Salmon Quantification - Comprehensive Reference Guide

Overview

Salmon is a fast, alignment-free tool for quantifying transcript and gene expression from RNA-seq data. It uses a k-mer-based quasi-mapping approach combined with an Expectation-Maximization (EM) algorithm to estimate transcript abundances without requiring full genome alignment.

Website: https://combine-lab.github.io/salmon/
Publication: Patro et al. Nature Methods 2017
Current Version: 1.10.0+

Applicable to: RNA-seq, single-cell RNA-seq, metatranscriptomics, spatial transcriptomics

---

What Salmon Does

Core Functions

1. **Quasi-mapping** - Maps reads to transcripts using k-mer matching (not full alignment)
2. **Transcript Quantification** - Estimates abundance for each transcript (TPM and counts)
3. **Gene-level Aggregation** - Can aggregate to gene level (or use tximport)
4. **Multi-mapper Resolution** - Probabilistically assigns reads mapping to multiple transcripts
5. **Bias Correction** - Corrects for GC, sequence, and positional biases

Salmon vs Traditional Methods

Feature	Salmon	STAR + featureCounts	kallisto
Speed	Very fast (2-5 min)	Slow (20-40 min)	Very fast (3-8 min)
Accuracy	Excellent	Excellent	Excellent
Output	Transcript counts	Gene counts	Transcript counts
BAM file	No	Yes	No
Memory	8-12GB	30-50GB	4-8GB
Bias correction	Built-in	Manual	Limited
Multi-mappers	EM algorithm	Discard or count once	EM algorithm
Gene families	Handles well	Problematic	Handles well

Salmon Quantification Approach

K-mer Based Quasi-Mapping

Traditional alignment (STAR, HISAT2):

```
Read → Find seeds → Extend alignment → Score → Report best alignment
Time: ~10-30 minutes per sample
Output: Full BAM file with exact positions
```

Salmon quasi-mapping:

```
Read → Hash k-mers → Find transcript matches → Score compatibility → Report
Time: ~2-5 minutes per sample
Output: Transcript abundance estimates (no BAM)
```

How It Works

```
Step 1: K-mer Extraction
  Read: ATGCGTACGATCGATCG...
  K-mers (k=31): ATGCGTACGATCGATCGATCGATCGATCGATC
                 TGCGTACGATCGATCGATCGATCGATCGATCG
                 GCGTACGATCGATCGATCGATCGATCGATCGA
                 ...

Step 2: Transcript Matching
  K-mer → Index lookup → Find transcripts containing this k-mer
  Build compatibility graph: Read ↔ {Transcript1, Transcript2, ...}

Step 3: Probabilistic Assignment (EM Algorithm)
  Initialize: Uniform probability across compatible transcripts
  Iterate:
    1. E-step: Assign reads to transcripts weighted by current estimates
    2. M-step: Update transcript abundances based on assignments
  Until convergence

  Result: Probability distribution of each read across transcripts
          → Fractional counts for each transcript
```

Why EM Algorithm?

Problem: Multi-mapping reads

```
Read maps to:
  - Transcript A (gene X)
  - Transcript B (gene X, different isoform)
  - Transcript C (pseudogene of X)

Traditional methods:
  Discard (lose information)
  Count once (underestimate)
  Count all (overestimate)

Salmon's EM:
  Assign fractionally based on:
    - Transcript abundance
    - Sequence compatibility
    - Fragment length distribution
  Iterate until stable
  Maintains total read count
```

Example:

```
Initial:
  Transcript A: 1000 reads (100 unique, 900 ambiguous)
  Transcript B: 100 reads (10 unique, 90 ambiguous)

EM realizes:
  - A is 10x more abundant
  - Ambiguous reads more likely from A

After EM:
  Transcript A: ~945 reads (probabilistic assignment)
  Transcript B: ~55 reads (probabilistic assignment)
```

Salmon Quantification Command

Basic Command

```
salmon quant \
    --index salmon_index \
    --libType A \
    --mates1 R1.fastq.gz \
    --mates2 R2.fastq.gz \
    --output sample_quant \
    --threads 4
```

