Identification and quantification of polyA sites from different 3'end seq data with the PolyAseqTrap package

Wenbin Ye, Xin Cheng, Xiaohui Wu

2024-12-12

Contents

1 Overview

We evaluated PolyAseqTrap against existing 3' sequencing pipelines using data from 16 different 3' sequencing techniques across multiple species. This comprehensive evaluation demonstrates the effectiveness and robustness of PolyAseqTrap. In this guide, we use demo data from three species—human, mouse, and Arabidopsis to illustrate how PolyAseqTrap can be applied for unified and user-friendly polyA site identification and analysis across different types of 3' sequencing data.

2 Preparations

2.1 Preprocessing of 3'seq dataset

The 3'seq FASTQ data can be obtained from the NCBI SRA database. Before analysis, it is important to preprocess the data using tools such as Cutadapt and UMI-tools to trim adapter sequences and handle unique molecular identifiers (UMIs). These preprocessing steps are essential to ensure the accuracy and quality of the data before downstream analysis with PolyAseqTrap.

Additionally, we used regular expressions with fuzzy matching to identify the polyA stretches at the 3'end or polyT stretches at the 5'end of reads, collectively referred to as potential polyA tails.This polyA tail information was recorded in the sequence header of the FASTQ file, and the polyA tails in the reads were subsequently trimmed.

Note that we used only a subset of the original data as demo data. Please download the full dataset from NCBI SRA (e.g. SRR1168402 and SRR11837378)

Method 1: using R scrip to identify and trim the polyT/polyA stretches

```
library(PolyAseqTrap)
## identify and trim the polyT stretches
file_T <- system.file("extdata", "SRR1168402_T.fastq", package = "PolyAseqTrap")
# The output is 'SRR11837378_A.A.fq', which can be used as input for alignment tools
findTailAT(infile=file_T, odir=NULL,
         poly='T', ml=20, mp=5, mg=10, mm=2, deep=FALSE,
         mtail=6, mper=0.75, mr=3, review=TRUE, debug=TRUE,
          bar=0, reg=1, suf=NULL)


## identify and trim the polyA stretches
file_A <- system.file("extdata", "SRR11837378_A.fastq", package = "PolyAseqTrap")
# The output is 'SRR11837378_A.A.fq', which can be used as input for alignment tools
findTailAT(infile=file_A, odir=NULL, poly='A',
          ml=20, mp=5, mg=10, mm=2,
          deep=0, mtail=6, mper=0.75,
          mr=3, review=TRUE, debug=TRUE, bar=0, reg=1, suf=NULL)
# Example of sequence headers in the FASTQ file
##@SRR1168402.78_TTTCTTTTTTTTTTTT
##@SRR11837378.19_TAACAAATT_AAAAAAAAAAAAA
```

Method 2: using Perl script to identify and trim the polyT/polyA stretches Given the potentially large size of the data, R may face performance limitations with large datasets. To mitigate this, we also provide corresponding Perl scripts as an alternative for faster data processing, especially for users working with large-scale datasets.

```
## Usage
#cd path_of_PolyAseqTra
perl ./PolyAseqTrap/scripts/MAP_findTailAT.pl -h

## identify and trim the polyA stretches
perl ./PolyAseqTrap/scripts/MAP_findTailAT.pl \
 -in  ./PolyAseqTrap/inst/extdata/SRR1168402_T.fastq \
  -poly T -ml 20 -mp 5 -mg 10 -mm 2 -mr 3 \
  -mper 0.75 -mtail 6 -deep F -reg 1 \
  -odir ./ -suf ""  -oraw F  -debug T -review T

## identify and trim the polyA stretches
```

```
perl ./PolyAseqTrap/scripts/MAP_findTailAT.pl \
 -in  ./PolyAseqTrap/inst/extdata/SRR11837378_A.fastq \
 -poly A -ml 20 -mp 5 -mg 10 -mm 2 -mr 3 \
 -mper 0.75 -mtail 6 -deep F -reg 1 \
 -odir ./ -suf ""  -oraw F  -debug T -review T
```

Then use alignment tool such as STAR to perform local alignment with soft-clipping allowed.

```
STAR --runThreadN ${cpu} --genomeDir ${REFindex}  \
--outFileNamePrefix  ${outpath}/sampleName \
--readFilesIn   ${input}  \
--outSAMtype BAM SortedByCoordinate \
--outFilterMultimapNmax 1 \
--outMultimapperOrder Random
```

2.2 Reference genome

In mapping process of alignment tool (e.g., STAR), the reference genomes and annotation files can be obtained from Ensembl, UCSC and NCBI database. For polyA site identification in PolyAseqTrap, the reference genome format BSgenome is required for detecting A-rich polyA sites and removing internal priming. The BSgenome of ***Arabidopsis thaliana (TAIR10)*** for this example can be downloaded from the PolyAseqTrap GitHub repository. If there is no corresponding BSgenome package for your species, please refer to the BSgenome package for instructions on creating a custom BSgenome object.

