcode + 12 bp UMI (28 bp total)
- R2: cDNA (90 bp)
- `--soloType CB_UMI_Simple`
- `--soloCBlen 16`
- `--soloUMIlen 12`
- `--soloBarcodeReadLength 28`
- `--soloBarcodeMate 2`

Whitelist filtering was enforced via `--soloCBwhitelist`.

10.4 Counting and outputs

Gene-level UMI counting was performed internally by STARsolo using `-soloFeatures Gene`. Both raw and filtered gene–cell matrices were generated. No external counting tools (e.g. featureCounts) were used.

10.5 BAM output

BAM output was disabled via `-outSAMtype None` to reduce disk usage and memory overhead. STARsolo gene–cell matrices are complete without BAM files. BAM generation can be enabled if required for downstream inspection.

10.6 Execution environment

Alignment was executed inside Docker under WSL2. Memory was configured to allow STAR index generation and alignment:

```
[wsl2]
memory=32GB
processors=8
swap=16GB
```

No additional host-specific tuning was required.

10.7 Alignment summary statistics

Alignment metrics were consistent across donors. Unique mapping rates ranged from 68.8% to 72.4%, multi-mapping from 19.3% to 21.7%, and reads unmapped due to short length from 7.6% to 9.4%. Mismatch rates were stable at approximately 0.21–0.23%. No chimeric reads were reported.

Donor	Input Reads (M)	Unique (%)	Multi (%)	Unmapped Short (%)	Mismatch (%)
Donor1	227.1	71.89	20.18	7.83	0.23
Donor2	207.0	72.41	19.33	8.16	0.22
Donor3	185.7	72.42	19.84	7.64	0.21
Donor4	207.8	68.77	21.74	9.37	0.22

Table 1: STARsolo alignment summary statistics (raw branch).

Mapping performance is consistent with high-quality 10x 3′ scRNA-seq libraries; donor4 shows a modestly lower unique mapping rate and higher short-read fraction but remains within expected bounds for PBMC datasets.

11 Downstream Analysis

11.1 Overview

Downstream analyses were performed on donor-specific expression datasets derived from STARsolo filtered gene–cell matrices. Seurat objects were used for quality control, normalization, clustering, and marker-based annotation. Pseudobulk differential expression and co-expression networks were derived from the resulting QC-filtered and annotated data representations.

The workflow proceeds in five stages: (i) Seurat object construction from STARsolo outputs, (ii) donor-specific quality control and normalization, (iii) clustering and marker-based cell-type annotation, (iv) donor-aware pseudobulk differential expression with equivalence testing, and (v) cell-type–specific co-expression network construction with cross-donor consensus integration. All analyses were executed inside the containerized R environment to ensure full reproducibility.

All Seurat-based analyses were performed using `Seurat v4.4.0` and `SeuratObject v4.1.4` as specified in the `renv.lock` file embedded in the Docker image.

12 Methods

12.1 Seurat object construction

Filtered STARsolo gene–barcode matrices were imported into Seurat. Feature names were set to gene symbols; symbol-less Ensembl placeholder rows were removed. Duplicate symbols were collapsed by summing counts to ensure a unique feature space per object.

12.2 Quality control filtering

Quality control was performed independently per donor using data-adaptive thresholds derived from empirical distributions of QC metrics.

Filtering rules:

- `percent.mt` $\leq \min(q_{90}, 25)$
- `nFeature_RNA` $\geq q_{05}$ and $\leq \min(q_{99}, 6000)$
- `nCount_RNA` $\geq q_{05}$ and $\leq q_{99}$
- `percent.hb` ≤ 1

For each donor, thresholds and filtering summaries were written to disk. Filtering decisions were therefore fully auditable.

12.3 Normalization and feature selection

QC-filtered objects were normalized using Seurat's `LogNormalize` method (library-size scaling followed by log-transformation; scale factor = 10,000). Highly variable genes (HVGs; $n = 2000$) were identified using the VST method.

Scaling and regression of `percent.mt` were applied to HVGs for dimensionality reduction only. Scaled values were not used for differential expression or network inference.

12.4 Clustering and marker-based annotation

Dimensionality reduction was performed using PCA (first 30 components), followed by nearest-neighbor graph construction and Louvain clustering (resolution = 0.3). UMAP embeddings were generated using a fixed random seed.

Cell type annotation was performed using predefined marker gene sets. For each marker set, module scores were computed using `AddModuleScore`. Each cell was assigned the label corresponding to its highest module score. Cluster-level majority labels were also computed for reporting and downstream grouping. PBMC marker genes represented in table 2.

Cell type	**Marker genes**
T-cells	TRAC, CD3D, CD3E, LTB
CD4 T-cells	IL7R, CCR7, LTB, MALAT1
CD8 T-cells	CD8A, CD8B, NKG7, GZMK
NK-cells	NKG7, GNLY, PRF1, FCGR3A, XCL1
B-cells	MS4A1, CD79A, CD74, HLA-DRA
Plasma cells	MZB1, XBP1, JCHAIN
CD14 Monocytes	LYZ, S100A8, S100A9, LGALS3, CTSS
FCGR3A Monocytes	LYZ, FCGR3A, MS4A7, LGALS3
Dendritic cells	FCER1A, CST3, CLEC10A
Platelets	PPBP, PF4, NRGN

Table 2: Canonical PBMC marker genes used for manual cell-type annotation. Marker sets were curated from established PBMC single-cell RNA-seq references and workflows, including the 10x Genomics PBMC 3k/10k datasets [5], dendritic cell characterization by [6], and Seurat-based PBMC analyses [7, 8].

12.5 Pseudobulk differential expression and equivalence testing

Differential expression was performed at the donor level using a pseudobulk strategy.

Pseudobulk construction: For each donor and coarse immune group (table 3.), raw UMI counts were summed across cells (minimum 50 cells per donor × group). Donor was treated as the experimental unit.

Statistical model and differential expression: Pseudobulk differential expression was performed using `DESeq2` v1.42.1 (Bioconductor 3.18). Counts were modeled using the design:

DEG group	Constituent cell types
T-like	T-cells, CD4 T-cells, CD8 T-cells, NK-cells
B-like	B-cells, Plasma cells
Mono-like	CD14 Monocytes, FCGR3A Monocytes

Table 3: Cell-type groupings used for pseudobulk differential expression analysis.

$$\sim \texttt{donor} + \texttt{group},$$

thereby controlling for donor-specific baseline effects while testing cell-type group contrasts. Wald tests were used for coefficient inference, and $\log_2$ fold changes correspond to the estimated group effect. P-values were adjusted for multiple testing using the Benjamini–Hochberg procedure.

Contrasts:

Contrast	Comparison
C1	T-like vs B-like
C2	T-like vs Mono-like
C3	B-like vs Mono-like

Table 4: Pseudobulk differential expression contrasts evaluated using DESeq2.

Marker genes: Strict marker genes were defined as those satisfying

$$\texttt{padj} < 10^{-10} \quad \text{and} \quad |\log_2 \text{FC}| > 3.$$

Equivalence testing (TOST): Two one-sided tests were performed on DESeq2 $\log_2$ fold-change estimates using a predefined margin $\delta = 0.75$. Genes were considered conserved if:

$$\texttt{padj\_equiv} < 0.05 \quad \text{and} \quad |\log_2 FC| < \delta$$

This distinguishes true similarity from non-significance.

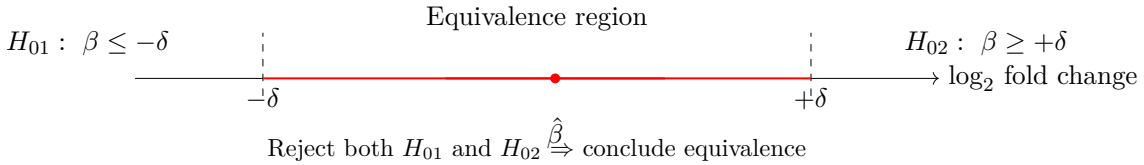


Figure 1: Two one-sided tests (TOST) for equivalence of $\log_2$ fold change within $[-\delta, +\delta]$.

12.6 Pathway enrichment analysis

Pathway analyses were performed using MSigDB Hallmark and C7 collections.

- GSEA (fgsea) on ranked gene lists (ranked by $\log_2$FC)

- ORA (clusterProfiler) on marker and conserved gene sets

Cross-contrast intersections and union gene sets were also tested for enrichment.

12.7 Cell-type–specific co-expression networks

Gene co-expression networks were constructed separately for each donor within predefined cell-type subsets (`CD4 T cells, B-cells, and CD14 monocytes`). For each donor and cell type, pairwise gene–gene associations were computed and converted into an undirected network.

Cross-donor integration was performed using a consensus approach: an edge was retained if observed in at least two independent donor-specific networks (consensus edge count ≥ 2). Given the limited number of biological replicates ($n = 4$), this requirement prioritizes reproducible co-expression relationships and reduces donor-specific noise.

Metacell aggregation: Cells were randomly pooled into metacells (size = 20) to stabilize correlation estimates.

Gene filtering: Genes detected (expression > 0) in at least 5% of metacells (not cells) within a donor $\times$ cell-type subset were retained.

Edge definition: Spearman correlation was computed across metacells. Edges were retained using a sparsification strategy designed to reduce unstable or donor-specific correlations:

- Top-k neighbors per gene ($k = 50$)
- $|\rho| \geq 0.25$
- Positive correlations only

The top-k constraint enforces local sparsity and limits hub inflation. The correlation threshold suppresses weak, noise-driven associations, which is particularly important given the limited donor count. Restricting to positive correlations improves interpretability in small-sample settings.

Per-donor graphs were reduced to their largest connected component to avoid fragmentation artifacts.

Cross-donor consensus and module detection: Donor-specific edge lists were merged into a consensus graph. For each gene pair, we computed:

- `support` = number of donors in which the edge was observed

- `median_cor` = median correlation across supporting donors

Edges were retained if observed in at least two donors with consistent sign. The final consensus network was restricted to its largest connected component.

Leiden clustering was performed on the consensus graph using `median_cor` as the edge weight.

Consensus edge weighting: In addition to `median_cor`, a stability-adjusted weight was computed for ranking and visualization:

$$\texttt{weight\_consensus} = \texttt{median\_cor} \times \texttt{support\_frac}, \qquad \texttt{support\_frac} = \frac{\texttt{support}}{\#\ \text{donors used}}$$

This support-scaled weight was not used for clustering, but is provided for edge prioritization and visualization.