Command Breakdown

```
salmon quant \                      # Quantification mode
    --index salmon_index \          # Path to Salmon index
    --libType A \                   # Auto-detect library type
    --mates1 R1.fastq.gz \          # Forward reads (PE)
    --mates2 R2.fastq.gz \          # Reverse reads (PE)
    --output sample_quant \          # Output directory
    --threads 4                      # Number of CPU cores
```

Single-end data:

```
salmon quant \
    --index salmon_index \
    --libType A \
    --unmatedReads reads.fastq.gz \
    --output sample_quant \
    --threads 4
```

Key Parameters Explained

Library Type (--libType)

Purpose: Specifies strand orientation of reads

Auto-detection (RECOMMENDED):

```
--libType A    # Auto-detect library type
```

Salmon will examine first 1-10 million reads and determine:

- **SE Libraries**: SF, SR, or U
- **PE Libraries**: ISF, ISR, or IU

Library type codes:

Code	Description	Read Orientation
SF	Single-end Forward	Read matches transcript strand
SR	Single-end Reverse	Read is reverse complement
U	Unstranded	Read could be either orientation
ISF	Inward Stranded Forward	R2 matches transcript, R1 is reverse
ISR	Inward Stranded Reverse	R1 matches transcript, R2 is reverse
IU	Inward Unstranded	Reads could be either orientation

Common protocols:

Protocol	Library Type
Illumina TruSeq	ISR (reverse stranded)
dUTP method	ISR (reverse stranded)
Ligation-based	ISR or ISF
SMARTer	Varies (use auto)
Nextera	Varies (use auto)
Old protocols	IU (unstranded)

Verification:

```
# Check detected library type
cat sample_quant/lib_format_counts.json

# Example output
{
    "expected_format": "ISR",
    "compatible_fragment_ratio": 0.9823,
    "num_compatible_fragments": 9823000,
    "num_assigned_fragments": 10000000,
    ...
}
```

If auto-detection fails:

```
# Manually specify (not recommended)
--libType ISR    # If you know it's reverse-stranded PE
```

Selective Alignment (--validateMappings)

Purpose: More stringent mapping validation

How it works:

Without --validateMappings (default quasi-mapping):

```
1. Find k-mer matches
2. Count matches per transcript
3. Assign based on counts
→ Fast but may include spurious matches
```

With --validateMappings (selective alignment):

```
1. Find k-mer matches
2. Build alignment chains
3. Score alignment quality
4. Filter low-quality matches
5. Assign based on validated matches
→ ~20% slower but more accurate
```

Accuracy improvement: ~2-5% more accurate quantification

Recommendation:

- Always use for publication-quality results
- Can skip for quick exploratory analysis

```
salmon quant \
    --validateMappings \    # RECOMMENDED for accuracy
    --index salmon_index \
    ...
```

Bias Corrections

Salmon provides three types of bias correction:

1. GC Bias (--gcBias) **Problem**: PCR amplification favors certain GC content

```
Without correction:
  High GC transcripts: Under-estimated
  Low GC transcripts: Over-estimated

Example:
  GC-rich gene (70% GC): 500 reads → Actually 600 reads
  GC-poor gene (30% GC): 500 reads → Actually 400 reads
```

How Salmon corrects:

```
1. Model GC content vs fragment count
2. Learn bias curve from data
3. Re-weight fragments based on GC content
4. Output corrected abundances
```

Command:

```
--gcBias    # Enable GC bias correction
```

Impact: ~3-5% improvement in accuracy

When to use:

- PCR-amplified libraries
- Non-UMI protocols
- Publication-quality analysis
- UMI-based methods (already corrected)

---

2. Sequence Bias (--seqBias) **Problem**: Random hexamer priming isn't truly random

```
Random hexamer priming:
  Primers: NNNNNN (supposedly random)
  Reality: Some hexamers bind better than others

Bias example:
  GCGCGC: Binds strongly → Over-represented
  ATATAT: Binds weakly → Under-represented

Result:
  Transcripts starting with preferred hexamers → Over-estimated
  Transcripts starting with poor hexamers → Under-estimated
```

How Salmon corrects:

```
1. Learn hexamer preferences from read starts
2. Model position-specific bias
3. Correct abundances based on sequence context
```

Command:

```
--seqBias    # Enable sequence-specific bias correction
```

Impact: ~2-3% improvement in accuracy

When to use:

- Random hexamer priming protocols
- Any RT-PCR based method
- Oligo-dT priming (different bias pattern)