- **Human genome (hg39)**

```
# for Homo sapiens (UCSC genome hg38)
library("BSgenome.Hsapiens.UCSC.hg38", quietly = TRUE)
bsgenome <-BSgenome.Hsapiens.UCSC.hg38
```

- **Mouse genome (mm39)**

```
# for Mus musculus (UCSC genome mm39)
library("BSgenome.Mmusculus.UCSC.mm39", quietly = TRUE)
bsgenome <-  BSgenome.Mmusculus.UCSC.mm39
```

- **Arabidopsis genome (TAIR10)**

```
# for Arabidopsis thaliana (Ensembl TAIR110)
library("BSgenome.Athaliana.ENSEMBL.TAIR10", quietly = TRUE)
bsgenome <- BSgenome.Athaliana.ENSEMBL.TAIR10
```

2.3 Genome annotation and 3'UTR region extraction

Genome annotation stored in a GFF/GTF file or a TXDB R object can be used to annotate PACs. The `parseGff` or `parseGenomeAnnotation` function from `movAPA` is employed to parse the provided annotation, and the processed annotation can then be saved as an Rdata object for future use. For more details, please refer to movAPA.

```
install.packages("devtools")
require(devtools)
install_github("BMILAB/movAPA")
library(movAPA)
browseVignettes('movAPA')
```

It is worth noting that 3'UTR annotations can be particularly useful for identifying V8 polyA sites in PolyAseqTrap, especially for 3'seq data lacking polyA tail information. The GRanges object of the 3'UTR annotations used in this study can be downloaded from the PolyAseqTrap GitHub repository, including human (hg39), mouse (mm39), and Arabidopsis (TAIR10).

- **for human (hg39)**

```r
library(movAPA)
# for Homo sapiens
# download Homo sapiens (hg38) from Ensembl
athGFF <- "Homo_sapiens.GRCh38.110.gtf"
gff<- parseGff(athGFF)
saveRDS(gff,file="Ensembl_Homo_sapiens.GRCh38.110.Rdata")

threeUTR.data <-subset(gff$anno.need,type=="three_prime_UTR")
threeUTRregion <- makeGRangesFromDataFrame(threeUTR.data,keep.extra.columns = F)
saveRDS(threeUTRregion,file="ThreeRegion_Homo_sapiens.Rdata")
```

- **for mouse (mm39)**

```r
library(movAPA)
# for Mus musculus
# download Mus musculus (mm39) from Ensembl
athGFF <- "Mus_musculus.GRCm39.110.gtf"
gff<- parseGff(athGFF)
saveRDS(gff,file="Ensembl_Mus_musculus.GRCm39.110.Rdata")
threeUTR.data <-subset(gff$anno.need,type=="three_prime_UTR")
threeUTRregion <- makeGRangesFromDataFrame(threeUTR.data,keep.extra.columns = F)
save
```

- **for Arabidopsis (TAIR10)**

```r
library(movAPA)
# for Arabidopsis thaliana
# download Arabidopsis (TAIR10) from Ensembl Plant
athGFF <- "Arabidopsis_thaliana.TAIR10.57.gff3"
gff<- parseGff(athGFF)
saveRDS(gff,file="Ensembl_Arabidopsis_thaliana.TAIR10.57.Rdata")
threeUTRregion <- makeGRangesFromDataFrame(threeUTR.data,keep.extra.columns = F)
saveRDS(threeUTRregion,file="ThreeRegion_Arabidopsis_thaliana.Rdata")
```

3 Identify PACs at varying confidence levels from BAM file

PolyAseqTrap employs a priority model to classify aligned reads into three categories based on the presence and composition of the polyA tail:

- reads with perfectly matched polyA tails (**C1**).
- reads with partially matched polyA tails (**C2**).
- reads without polyA tails but enriched at the 3'end (**C3**).

