Recovering Undetermined Reads: The Local Demultiplexer Pipeline

Akhilesh Kaushal

January 27, 2025

1 Introduction

Next-generation sequencing (NGS) technologies have transformed genomics by producing massive volumes of DNA and RNA data with high throughput. While these advances enable detailed characterization of genomes and transcriptomes, they also generate substantial computational challenges. Among the first steps in processing raw sequencing data is **demultiplexing** — the assignment of reads to their respective samples using barcode sequences (commonly i7 and i5 indices). Errors in demultiplexing, such as incorrect or incomplete sample sheets, can lead to large fractions of unassigned reads, reducing data quality and wasting sequencing resources.

The **Local Demultiplexer** is an automated, modular pipeline designed to address these issues. It provides a recovery strategy for unassigned reads and ensures accurate sample-level assignment. By leveraging high-performance computing (HPC) environments, the Local Demultiplexer enables efficient, large-scale processing of FASTQ data. Its components include:

- Splitting large FASTQ files into smaller chunks for parallelization.
- Re-demultiplexing reads based on validated dual-index (i7, i5) barcode pairs.
- Concatenating and validating final per-sample FASTQ files for downstream analyses.

2 Workflow Overview

The Local Demultiplexer consists of three major stages: splitting, demultiplexing, and concatenation/validation. Each stage is implemented as independent modules, making the pipeline flexible and reusable.

2.1 Workflow Diagram (with QC Integration, compact)

2.2 Splitting FASTQ Files

To enable efficient processing, large FASTQ files are split into smaller parts. This is particularly important when files exceed system memory limits or when distributing work across HPC nodes.

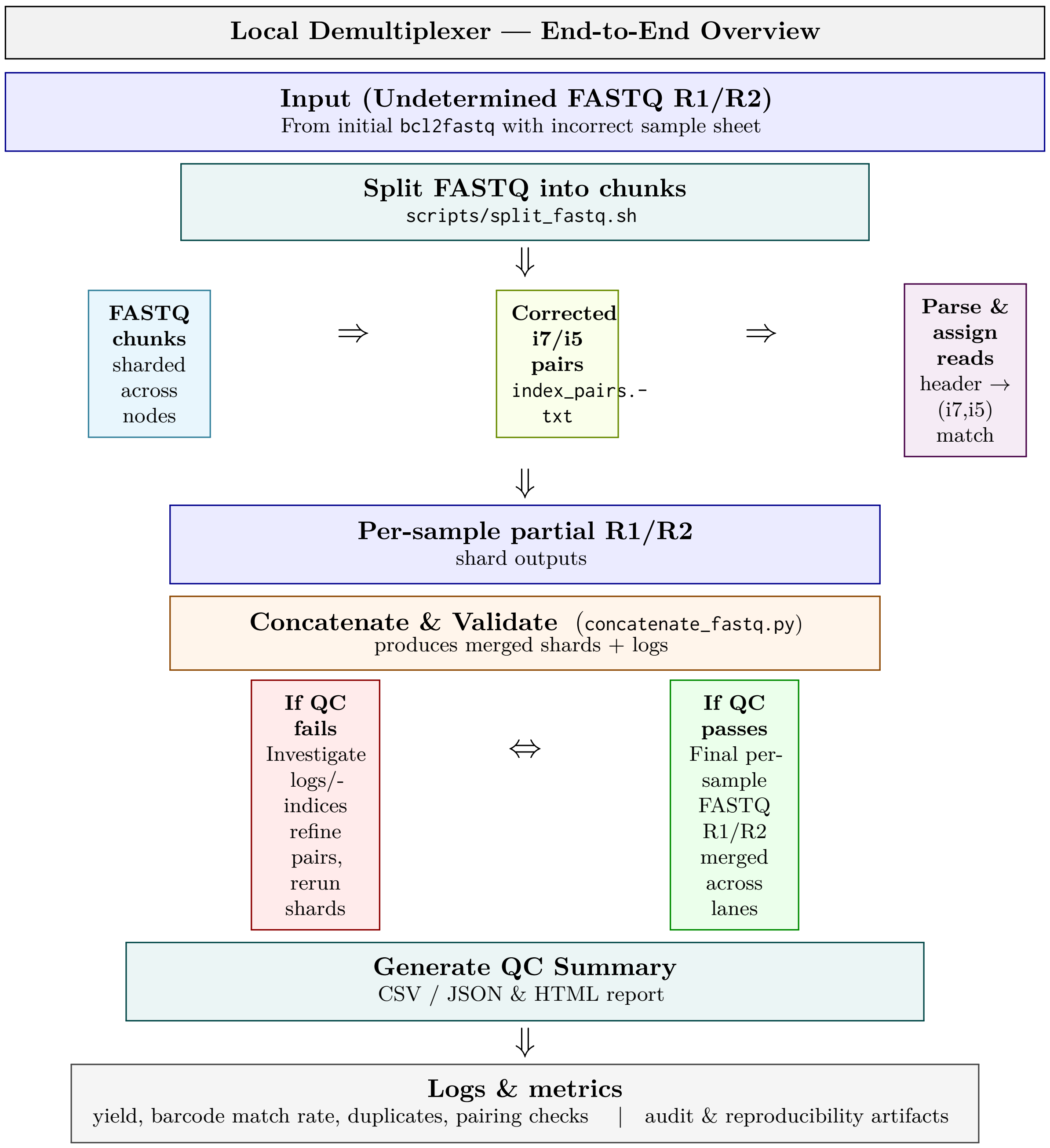
Local Demultiplexer — End-to-End Overview
Input (Undetermined FASTQ R1/R2)
From initial bcl2fastq with incorrect sample sheet
Split FASTQ into chunks
scripts/split_fastq.sh
FASTQ chunks sharded across nodes
Corrected i7/i5 pairs index_pairs.-txt
Parse & assign reads header → (i7,i5) match
Per-sample partial R1/R2
shard outputs
Concatenate & Validate (concatenate_fastq.py)
produces merged shards + logs
If QC fails Investigate logs/-indices refine pairs, rerun shards
If QC passes Final per-sample FASTQ R1/R2 merged across lanes
Generate QC Summary
CSV / JSON & HTML report
Logs & metrics
yield, barcode match rate, duplicates, pairing checks | audit & reproducibility artifacts

