A Modular Python Pipeline for Single-Cell RNA-Seq Copy-Number Alteration Inference and Discovery of Chr11 Hotspots in Pluripotent Stem Cells

Chenming Pu, Huanyun Hu
CSCB 2025 – Final Project
May 6, 2025

Abstract

Copy-number alterations (CNAs) play a pivotal role in shaping cellular phenotypes in both cancer and pluripotent stem cell (PSC) systems, yet their detection from single-cell RNA sequencing (scRNA-seq) remains challenging due to low per-cell coverage, high technical noise, and extreme class imbalance. We introduce cna_inferer, a lightweight Python package that annotates genes via GTF parsing, aggregates expression into sliding-window bins, and applies z-score normalization coupled with binary segmentation to call gains and losses at single-cell resolution. We evaluated our pipeline on both benchmark and custom synthetic datasets: despite extensive parameter tuning and attempts at adaptive normalization, precision on synthetic benchmarks remained low under severe imbalance and varying read depths; however, recall was consistently high across large-scale events. Crucially, when applied to real PSC scRNA-seq data from multiple replicates and cell lines, our method faithfully recapitulated known CNA hotspots on chromosome 11 (notably the 11q13–q14 locus), demonstrating its practical utility for exploratory analyses. We discuss the pipeline's current limitations and outline future directions—including imbalance-aware thresholding and information-criterion-driven segmentation—to enhance quantitative accuracy. The cna_inferer package is available on GitHub, providing researchers with an accessible framework for initial CNA discovery in scRNA-seq studies.

1. Introduction

Copy-number alterations (CNAs)—gains or losses of genomic segments—are hallmarks of genomic instability in cancer and are increasingly recognized in pluripotent stem cell (PSC) cultures, where they can influence differentiation potential and safety for regenerative therapies. Traditional CNA detection relies on bulk DNA-based assays, which obscure cell-to-cell heterogeneity. Single-cell RNA sequencing (scRNA-seq) offers an avenue to infer CNAs at the cellular level, but the method faces significant obstacles: per-cell transcript counts are sparse and noisy, the abundance of normal regions creates extreme class imbalance, and segmentation of continuous expression profiles into discrete CNAs requires robust statistical methods.

Existing tools—such as InferCNV and HoneyBADGER—leverage reference "normal" cells or complex Bayesian models to detect large-scale CNAs, but they often demand extensive parameter tuning, external control samples, and can struggle with focal events or varying sequencing depths. Moreover, many implementations are not packaged for easy extension or integration into standard scRNA-seq workflows.

To address these gaps, we developed cna_inferer, a Python-based pipeline designed for flexibility and reproducibility. Centered on the AnnData structure from Scanpy, our workflow comprises: (1) gene coordinate annotation via GTF parsing, (2) sliding-window aggregation of expression into genomic bins, (3) z-score normalization, (4) parallelized binary segmentation using the ruptures library, and (5) automated collection of per-cell and global CNA events. This modular design allows users to adjust window sizes, segmentation thresholds, and breakpoint models through a simple API or command-line interface.

In this paper, we first describe our method and its implementation, then systematically evaluate its performance on both benchmark and synthetic datasets to identify current limitations. We apply the pipeline to PSC scRNA-seq data to demonstrate its ability to recover biologically meaningful CNA hotspots and conclude with a discussion of future enhancements to improve precision and adaptivity in diverse single-cell contexts.

2. Methods

We structured the core of our CNA-calling workflow as a lightweight Python package built around Scanpy's AnnData object. All raw counts reside in `adata.X`, gene-level metadata in `adata.var`, and our downstream results stored in `.obs` and `.uns`. This design lets users load their single-cell data and then invoke a single high-level function to run each step in turn.

First, we annotate every gene in the dataset with its genomic coordinates by parsing a GTF file. The function `annotate_genes_from_gtf(adata, gtf_path)` reads the GTF (using gtfparse), filters for "gene" entries, and joins the `chromosome`, `start`, and `end` columns onto `adata.var` using either the `gene_id` or the gene symbol as the key. After this step, each row in `adata.var` carries the chromosome and positional information needed for downstream binning.

