Single-cell RNA-seq classification of LMNA-related patient/control status using TensorFlow/Keras

Vladislav Karpe

02.01.2026

Abstract

This work trains TensorFlow/Keras neural networks to classify **patient vs control status** from single-cell RNA-seq (scRNA-seq). We use the public GEO dataset GSE269705 (12 GSM sequencing libraries) measured across seven differentiation time points (days 0, 2, 4, 9, 16, 19, 30). Each cell is represented by a gene-expression vector. Labels (patient vs control and differentiation day) are inherited from the originating GSM library. Preprocessing follows a pipeline implemented under RAM constraints: per-cell quality control (min 200 genes, max 6000 genes, max 20% mitochondrial reads), random subsampling up to 3000 cells per sample, total-count normalization to 10,000 counts per cell, log1p transform, selection of 1000 highly-variable genes by variance, and per-gene scaling by standard deviation with clipping at 10. To prevent preprocessing leakage, HVG selection and scaling factors are fit on **training GSMs only** and then applied unchanged to validation/test. We avoid library leakage via GSM-level splits, and we reduce day–genotype confounding with day-matched evaluation. We report results for a genotype-only multilayer perceptron (Model A) and a multitask variant with an auxiliary day-prediction head (Model B), and we include a small sweep over core hyperparameters (depth, epochs, batch size, activation, optimizer).

1 Introduction

Mutations in *LMNA* (lamin A/C) are linked to laminopathies with cardiac and skeletal muscle involvement. Disease-associated effects can be reflected in cellular gene expression programs. Single-cell RNA sequencing (scRNA-seq) quantifies gene expression at the level of individual cells, enabling analysis of heterogeneous cell populations.

In this project, we frame a supervised classification task: given a single cell's expression vector, predict whether the cell originates from a patient or a control sample. The dataset also includes multiple differentiation time points. We test whether adding the time point as an auxiliary prediction target improves patient/control classification.

A practical challenge in scRNA-seq machine learning is avoiding information leakage. Cells sequenced within the same library share library-specific technical signatures (e.g., sequencing depth, ambient RNA, and batch effects). Here, one sequencing library corresponds to one GEO `GSM` sample. If individual cells were split randomly across train/validation/test, the model could exploit these library "fingerprints," leading to overly optimistic test performance. Therefore, we use a `GSM`-level split: all cells from a given `GSM` are assigned to exactly one partition, and `GSM`s are disjoint across train/validation/test.

Contributions. (i) A RAM-safe pipeline that converts scRNA-seq count matrices into an ML-ready dataset. (ii) A small hyperparameter sweep in TensorFlow/Keras (epochs, batch size, depth, activation, optimizer). (iii) Sample-level evaluation of a patient/control classifier (Model A) and a time-aware multi-task variant (Model B), including a split scheme that reduces day–genotype confounding.

2 Methods

2.1 Dataset

We used a public scRNA-seq dataset from NCBI GEO under accession **GSE269705**. The study contains **12 sequencing libraries** (GEO accessions `GSM8325046`–`GSM8325057`). In this report, each `GSM` is treated as one **sample/library**.

10x-style files. Each GSM is stored in the common "10x Genomics sparse-matrix format" layout: `matrix.mtx.gz` (sparse count matrix), `barcodes.tsv.gz` (cell IDs), `features.tsv.gz` (gene names). The matrix is sparse because most gene counts are zero for a given cell.

2.2 Labels

Each cell inherits labels from its source `GSM`:

- **Genotype/condition** $y_{\text{geno}} \in \{0, 1\}$, where 0=Control and 1=Patient.
- **Time point** $y_{\text{day}} \in \{0, \ldots, 6\}$, encoding the differentiation day via the mapping $(0, 2, 4, 9, 16, 19, 30) \rightarrow (0, \ldots, 6)$.

2.3 Train/validation/test splitting

2.3.1 GSM-level splitting (no library leakage)

All splits are performed at the `GSM` (library) level to avoid within-library leakage.