---

3. Positional Bias (--posBias) **Problem**: Fragments not uniformly distributed along transcripts

```
Causes:
  - 3' bias: RNA degradation, poly-A selection
  - 5' bias: Some library prep methods
  - Coverage holes: Secondary structure

Without correction:
  3' biased sample:
    Gene A (5' region expressed): Under-estimated
    Gene B (3' region expressed): Over-estimated
```

How Salmon corrects:

```
1. Model fragment position distribution
2. Learn coverage pattern along transcripts
3. Re-weight fragments based on position
```

Command:

```
--posBias    # Enable positional bias correction
```

Impact: ~1-2% improvement (varies by sample quality)

When to use:

- Degraded RNA samples
- Poly-A selected libraries (mild 3' bias)
- High-quality, uniform coverage

Combined Bias Correction **Standard practice**: Enable all three

```
salmon quant \
    --validateMappings \
    --gcBias \
    --seqBias \
    --posBias \
    --index salmon_index \
    --libType A \
    --mates1 R1.fastq.gz \
    --mates2 R2.fastq.gz \
    --output sample_quant \
    --threads 4
```

Total improvement: ~5-10% more accurate quantification

Trade-off: ~10-15% slower runtime (still very fast overall)

Mapping Parameters

--minAssignedFrags **Purpose**: Minimum fragments assigned to a transcript for it to be reported

Default: 10

Why it matters:

```
Low-count transcripts (1-9 fragments):
  - High uncertainty
  - Likely noise or mapping artifacts
  - Unstable estimates

Filtering helps:
    Reduces false positives
    Improves statistical power
    Cleaner downstream analysis
```

Recommendations:

```
# Standard analysis
--minAssignedFrags 10    # Default, good balance

# Sensitive analysis (detect low-abundance)
```

```
--minAssignedFrags 1     # Keep everything

# Conservative analysis
--minAssignedFrags 50    # Only well-supported transcripts
```

--rangeFactorizationBins **Purpose**: Controls EM algorithm precision

Default: 4

What it does:

```
Bins for fragment length distribution:
  More bins → More precise → Slower
  Fewer bins → Less precise → Faster

Default (4): Good balance
Higher (8-16): Marginal improvement, longer runtime
Lower (2): Faster but less accurate
```

Recommendation: Keep default unless you have specific needs

Output Files

Salmon creates an output directory with multiple files:

Directory Structure

```
sample_quant/
  quant.sf                      # Main output: Transcript abundances
  quant.genes.sf                # Gene-level abundances (if --geneMap used)
  lib_format_counts.json        # Library type detection results
  meta_info.json                # Run metadata and parameters
  cmd_info.json                 # Command line executed
  aux_info/
     ambig_info.tsv           # Ambiguously mapped fragment info
     expected_bias.gz         # Learned bias models
     fld.gz                   # Fragment length distribution
     meta_info.json           # Auxiliary metadata
     observed_bias.gz         # Observed bias patterns
  logs/
     salmon_quant.log          # Detailed log file
```

---

quant.sf (Main Output)

Format: Tab-separated values (TSV)

Columns:

Column	Name	Description
1	**Name**	Transcript ID (e.g., ENST00000000001)
2	**Length**	Transcript length in base pairs
3	**EffectiveLength**	Effective length accounting for biases
4	**TPM**	Transcripts Per Million (normalized abundance)
5	**NumReads**	Estimated number of reads from this transcript

Example:

```
Name                    Length  EffectiveLength  TPM        NumReads
ENST00000000001.1       2000    1850            1250.50    10000.5
ENST00000000002.1       1500    1350            500.25     3500.2
ENST00000000003.1       3000    2850            2500.75    25000.8
```

Column details:

Length

- Raw transcript length from FASTA
- Includes all exons (spliced)
- Fixed value for each transcript

EffectiveLength

- Adjusted length accounting for:
  - Fragment length distribution
  - Positional bias
  - Edge effects (fragments can't start/end anywhere)

```
Formula: EffectiveLength  Length -   + 1
  where   = mean fragment length

Example:
  Transcript length: 2000bp
  Mean fragment length: 200bp
  Effective length: 2000 - 200 + 1 = 1801bp

Why: A 200bp fragment can't start in last 199bp of transcript
```

