**Figure 2A**

Codes of analysis RiboD-PETRI scRNA-seq sequencing data

1.Since both reads1 and reads2 files contain information about barcode and genome, in order to improve the data utilisation efficiency of reads1 and reads2 files, we pre-processed the reads1 and reads2 files respectively. We used codes to combine the reads1 and reads2 files of scRNA-seq clean data (such as: "{sample}\_R1.fq.gz" and "{sample}\_R2.fq.gz") to mention barcode sequences and genome sequences, respectively, and processed them into four fastq files for downstream splitting of barcode and comparison processes. Make sure all fastq files have names in the form: {sample name}\_{S#}\_L00{#}\_R1\_001.fastq.gz , {sample name}\_{S#}\_L00{#}\_R2\_001.fastq.gz , {sample name}B\_{S#}\_L00{#}\_R1\_001.fastq.gz and {sample name}B\_{S#}\_L00{#}\_R2\_001.fastq.gz.

The codes are:

$cutadapt -g NNNNNNNNAGAATACACGACGCTCTTCCGATCT -o r1.fastq {sample}\_R1.fq.gz

$cutadapt -g NNNNNNNNAGAATACACGACGCTCTTCCGATCT -o r2.fastq {sample}\_R2.fq.gz

$cat r1.fastq | seqkit subseq -r 1:85 > {sample}\_S5\_L001\_R1\_001.fastq

$cat r2.fastq | seqkit subseq -r 80:100 > {sample}\_S5\_L001\_R2\_001.fastq

$cat r2.fastq | seqkit subseq -r 1:85 > {sample}B\_S5\_L001\_R1\_001.fastq

$cat r1.fastq | seqkit subseq -r 80:100 > {sample}B\_S5\_L001\_R2\_001.fastq

$gzip {sample}\_S5\_L001\_R1\_001.fastq

$gzip {sample}\_S5\_L001\_R2\_001.fastq

$gzip {sample}B\_S5\_L001\_R1\_001.fastq

$gzip {sample}B\_S5\_L001\_R2\_001.fastq

2.Create a folder named demo and then in the demo folder create the folders with the sample names : {sample} and {sample}B. Put {sample}\_S5\_L001\_R1\_001.fastq and {sample}\_S5\_L001\_R2\_001.fastq in the folder named {sample}; Put {sample}B\_S5\_L001\_R1\_001.fastq and {sample}B\_S5\_L001\_R2\_001.fastq in the folder named {sample}B.

3.Then we performed single-cell sequencing data analyses according to the processes and scripts of previous articles "Prokaryotic single-cell RNA sequencing by in situ combinatorial indexing". The scripts folder was also been put in "scripts" folder which we made some modifications owing to differences in python versions. The data are all in one lane, so we removed the step of merging the lanes. Then in the "demo" folder, run the following codes:

$python [path]/scripts/sc\_pipeline\_11.py {sample}\_S5 {n\_lanes}

$python [path]/scripts/sc\_pipeline\_11.py {sample}B\_S5 {n\_lanes}

# “sample” is sample name and S number (eg first1000\_S5)

# n\_lanes is the number of sequencing lanes for analysis - if lanes are merged, then the single file should be names with suffix \_L001\_R1\_001.fastq.gz and n\_lanes set to 1. The script will count lanes from 1 to n\_lanes so always start numbering from 1.

# sc\_pipeline\_11 runs fastqc, quality filter, and barcode demultiplexing

For example:

$python path\_to/scripts/sc\_pipeline\_11.py 1214-7\_S5 1

$python path\_to/scripts/sc\_pipeline\_11.py 1214\_7B\_S5 1

4.Look at {sample}\_bc1\_ReadsPerBC.eps, {sample}\_bc1\_kneePlot.eps, {sample}B\_bc1\_ReadsPerBC.eps and {sample}B\_bc1\_kneePlot.eps to determine number of BCs to include in further analysis. In the following analysis step, we only need to generate statistical tables for UMI, genes and cells, so we remove the step of synthesising the matrix. This script generates two files named as ("{sample}\_CDS\_v11\_threshold\_0\_filtered\_mapped\_UMIs.txt" and "{sample}B\_CDS\_v11\_threshold\_0\_filtered\_mapped\_UMIs.txt").