Figure 1: Color-card roadmap of the Local Demultiplexer

```
bash scripts/split_fastq.sh input_folder output_folder
```

Example: A 100 GB FASTQ file may be split into 50 smaller files (2 GB each), allowing independent parallel jobs to run on an HPC cluster.

2.3 Demultiplexing Reads

Reads are reassigned to samples based on corrected dual-index pairs (i7, i5). This step is critical when the original demultiplexing (e.g., via `bcl2fastq`) was performed with an incorrect sample sheet, leaving many reads in `Undetermined` FASTQ files.

```
bash scripts/submit_sequential_log.sh job_list.txt
```

Example input file (index_pairs.txt):

```
GATCGGAAG AGCTTAGC
CGATGTTAG TTAGGCCA
ACGTAGCTA CGTAGCTA
```

Each line represents a valid i7–i5 index pair corresponding to a sample.

2.4 Concatenating and Validating Reads

Once demultiplexing is complete, read fragments are merged into final paired-end FASTQ files. Validation checks confirm correct pairing and completeness of the outputs.

```
python local_demultiplexer/concatenate_fastq.py \
  --r1_dir R1_files/ \
  --r2_dir R2_files/ \
  --output_dir merged/ \
  --log_file process.log
```

The script ensures:

- Correct pairing of R1 and R2 reads across all lanes.
- No data loss during merging.
- A detailed log file for reproducibility and debugging.

3 Read Header Anatomy and Index Extraction

Most Illumina FASTQ headers are whitespace-delimited fields, with the dual index string commonly present as the **10th field** for Undetermined reads generated by certain demultiplexers. In our recovery use case, the Local Demultiplexer extracts the **i7** and **i5** indices by splitting that field on "+" (e.g., `ACGTACGT+TTGGAACC`).

Example Read Header (schematic)

```
@INSTR:RUN:FLOWCELL:LANE:TILE:X:Y 1:N:0:ACGTACGT+TTGGAACC
```

The right-hand token (after the space) contains colon-separated subfields; the final subfield is typically the dual index string `i7+i5`.

3.1 Header Anatomy and Parsing

`@INSTR:RUN:FLOWCELL:LANE:TILE:X:Y  1:N:0:ACGTACGT+TTGGAACC`

Left fields: instrument/run/flowcell/position • Middle: read/filter • Right: dual index field `i7+i5`

Extraction: split last token on "+" ⇒ `i7`=`ACGTACGT`, `i5`=`TTGGAACC`

Mapping: (`i7`,`i5`) ↔ Sample ID via validated sample sheet

Figure 2: FASTQ header anatomy and dual-index extraction (no TikZ).

Illustrative Pseudocode

```
# header right token: "1:N:0:ACGTACGT+TTGGAACC"
dual_index=$(echo "$token" | awk -F: '{print␣$NF}')
i7=$(echo "$dual_index" | awk -F+ '{print␣$1}')
i5=$(echo "$dual_index" | awk -F+ '{print␣$2}')
# lookup (i7,i5) -> sample in validated sample-sheet
```

4 Matching Policy: Allowed Mismatches / Hamming Distance

To tolerate occasional sequencing errors in index reads while preventing cross-sample bleed, a simple, auditable Hamming-distance policy is applied to both indices independently and jointly. Thresholds can be tuned per run.

Table 1: Allowed mismatches policy for dual indices.

