**SynthLongRead: Getting Started Guide**

This guide will walk you through the process of installing and running SynthLongRead for the first time with your own data.

**Table of Contents**

1. [Introduction](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#introduction)
2. [Installation](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#installation)
3. [Input Data Requirements](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#input-data-requirements)
4. [Basic Workflow](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#basic-workflow)
5. [Examining Results](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#examining-results)
6. [Validating Synthetic Data](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#validating-synthetic-data)
7. [Benchmarking Tools](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#benchmarking-tools)
8. [Troubleshooting](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#troubleshooting)
9. [Advanced Usage](https://claude.ai/chat/63010c1c-8e02-409b-aa4d-9c8f82609fa1#advanced-usage)

**Introduction**

SynthLongRead is a framework for generating synthetic long-read single-cell RNA sequencing data with realistic error profiles. It learns from real data to create synthetic reads that mimic the characteristics of the original data, but with a known ground truth of which transcripts are present in each cell.

**Installation**

Follow these steps to install SynthLongRead:

# Clone the repository

git clone https://github.com/yourusername/SynthLongRead.git

cd SynthLongRead

# Install dependencies

pip install -r requirements.txt

# Install the package in development mode

pip install -e .

**Prerequisites**

Make sure you have the following software installed and available in your PATH:

* Python 3.8 or higher
* minimap2 (for read alignment)
* samtools (for BAM file processing)
* FLAMES (optional, only if you want to benchmark FLAMES)

**Input Data Requirements**

To run SynthLongRead, you need the following input files:

1. **Real long-read FASTQ file**: A representative dataset from Oxford Nanopore (ONT) or Pacific Biosciences (PacBio) sequencing of single-cell or single-nucleus RNA.
2. **Reference transcriptome**: A FASTA file containing transcript sequences. This should match the annotation version used in your GTF file.
3. **Reference GTF**: A gene annotation file in GTF format. This is used to establish gene-isoform relationships.
4. **Reference genome** (optional): Only required if you want to benchmark FLAMES or other tools that need a reference genome.

**Basic Workflow**

**1. Run the Basic Workflow Script**

The simplest way to start is using the provided example script:

python examples/basic\_workflow.py \

--reference\_transcriptome /path/to/transcriptome.fa \

--reference\_gtf /path/to/annotation.gtf \

--real\_fastq /path/to/your\_data.fastq \

--output\_dir ./synthlongread\_output \

--platform ONT \

--n\_cells 100 \

--max\_reads 100000

**2. Key Parameters**

* --reference\_transcriptome: Path to your reference transcriptome FASTA file
* --reference\_gtf: Path to your reference GTF annotation file
* --real\_fastq: Path to your real long-read FASTQ file
* --output\_dir: Directory where outputs will be saved
* --platform: Sequencing platform, either "ONT" or "PacBio"
* --n\_cells: Number of synthetic cells to generate
* --max\_reads: Maximum number of reads to generate (use this to limit computation time)
* --threads: Number of CPU threads to use (default: 4)
* --seed: Random seed for reproducibility (default: 42)

**3. Expected Runtime**

SynthLongRead has three main computational steps:

1. **Error profile extraction**: ~1-2 hours for a typical dataset
2. **Model training**: ~2-4 hours depending on dataset size and available hardware
3. **Read generation**: ~1 hour per 100,000 reads

For a first test run, consider limiting --max\_reads to 10,000-50,000 to get results faster.

**Examining Results**

After running the basic workflow, check your output directory for the following files:

synthlongread\_output/

├── error\_profiles.pkl # Serialized error profiles learned from real data

├── models/ # Directory containing trained neural network models

│ ├── seq\_error\_model.pt # Sequence error model (PyTorch)

│ ├── quality\_model.pt # Quality score model (PyTorch)

│ └── read\_length\_model.pkl # Read length distribution model

├── synthetic\_data.fastq # Generated synthetic FASTQ file

└── ground\_truth.csv # True expression matrix (transcript counts per cell)

The two key output files are:

* synthetic\_data.fastq: The synthetic FASTQ file you can analyze with your tools
* ground\_truth.csv: The "answer key" showing what transcripts are actually in each cell

**Validating Synthetic Data**

To check how well your synthetic data matches the characteristics of the real data:

python examples/benchmark\_flames.py \

--synthetic\_fastq ./synthlongread\_output/synthetic\_data.fastq \

--real\_fastq /path/to/your\_data.fastq \

--output\_dir ./validation\_results

This will generate comparison plots in ./validation\_results/read\_metrics/:

* read\_length\_comparison.png: Histograms of read lengths
* quality\_score\_comparison.png: Quality score distributions
* gc\_content\_comparison.png: GC content distributions
* homopolymer\_comparison.png: Homopolymer length distributions
* base\_composition\_comparison.png: Base frequency comparisons

Review these plots to ensure your synthetic data has similar characteristics to your real data. If there are major discrepancies, you may need to adjust parameters or check your input data.

