Preprocessing pipeline (CHATGTP GENERATED)

PROMPT:

Share preprocessing\_pipeline.R and prompt: This code is quite complex and I'm trying to understand the process flow. Could you attempt to explain this to me. If you struggle to make sense of the code, feel free to ask me some questions that could help.

**Big picture**

You start with a **gene × cell UMI matrix** and **per-cell metadata**. The pipeline then builds multiple training datasets by varying:

* **Normalization strategy**: global, relative, read\_depth
* **Cell-type subset**: entries in CELL\_TYPE\_SETS (e.g., lcam\_hi, macrophages, or all\_clusters = NULL meaning “all”)
* **Gene subset**: metabolic, nonmetabolic, random (size-matched to the metabolic set)

For each combo, it writes:

* a **wide training matrix** (rows = samples, columns = gene\_cluster features),
* **patient/tissue metadata**, and a **70/30 split** (X\_train/X\_test, y\_train/y\_test),
* **QC totals** by cell type,
* and triggers **fold-change** calculations.

**Key objects (what they look like)**

* umitab (sparse Matrix): genes × cells raw UMI counts.
* cell\_metadata\_final (tibble): cell\_ID, sample\_ID, cluster\_ID for *kept* cells.
* table\_s1 (tibble): sample-level metadata: sample\_ID, patient\_ID, tissue, etc.
* annots\_list (tibble): cluster annotations (e.g., lineage, sub\_lineage).
* hsa01100\_genes (table): KEGG metabolic genes (column Symbol).
* gene\_cluster (string): "<GENE>\_<cluster\_ID>" → becomes a feature (column) in the wide matrix.

**The control flow (chronological)**

1. **Load & prepare data**
   * datasets <- load\_all\_datasets() → pulls in lung\_ldm, table\_s1, annots\_list, hsa01100\_genes.
   * raw\_umitab <- lung\_ldm$dataset$umitab
   * cell\_metadata <- prepare\_cell\_metadata(lung\_ldm, table\_s1, DOUBLETS) → builds clean per-cell metadata and handles doublets.
   * umitab\_filtered <- filter\_umitab(raw\_umitab, cell\_metadata) → drop cells not in metadata.
   * cell\_metadata\_final <- cell\_metadata %>% filter(cell\_ID %in% colnames(umitab\_filtered))
2. **Cell-level CP10K (saved once for re-use)**
   * cp10k\_normalized\_umitab <- apply\_cp10k\_normalization(umitab\_filtered)

Per-cell library size → scale to 10,000, drop empty cells, log1p.

* + Save to disk: cp10k\_normalized\_umitab.rds + cell\_metadata\_final.csv.

1. **Aggregate to sample × cluster × gene (long format)**
   * aggregated\_data <- aggregate\_umitab\_to\_long(cp10k\_normalized\_umitab, cell\_metadata\_final)

Internals:

* + - summary() → triplet nonzeros
    - add gene & cell\_ID
    - **inner\_join** with cell\_metadata
    - group\_by(sample\_ID, cluster\_ID, gene) %>% summarise(count = sum(count))
    - (Important: here “count” is actually the **sum of log-CP10K values** because the source was CP10K+log1p; it’s named count for convenience.)

1. **Run the three approaches** (each iterates over cell-type sets × gene sets and writes outputs)

**a) Global approach**

* + global\_ctnorm\_data <- apply\_celltype\_normalization(aggregated\_data)

Steps:

* + - compute totals per (sample\_ID, cluster\_ID),
    - **single global mean** = mean of those totals,
    - within each (sample, cluster): fraction = count / celltype\_total,
    - rescale: normalized\_count = fraction \* global\_global\_mean.
  + For each cell\_type in CELL\_TYPE\_SETS:
    - subset clusters if not NULL.
    - For each gene\_type in GENE\_TYPE\_SETS:
      * choose genes via filter\_genes\_by\_type() (metabolic / size-matched nonmetabolic / size-matched random).
      * save\_celltype\_total\_counts(...) for QC plots.
      * call **fold-changes** (your code has an elided call like calculate\_fold\_changes\_for\_normalization(gene\_filtered...)).
      * add log\_normalized\_count = log1p(normalized\_count) and gene\_cluster.
      * save\_results(...) → pivot to wide, attach metadata, split patients 70/30, and write all CSVs.

**b) Relative approach**

* + Works **per cell-type subset**:
    - subset cells by selected clusters,
    - apply\_cp10k\_normalization() again **on that subset** (so CP10K reflects only those cells),
    - triplet → join cell\_metadata, aggregate to (sample, cluster, gene) **by summing log values** → column log\_normalized\_count,
    - rename to count and feed into apply\_celltype\_normalization() (rescale to the same single global mean),
    - back to log1p (on normalized\_count), define gene\_cluster,
    - gene set selection, QC totals (treat log\_normalized\_count as normalized\_count just for the totals function), fold-changes, and save\_results().