The codes were:

SCRIPT: [path]/scripts/pipeline.sh {sample} {n\_BCs} {fasta} {gtf} {custom\_name}

# n\_BCs is number of BCs to include in further analysis (typically 10000-80000)

# fasta is location and name of fasta for alignment

# gtf is location and name of gtf for feature calling - see example.gtf for example format (specifically, gene names should be indicated by 'name=')

# Custom name is a new name for the sample, corresponding to maybe the gtf used or other specific input of the pipeline. For example, we might analyze the same cells by CDS or by exon and would indicate that in the custom name. Custom name can be the same as sample name if desired.

# pipeline.sh includes a number of cleanup commands at the end. If interested in intermediate files, these can be easily commented out.

For example:

$sh ./scripts/pipeline.sh 1214\_7 10000 Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.dna.toplevel.fa.gz Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.46.gtf 1214\_7\_CDS

$sh ./scripts/pipeline.sh 1214\_7B 10000 Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.dna.toplevel.fa.gz Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.46.gtf 1214\_7B\_CDS

5.The files ("{sample}\_CDS\_v11\_threshold\_0\_filtered\_mapped\_UMIs.txt" and "{sample}B\_CDS\_v11\_threshold\_0\_filtered\_mapped\_UMIs.txt") obtained in step 4 are combined into a matrix containing UMI, gene and barcode information.

The codes were:

$cat {sample}\_CDS\_v11\_threshold\_0\_filtered\_mapped\_UMIs.txt {sample}B\_CDS\_v11\_threshold\_0\_filtered\_mapped\_UMIs.txt > {sample}.txt

$sed -i "s/{sample}B/{sample}/g" {sample}.txt

$sed -i "s/Chromosome://g" {sample}.txt

$awk 'BEGIN{ FS=" ";OFS="+" }{ print $1,$2,$3 }' {sample}.txt > {sample}.2.txt

$cat {sample}.2.txt | sort | uniq > {sample}.3.txt

$awk 'BEGIN{ FS="+";OFS=" " }{ print $1,$3 }' {sample}.3.txt > {sample}.4.txt

$awk 'BEGIN{ FS=" ";OFS="+" }{ print $1,$2 }' {sample}.4.txt > {sample}.5.txt

$cat {sample}.5.txt | sort | uniq -c > {sample}.6.txt

$awk 'BEGIN{ FS=" ";OFS="\t" }{ print $2,$3,$1 }' {sample}.6.txt > {sample}.7.txt

$awk 'BEGIN{ FS="+";OFS="\t" }{ print $1,$2,$3 }' {sample}.7.txt > {sample}.8.txt

$awk 'BEGIN{ FS="\t\t";OFS="\t" }{ print $1,$2,$3 }' {sample}.8.txt > {sample}.9.txt

$awk 'BEGIN{ FS="\t";OFS="+" }{ print $1,$2,$3 }' {sample}.9.txt > {sample}.10.txt

$sed -i '1i\cell+gene+count' {sample}.10.txt

$awk 'BEGIN{ FS="+";OFS="\t" }{ print $1,$2,$3 }' {sample}.10.txt > {sample}.11.txt

$awk 'BEGIN{ FS="\t";OFS="," }{ print $1,$2,$3 }' {sample}.11.txt > {sample}.12.csv

>>>python

>>>import pandas as pd

>>>import numpy as np

>>>table = pd.read\_csv("{sample}.12.csv")

>>>table=pd.pivot\_table(table,index=["cell"],columns=["gene"],values=["count"],fill\_value=0)

>>>df = pd.DataFrame(table)

>>>df.to\_csv('{sample}.csv')

1. Download "{sample}.csv", removed rows and columns that clearly not belong to cells and genes (such as "cell", "16" and so on.), and count the number of UMI in each cell in the gene expression matrix, and select cells with a UMI count greater than 15 to calculate the median UMI count.

**Figure 2B, C**

1. Based on the gene expression matrix obtained in the previous step, count the number of genes detected in each cell. For gene/cell analysis, we used the matrix {sample}.csv, removed rows and columns that clearly not belong to cells and genes (such as "cell", "16" and so on.), and convert it to "txt" format file ({sample}.UMI-cell.txt) to counted the number of genes in each cell using the following codes in R software:

>a <- read.table("{sample}.UMI-cell.txt")

>a1 <- a[-1,-1]

>a1[a1 != 0] <- 1

>write.table(a1,"{sample}.gene-cell.xlsx",sep="\t", quote=F, row.names=T,col.names = T)

1. Sort the cells based on the number of genes and UMIs counted in each cell, and take the top 1,000, 5,000, 10,000 cells to calculate the median number of UMIs or genes contained in these cells.