Functional enrichment (ORA): Module gene sets were tested for over-representation using MSigDB Hallmark (H) and Immunologic Signatures (C7) collections. Enrichment was performed against the network gene universe and summarized in tabular and dot-plot form.

12.8 Network exports for Gephi

For each consensus network, Gephi-compatible tables were generated:

Edge list (`edges.csv`) including `median_cor`, `support`, `support_frac`, and `weight_consensus`.
Node table (`nodes.csv`) including module assignment, degree, and betweenness centrality.
Module membership table (`modules.csv`).

Nodes are additionally annotated with canonical marker membership flags and DEG-derived indicators (strict marker and conserved sets), enabling direct biological interpretation within the graph representation.

12.9 Network visualization with Gephi

Consensus co-expression networks were visualized using Gephi v0.10.1 (Java 11 runtime, Windows 11 environment). Gephi was used solely for layout and visual rendering; all graph construction, filtering, weighting, and module detection were performed within the reproducible R pipeline.

Layout algorithm Node positions were computed using the ForceAtlas2 layout algorithm with default continuous force-directed dynamics. The following key parameters were used:

- Scaling = 5.0
- Gravity = 5.0
- Edge weight influence = 1.0
- LinLog mode = disabled
- Prevent overlap = disabled
- Approximate repulsion = enabled

Layouts were allowed to converge visually to a stable configuration before export. No manual node repositioning was performed.

Visual encoding Visual attributes were mapped deterministically as follows:

- Node color: Leiden module assignment (partition coloring)
- Node size: Degree centrality (linear scaling)
- Edge thickness: Median consensus correlation weight

Node size scaling was applied uniformly across modules to preserve relative degree differences within each network. No manual recoloring or post-hoc adjustment of individual nodes was performed.

Export Final layouts were exported as vector graphics (PDF/SVG) for inclusion in the report. These visualizations are intended to illustrate module organization and global network structure; quantitative analyses and enrichment results are derived exclusively from the reproducible computational pipeline.

13 Results

13.1 Quality control and filtering outcomes

Initial cell recovery ranged from 4,612 to 5,525 cells per donor (Table 5). Median detected features per cell were consistent across donors (1,786–2,127 genes), with donor4 exhibiting slightly lower complexity. Median UMI counts ranged from 6,819 to 8,608.

Donor	Cells	Med. nFeat	Med. nCount	Med. %mt	%mt>20	Removed
Donor1	5510	2058	7889	15.77	27.33	1506
Donor2	5525	2127	7932	14.61	21.38	1181
Donor3	4612	2062	8608	15.72	23.24	1072
Donor4	5206	1786	6819	23.68	63.81	3322

Table 5: Pre-filtering quality control metrics per donor (mitochondrial threshold = 20%).

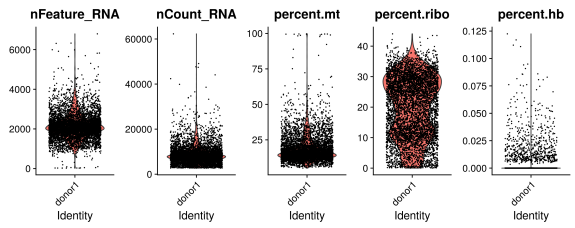
Donor	**Median %ribo**	**Median %hb**
Donor1	21.24	0.00
Donor2	15.12	0.00
Donor3	21.01	0.00
Donor4	11.84	0.00

Table 6: Additional QC composition metrics per donor.

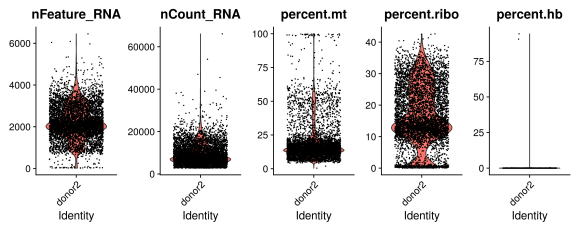
Mitochondrial content differed substantially between donors. Donors 1–3 showed comparable median mitochondrial percentages (∼14–16%), whereas donor4 displayed elevated mitochondrial content (median 23.7%). Consequently, 63.8% of donor4 cells exceeded the 20% mitochondrial threshold compared to 21–27% in the other donors (Table 5).

Feature-count extremes were rare: fewer than 1.5% of cells exhibited < 500 detected genes, and $< 0.2\%$ exceeded 6,000 genes in any donor, indicating minimal evidence of low-quality empty droplets or high-confidence doublets based on feature count alone.

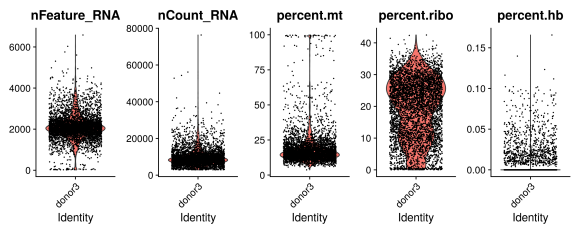
Overall, quality distributions were broadly consistent across donors with the exception of donor4, which showed a right-shifted mitochondrial distribution and lower median transcript complexity (Figure 2).



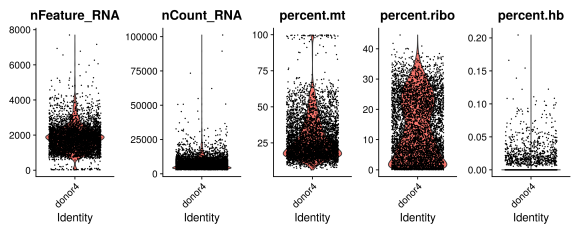
(a) Donor1



(b) Donor2



(c) Donor3



(d) Donor4

Figure 2: Quality control distributions per donor showing nFeature_RNA, nCount_RNA, percent.mt, percent.ribo and percent.hb prior to filtering.

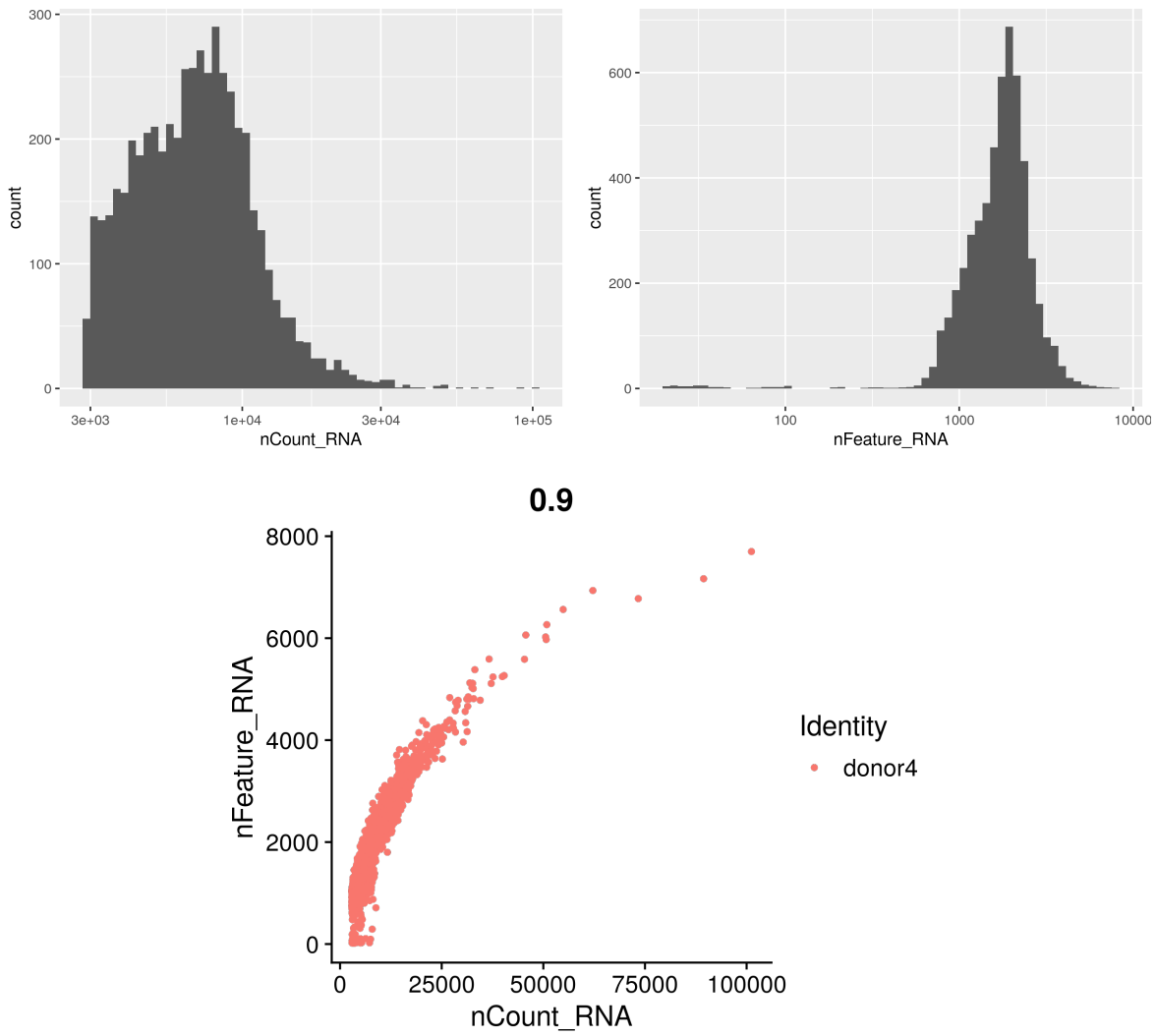


Figure 3: Quality control metrics for donor4 prior to filtering. Histograms show UMI count and detected gene distributions; scatter plot shows the relationship between UMI counts and detected genes per cell.

13.2 Filtering yield and failure modes

Quality control was performed independently for each donor using donor-specific empirical thresholds.

Across donors, 14–18% of cells were removed for Donors 1–3, whereas donor4 exhibited substantially higher mitochondrial content, resulting in removal of 47.6% of cells (Table 7).

Mitochondrial percentage was the dominant filtering driver across donors (Tables 7–8).