Additionally, each category is further subdivided into subclasses (**V1** to **V8**) by considering factors such as polyA tail length, base composition, and alignment results. If an aligned read cannot reliably pinpoint a polyA site, it is labeled as `count`. The `use.as.count` column is set to 1 if aligned reads are within 24 nt (default) of identified PACs, indicating that these reads can be used to quantify PACs.

3.1 Identify PACs in human genome

Here we demonstrate how to use PolyAseqTrap to identify and quantify polyA sites using two different 3'seq datasets from the human genome: PolyA-Seq (SRR299116, polyT stretches) and PAS-seq (SRR11837378, polyA stretches). For simplicity, we focus on chromosome 22 in this example. The corresponding BAM files are available in the PolyAseqTrap GitHub repository. The demonstration human PACs results can be loaded with `data(PACs_human)`.

Note: here the `adjust.chr` parameter in the `FindPTA` function is set to `TRUE` to add the "chr" prefix to chromosome names in the BAM file, as the reference genome used for alignment from the Ensembl does not include the "chr" prefix.

Example 1: Identify and quantify polyA sites in 3'end data with polyT stretches

```
library(PolyAseqTrap,  warn.conflicts = FALSE, quietly=TRUE)
library(BSgenome.Hsapiens.UCSC.hg38)
bsgenome <-  BSgenome.Hsapiens.UCSC.hg38

# load 3'UTR annotation for detecting V8 polyA site
threeUTR_path <- system.file("extdata",
                             "ThreeRegion_Homo_sapiens.Rdata",
                             package = "PolyAseqTrap")
threeUTRregion <- readRDS(threeUTR_path)

# get bam file
bam_T_file <- system.file("extdata",
                          "SRR299116_T_chr22_hg_sorted.bam",
                          package = "PolyAseqTrap")


# identify and quantify PACs, it wouldn't predict V8 polyA site if
# without providing 3'UTR annotation.
# here "adjust.chr" is set to TRUE to add "chr" prefix
pa.hg.result <- FindPTA(bam=bam_T_file,
        yieldSize=10^7,
        reverse=F,
        bsgenome=bsgenome,
        d=24,
        poly='T',
        adjust.chr=TRUE,
        threeUTRregion=threeUTRregion,
        cutoffCount = 5,
        ext3UTRlen =   1000 ,
        isDRS = FALSE,
        run.quantify=TRUE)
#> [1] "The program is reading the BAM files."
#> [1] "Extract reference sequence around polyA coordinate."
#> [1] "Detecting reads with polyA tail and adjust polyA coordinate"
#> [1] "C1: processing reads with 100% An tails"
#> [1] "Detecting V1 polyA sites"
#> [1] "C2: processing reads with !100% An tails"
#> [1] "Excluding reads with perfectly matched polyA tails but polyA tail actually from ..."
#> [1] "Processing reads without soft-clippingand and with partially matched polyA tails"
#> [1] "Detecting V2 polyA sites"
#> [1] "Detecting part of polyA tail from reference genome"
#> [1] "Detecting V7 polyA sites"
#> [1] "Detecting V3 and V4 polyA sites"
#> [1] "Detecting V5 polyA sites"
#> [1] "Detecting V6 polyA sites"
#> [1] "C3: reads without polyA tails"
#> [1] "Identifying potential polyA sites affected by SNPs"
#> [1] "Searching for A-rich fragments surrounding polyA sites"
#> [1] "C3: detecting V8 polyA sites"
#> [1] "Completed: Identification and quantification of polyA sites"
# Display details of alignment and category of aligned reads
rmarkdown::paged_table(head(pa.hg.result$pa.table[,c("readName","cigar","seq",
                                                     "softClipFragment","trimmed_seq",
                                                     "unmapped_seq",
                                                     "reference_seq","is_Arich",
                                                     "chr","strand","coord",
                                                     "level","class","use.as.count")]),
                       options = list(rows.print = 5, cols.print = 5))
```

```
#category of aligned reads
t(table(pa.hg.result$pa.table$class))
#>
#>              C1     C2     C3  Count
#>    [1,]    1012   7814 173173  30036
#subclasses of aligned reads
t(table(pa.hg.result$pa.table$level))
#>
#>              V1     V2     V3     V4     V5     V6     V7     V8  Count
#>    [1,]     908     55    103    727     49      3   6981 173173  30036

# Display details of PACs
rmarkdown::paged_table(head(pa.hg.result$pa.coord),
                       options = list(rows.print = 5, cols.print = 5))
```

```
#filter PACs that were supported by at least five reads
pac5.hg <- subset(pa.hg.result$pa.coord,total.count>=5)
```