TPM (Transcripts Per Million)

- Normalized abundance measure
- Comparable across samples
- Sums to 1 million within each sample

```
Formula:
  1. Counts per base: NumReads / EffectiveLength
  2. Normalize: (Counts per base / Sum of all counts per base) × 1,000,000

Properties:
    Sum of all TPMs = 1,000,000
    Accounts for transcript length
    Comparable across samples
    NOT raw counts (don't use for DESeq2/edgeR directly)
```

TPM interpretation:

```
TPM = 1000:    1 out of every 1000 transcripts in the sample
TPM = 100:     Moderate expression
TPM = 10:      Low expression
TPM = 1:       Very low expression
TPM < 1:       Barely detectable
```

NumReads

- Estimated number of reads from transcript
- Can be fractional (due to multi-mapping)
- Sum across transcripts   total mapped reads

```
Example:
  Read maps to Transcript A and B

  If A is 3x more abundant than B:
    Transcript A: +0.75 reads
    Transcript B: +0.25 reads

  Result: NumReads can be non-integer
```

---

lib_format_counts.json

Purpose: Library type detection results

Example:

```json
{
    "expected_format": "ISR",
    "compatible_fragment_ratio": 0.9823,
    "num_compatible_fragments": 9823000,
    "num_assigned_fragments": 10000000,
    "num_consistent_mappings": 9823000,
    "num_inconsistent_mappings": 177000,
    "MSF": 50000,
    "MSR": 127000,
    "MU": 0,
    "SF": 25000,
    "SR": 25000,
    "U": 0,
    "ISF": 88500,
    "ISR": 9823000,
    "IU": 0
}
```

Key fields:

- `expected_format`: Detected library type
- `compatible_fragment_ratio`: Fraction of reads matching expected orientation
- High ratio (>0.95): Confident detection
- Low ratio (<0.80): Mixed or poor quality

Red flags:

- Compatible ratio <0.80: Library type unclear
- Multiple types with similar counts: Mixed libraries
- Unexpected type: Wrong protocol or sequencing issue

---

meta_info.json

Purpose: Run metadata and parameters

Example:

```
{
    "salmon_version": "1.10.0",
    "samp_type": "paired",
    "opt_type": "vbem",
    "quant_errors": [],
    "num_libraries": 1,
    "library_types": ["ISR"],
    "frag_dist_length": 1000,
    "num_targets": 234567,
    "num_bootstraps": 0,
    "mapping_type": "mapping",
    "index_seq_hash": "a1b2c3d4e5f6...",
    "index_name_hash": "f6e5d4c3b2a1...",
    "index_decoy_seq_hash": "1a2b3c4d5e6f...",
    "index_keep_duplicates": false,
    "keep_duplicates": false
}
```

Important fields:

- `salmon_version`: Version used (important for reproducibility)
- `num_targets`: Number of transcripts quantified
- `library_types`: Detected library type(s)
- `num_bootstraps`: If bootstrapping was run

---

aux_info/ Directory

Contains auxiliary information for downstream tools:

fld.gz

- Fragment length distribution
- Used by tximport for length correction
- Histogram of insert sizes

expected_bias.gz / observed_bias.gz

- Learned bias models
- Used for bias correction
- Can be visualized for QC

ambig_info.tsv

- Information about ambiguously mapped fragments
- Useful for debugging low mapping rates

Downstream Analysis with tximport

Why Use tximport?

Problem: Salmon outputs transcript-level abundances, but we often want gene-level

Solutions:

1. Sum transcript counts directly → Loses statistical information
2. Use longest isoform only → Ignores real biology
3. Use tximport → Properly aggregates with statistical awareness

tximport Advantages

- Aggregates transcripts to genes correctly
- Preserves inferential uncertainty
- Accounts for transcript length differences
- Compatible with DESeq2/edgeR
- Handles multi-isoform genes properly