Representative pre- and post-filtering distributions for donor4 are shown in Figures 3 and 4. Filtering reduced high-mitochondrial outliers and extreme library size artifacts while preserving the main cellular population.

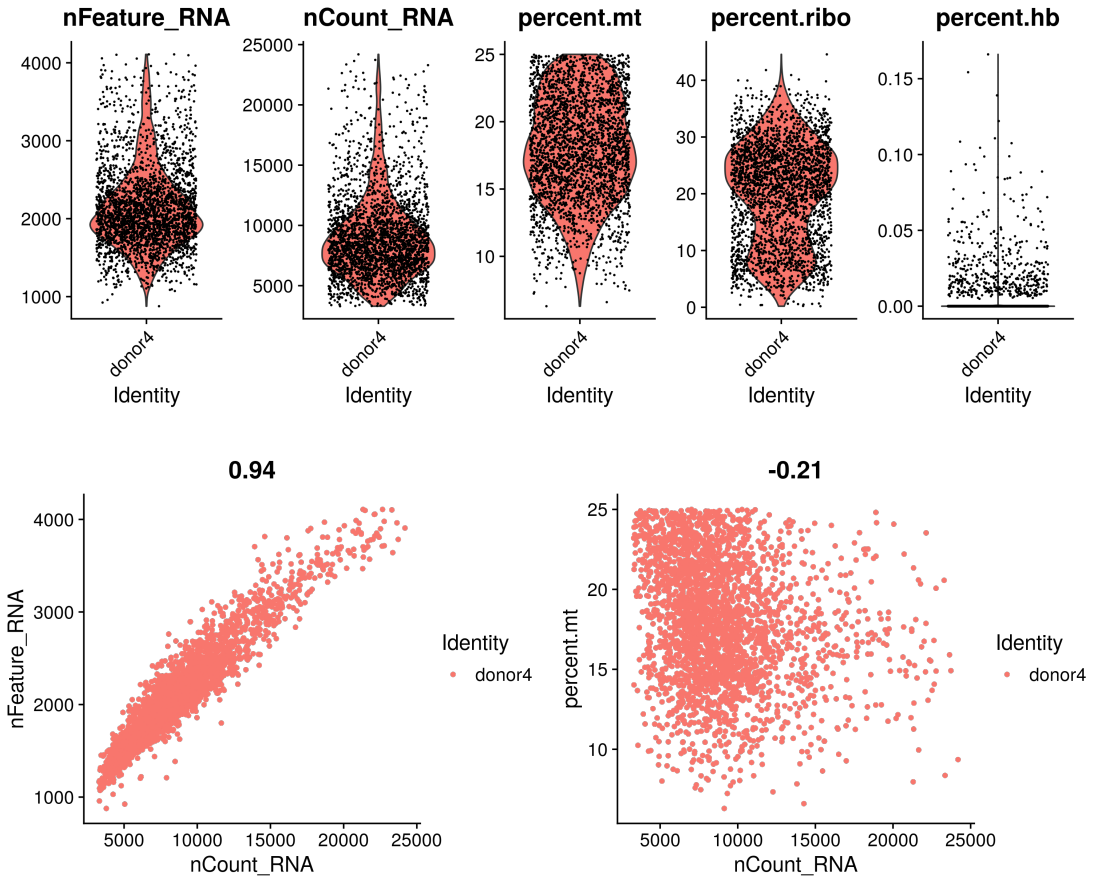


Figure 4: Quality control metrics for donor4 after quality filtering and normalization. Top: distribution of detected features, UMI counts, and mitochondrial percentage. Bottom: relationships between UMI counts and features (left) and mitochondrial content (right).

13.3 Clustering structure and annotation consistency

UMAP embeddings revealed well-resolved immune lineages across all donors (Figure 5). Major PBMC populations—including CD4_T, CD8_T, NK, B-cells, CD14 monocytes, and FCGR3A monocytes—formed discrete and reproducible clusters.

CD4_T cells represented the largest population in all donors (1,316–3,003 cells), followed by CD14_Mono and B-cells (Table 9). CD8_T and NK cells formed smaller but clearly separable clusters. Minor populations including dendritic cells, plasma cells, and platelets were consistently detected at low frequency.

Cluster topology was conserved across donors. T cell subsets localized to adjacent regions, monocyte subsets formed distinct but proximal clusters, and B-lineage cells separated clearly from myeloid populations. No major donor-specific distortions of global manifold structure were observed.

Low-count generic "T_cells" annotations were minimal (0–7 cells per donor), indicating strong agreement between per-cell predictions and majority cluster identities. Overall, clustering and an-

Donor	Cells (before)	Cells (after)	% dropped	MT fails
Donor1	5510	4499	18.35	885
Donor2	5525	4618	16.42	808
Donor3	4612	3951	14.33	541
Donor4	5206	2729	47.58	2400

Table 7: Cell filtering summary per donor.

Donor	Cells (pre)	Cells removed	Cells retained	% removed
Donor1	5510	1506	4004	27.3
Donor2	5525	1181	4344	21.4
Donor3	4612	1072	3540	23.2
Donor4	5206	3322	1884	63.8

Table 8: Cells removed based on mitochondrial content threshold ($> 20\%$).

notation were consistent across donors, supporting downstream pseudobulk and network analyses.

13.4 Differential expression between coarse immune groups

Pseudobulk differential expression was performed across three coarse immune contrasts with balanced donor support (8 pseudobulk samples derived from 4 paired donors per contrast; Table 10).

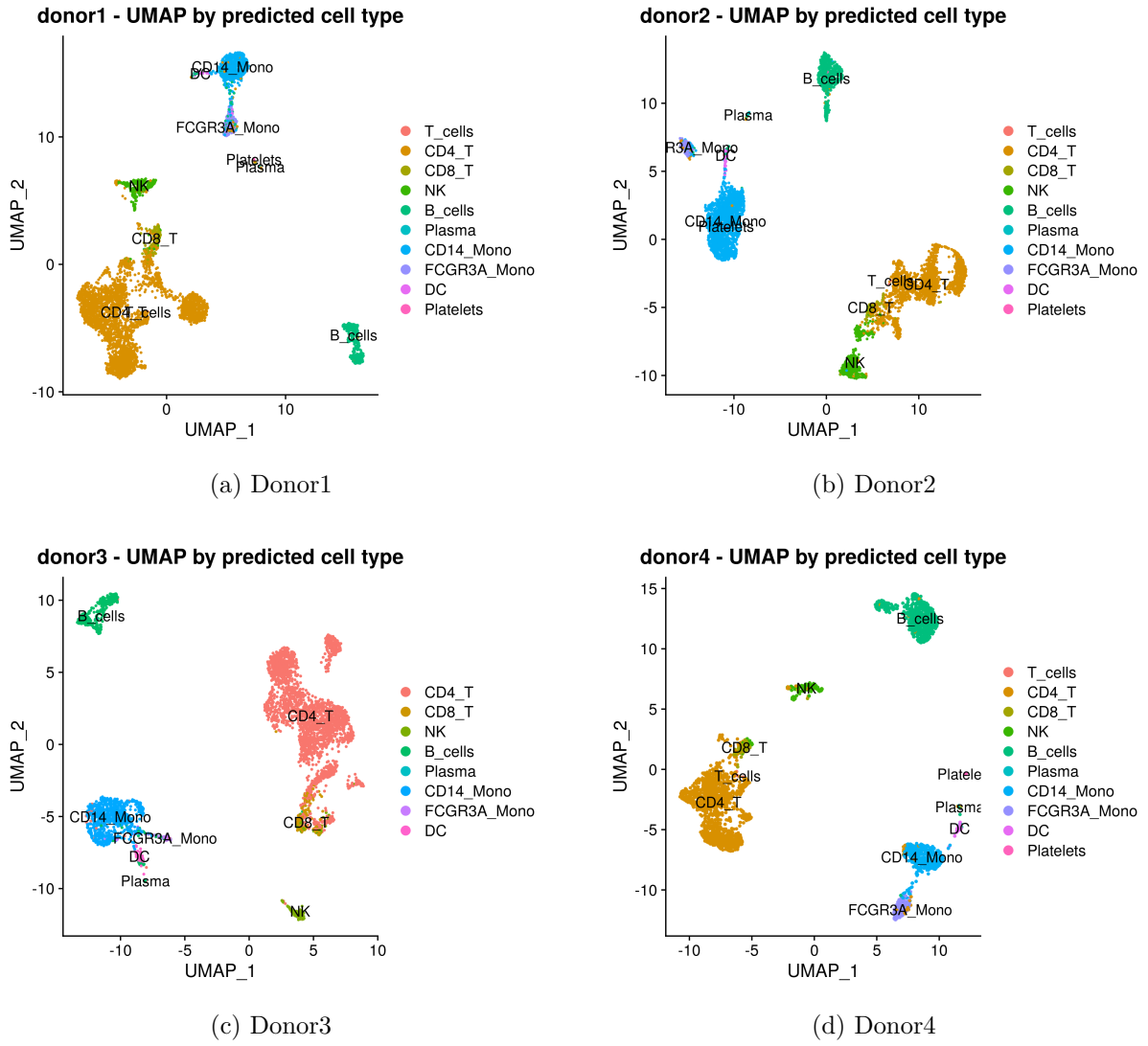
Strict marker genes were most abundant for T-like vs Mono-like (1,366 genes), followed by B-like vs Mono-like (674) and T-like vs B-like (464). In contrast, equivalence testing identified the largest conserved set for T-like vs B-like (2,266 genes), with substantially fewer conserved genes in contrasts involving monocyte populations (641 and 199 genes, respectively).

Volcano plots show clear separation between large-effect marker genes and small-effect conserved genes across contrasts (Figure 6). Effect-size distributions demonstrate that most genes concentrate near zero fold change, with strict markers occupying the distribution tails and equivalence calls localized near the origin (Figure 7).

Together, these results indicate strong transcriptional divergence between lymphoid and myeloid compartments, while also identifying contrast-specific sets of genes with statistically supported minimal differences.

Cross-contrast aggregation highlights the structure of differential and equivalence signals (Table 11). A total of 1,763 genes were identified as strict markers in at least one contrast, whereas only 13 genes were shared as markers across all three contrasts, indicating strong contrast specificity of large-effect transcriptional differences.