2.3.2 Two split schemes

We evaluate two split schemes implemented in `prepare_dataset.py`:

Original split (hard-coded GSM lists).

- Test: `GSM8325052` (Control, day 19) and `GSM8325056` (Patient, day 16)
- Validation: `GSM8325049` (Control, day 9) and `GSM8325054` (Patient, day 0)
- Train: `GSM8325046, 5047, 5048, 5050, 5051, 5053, 5055, 5057`

This split keeps GSM disjoint, but it mixes differentiation days across partitions, which can induce day–genotype confounding.

Day-matched split (confound-reduced). To reduce day–genotype confounding, we define a day-matched split by selecting two days: a `test_day` and a `val_day`.

For `test_day=19` and `val_day=0`, the split becomes:

- Test: `GSM8325052` (Control, day 19) and `GSM8325057` (Patient, day 19)
- Validation: `GSM8325046` (Control, day 0) and `GSM8325054` (Patient, day 0)
- Train: all remaining `GSM`s from days 2, 4, 9, 16, 30

2.4 Preprocessing

All preprocessing is implemented in `prepare_dataset.py` with fixed defaults (seed 42). Key parameters:

- Max cells per sample: `max_cells_per_sample = 3000`
- QC thresholds: `min_genes=200`, `max_genes=6000`, `max_mito_pct=20.0`
- Normalization target: `target_sum = 10000`
- Feature filter: remove genes expressed in < 10 cells (`min_cells_per_gene=10`)
- Highly variable genes: `n_hvg = 1000`
- Scaling clip: `scale_clip = 10.0`

2.4.1 Quality control and subsampling

For each cell we compute detected gene count and mitochondrial fraction (genes starting with `MT-`). Cells are kept if:

$$200 \leq \texttt{n\_genes} \leq 6000, \quad \texttt{pct\_mt} \leq 20\%.$$

To control RAM/time, we randomly subsample up to 3000 cells per GSM (fixed seed).

2.4.2 Normalization and log transform

Counts are normalized per cell to a fixed total (`target_sum=10000`) and log-transformed:

$$x \leftarrow \log(1 + x).$$

2.4.3 Highly variable genes (HVG)

To reduce dimensionality, we select 1000 genes with the highest variance on log-normalized values (after removing genes expressed in fewer than 10 cells). These HVGs are used as the model input features.

2.4.4 Scaling

Each gene is scaled by dividing by its standard deviation and clipped to $[-10, +10]$. We do not mean-center features to avoid densifying sparse matrices.

2.4.5 Two-pass, RAM-safe implementation

To avoid concatenating all data in memory, preprocessing runs in two passes:

- Pass 1: iterate over training GSMs, apply QC + normalization/log1p, and accumulate per-gene statistics to select HVGs and compute scaling factors.
- Pass 2: iterate over all GSMs, apply the HVG list and scaling factors from Pass 1, and build the final sparse matrix.

2.5 Models

Both models use the same backbone implemented in `train.py`. Default training hyperparameters are: learning rate 10^{-3}, width 512, depth 2, dropout 0.3, ReLU, Adam, batch size 256, max epochs 40, early stopping with patience 8 (monitoring validation AUC).

2.5.1 Model A: genotype-only MLP

Model A predicts genotype with a multilayer perceptron: Dense $\rightarrow$ BatchNorm $\rightarrow$ Activation $\rightarrow$ Dropout repeated for each hidden layer, followed by a sigmoid output. Loss is binary cross-entropy.

2.5.2 Model B: multitask genotype + time point

Model B shares the same backbone but adds a 7-way softmax head predicting the time point. The combined loss is:

$$L = L_{\text{geno}} + \alpha L_{\text{day}},$$

where α corresponds to `loss_w_day` (default 0.3).

2.6 Hyperparameter variations (assignment requirement)

To satisfy the assignment requirement to test different configurations, we ran a small sweep (via `sweep.py`) changing:

- Depth: 1 vs 2
- Epoch budget: 20 vs 50 (with early stopping active)
- Batch size: 128 vs 512
- Activation: ReLU vs tanh
- Optimizer: Adam vs SGD

This is not an exhaustive search. It is a targeted set of variations covering the required categories.