Example 2: Identifying and quantifying polyA sites in 3'seq data with polyA stretches

```
library(PolyAseqTrap, warn.conflicts = FALSE, quietly=TRUE)
library(BSgenome.Hsapiens.UCSC.hg38)
bsgenome <-  BSgenome.Hsapiens.UCSC.hg38

# load 3'UTR annotation for detecting V8 polyA site
threeUTR_path <- system.file("extdata",
                             "ThreeRegion_Homo_sapiens.Rdata",
                             package = "PolyAseqTrap")
threeUTRregion <- readRDS(threeUTR_path)

# get bam file
bam_A_file <- system.file("extdata",
                          "SRR11837378_A_chr22_hg_sorted.bam",
                          package = "PolyAseqTrap")

# identify and quantify PACs, it wouldn't predict V8 polyA site if
#without providing 3'UTR annotation.
pa.hg.result <- FindPTA(bam=bam_A_file,
                        yieldSize=10^7,
                        reverse=F,
                        bsgenome=bsgenome,
                        d=24,
                        poly='A',
                        adjust.chr=TRUE,
                        threeUTRregion=threeUTRregion,
                        cutoffCount = 5,
                        ext3UTRlen =   1000 ,
                        isDRS = FALSE,
                        run.quantify=TRUE)
# Display details of alignment and category of aligned reads
rmarkdown::paged_table(head(pa.hg.result$pa.table[,c("readName","cigar","seq",
                                                     "softClipFragment","trimmed_seq",
                                                     "unmapped_seq",
                                                     "reference_seq","is_Arich",
                                                     "chr","strand","coord",
                                                     "level","class","use.as.count")]),
                       options = list(rows.print = 5, cols.print = 5))
#category of aligned reads
knitr::kable(t(table(pa.hg.result$pa.table$class)))
#subclasses of aligned reads
knitr::kable(t(table(pa.hg.result$pa.table$level)))

# Display details of PACs
rmarkdown::paged_table(head(pa.hg.result$pa.coord),
                       options = list(rows.print = 5, cols.print = 5))
#filter PACs that were supported by at least five reads
pac5.hg <- subset(pa.hg.result$pa.coord,total.count>=5)
```

3.2 Identify PACs in mouse genome

Here we use 3P-Seq data (SRR766743, polyT stretches) to demonstrate how to use PolyAseqTrap to identify and quantify polyA sites in mouse genome. For simplicity, we focus on chromosome 19 in this example. The corresponding BAM files are available in the PolyAseqTrap GitHub repository. The demonstration mouse PACs results can be loaded with `data(PACs_mouse)`.