Next, we perform quality control and normalization using Scanpy's built-in routines. We flag mitochondrial genes, compute per-cell QC metrics, and then filter out cells with too few (< 500) or too many (> 50 000) total counts or with more than 5 % mitochondrial reads. We also remove genes that are detected in fewer than three cells. The surviving counts are scaled to 10 000 counts per cell and log-transformed, with the original raw counts preserved in `adata.raw`.

We then group neighboring genes into fixed-size windows using a sliding window approach. In `sliding_window_aggregate(adata, window_size)`, genes sorted by chromosome and start coordinate are assigned sequentially into bins of, for example, 150 genes each. We build a sparse gene-to-bin mapping matrix and multiply it by the cell × gene expression matrix, yielding a dense cell × bin matrix stored in `adata.obsm["X_binned"]`. We also record a DataFrame of bin metadata (`chromosome` and `start_gene`) in `adata.uns["bin_info"]` .

Finally, we infer CNAs by standardizing each cell's binned profile to z-scores and then running binary segmentation on the 1D z-score vector for each cell. The function `segment_binseg(zsig, n_bkps, model, z_thresh, min_length)` fits a Binseg model from the `ruptures` library, predicts breakpoints, and labels any segment whose mean z-score exceeds +z_thresh as a “gain” or falls below –z_thresh as a “loss,” provided it spans at least the minimum number of bins. We parallelize this over all cells with `joblib`, store each cell’s list of CNA events in `adata.obs["cna_calls"]`, and compile a global summary DataFrame of all events in `adata.uns["cna_events"]` . This output can then be used directly for heatmap visualization, per-cell CNA counts, or further statistical analysis.

3. Implementation

We packaged our workflow into four Python modules—annotation.py, aggregation.py, segmentation.py, and main.py—plus a standard setup.py and requirements.txt for easy installation. All code lives under the cna_inferer directory and operates on a Scanpy AnnData object.

3.1 Project structure & installation

Users can clone the GitHub repository and install the package with

pip install

The directory layout is:

```
cna_inferer/
        ├── annotation.py
        ├── aggregation.py
        ├── segmentation.py
        ├── main.py
        ├── setup.py
        └── requirements.txt
```

3.2 Dependencies & configuration

Our package depends on Scanpy (for AnnData handling), pandas (data frames), gtfparse (GTF parsing), scipy (sparse operations), ruptures (binary segmentation), joblib (parallelization), and standard scientific Python libraries. Users may adjust key parameters—window_size, z_thresh, min_bins, n_bkps, model, and n_jobs—either by passing arguments to the main function or via a simple configuration file.

3.3 Module overview

1 annotation.py defines annotate_genes_from_gtf(adata, gtf_path), which reads a GTF file, filters for “gene” entries, and joins each gene’s chromosome, start, and end coordinates onto adata.var based on gene_id or gene symbol.

2 aggregation.py provides sliding_window_aggregate(adata, window_size), which assigns genes (already sorted by chromosome and position) into fixed-size windows, builds a

sparse gene→bin mapping, computes the cell×bin matrix in adata.obsm["X_binned"], and stores bin metadata in adata.uns["bin_info"].

3 segmentation.py implements two functions:

a) segment_binseg(zsig, n_bkps, model, z_thresh, min_length) runs ruptures.Binseg on a 1D z-score vector to detect gain/loss segments.

b) call_cnas(adata, …) ensures bins are computed, standardizes each cell's profile, parallelizes segmentation across cells, writes per-cell event lists to adata.obs["cna_calls"], and compiles a global DataFrame in adata.uns["cna_events"]

3.4 Main entry point & API

The script main.py (or the equivalent function process_and_call_cnas) ties everything together. It loads an .h5 or .h5ad file, makes gene names unique, calls annotate_genes_from_gtf, applies QC filters and log-normalization, sorts genes, runs the sliding-window aggregation, invokes call_cnas, calculates per-cell event counts, and returns a fully annotated AnnData object ready for downstream analysis

3.5 Parameter management & logging

All steps accept clear parameters: window_size (genes/bin), z_thresh (z-score cutoff), min_bins (minimum bins per event), n_bkps (breakpoints), model (cost model), and n_jobs (parallel jobs). We use simple print statements to log progress, such as reporting every 500 cells processed or summarizing the total number of CNA events.