In contrast, conserved (equivalence) genes were more stable across comparisons: 2,588 genes were conserved in at least one contrast and 92 were conserved across all three contrasts, suggesting the presence of a shared transcriptional core with statistically supported small effect sizes.



(a) Donor1

(b) Donor2

(c) Donor3

(d) Donor4

Figure 5: UMAP embeddings per donor colored by predicted cell type. Major PBMC lineages form discrete, well-separated clusters across donors.

Overlap between marker and conserved sets was limited (20 genes), consistent with the mutually exclusive effect-size definitions used for classification.

13.5 Pathway enrichment patterns

Pathway enrichment was evaluated per contrast using two complementary approaches: GSEA on ranked gene lists and ORA on discrete strict marker and conserved gene sets. For concise presentation, results are additionally summarized using aggregated ("any contrast") gene sets to illustrate shared pathway-level trends across comparisons.

Hallmark ORA on marker genes identified enrichment of pathways consistent with large, contrast-specific transcriptional differences (Figure 8). In contrast, ORA on conserved genes highlighted pathways over-represented among genes with statistically supported small effects, providing a

Cell type (%)	Donor1	Donor2	Donor3	Donor4
CD4_T	66.8	38.3	62.1	48.2
CD8_T	2.8	1.9	3.5	2.9
T_cells	0.1	0.2	0.0	0.2
NK	6.0	9.6	4.1	5.3
B_cells	8.8	12.8	8.4	22.3
Plasma	0.1	0.3	0.2	0.3
CD14_Mono	12.2	32.8	19.1	14.3
FCGR3A_Mono	2.4	2.8	1.4	5.0
DC	0.9	1.3	1.2	1.2
Platelets	0.0	0.1	0.0	0.4

Table 9: Relative cell-type composition per donor (percent of post-filter cells).

Contrast	Samples	Donors	Strict markers	Conserved (TOST)
T-like vs B-like	8	4	464	2266
T-like vs Mono-like	8	4	1366	641
B-like vs Mono-like	8	4	674	199

Table 10: Pseudobulk DESeq2 + TOST summary per contrast. Samples denote donor-by-group pseudobulks retained after donor pairing and minimum cell-count filtering. Strict markers use padj $< 10^{-10}$ and $|\log_2 \mathrm{FC}| > 3$. Conserved genes are equivalence calls (TOST) with FDR < 0.05 and $|\log_2 \mathrm{FC}| < \delta$ (with expression filtering by baseMean).

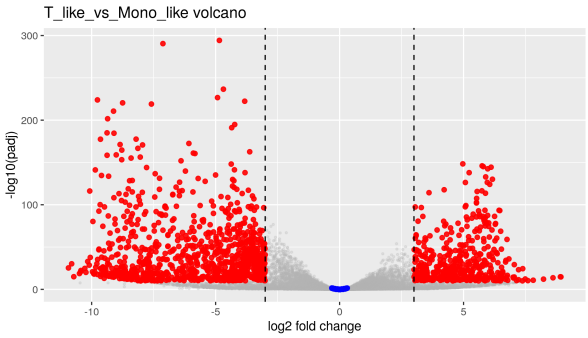
complementary view of stable transcriptional programs across comparisons (Figure 8).

C7 ORA identified immune-context signatures and provided higher-resolution immunologic annotation of the marker and conserved sets (Figure 9). A representative Hallmark GSEA analysis illustrates concordant pathway-level directionality when using continuous ranked statistics (Figure 10).

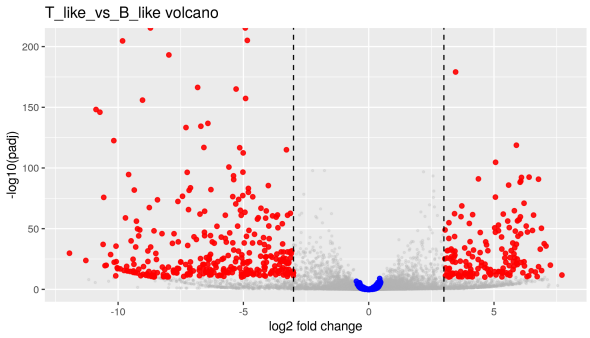
Hallmark over-representation analysis of aggregated strict marker genes was dominated by immune activation and inflammatory signaling pathways, including inflammatory response, allograft rejection, TNFA signaling via NFκB, complement, and cytokine signaling programs (IL2/STAT5 and IL6/JAK/STAT3). These enrichments are consistent with strong transcriptional divergence between lymphoid and myeloid compartments.

In contrast, conserved (equivalence) gene sets were enriched for MYC targets, oxidative phosphorylation, G2M checkpoint, unfolded protein response, protein secretion, and DNA repair. These pathways reflect shared metabolic, proliferative, and housekeeping programs across immune compartments.

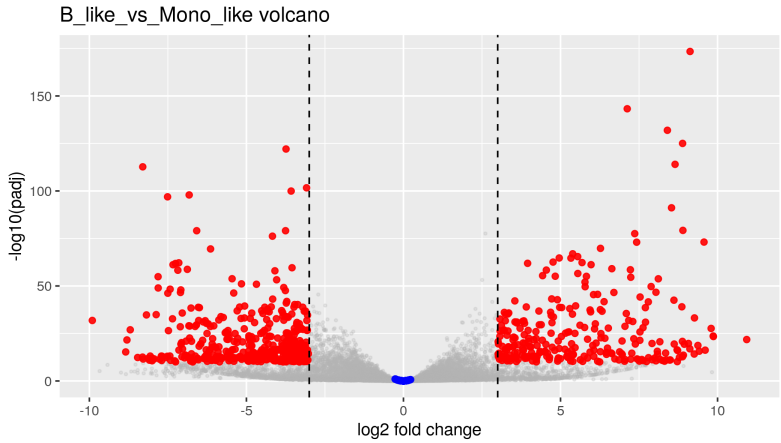
The minimal overlap in pathway-level enrichment between marker and conserved sets supports the effect-size-based separation of contrast-specific versus core transcriptional programs.



(a) T-like vs Mono-like



(b) T-like vs B-like



(c) B-like vs Mono-like

Figure 6: Volcano plots from pseudobulk DESeq2 contrasts. Red points indicate strict markers; blue points indicate genes passing equivalence (TOST). Dashed lines mark the strict marker log2FC threshold.

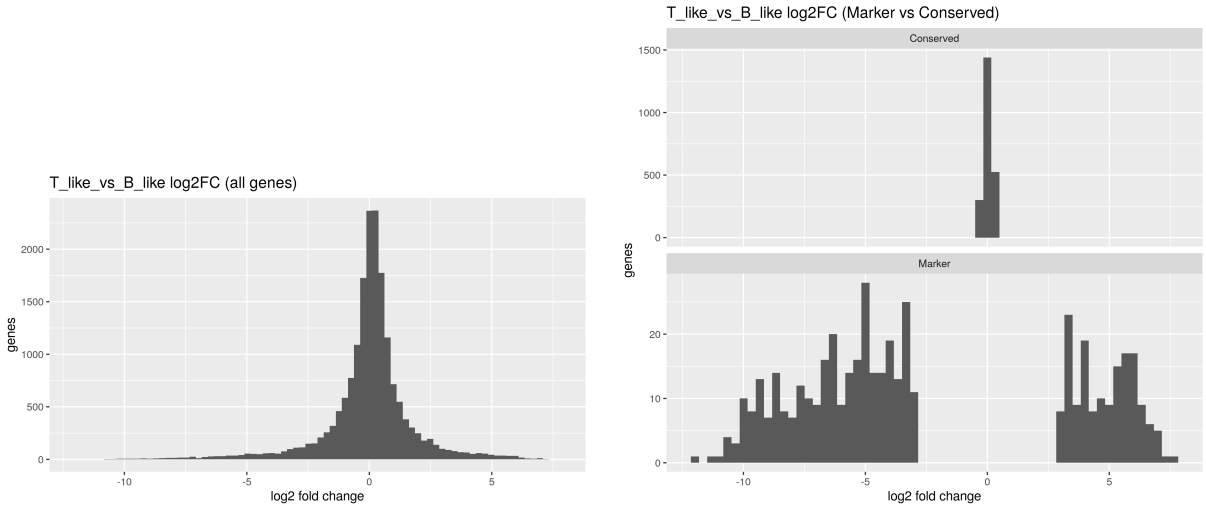
13.6 Consensus co-expression network properties

Consensus co-expression networks were constructed for CD4 T cells, B cells, and CD14 monocytes using donor-level metacell correlations followed by cross-donor support filtering and restriction to the largest connected component (LCC).

Final consensus networks were obtained after donor-level sparsification, cross-donor support filtering (≥ 2 donors), and restriction to the largest connected component. All three cell-type networks formed a single connected component post consensus-LCC (Table 13).

Edge support distributions indicate that most candidate edges were donor-specific prior to filtering, whereas retained consensus edges are predominantly supported at the minimum donor threshold, with a smaller fraction supported by three or four donors (Figure 11).

Consensus edge weights (median Spearman correlation across supporting donors) are concentrated in a moderate positive range with a right tail of higher-correlation edges (Figure 13). Module size distributions are right-skewed, with numerous small modules and fewer large modules (Figure 12).



(a) All genes (b) Marker vs conserved subsets

Figure 7: Distribution of DESeq2 log2 fold-changes for a representative contrast (T-like vs B-like). Most genes concentrate near 0, while strict marker calls occupy the tails; equivalence calls concentrate near 0 by construction.

Set	Genes
Markers (any contrast)	1763
Markers (all contrasts)	13
Conserved (any contrast)	2588
Conserved (all contrasts)	92
Marker ∩ Conserved	20

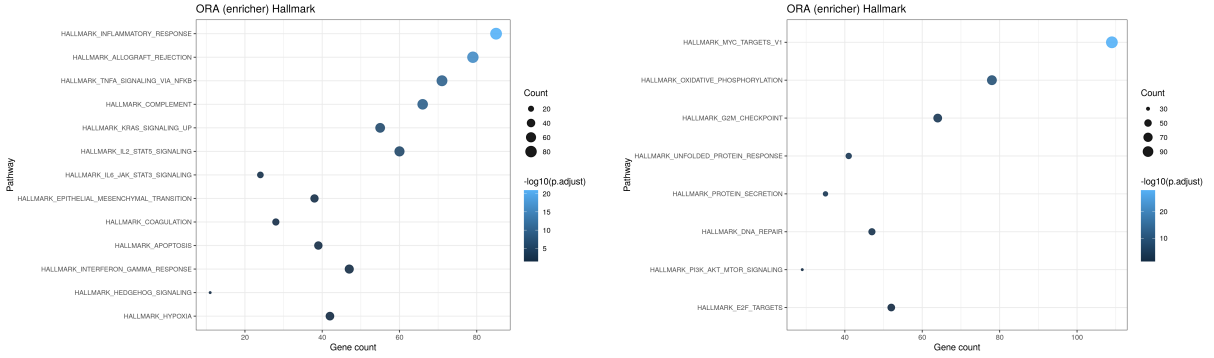
Table 11: Cross-contrast aggregation of strict marker and conserved (equivalent) gene sets.