2.7 Evaluation metrics

We report:

- **Loss** (cross-entropy objective),
- **Accuracy** at threshold 0.5,
- **ROC-AUC** (threshold-free ranking metric).

Plots (loss curve, accuracy curve, ROC curve, confusion matrix) are generated by `evaluate.py` and included in the submission ZIP as PNG.

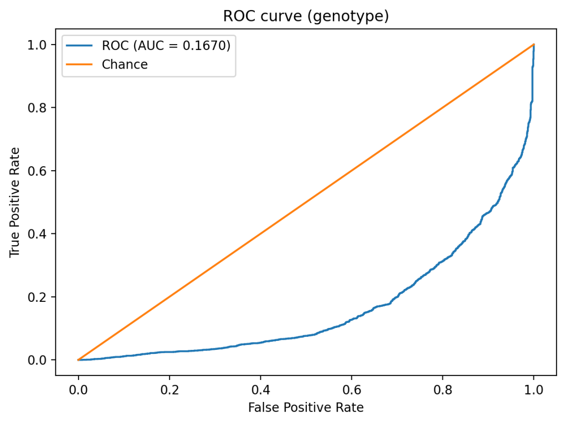
3 Results

3.1 Split scheme matters: original vs day-matched

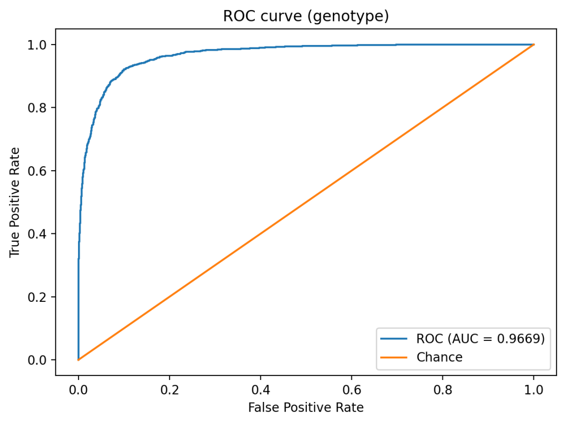
Table 1 shows genotype performance for several splits. The original mixed-day split produces an **inverted ROC** (AUC well below 0.5), consistent with the model learning a day-associated shortcut that does not generalize. Day-matched splits yield substantially higher AUC, indicating improved generalization when day–genotype confounding is reduced. Accuracy is reported at a fixed threshold of 0.5 and can therefore look poor even when AUC is high.

Run (split)	Model	n_{test}	Test AUC	Test acc	Genotype loss	Combined loss
original	A	4130	0.239	0.279	10.056	10.056
day-matched (test=0, val=16)	A	4630	0.820	0.598	1.968	1.968
day-matched (test=16, val=0)	A	4741	0.944	0.803	0.734	0.734
day-matched (test=19, val=0)	A	3287	0.871	0.555	1.708	1.708
original	B	4130	0.141	0.200	6.819	7.002
day-matched (test=0, val=16)	B	4630	0.641	0.505	3.181	7.435
day-matched (test=16, val=0)	B	4741	0.960	0.908	0.376	4.077
day-matched (test=19, val=0)	B	3287	0.936	0.773	0.681	4.387

Table 1: Split-scheme comparison. The original mixed-day split yields inverted performance. Day-matched splits substantially improve generalization. For Model B, the combined loss includes the auxiliary day objective; genotype loss refers to the genotype head.



(a) Original split (inverted ROC).



(b) Day-matched split (test day 16, val day 0).

Figure 1: ROC curves for Model A under two split schemes.

3.2 Hyperparameter sweep (Model A on day-matched test=19, val=0)

We ran a small sweep covering the required configuration categories. Table 2 shows that AUC varied from 0.888 to 0.936 across tested settings. The best AUC in this sweep was obtained using tanh activation (AUC 0.936). Increasing batch size to 512 also improved AUC relative to the baseline 128-batch run in this split.