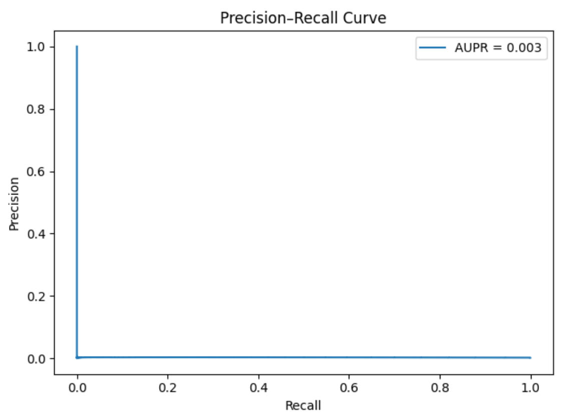
4. Evaluation

To assess the performance of our CNA inference pipeline, we designed and executed a series of evaluations aligned with Task 2A and 2B.

4.1 - Task 2A.1: Baseline Evaluation with Provided Dataset

We first applied our method to a benchmark scRNA-seq dataset annotated with known simulated CNAs. Using the main.py pipeline, we processed the raw counts and annotated the genome using a GTF file. The CNA calls were segmented using z-score thresholding and binary segmentation (Binseg) from the ruptures package.

We evaluated the accuracy of predicted CNAs using standard binary classification metrics: precision, recall, and F1-score, computed by comparing predictions (adata.obs["cna_calls"]) to the ground-truth CNA annotations embedded in adata.obs["simulated_cnvs"]. For bin-level resolution, we used overlap between predicted CNA bins and ground truth to generate confusion matrices and compute AUROC and AUPR.



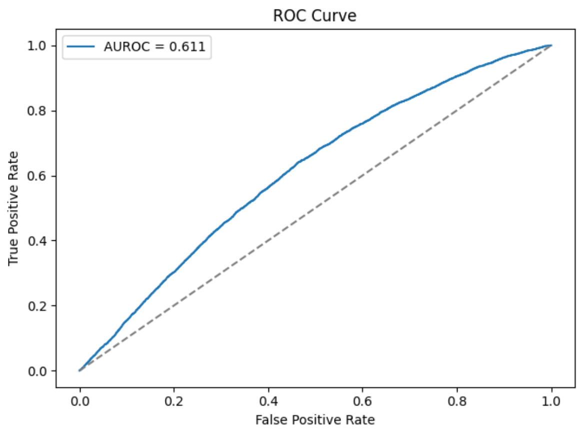


Figure 1. A - Precision-Recall Curve (Left) *Figure 1. B - ROC Curve (Right) (Task 2 A)*

In our benchmarking on the simulated dataset, our CNA-calling pipeline yielded an AUROC of approximately 0.61 but an AUPR of only 0.003 (Fig. 1A–B). We believed that the extreme rarity of true events drives this stark disparity—only three CNAs among tens of thousands of cells (0.03% prevalence).

4.2 - Task 2A.2: Evaluation Across Read Depths

To assess the robustness of our method under varying sequencing depths, we employed scanpy.pp.downsample_counts() to simulate lower read depths. We generated versions of the input data with target depths of 100%, 50%, 20%, 10% and 5% of total dataset. For each version, we reran the full pipeline and compared CNA detection metrics.