**Benchmarking Tools**

**Running FLAMES Benchmark**

If you want to benchmark FLAMES on your synthetic data:

python examples/benchmark\_flames.py \

--synthetic\_fastq ./synthlongread\_output/synthetic\_data.fastq \

--ground\_truth ./synthlongread\_output/ground\_truth.csv \

--reference\_genome /path/to/genome.fa \

--reference\_gtf /path/to/annotation.gtf \

--output\_dir ./flames\_benchmark

This will:

1. Run FLAMES on your synthetic data
2. Compare FLAMES results to the ground truth
3. Generate evaluation metrics and visualizations

The benchmark results will be in ./flames\_benchmark/:

* metrics/: Detailed metrics at transcript and gene level
* summary\_metrics.csv: Overall performance summary
* Various visualization plots showing detection accuracy, quantification correlation, etc.

**Benchmarking Other Tools**

To benchmark other tools, run them on the synthetic FASTQ file, then use the evaluation module to compare results:

from synthlongread.evaluation import IsoformBenchmark

benchmark = IsoformBenchmark(

ground\_truth\_file="./synthlongread\_output/ground\_truth.csv",

results\_file="/path/to/tool\_results.csv",

output\_dir="./benchmark\_results"

)

metrics = benchmark.run\_all\_evaluations()

**Troubleshooting**

**Common Issues**

1. **Memory errors**:
   * Reduce --max\_reads to generate fewer reads
   * Run on a machine with more RAM
   * Use smaller input datasets for initial testing
2. **Runtime issues**:
   * Increase --threads to use more CPU cores
   * Use smaller input datasets for initial testing
   * Consider running on an HPC system for large datasets
3. **Alignment errors**:
   * Ensure your reference transcriptome and GTF files are compatible versions
   * Check if minimap2 is properly installed and in your PATH
   * Try using a different aligner by providing a pre-aligned BAM file with --alignment\_file
4. **Error profile issues**:
   * Ensure your input FASTQ has enough reads (at least 10,000 recommended)
   * Check if your input FASTQ matches the specified platform (ONT or PacBio)
   * Try using a different subset of your data if current subset has quality issues

**Quick Test Run**

For a quick test to verify everything is working correctly:

python examples/basic\_workflow.py \

--reference\_transcriptome /path/to/transcriptome.fa \

--reference\_gtf /path/to/annotation.gtf \

--real\_fastq /path/to/your\_data.fastq \

--output\_dir ./test\_run \

--platform ONT \

--n\_cells 10 \

--max\_reads 10000

This will produce a small synthetic dataset with just 10 cells and up to 10,000 reads, which should complete relatively quickly.

**Advanced Usage**

**Custom Expression Patterns**

To generate synthetic data with specific expression patterns:

python examples/custom\_expression.py \

--reference\_transcriptome /path/to/transcriptome.fa \

--reference\_gtf /path/to/annotation.gtf \

--real\_fastq /path/to/your\_data.fastq \

--output\_dir ./custom\_output

This creates synthetic data with predefined cell types and expression patterns. You can edit the script to define your own patterns.

**Using Pre-trained Models**

If you've already run SynthLongRead once, you can reuse the trained models:

python examples/basic\_workflow.py \

--reference\_transcriptome /path/to/transcriptome.fa \

--reference\_gtf /path/to/annotation.gtf \

--real\_fastq /path/to/your\_data.fastq \

--output\_dir ./new\_output \

--model\_dir ./previous\_run/models

**Programmatic API**

For more advanced control, you can use the SynthLongRead API directly in your own Python scripts:

from synthlongread import SynthLongRead

# Initialize

synth = SynthLongRead(

reference\_transcriptome="/path/to/transcriptome.fa",

reference\_gtf="/path/to/annotation.gtf",

platform="ONT",

output\_dir="./output"

)

# Learn from real data

synth.learn\_from\_real\_data(

real\_fastq="/path/to/your\_data.fastq"

)

# Generate synthetic dataset

synthetic\_fastq, ground\_truth = synth.generate\_synthetic\_dataset(

n\_cells=100,

sparsity=0.8,

max\_reads=100000

)

This gives you more flexibility to customize the process for your specific needs.