Run	Depth	Batch	Act / Opt	Test AUC	Test acc / loss
A_d1_e20_b128_relu_adam	1	128	ReLU / Adam	0.907	0.834 / 0.559
A_d2_e20_b128_relu_adam	2	128	ReLU / Adam	0.913	0.848 / 0.520
A_d1_e50_b128_relu_adam	1	128	ReLU / Adam	0.907	0.834 / 0.559
A_d1_e20_b512_relu_adam	1	512	ReLU / Adam	0.929	0.854 / 0.521
A_d1_e20_b128_tanh_adam	1	128	tanh / Adam	**0.936**	0.847 / 0.401
A_d1_e20_b128_relu_sgd	1	128	ReLU / SGD	0.888	0.815 / 0.447

Table 2: Model A hyperparameter sweep results (day-matched split: test day 19, validation day 0). Early stopping was active, so the requested epoch budget does not imply the model trained all epochs.

3.3 Model B does not consistently improve genotype AUC

Model B was designed to test whether auxiliary supervision (predicting the differentiation time point) improves genotype generalization. Across the evaluated day-matched splits, Model B improved genotype AUC for test day 16 (0.960 vs 0.944) and test day 19 (0.936 vs 0.871), but underperformed on test day 0 (0.641 vs 0.820). This suggests that the auxiliary objective can help in some held-out-day settings, but the effect is not robust across splits in this small dataset (12 GSM libraries).

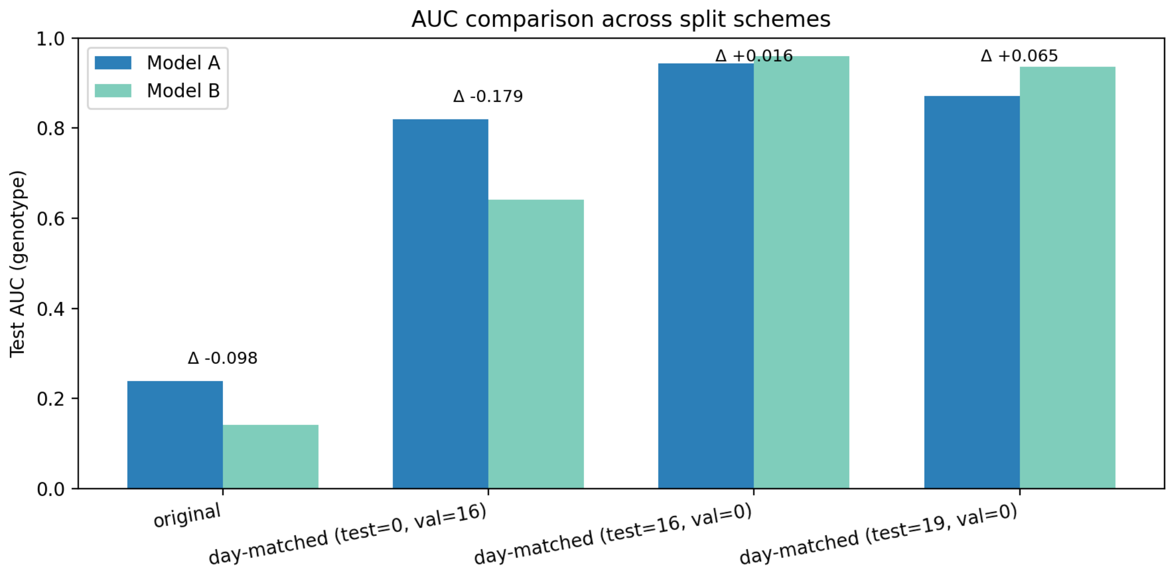


Figure 2: Genotype ROC-AUC across split schemes for Model A and Model B. Δ denotes (Model B − Model A). Model B improves AUC for some held-out days (16 and 19) but not consistently (day 0).