13.7 Module-level functional enrichment and network visualization with Gephi

Across the three consensus networks (C4, B-cell, and C14-monocyte; Figures 14, 16, and 18), module-level ORA revealed clear functional compartmentalization, with distinct biological programs segregating into specific Leiden modules rather than being diffusely distributed across the networks.

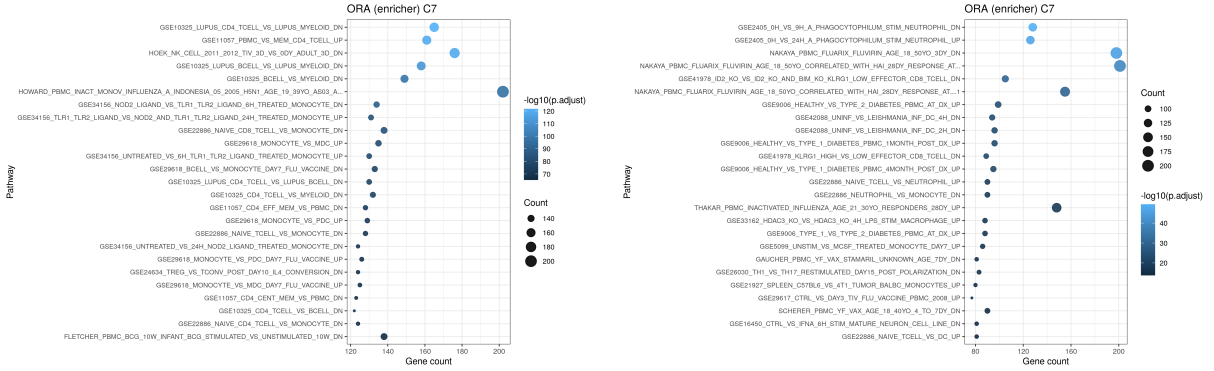
C4 consensus network.
Hallmark enrichment (Figure 15, top) identified modules associated with TNFα signaling via NFκB, interferon (alpha and gamma) responses, inflammatory response, allograft rejection, and MYC targets. These signatures localize to separate modules in the C4 network (Figure 14), indicating structured separation between inflammatory, interferon-driven, and proliferation-associated programs.

C7 enrichment (Figure 15, bottom) further supported this organization, with modules enriched for T-cell activation, vaccine-response datasets, and monocyte/inflammatory contrasts. Strong enrichment signals were concentrated in specific modules within the C4 network (Figure 14), suggesting coherent functional hubs within the consensus network.



(a) Markers (any contrast) (b) Conserved (any contrast)

Figure 8: Over-representation analysis (Hallmark) for aggregated gene sets across contrasts.



(a) Markers (any contrast) (b) Conserved (any contrast)

Figure 9: Over-representation analysis (C7) for aggregated gene sets across contrasts.

B-cell consensus network.

In the B-cell network (Figure 16), Hallmark ORA (Figure 17, top) highlighted dominant TNFα/NFκB signaling, complement activation, inflammatory response, and interferon-related pathways, with MYC targets present in a distinct module. Compared to C4, enrichment was more concentrated into fewer modules, consistent with the modular structure shown in Figure 16.

C7 results (Figure 17, bottom) showed strong enrichment for vaccine-response and infection-associated signatures, as well as contrasts reflecting naive versus memory or activation states. Overall, the modular structure in Figure 16 captures both inflammatory activation and differentiation-related programs within the B-cell context.

C14-monocyte consensus network.

The monocyte network (Figure 18) exhibited the most pronounced innate immune signature. Hallmark enrichment (Figure 19, top) was dominated by TNFα/NFκB signaling, interferon responses, complement, and inflammatory pathways, each localized to specific modules within the network (Figure 18).

C7 enrichment (Figure 19, bottom) showed extensive representation of LPS stimulation, viral

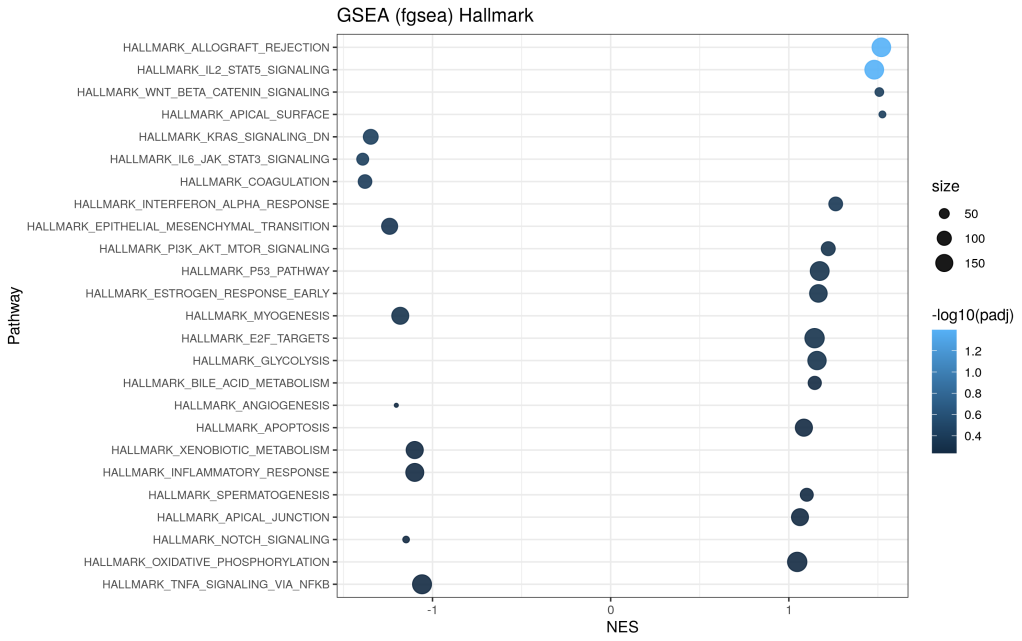


Figure 10: GSEA (Hallmark) for T-like vs B-like ranked by $\log_2$FC.

infection, TLR activation, and vaccine-response datasets, consistent with strong innate immune activation programs. Compared to the other networks, enrichment signals were broader and more uniformly aligned with pathogen-response biology (Figure 19).

Cross-network interpretation.

Across all networks, enrichment patterns (Figures 15, 17, and 19) support the biological validity of the consensus modular structure. Recurrent immune pathways (interferon and TNFα/NFκB signaling) appear across cell types but segregate into context-specific modules within each network (Figures 14, 16, and 18). The C4 network reflects mixed adaptive and inflammatory programs, the B-cell network emphasizes activation and differentiation structure, and the monocyte network is dominated by innate inflammatory responses.

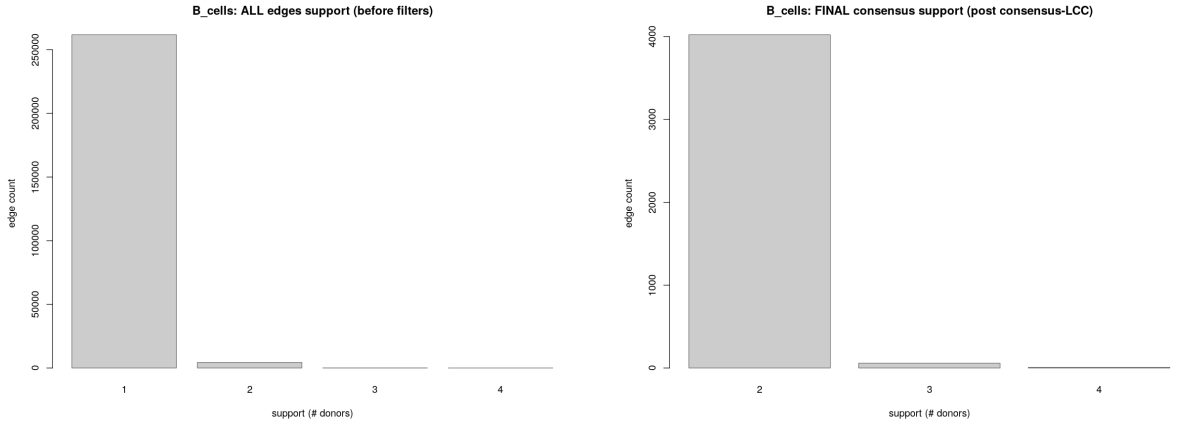
Given the relatively small module sizes in some cases and the use of ORA, these interpretations should remain conservative; however, the enrichment structure is internally coherent and consistent with expected cell-type–specific biology.

Gene set	Markers (any contrast)	Conserved (any contrast)
Immune activation	Inflammatory response	–
Antigen response	Allograft rejection	–
NFκB axis	TNFA signaling via NFκB	–
Innate immunity	Complement	–
Cytokine signaling	IL2/STAT5, IL6/JAK/STAT3	–
Proliferation	E2F targets	G2M checkpoint, E2F targets
Growth programs	KRAS signaling	MYC targets v1
Metabolism	–	Oxidative phosphorylation
Protein handling	–	Unfolded protein response, Protein secretion
Genome stability	–	DNA repair
mTOR axis	PI3K/AKT/mTOR	PI3K/AKT/mTOR

Table 12: Top enriched Hallmark pathways for aggregated strict marker genes and conserved (TOST) genes across contrasts. Markers are dominated by immune activation and inflammatory signaling programs, whereas conserved genes are enriched for shared metabolic, proliferative, and housekeeping pathways.