Basic tximport Usage

Step 1: Prepare tx2gene mapping

```r
library(tximport)
library(dplyr)

# Option A: From Ensembl BioMart
library(biomaRt)
mart <- useEnsembl("ensembl", dataset = "hsapiens_gene_ensembl")
tx2gene <- getBM(
    attributes = c("ensembl_transcript_id", "ensembl_gene_id"),
    mart = mart
)
colnames(tx2gene) <- c("transcript_id", "gene_id")

# Option B: From GTF file
library(GenomicFeatures)
txdb <- makeTxDbFromGFF("annotation.gtf")
k <- keys(txdb, keytype = "TXNAME")
tx2gene <- select(txdb, k, "GENEID", "TXNAME")
colnames(tx2gene) <- c("transcript_id", "gene_id")

# Option C: Manual (if you have a file)
tx2gene <- read.table("tx2gene.txt", header = TRUE)
```

Format:

```
transcript_id      gene_id
ENST00000000001    ENSG00000000001
ENST00000000002    ENSG00000000001
ENST00000000003    ENSG00000000002
```

Step 2: Collect Salmon output files

```r
# List all quant.sf files
samples <- c("sample1", "sample2", "sample3", ...)
```

```r
files <- file.path("salmon_output", samples, "quant.sf")
names(files) <- samples

# Verify files exist
all(file.exists(files))
```

Step 3: Import with tximport

```r
# Import transcript abundances
txi <- tximport(
    files,
    type = "salmon",
    tx2gene = tx2gene,
    countsFromAbundance = "lengthScaledTPM"  # Recommended
)

# Output structure
names(txi)
# [1] "abundance"           # Gene-level TPMs
# [2] "counts"              # Gene-level estimated counts
# [3] "length"              # Average transcript length per gene
# [4] "countsFromAbundance" # Method used
```

countsFromAbundance options:

Option	Description	Use Case
"no"	Use estimated counts directly	Standard
"scaledTPM"	Scale TPMs to library size	Simple normalization
"lengthScaledTPM"	Scale TPMs & adjust for length	**RECOMMENDED**
"dtuScaledTPM"	For DTU analysis	Isoform switching

Step 4: Create DESeq2 object

```r
library(DESeq2)

# Sample metadata
coldata <- data.frame(
    sample = samples,
    condition = c("control", "control", "treated", ...),
    row.names = samples
)
```

```
# Create DESeqDataSet from tximport
dds <- DESeqDataSetFromTximport(
    txi,
    colData = coldata,
    design = ~ condition
)

# Run differential expression
dds <- DESeq(dds)
results <- results(dds)
```

Step 5: Explore results

```
# View results
head(results)

# MA plot
plotMA(results)

# PCA
vsd <- vst(dds)
plotPCA(vsd, intgroup = "condition")

# Top genes
res_ordered <- results[order(results$padj), ]
head(res_ordered, 20)
```

Advanced tximport Features

Import with bootstraps (if run)

```
# Run Salmon with bootstraps
salmon quant \
    --numBootstraps 30 \    # 30 bootstrap samples
    ...
```

```
# Import with uncertainty
txi <- tximport(
    files,
    type = "salmon",
    tx2gene = tx2gene,
    txOut = FALSE,           # Gene-level
    varReduce = TRUE         # Compute inferential variance
)
```

```r
# Now txi includes variance estimates
names(txi)
# Includes "variance" element
```

When to use bootstraps:

- Publication-quality DEG analysis
- When uncertainty quantification matters
- Low-count genes
- Exploratory analysis (overkill)

Transcript-level analysis

```r
# Keep transcript-level (don't aggregate to genes)
txi_transcripts <- tximport(
    files,
    type = "salmon",
    txOut = TRUE,             # Keep transcript-level
    countsFromAbundance = "lengthScaledTPM"
)

# Use for isoform-level analysis
dds_tx <- DESeqDataSetFromTximport(
    txi_transcripts,
    colData = coldata,
    design = ~ condition
)
```

Use cases:

- Isoform switching analysis
- Alternative splicing detection
- Transcript-specific regulation

Interpreting Salmon Logs

Mapping Rate

Check salmon_quant.log:

```
[info] Mapping rate = 85.23%
```

Interpretation:

Mapping Rate	Quality	Action
>80%	Excellent	Proceed
70-80%	Good	OK, but check contamination
60-70%	Acceptable	Investigate low mapping
<60%	Poor	Check index, contamination, degradation

Common causes of low mapping:

- Wrong index (different species or annotation version)
- Contamination (bacterial, adapter, rRNA)
- Degraded RNA
- Poor sequencing quality