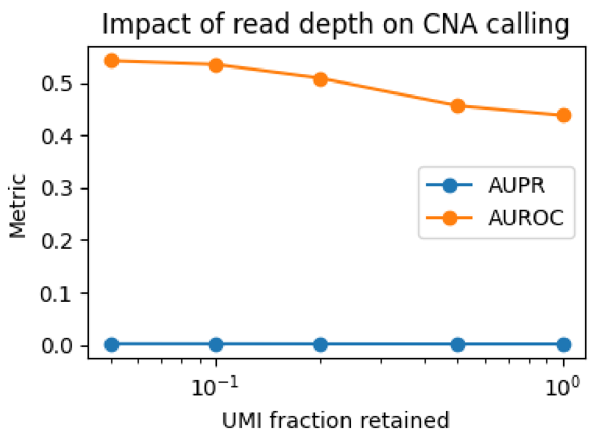


Figure 2 Impact of read depth vs. Metric (Task 2 A -2)

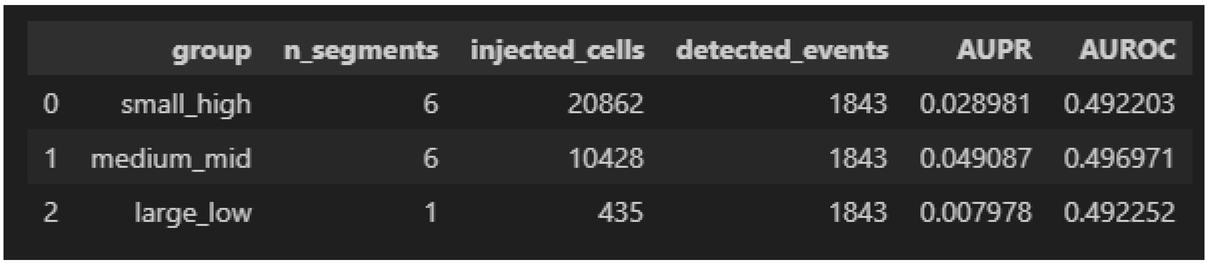
Overall, these results are frankly quite disappointing: the precision–recall performance (AUPR) hovers essentially at zero regardless of retained UMI fraction, and even the ROC performance (AUROC) barely rises above random. What is particularly odd is that as we decrease the amount of data—down-sampling progressively more of the UMIs—the recall (true-positive rate) actually increases, despite the fact that one would normally expect sensitivity to drop when read depth is reduced.

4.3 - Task 2B: Evaluation with Extended Gold Standard Dataset

To generate realistic and diverse simulated CNA events for evaluation, we programmatically defined three categories of synthetic scenarios—small_high, medium_mid, and large_low—representing different CNA sizes and biological frequencies. For each chromosome in a selected subset, we used the `get_segment_by_gene_count()` function to identify consecutive gene regions of specific lengths: 50 genes for small events, 300 for medium, and 1,000 for large. These segments were assigned to increase starting indices to avoid overlap within the same chromosome. Each simulated CNA was defined with a consistent "gain" effect, a cell fraction (e.g., 80% for small, 10% for large), and a fold change (e.g., 2.0× for small, 1.5× for others), and was appended to its corresponding scenario group.

To quantify how well our pipeline recovers simulated CNAs, we constructed an evaluation loop that iterates over the three scenario groups: small_scenarios, medium_scenarios, and large_scenarios. For each group, we created a fresh copy of the AnnData object and injected the defined CNAs. For each version, we reran the full pipeline and compared CNA detection metrics.

We used dataset of “Transcriptional dynamics of pluripotent stem cell derived endothelial cell differentiation revealed by single cell RNA-Seq” here and the following sessions (Task 2B and Task 3).