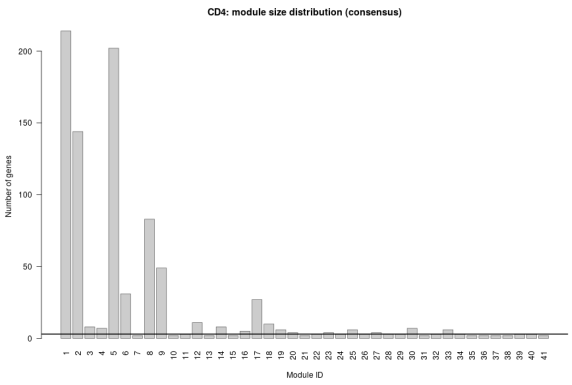
Network	Donors	Universe	Nodes	Edges	Avg deg	Density	Comp	Modules
CD4 T cells	4	–	888	4237	9.54	0.01076	1	–
B cells	4	–	2071	4089	3.95	0.00191	1	–
CD14 monocytes	4	–	2194	4523	4.12	0.00188	1	–

Table 13: Consensus network summary statistics (post consensus-LCC). Universe denotes genes eligible for consensus (present in ≥N donors after donor-level filtering). Nodes and edges refer to the final consensus graph. Comp = number of connected components.

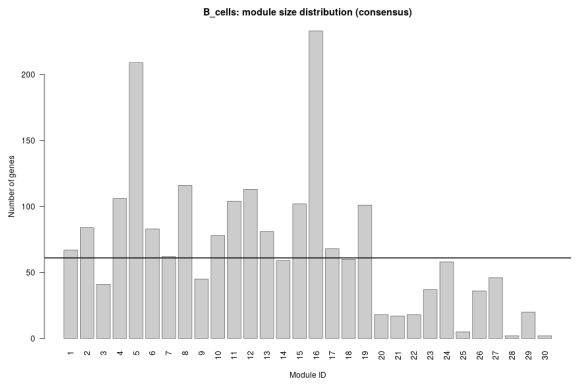


(a) All candidate edges (pre-filter) (b) Final consensus edges (post-filter)

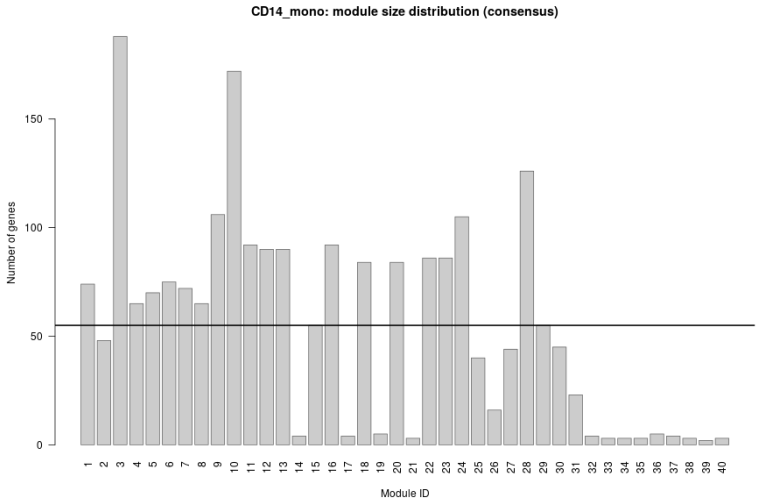
Figure 11: Edge support distributions for a representative network. Support denotes the number of donor-specific networks in which a gene–gene edge is observed.



(a) CD4 T cells



(b) B cells



(c) CD14 monocytes

Figure 12: Module size distributions for post-consensus, post-LCC networks. Bars indicate the number of genes per Leiden module; the horizontal line denotes the median module size.

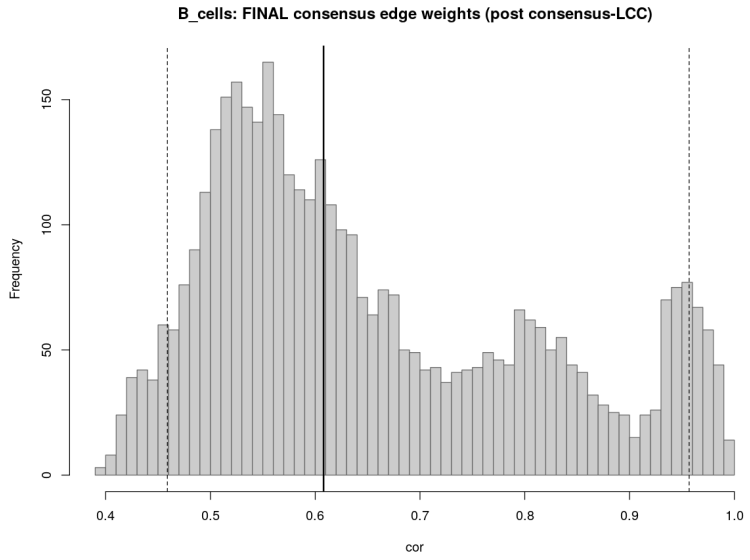


Figure 13: Distribution of consensus edge weights (median Spearman correlation across supporting donors) after consensus filtering and LCC restriction (B cells shown).

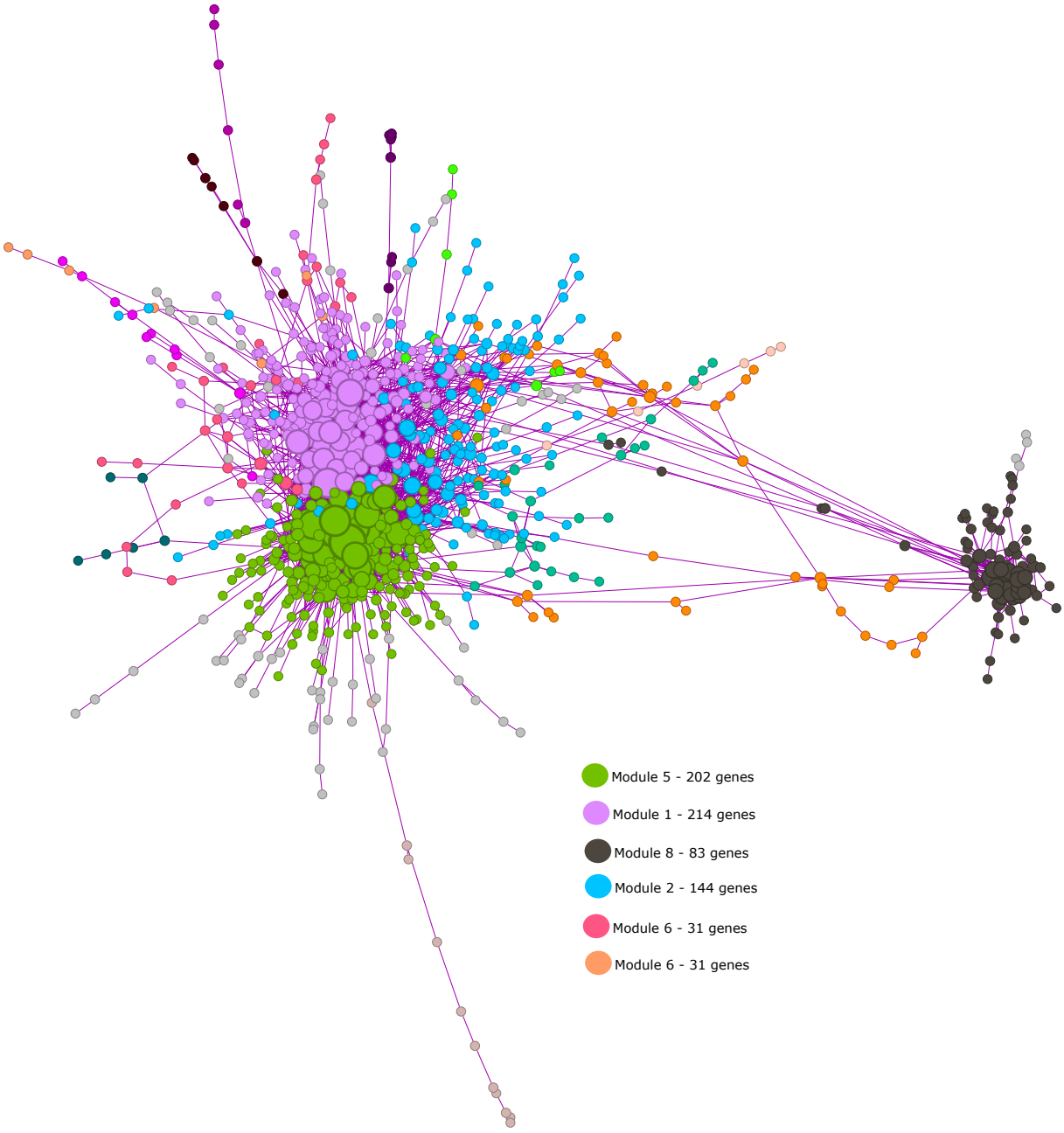


Figure 14: C4 consensus network (post-consensus, post-LCC) colored by Leiden module.