Fragment Length Distribution

Check aux_info/fld.gz:

```r
# Visualize in R
library(dplyr)
library(ggplot2)

fld <- read.table(gzfile("sample_quant/aux_info/fld.gz"))
colnames(fld) <- c("length", "count")

ggplot(fld, aes(x = length, y = count)) +
    geom_line() +
    labs(title = "Fragment Length Distribution",
         x = "Fragment Length (bp)",
         y = "Count")
```

Expected patterns:

```
Normal distribution (good):




  100   200   300  (bp)
```

```
Bimodal (mixed libraries):




  100   200   300
```

```
Very narrow (size selection issue):



    200

Very wide (poor quality):


  50     300    500
```

Library Type Consistency

Check across samples:

```
# Extract library types from all samples
for sample in sample1 sample2 sample3; do
    echo -n "${sample}: "
    grep "expected_format" ${sample}/lib_format_counts.json
done

# Expected: All samples same type
sample1: "expected_format": "ISR"
sample2: "expected_format": "ISR"
sample3: "expected_format": "ISR"

# Red flag: Mixed types
sample1: "expected_format": "ISR"
sample2: "expected_format": "IU"    # ← Problem!
sample3: "expected_format": "ISR"
```

If inconsistent:

- Check library prep protocol
- Verify not mixing different library types
- May indicate sample swap or mislabeling

Resource Requirements

Memory (RAM)

Component	Requirement
Salmon index in RAM	5-8GB (human)
Quantification buffer	2-4GB

Component	Requirement
Total	8-12GB

Scaling by species:

- Mouse: 6-10GB
- Zebrafish: 4-6GB
- Drosophila: 2-4GB
- C. elegans: 1-2GB
- Arabidopsis: 2-3GB

CPU Cores

Scaling efficiency:

- 1 core: Baseline
- 4 cores: 3.5x faster (good efficiency)
- 8 cores: 6x faster (diminishing returns)
- 16 cores: 8x faster (poor efficiency)

Recommendation: 4-8 cores optimal

Disk Space

Per sample:

- Input FASTQ: 2-8GB (compressed)
- Output directory: 50-200MB
- Temporary files: Minimal

For 100 samples: ~5-20GB total output

Time

Sample Size	4 cores	8 cores
10M reads	2 min	1.5 min
25M reads	4 min	2.5 min
50M reads	7 min	4 min
100M reads	12 min	7 min

With bias correction: +10-20% time

Workflow-Specific Considerations

Bulk RNA-seq

Standard settings:

```
salmon quant \
    --validateMappings \
    --gcBias \
    --seqBias \
    --posBias \
    --libType A \
    --index salmon_index \
    --mates1 R1.fastq.gz \
    --mates2 R2.fastq.gz \
    --threads 4 \
    --output sample_quant
```

Expected metrics:

- Mapping rate: >75%
- Library type: Consistent across samples
- Fragment length: 150-300bp mean

Single-cell RNA-seq

Special considerations:

```
# alevin mode (for UMI-based scRNA-seq)
salmon alevin \
    --index salmon_index \
    --libType ISR \
    --mates1 R1.fastq.gz \      # Cell barcode + UMI
    --mates2 R2.fastq.gz \      # cDNA read
    --chromiumV3 \              # Or other chemistry
    --output sample_alevin \
    --threads 8
```

Differences:

- Uses alevin mode (not quant)
- Handles UMIs and cell barcodes
- Outputs count matrix (genes × cells)
- No bias correction (UMIs handle that)

Metatranscriptomics

Challenges:

- Multiple species in index
- High multi-mapping
- Taxonomic assignment needed

Recommendations:

```
salmon quant \
    --validateMappings \
    --allowDovetail \            # For short fragments
    --recoverOrphans \           # Keep unpaired reads
    --libType A \
    --index metatranscriptome_index \
    --mates1 R1.fastq.gz \
    --mates2 R2.fastq.gz \
    --threads 8 \
    --output sample_quant
```

Downstream:

- Use taxonomic classification
- Aggregate by species/genus
- Account for shared genes

Troubleshooting

Low Mapping Rate (<50%)

Diagnostic steps:

1. **Check index matches data**:

```
# Verify species
head -n 1 salmon_index/info.json

# Check number of transcripts
grep "num_targets" sample_quant/meta_info.json
```