Index	**Length**	**Max Hamming Distance**	**Action**
i7	6–10 bp	≤ 1 (default)	accept if unique; tie → reject
i5	6–10 bp	≤ 1 (default)	accept if unique; tie → reject
Both	—	if both pass and pair exists	assign to sample ID
Either fails	—	—	mark as *unassigned*; log detail
Tie/ambiguous	—	—	*unassigned*; log all candidates

Notes.

- Tighten to distance 0 for highly multiplexed panels or when barcode edit-distances are small.
- Always ship logs with: per-barcode yield, rejection reasons, ambiguity counts, and top offending barcodes.

5 QC Metrics and Logging

A critical feature of the Local Demultiplexer is its structured logging system. Every run produces both a human-readable log file (`process.log`) and machine-parsable summaries (CSV/JSON). These artifacts enable reproducibility, audit trails, and rapid troubleshooting.

5.1 Core Metrics

- **Total reads processed:** Raw count across all input FASTQ chunks.
- **Reads assigned:** Number and percentage mapped to valid (i7,i5) pairs.
- **Unassigned reads:** Count and fraction; broken down by reason (no match, mismatch threshold exceeded, ambiguous match).
- **Per-sample yield:** Absolute counts and relative percentages per sample.
- **Ambiguity rate:** Fraction of reads mapping equally well to multiple samples.
- **Duplicate headers:** Detected and logged if present.

5.2 Illustrative Log Output

```
[2025-01-27 10:14:02] INFO Total reads processed: 48,126,390
[2025-01-27 10:14:02] INFO Reads assigned: 45,732,881 (95.0%)
[2025-01-27 10:14:02] INFO Reads unassigned: 2,393,509 (5.0%)
    - 1,875,221 no index match
    - 412,188 exceeded mismatch threshold
    - 106,100 ambiguous (multiple valid matches)
[2025-01-27 10:14:02] INFO Per-sample yields:
    Sample01: 12,345,678 (25.6%)
    Sample02: 11,987,654 (24.9%)
    Sample03: 8,765,432 (18.5%)
    ...
[2025-01-27 10:14:02] INFO Top offending indices:
    ACGTAGCT+TTGGAACC -> 156,231 rejects
    CGTACGTA+AGCTTAGC -> 124,788 rejects
[2025-01-27 10:14:02] INFO Run completed successfully.
```

Table 2: Example summary metrics for a demultiplexing run.

Metric	Count	Percent	Notes
Total reads	48,126,390	100%	All input reads
Assigned	45,732,881	95.0%	Successfully matched to sample
Unassigned	2,393,509	5.0%	See breakdown below
No match	1,875,221	3.9%	Index not in sample sheet
Threshold fail	412,188	0.9%	>1 mismatch
Ambiguous	106,100	0.2%	Equal best match to ≥ 2 samples

5.3 Summary Table Example

5.4 Best Practices

- Archive logs and QC summaries alongside final FASTQs (same directory tree).
- Review ambiguous cases; frequent ambiguities suggest insufficient index edit-distance.
- For large projects, ingest summary CSV into LIMS or a QC dashboard.

6 Pipeline Architecture and Features

The Local Demultiplexer integrates shell and Python scripts into a modular architecture:

- **Scalability:** Compatible with HPC clusters for parallel execution.
- **Error Handling:** Built-in logging and warnings for unexpected index mismatches.
- **Flexibility:** Modular design enables substitution of individual components.
- **Recovery-Oriented:** Specifically designed to salvage reads misclassified as unassigned by bcl2fastq.
- **Reproducibility:** Version-controlled scripts and detailed logs ensure consistent results.

7 Use Case: Recovery from Incorrect Sample Sheets

The pipeline was initially developed to address a common sequencing problem: incorrect sample sheets provided to `bcl2fastq`, resulting in tens of millions of unassigned reads. By re-parsing headers and applying corrected i7–i5 barcodes, the Local Demultiplexer successfully re-assigned reads to their respective samples. This recovery strategy ensures:

- Minimal data loss.
- Accurate downstream analyses.
- Significant cost savings for sequencing facilities.

8 Conclusion

The Local Demultiplexer offers a robust, efficient, and scalable approach to NGS data preprocessing. By automating the recovery of misassigned reads and streamlining sample-level FASTQ generation, it provides sequencing facilities and bioinformatics researchers with a reliable preprocessing solution. Its modular structure, ease of deployment, and compatibility with HPC environments make it an indispensable component of the genomic analysis pipeline.

9 Availability

The source code is freely available on GitHub:

```
git clone https://github.com/akaushaluams/demultiplexer.git
```

10 References

1. Smith J. et al. (2020). Next-generation sequencing and data processing. *Bioinformatics Journal*, 35(4), 1234–1245.

2. Brown A. et al. (2018). Efficient demultiplexing strategies for sequencing data. *Genomics Review*, 27(2), 567–578.