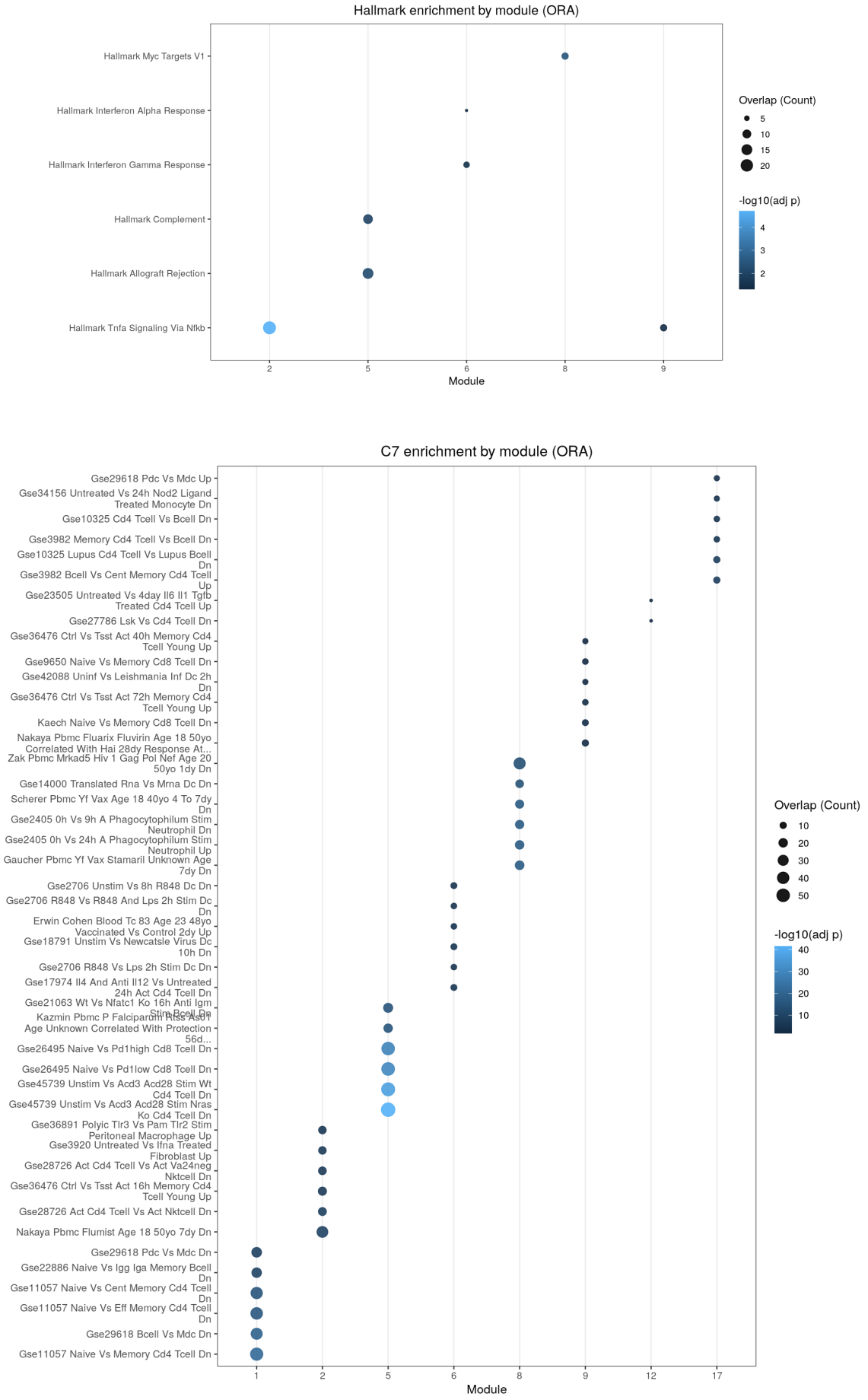


Figure 15: Module-level functional enrichment for the C4 consensus network. Top: Hallmark ORA. Bottom: C7 ORA.

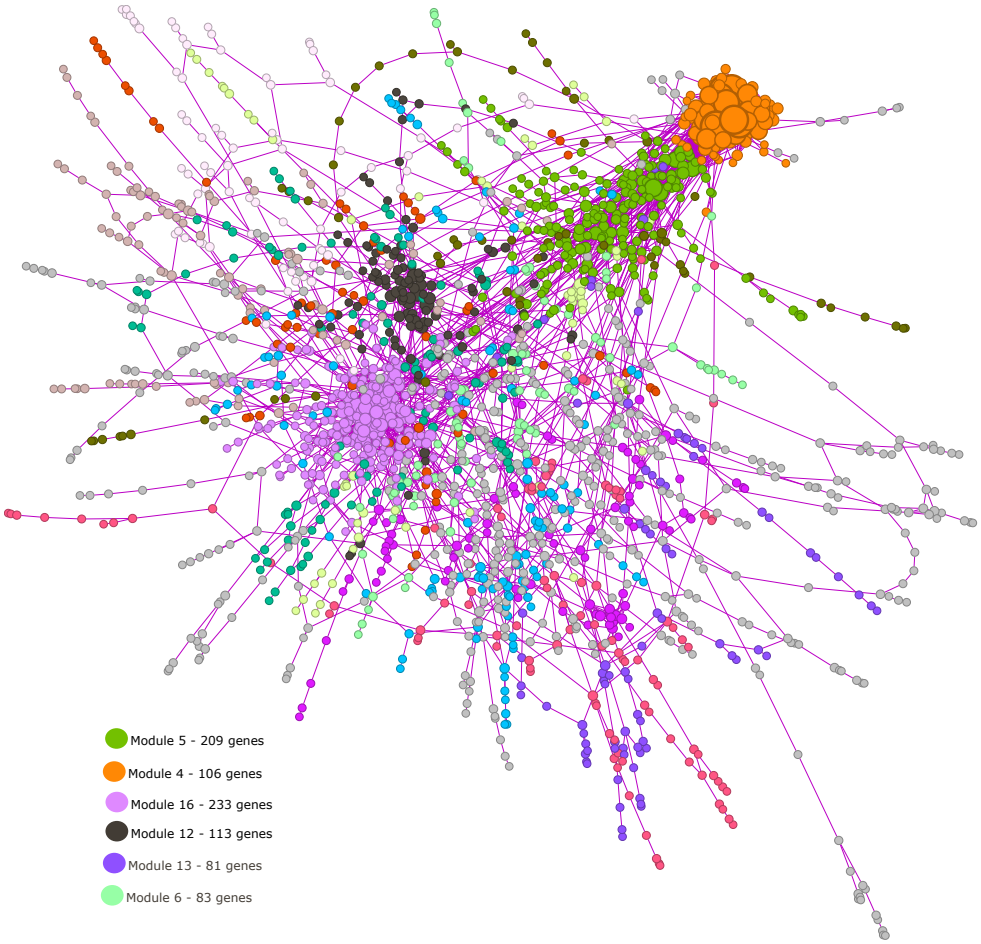


Figure 16: B-cell consensus network (post-consensus, post-LCC) colored by Leiden module.

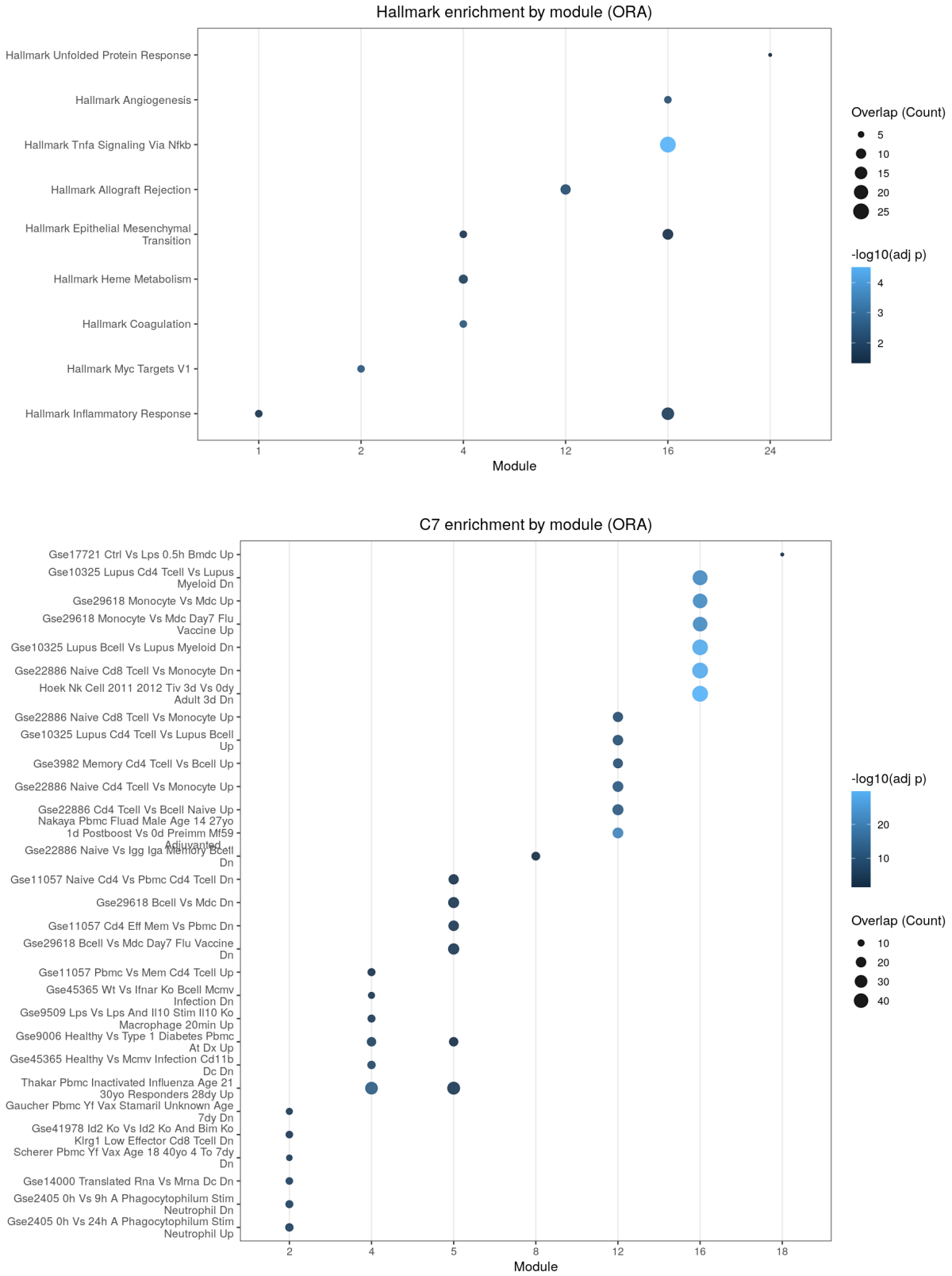


Figure 17: Module-level functional enrichment for the B-cell consensus network. Top: Hallmark ORA. Bottom: C7 ORA.