4 Discussion

4.1 Main takeaway: split design dominates performance

The biggest observed factor was the split scheme. Even with GSM-level splitting (no shared libraries across partitions), mixing differentiation days across train/val/test can create a shortcut where the network learns day-associated expression structure that correlates with genotype in the training set. When this association changes in validation/test, performance can collapse and even invert (AUC $<$ 0.5). Day-matched splitting reduces this confound by forcing evaluation on a held-out day with both genotypes present.

4.2 Limitations

- **Small number of libraries.** With only 12 GSMs, estimates of generalization can be noisy and sensitive to which day is held out.
- **Biological replication.** "Patient vs control" may partly reflect line or batch specific effects. Stronger conclusions would require more donors.
- **Non-independence of cells.** Cell-level metrics can overstate confidence because cells within the same GSM library are correlated. The effective number of independent units is closer to the number of GSM libraries.

5 Conclusion

We implemented a RAM-safe scRNA-seq preprocessing pipeline and trained TensorFlow/Keras neural networks to classify patient/control status from single-cell expression vectors. Evaluation was performed with GSM-level splitting to avoid library leakage and with an additional day-matched split scheme to reduce day–genotype confounding. Under day-matched splits, a simple MLP (Model A) achieved high ROC-AUC (up to 0.94), while a multitask genotype+day model (Model B) did not consistently improve genotype performance. Overall, the results highlight that careful split design can matter more than model complexity for scRNA-seq classification.

References

[1] NCBI Gene Expression Omnibus (GEO): GSE269705. <https://www.ncbi.nlm.nih.gov/geo/>

[2] TensorFlow. <https://www.tensorflow.org/>

[3] Keras. <https://keras.io/>

Appendix: Reproducibility

Here are command snippets that reproduce the main results shown in Section 3. The full pipeline instructions are in `README.md`.

Dataset preparation.

```
python src/prepare_dataset.py --out_dir data/processed_original  --split_scheme original

python src/prepare_dataset.py --out_dir data/processed_dm_t0_v16  --split_scheme day_matched
↪ --split_test_day 0  --split_val_day 16
python src/prepare_dataset.py --out_dir data/processed_dm_t16_v0  --split_scheme day_matched
↪ --split_test_day 16 --split_val_day 0
python src/prepare_dataset.py --out_dir data/processed_dm_t19_v0  --split_scheme day_matched
↪ --split_test_day 19 --split_val_day 0
```

Training.

```
python src/train.py --data_dir data/processed_original   --out_dir outputs/runs/orig_A
↪ --model A
python src/train.py --data_dir data/processed_original   --out_dir outputs/runs/orig_B
↪ --model B
```

```
python src/train.py --data_dir data/processed_dm_t0_v16   --out_dir outputs/runs/dm_t0_v16_A
↪ --model A
python src/train.py --data_dir data/processed_dm_t0_v16   --out_dir outputs/runs/dm_t0_v16_B
↪ --model B
python src/train.py --data_dir data/processed_dm_t16_v0   --out_dir outputs/runs/dm_t16_v0_A
↪ --model A
python src/train.py --data_dir data/processed_dm_t16_v0   --out_dir outputs/runs/dm_t16_v0_B
↪ --model B
python src/train.py --data_dir data/processed_dm_t19_v0   --out_dir outputs/runs/dm_t19_v0_A
↪ --model A
python src/train.py --data_dir data/processed_dm_t19_v0   --out_dir outputs/runs/dm_t19_v0_B
↪ --model B
```

Evaluation plots.

```
python src/evaluate.py --run_dir outputs/runs/orig_A
python src/evaluate.py --run_dir outputs/runs/orig_B
python src/evaluate.py --run_dir outputs/runs/dm_t0_v16_A
python src/evaluate.py --run_dir outputs/runs/dm_t0_v16_B
python src/evaluate.py --run_dir outputs/runs/dm_t16_v0_A
python src/evaluate.py --run_dir outputs/runs/dm_t16_v0_B
python src/evaluate.py --run_dir outputs/runs/dm_t19_v0_A
python src/evaluate.py --run_dir outputs/runs/dm_t19_v0_B
```