**Figure 2D, E, G, H, I and Figure 2-figure supplement 2**

1. Import the {sample}.UMI-cell.txt into Seurat v4.3.0 (https://satijalab.org/seurat/articles/get\_started.html) for single-cell analysis. Taking sample 0427-3 as an example.

The Seurat codes were:

For Single-cell RiboD-PETRI data of E .coli 3h:

>Ecoli3h.data<-read.table("F:/0427-3.txt")

>Ecoli3h.data1 <- Ecoli3h.data[-1,-1]

>colnames(Ecoli3h.data1) <- Ecoli3h.data[1,-1]

>rownames(Ecoli3h.data1) <- Ecoli3h.data[-1,1]

>Ecoli3h2 <- CreateSeuratObject(counts = t(Ecoli3h.data1), project = "3", min.cells = 1, min.features =1)

>Ecoli3h3 <- subset(Ecoli3h2, subset = nFeature\_RNA >200 & nFeature\_RNA < 5000)

>VlnPlot(Ecoli3h3, features = c("nFeature\_RNA", "nCount\_RNA"), ncol = 2)

>plot2 <- FeatureScatter(Ecoli3h3, feature1 = "nCount\_RNA", feature2 = "nFeature\_RNA")

>Ecoli3h4 <- NormalizeData(Ecoli3h3, normalization.method = "LogNormalize", scale.factor = 1e4)

>Ecoli3h5 <- FindVariableFeatures(Ecoli3h4, selection.method = 'vst', nfeatures = 500)

>Ecoli3h6 <- ScaleData(Ecoli3h5)

>Ecoli3h7 <- RunPCA(Ecoli3h6,features = VariableFeatures(object = Ecoli3h6))

>Ecoli3h10 <- FindNeighbors(Ecoli3h7, dims = 1:6)

>Ecoli3h11 <- FindClusters(Ecoli3h10, resolution = 0.3)

>Ecoli3h <- RunUMAP(Ecoli3h11, dims = 1:6)

>DimPlot(Ecoli3h, reduction = 'umap', label = TRUE, pt.size = 0.2)

>DimPlot(Ecoli3h, reduction = 'umap', label = TRUE, pt.size = 0.5)

>cluster1.markers <- FindMarkers(Ecoli3h , ident.1 = 0 , min.pct = 0)

>Ecoli3h.markers <- FindAllMarkers(Ecoli3h, only.pos = TRUE, min.pct = 0.25, logfc.threshold = 0.25)

>FeaturePlot(Ecoli3h, features = c("b2513","b3424","b2917","b0920","b3947","b0033","b4067","b1278"), pt.size = 0.85)

>FeaturePlot(Ecoli3h, features = c("b0833"))

>DimPlot(Ecoli3h, reduction = 'umap', label = TRUE, pt.size = 0.5)

>FeaturePlot(Ecoli3h,features = c("nCount\_RNA"), pt.size = 0.85)

>Ecoli3h.markers %>% group\_by(cluster) %>% top\_n(n = 10, wt = avg\_log2FC) -> top10

>DoHeatmap(Ecoli3h, features = top10$gene) + NoLegend()

>VlnPlot(object = Ecoli3h,features =c("b1886","b0483","b3985","b2917","b0920","b4067","b3424","b3947","b2452","b0586","b0033","b1278","b2513"))

**Figure 2F**

Pathway enrichment analysis of marker gene, we used the R package clusterProfiler{Yu, 2012 #50}. The marker genes were screened by logFC > 0.2, pvalue < 0.001.

>library("clusterProfiler")

>library(org.EcK12.eg.db)

>library(R.utils)

>R.utils::setOption("clusterProfiler.download.method",'auto')

>Tn2<-read.table("3h-KEGG-p0.001.txt")

>Tn5<-Tn2[,1]

>kegg <- enrichKEGG(

gene = Tn5,

organism = 'eco',

pAdjustMethod = 'fdr',

pvalueCutoff = 0.2,

qvalueCutoff = 0.2,

)

>barplot(kegg,color = "pvalue")