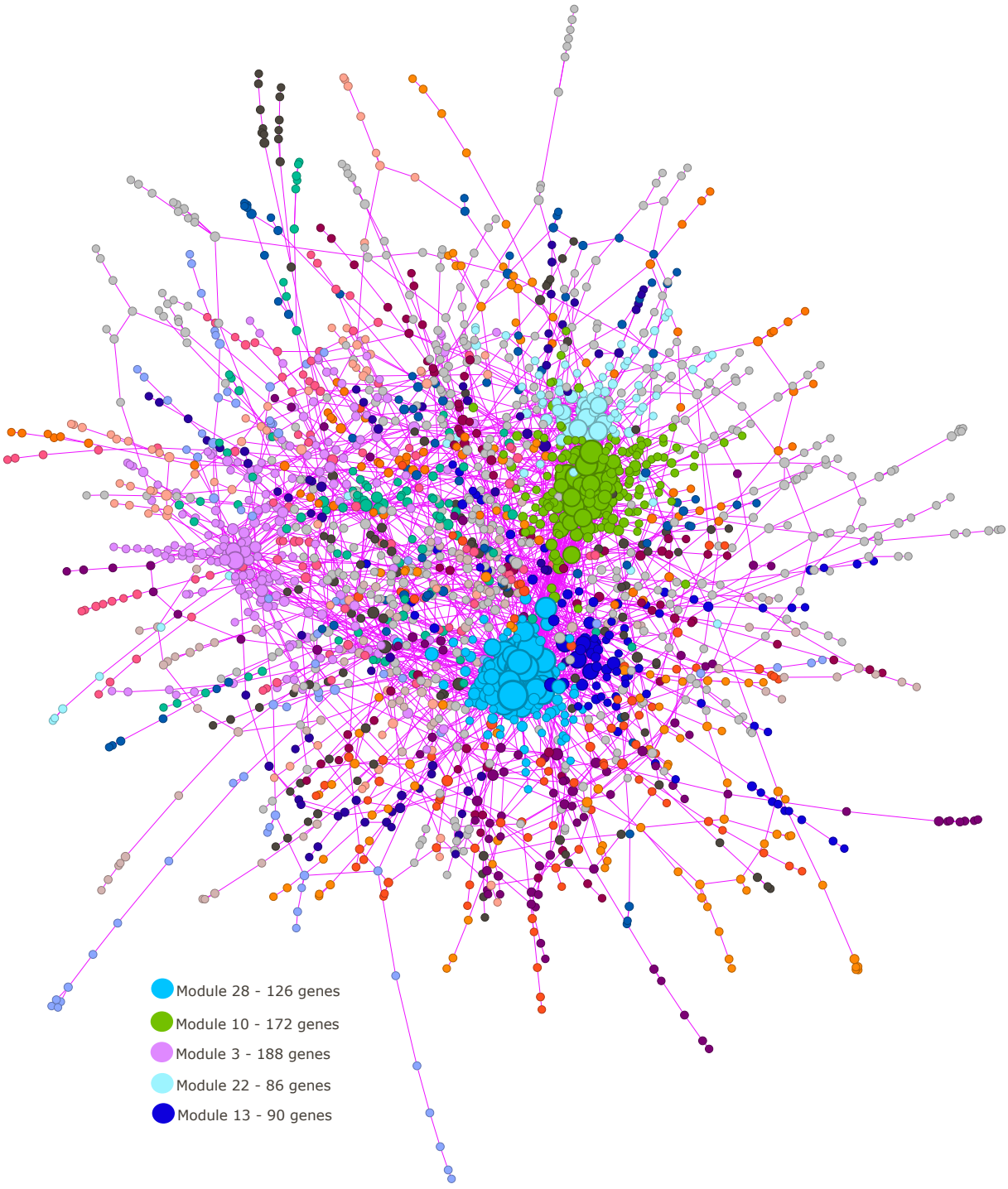


Figure 18: C14-monocyte consensus network (post-consensus, post-LCC) colored by Leiden module.

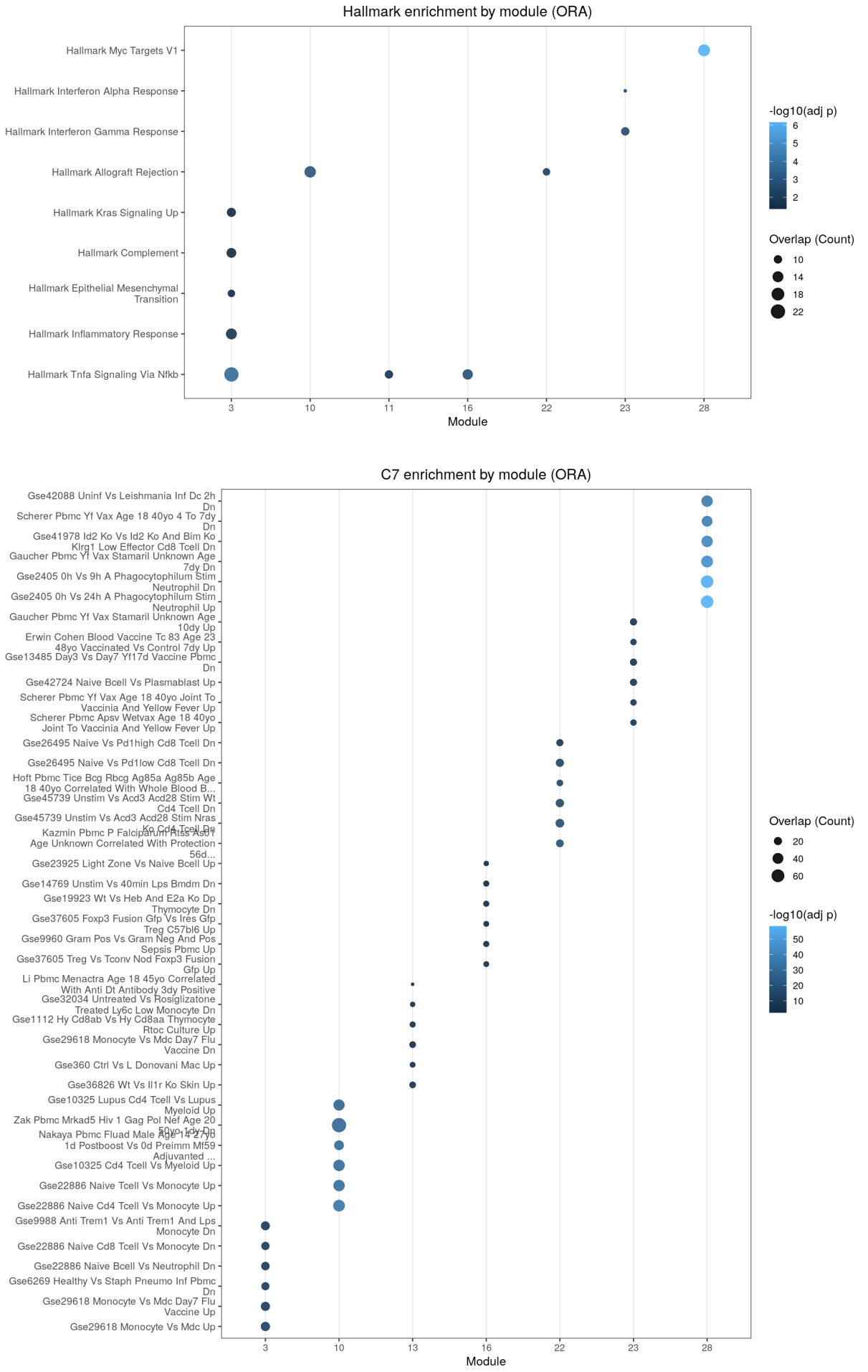


Figure 19: Module-level functional enrichment for the C14-monocyte consensus network. Top: Hallmark ORA. Bottom: C7 ORA.

14 Conclusions

This project demonstrates a complete, reproducible scRNA-seq workflow from raw FASTQs to cell-type annotation, donor-aware pseudobulk inference, pathway enrichment, and cross-donor consensus co-expression networks. The engineering design—Snakemake as the authoritative DAG, a section-based Python wrapper for safe execution, and a digest-pinned Docker image with a version-locked `renv` R environment (including vendored Seurat sources)—provides strong guarantees of portability, auditability, and long-term re-executability. In practice, the pipeline produced stable QC and alignment behavior across donors, recovered expected PBMC lineage structure, and yielded interpretable downstream outputs without reliance on vendor pipelines or opaque defaults.

Biologically, the donor-specific analyses and consensus network integration produced coherent, cell-type–appropriate signals. Pseudobulk DESeq2 contrasts captured strong lymphoid–myeloid divergence while equivalence testing (TOST) separated truly small-effect genes from non-significant ones, enabling a clearer distinction between contrast-specific markers and shared transcriptional cores. At the network level, consensus construction and module detection yielded structured module organization with functionally meaningful compartmentalization: inflammatory and interferon-driven programs, proliferation-related signatures, and innate immune activation pathways segregated into distinct Leiden modules and showed consistent enrichment patterns across cell types (Figures 14–19). Overall, these results support both the technical validity of the workflow and the internal coherence of the analytical outputs, while remaining conservative where module size and ORA limitations warrant restraint.

References

[1] Hao Y, Stuart T, Kowalski MH, et al. *Dictionary learning for integrative, multimodal and scalable single-cell analysis.* Nature Biotechnology. 2023. doi:10.1038/s41587-023-01767-y.

[2] Satija R, Butler A, Hoffman P, Stuart T. *SeuratObject: Data Structures for Single Cell Data.* R package version 4.1.4. 2023.

[3] Liberzon A, Birger C, Thorvaldsdóttir H, Ghandi M, Mesirov JP, Tamayo P. *The Molecular Signatures Database (MSigDB) hallmark gene set collection.* Cell Systems. 2015.

[4] Godec J, Tan Y, Liberzon A, Tamayo P, Bhattacharya S, Butte AJ, Mesirov JP, Haining WN. *Compendium of Immune Signatures Identifies Conserved and Species-Specific Biology in Response to Inflammation.* Immunity. 2016.

[5] Zheng GX, Terry JM, Belgrader P, et al. *Massively parallel digital transcriptional profiling of single cells.* Cell. 2017;171(5):1202–1214.e15.

[6] Villani AC, Satija R, Reynolds G, et al. *Single-cell RNA-seq reveals new types of human blood dendritic cells, monocytes, and progenitors.* Science. 2017;356(6335).

[7] Stuart T, Butler A, Hoffman P, et al. *Comprehensive Integration of Single-Cell Data.* Cell. 2019;177(7):1888–1902.e21.

[8] Hao Y, Hao S, Andersen-Nissen E, et al. *Integrated analysis of multimodal single-cell data.* Nature Biotechnology. 2021;39: 856–865.

[9] Aran D, Looney AP, Liu L, et al. *Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage.* Nature Immunology. 2019;20(2):163–172.