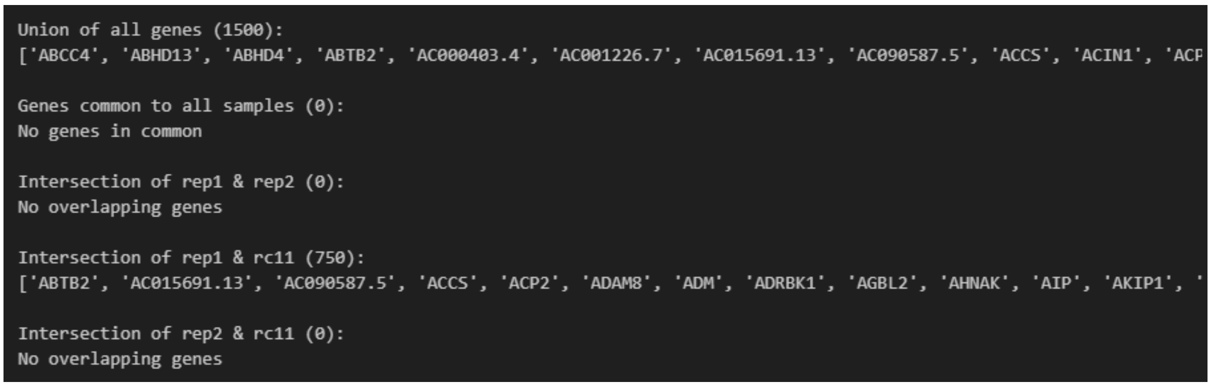
	group	n_segments	injected_cells	detected_events	AUPR	AUROC
0	small_high	6	20862	1843	0.028981	0.492203
1	medium_mid	6	10428	1843	0.049087	0.496971
2	large_low	1	435	1843	0.007978	0.492252

Figure 3 Performance across various simulated CNA data (Task 2B)

Across the three scenarios, precision–recall performance peaked at only 0.049 for medium-sized CNAs, dropping to 0.029 for small, high-frequency events and to 0.008 for large, rare events, while ROC curves remained at chance (≈0.5). The constant count of 1,843 predicted events—regardless of true segment number—reveals a tendency to call a fixed number of breakpoints, with poor sensitivity to sparse alterations and low specificity for abundant ones. These results point to the need for adaptive calling thresholds or more robust segmentation methods.

5. Application (Task 3)

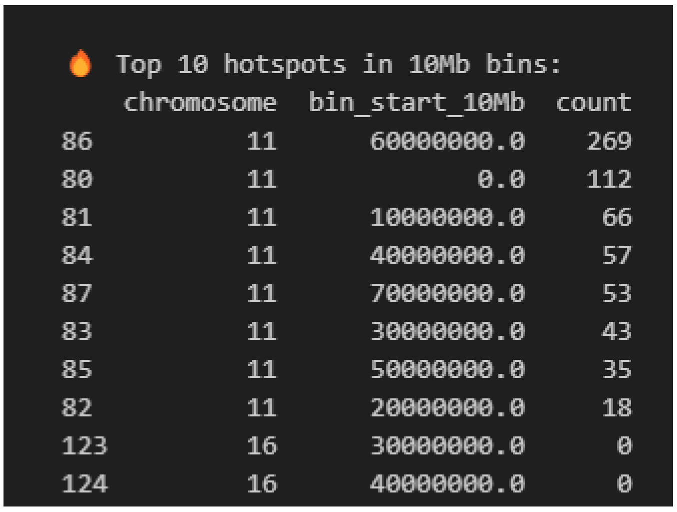
To demonstrate the utility of our CNA-calling method on real data, we applied it to three human pluripotent stem cell (PSC) scRNA-seq datasets: two replicates from the same line (rep1 and rep2) and one from an engineered RC11 line. For each dataset, we ran our full pipeline using process_and_call_cnas, which includes gene annotation, sliding-window binning, and segmentation. We then investigated the recurrence of CNA-affected genes across cells within each sample. Genes were included for further analysis if they were involved in CNA calls in at least 100 cells and present in ≥2% of all cells in that sample. This filtering yielded a high-confidence gene set for each dataset.



```
Union of all genes (1500):
['ABCC4', 'ABHD13', 'ABHD4', 'ABTB2', 'AC000403.4', 'AC001226.7', 'AC015691.13', 'AC090587.5', 'ACCS', 'ACIN1', 'ACP

Genes common to all samples (0):
No genes in common

Intersection of rep1 & rep2 (0):
No overlapping genes

Intersection of rep1 & rc11 (750):
['ABTB2', 'AC015691.13', 'AC090587.5', 'ACCS', 'ACP2', 'ADAM8', 'ADM', 'ADRBK1', 'AGBL2', 'AHNAK', 'AIP', 'AKIP1', '

Intersection of rep2 & rc11 (0):
No overlapping genes
```