2. **Check for contamination**:

```
# Run FastQC
fastqc R1.fastq.gz R2.fastq.gz

# Check adapter content
# Check overrepresented sequences
```

3. **Test with subset**:

```
# Take first 100K reads
zcat R1.fastq.gz | head -400000 | gzip > R1.subset.fq.gz
zcat R2.fastq.gz | head -400000 | gzip > R2.subset.fq.gz

# Run Salmon
salmon quant --index salmon_index --libType A \
    --mates1 R1.subset.fq.gz --mates2 R2.subset.fq.gz \
    --output test_quant

# Check mapping rate
cat test_quant/logs/salmon_quant.log | grep "Mapping rate"
```

Library Type Detection Fails

Symptoms:

```
{
    "expected_format": "IU",
    "compatible_fragment_ratio": 0.45,    // Low!
    ...
}
```

Causes & solutions:

1. **Truly unstranded library**:
    - OK if protocol is unstranded
    - Check library prep method
2. **Mixed orientation**:
    - Sample contamination or mixing
    - Check sample sheet
3. **Very low depth**:
    - Not enough reads for confident detection
    - Use `--libType ISR` (or appropriate type) manually
4. **Poor quality**:
    - High error rate confuses detection
    - Check FastQC results

Out of Memory

Error: ”std::bad_alloc” or ”Cannot allocate memory”

Solutions:

1. **Increase RAM allocation**:
   - Allocate 16GB minimum
   - 32GB for very large transcriptomes
2. **Reduce index size** (rebuild with --sparse):

   ```
   salmon index \
       --sparse \              # Sparse index (less memory)
       --transcripts gentrome.fa \
       --index salmon_index_sparse
   ```
3. **Use --reduceGCMemory**:

   ```
   salmon quant \
       --reduceGCMemory \      # Reduce garbage collection overhead
       --index salmon_index \
       ...
   ```

Very Low Fragment Count

Symptoms: Most transcripts have <10 fragments assigned

Causes:

1. **Very low sequencing depth**:
   - Check total read count
   - May need more sequencing
2. **Wrong index**:
   - Using transcriptome index when should use genome
   - Or vice versa
3. **Severe degradation**:
   - Fragments too short
   - Check fragment length distribution

Inconsistent Results Across Samples

Check:

1. **Library types match**:

   ```
   grep "expected_format" */lib_format_counts.json
   ```
2. **Mapping rates similar**:

   ```
   grep "Mapping rate" */logs/salmon_quant.log
   ```

3. **Fragment lengths similar**:

   ```
   # Compare fld.gz across samples
   ```

4. **Same index used**:

   ```
   grep "index_seq_hash" */meta_info.json | sort -u
   ```

Best Practices

Standard RNA-seq Quantification

```
salmon quant \
    --validateMappings \           # Accuracy
    --gcBias \                     # Correct GC bias
    --seqBias \                    # Correct hexamer bias
    --posBias \                    # Correct positional bias
    --libType A \                  # Auto-detect
    --index salmon_index \
    --mates1 R1.fastq.gz \
    --mates2 R2.fastq.gz \
    --threads 4 \
    --output sample_quant
```

Quick QC Run (Speed Priority)

```
salmon quant \
    --libType A \                  # Auto-detect only
    --index salmon_index \
    --mates1 R1.fastq.gz \
    --mates2 R2.fastq.gz \
    --threads 8 \                  # More cores
    --output sample_quant
    # No bias corrections → 2x faster
```

Publication-Quality (Accuracy Priority)

```
salmon quant \
    --validateMappings \
    --gcBias \
    --seqBias \
    --posBias \
    --numBootstraps 30 \           # Uncertainty quantification
    --libType A \
```

```
    --index salmon_index \
    --mates1 R1.fastq.gz \
    --mates2 R2.fastq.gz \
    --threads 8 \
    --output sample_quant
```

Related Documentation

- **Salmon Index**: `docs/salmon_index.md` - Building the index
- **tximport**: https://bioconductor.org/packages/tximport/ - Importing to R
- **DESeq2**: `docs/deseq2.md` - Differential expression
- **Salmon Manual**: https://salmon.readthedocs.io/

Document Version: 1.0
Last Updated: January 2026
Salmon Version: 1.10.0+
Applicable to: Bulk RNA-seq, scRNA-seq (alevin mode), metatranscriptomics