**c) Read-depth baseline**

* + Start from aggregated\_data (the sample×cluster×gene table built from the CP10K+log matrix).
  + For each subset:
    - choose genes,
    - treat count as normalized\_count (i.e., **no rescaling**),
    - save QC totals, fold-changes, then
    - log1p(normalized\_count) and add gene\_cluster,
    - save\_results().

**The important helper functions (what they really do)**

* aggregate\_umitab\_to\_long(umitab, cell\_metadata)

Converts a sparse matrix to long form per (sample, cluster, gene) by **summing cell-level values**. Because your input here is **already CP10K+log1p**, the “counts” summed are **log-transformed** values, not raw UMIs.

* apply\_celltype\_normalization(long\_data)

Rescales each (sample, cluster) to the **same global average**:

* + compute total per cell type for each sample,
  + take the mean across all these totals,
  + within each group, make fractions, then scale by that global mean.
* filter\_genes\_by\_type(all\_genes, gene\_type)

Picks:

* + metabolic genes present,
  + size-matched sample of non-metabolic genes, or
  + size-matched random genes.

(Size matching gives a fair comparison across gene-set types; set.seed() stabilizes the exact picks.)

* save\_results(long\_data, method\_name, cell\_type, gene\_type)
  + Adds gene\_cluster upstream (or expects it),
  + Pivots to wide training\_matrix.csv,
  + Joins patient\_ID and tissue,
  + Splits **by patient** into 70/30 (to avoid leakage),
  + Writes X\_train.csv, X\_test.csv, y\_train.csv, y\_test.csv, metadata\_train.csv, metadata\_test.csv.
* save\_celltype\_total\_counts(normalized\_data, ...)
  + Totals per (sample, cluster),
  + Attaches tissue and cluster annotations,
  + Writes for QC (output/6. plots/data/total\_counts/...).

**Outputs (where everything lands)**

* CP10K cache:

output/6. plots/data/cp10k/cp10k\_normalized\_umitab.rds,

output/6. plots/data/cp10k/cell\_metadata\_final.csv

* Per-combo artifacts (for each normalization × celltypes × genetype):

output/1. data preprocessing/training datasets/method=<...>/celltypes=<...>/genetype=<...>/

* + training\_matrix.csv
  + training\_matrix\_with\_meta.csv
  + X\_train.csv, X\_test.csv
  + y\_train.csv, y\_test.csv
  + metadata\_train.csv, metadata\_test.csv
* QC totals per combo:

output/6. plots/data/total\_counts/method=<...>/celltypes=<...>/genetype=<...>/celltype\_total\_counts.csv

**Subtleties & “why is it like this?”**

* **“Global” & “Read-depth” both use aggregated\_data built from CP10K+log1p.**

That means you’re **summing log-transformed values** at aggregation time. It’s consistent inside this script, but it’s not the same as “sum raw UMIs, then normalize.” If you intended global/read-depth to reflect raw counts, you’d aggregate on the raw umitab\_filtered instead.

* **Relative approach re-does CP10K on the subset**.

Because library size scaling is per cell, doing it on the subset can change the normalization relative to the “global” CP10K matrix. This is fine (by design) but worth being aware of.

* **Fold-change calculation** is called but not shown (calculate\_fold\_changes\_for\_normalization(gene\_filtered...)), so whatever that function expects as input (raw vs log, per-sample vs per-patient) determines the interpretation of those outputs.
* **Randomness** is controlled via set.seed(42) and set.seed(43) so gene sampling and patient splits are reproducible.

**A mental model (DAG-ish)**



**A few focused questions (optional, to confirm intent)**

1. **Global/read-depth source**: Did you *intend* those branches to operate on **CP10K+log1p aggregated values** (as they do now), or on **raw UMI aggregates**? If the latter, we’d point aggregate\_umitab\_to\_long() at umitab\_filtered instead.
2. **Fold changes**: What inputs does calculate\_fold\_changes\_for\_normalization() expect (raw vs log; per sample vs grouped)? I can sanity-check that we’re feeding it the right structure and scale.
3. **prepare\_cell\_metadata filter rules**: Besides doublets, does it do any other filtering (dead cells, QC thresholds)? Useful for understanding why some cells are missing.
4. **Metabolic gene source**: Is hsa01100\_genes$Symbol guaranteed to be HGNC symbols that match your umitab rownames? If not, we may want a join that is robust to synonyms.

If you share the fold-change function signature (or confirm the global/read-depth intent), I can refine the flow to match your exact design and, if you want, propose a tiny refactor that keeps outputs identical but makes the “raw vs CP10K” pathways explicit.