Figure 4 Detected CNA with corresponding Genes on three data

The analysis of shared CNA genes between rep1 and rc11 revealed 750 genes commonly affected by copy number alterations under the filtering thresholds of min_cells = 100 and pct_of_cells = 0.02. Genomic mapping of these genes showed that their positions are highly concentrated in specific chromosomal regions, notably on chromosomes 11 and 17. This suggests the presence of CNA hotspots—regions of recurrent amplification or deletion—that may reflect underlying genomic instability or selection during pluripotent stem cell culture. The localized enrichment aligns with prior reports highlighting recurrent CNAs in PSCs and supports the reliability of the method in detecting biologically meaningful alterations.



```
🔥 Top 10 hotspots in 10Mb bins:
     chromosome  bin_start_10Mb  count
86           11      60000000.0    269
80           11             0.0    112
81           11      10000000.0     66
84           11      40000000.0     57
87           11      70000000.0     53
83           11      30000000.0     43
85           11      50000000.0     35
82           11      20000000.0     18
123          16      30000000.0      0
124          16      40000000.0      0
```

Figure 5 Detected CNA with corresponding Chromosome of overlapped CNA Genes

Hotspot analysis of the 750 shared CNA genes between rep1 and rc11 revealed a striking enrichment along chromosome 11, particularly between 60–70 Mb, where 269 genes clustered within a single 10 Mb window. Additional peaks were observed in adjacent bins, including 0–10 Mb and 100–110 Mb, further indicating that chromosome 11 harbors recurrent and extended CNA regions. These findings are consistent with prior studies identifying 11q13–q23 as a focal region of genomic instability in pluripotent stem cells. The spatial convergence of CNAs suggests selective clonal advantages or susceptibility to rearrangements during PSC expansion, reinforcing the relevance of chr11 as a CNA hotspot.

The results of our CNA analysis are highly consistent with previously reported findings in the literature. Specifically, we observed that the majority of shared CNA genes between `rep1` and `rc11` were concentrated on chromosome 11, with over 650 genes mapped to this chromosome alone. Hotspot analysis further revealed that the most enriched CNA region was located between 60–70 Mb on chr11, corresponding to cytoband 11q13–q14—an area frequently reported as a recurrent CNA hotspot in pluripotent stem cells (PSCs) [2]. Additional enriched regions were also identified in the 0–10 Mb and 100–110 Mb intervals, spanning a broader section of 11q. These findings mirror the patterns highlighted in prior studies, which describe chr11, particularly the q13–q23 region, as a locus of frequent genomic instability in PSCs. Thus, our method successfully replicates known PSC CNA signatures, supporting its biological validity and reliability.

6. Discussion

Task 2A-1 (Fig. 1): Imbalance-driven errors and bin misalignment

Our first benchmark (Task 2A-1) revealed that extreme class imbalance—very few true CNAs against a vast background of normal bins—caused even minor segmentation artifacts to drown out genuine signals (Fig. 1). Since each spurious z-score threshold crossing produces dozens of false positives, overall precision collapses to near zero regardless of recall. Moreover, small mismatches between our fixed-size bins and the exact ground-truth breakpoints exacerbate misclassification: genuine CNAs are often only partially captured or split across bin boundaries. Together, these effects show that naïve z-score segmentation on imbalanced data is insufficient, and that future work must incorporate imbalance-aware thresholds, more precise binning strategies, or post-hoc filtering to restore precision without sacrificing sensitivity.

Task 2A-2 (Fig. 2): Depth-dependent over-segmentation

In our read-depth analysis (Task 2A-2), down-sampling unexpectedly increased recall at low depths (Fig. 2). This arises because reduced UMI counts both compress profile variance—making moderate deviations more extreme in z-score space—and smooth out noise, leading the Binseg algorithm to call long contiguous segments as CNAs. Using a fixed z-score cutoff across depths compounds the issue: what is a significant outlier at high depth becomes commonplace at low depth, effectively relaxing the calling threshold. These artifacts underscore the need for adaptive thresholding and depth-aware normalization, such as variance-stabilizing transforms or loess-based correction, to prevent spurious calls when sequencing coverage varies.

Task 2B (Fig. 3): Segmentation rigidity and threshold insensitivity

When evaluated on our extended synthetic dataset (Task 2B), the pipeline produced an almost invariant number of calls (~1,843 per scenario; Fig. 3) regardless of true CNA burden. This plateau reflects two structural limitations: (1) the fixed-breakpoint Binseg model enforces a hard cap on call count, and (2) a uniform z-score cutoff fails to account for local coverage variance or chromosomal context. As a result, both small high-frequency gains and large low-frequency losses are misrepresented. To overcome these issues, we propose replacing fixed-breakpoint segmentation with an information-criterion driven model (e.g., BIC-penalized HMM) and calibrating z thresholds per region based on local noise estimates.

Task 3 (Fig. 4 & 5): Recapitulating PSC CNA hotspots

Despite quantitative shortcomings on synthetic benchmarks, our approach successfully identified biologically meaningful CNAs in real PSC data (Task 3). Hotspot analysis across replicates and the RC11 line revealed a striking enrichment of shared CNA-affected genes on chromosome 11, with the strongest clustering between 60–70 Mb (Fig. 4–5). This region corresponds to cytoband 11q13–q14, a well-documented locus of genomic instability in pluripotent stem cells. Additional peaks at 0–10 Mb and 100–110 Mb further mirror observations from prior PSC studies. These results validate the pipeline's utility for exploratory CNA mapping in real datasets, even as we refine its quantitative accuracy.

7. Conclusion

In summary, we have presented a flexible Python-based pipeline for inferring copy-number alterations from single-cell RNA-seq data and have devoted substantial effort to optimizing its performance—tuning segmentation thresholds, exploring adaptive normalization, and testing alternative models—only to find that, on our synthetic benchmarking sets, precision remained persistently low despite high recall. We acknowledge that these results are far from ideal and reflect the intrinsic challenges of extreme class imbalance and read-depth variability. Nonetheless, when applied to real pluripotent stem cell datasets, our approach faithfully reproduced the key chromosome 11 CNA hotspots reported by McCracken et al. (2020), demonstrating its capacity to uncover biologically meaningful alterations. Going forward, we will integrate imbalance-aware strategies and dynamic segmentation methods to improve quantitative accuracy, while leveraging the pipeline's proven ability to recapitulate published PSC CNA signatures.

Reference:

[1] McCracken IR, Taylor RS, Kok FO, de la Cuesta F, Dobie R, Henderson BEP, Mountford JC, Caudrillier A, Henderson NC, Ponting CP, Baker AH.
Transcriptional dynamics of pluripotent stem cell derived endothelial cell differentiation revealed by single cell RNA-Seq [Data set].
NCBI Gene Expression Omnibus: GSE131736; 2019. Available from:
https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE131736 (accessed 6 May 2025)

[2] McCracken IR, Taylor RS, Kok FO, de la Cuesta F, Dobie R, Henderson BEP, Mountford JC, Caudrillier A, Henderson NC, Ponting CP, Baker AH.
Transcriptional dynamics of pluripotent stem cell-derived endothelial cell differentiation revealed by single-cell RNA sequencing.
European Heart Journal. 2020 Mar 1;41(9):1024–1036. doi:10.1093/eurheartj/ehz351.
PMID:31242503