Note: here the `adjust.chr` parameter in the `FindPTA` function is set to `TRUE` to add the "chr" prefix to chromosome names in the BAM file, as the reference genome used for alignment from the Ensembl does not include the "chr" prefix.

```
library(PolyAseqTrap,  warn.conflicts = FALSE, quietly=TRUE)
library(BSgenome.Mmusculus.UCSC.mm39)
bsgenome <-  BSgenome.Mmusculus.UCSC.mm39
# load 3'UTR annotation for detecting V8 polyA site
threeUTR_path <- system.file("extdata",
                             "ThreeRegion_Mus_musculus.Rdata",
                             package = "PolyAseqTrap")
threeUTRregion <- readRDS(threeUTR_path)

# get bam file
bam_T_file <- system.file("extdata",
                          "SRR766743_T_chr19_mm_sorted.bam",
                          package = "PolyAseqTrap")
```

```
# identify and quantify PACs, it wouldn't predict V8 polyA site if
#without providing 3'UTR annotation
pa.mm.result <- FindPTA(bam=bam_T_file,
                        yieldSize=10^7,
                        reverse=F,
                        bsgenome=bsgenome,
                        d=24,
                        poly='T',
                        adjust.chr=TRUE,
                        threeUTRregion=threeUTRregion,
                        cutoffCount = 5,
                        ext3UTRlen =   1000 ,
                        isDRS = FALSE,
                        run.quantify=TRUE)
#> [1] "The program is reading the BAM files."
#> [1] "Extract reference sequence around polyA coordinate."
#> [1] "Detecting reads with polyA tail and adjust polyA coordinate"
#> [1] "C1: processing reads with 100% An tails"
#> [1] "Detecting V1 polyA sites"
#> [1] "C2: processing reads with !100% An tails"
#> [1] "Excluding reads with perfectly matched polyA tails but polyA tail actually from ..."
#> [1] "Processing reads without soft-clippingand and with partially matched polyA tails"
#> [1] "Detecting V2 polyA sites"
#> [1] "Detecting part of polyA tail from reference genome"
#> [1] "Detecting V7 polyA sites"
#> [1] "Detecting V3 and V4 polyA sites"
#> [1] "Detecting V5 polyA sites"
#> [1] "Detecting V6 polyA sites"
#> [1] "C3: reads without polyA tails"
#> [1] "Identifying potential polyA sites affected by SNPs"
#> [1] "Searching for A-rich fragments surrounding polyA sites"
#> [1] "C3: detecting V8 polyA sites"
#> [1] "Completed: Identification and quantification of polyA sites"


# Display details of alignment and category of aligned reads
rmarkdown::paged_table(head(pa.mm.result$pa.table[,c("readName","cigar","seq",
                                                     "softClipFragment","trimmed_seq",
                                                     "unmapped_seq",
                                                     "reference_seq","is_Arich",
                                                     "chr","strand","coord",
                                                     "level","class","use.as.count")]),
                       options = list(rows.print = 5, cols.print = 5))
```

```
#category of aligned reads
t(table(pa.mm.result$pa.table$class))
#>
#>           C1     C2     C3  Count
#>   [1,] 224233   3339 161150  39634
#subclasses of aligned reads
t(table(pa.mm.result$pa.table$level))
#>
#>           V1     V2     V3     V4     V5     V6     V7     V8  Count
#>   [1,] 221261   2092    249   1246    880      2   1842 161150  39634
```

```r
# Display details of PACs
rmarkdown::paged_table(head(pa.mm.result$pa.coord),
                        options = list(rows.print = 5, cols.print = 5))
```

```r
#filter PACs that were supported by at least five reads
pac5.mm <- subset(pa.mm.result$pa.coord,total.count>=5)
```

3.3 Identify PACs in Arabidopsis genome

Here we use PolyA-Tag-seq data (SRR5055884, polyT stretches) to demonstrate how to use PolyAseqTrap to identify and quantify polyA sites in Arabidopsis genome. For simplicity, we focus on chromosome 2 in this example. The corresponding BAM files are available in the PolyAseqTrap GitHub repository. The demonstration Arabidopsis PACs results can be loaded with `data(PACs_tair)`.

```r
library(PolyAseqTrap, warn.conflicts = FALSE, quietly=TRUE)
library(BSgenome.Athaliana.ENSEMBL.TAIR10)
bsgenome <-  BSgenome.Athaliana.ENSEMBL.TAIR10

# load 3'UTR annotation for detecting V8 polyA site
threeUTR_path <- system.file("extdata",
                             "ThreeRegion_Arabidopsis_thaliana.Rdata",
                             package = "PolyAseqTrap")
threeUTRregion <- readRDS(threeUTR_path)

# get bam file
bam_T_file <- system.file("extdata",
                          "SRR5055884_T_ch2_tair_sorted.bam",
                          package = "PolyAseqTrap")
# identify and quantify PACs, it wouldn't predict V8 polyA site if
# without providing 3'UTR annotation
pa.tair.result <- FindPTA(bam=bam_T_file,
                          yieldSize=10^7,
                          reverse=F,
                          bsgenome=bsgenome,
                          d=24,
                          poly='T',
                          adjust.chr=FALSE,
                          threeUTRregion=threeUTRregion,
                          cutoffCount = 5,
                          ext3UTRlen =   1000 ,
                          isDRS = FALSE,
                          run.quantify=TRUE)
#> [1] "The program is reading the BAM files."
#> [1] "Extract reference sequence around polyA coordinate."
#> [1] "Detecting reads with polyA tail and adjust polyA coordinate"
#> [1] "C1: processing reads with 100% An tails"
#> [1] "Detecting V1 polyA sites"
#> [1] "C2: processing reads with !100% An tails"
#> [1] "Excluding reads with perfectly matched polyA tails but polyA tail actually from ..."
#> [1] "Processing reads without soft-clippingand and with partially matched polyA tails"
#> [1] "Detecting V2 polyA sites"
#> [1] "Detecting part of polyA tail from reference genome"
#> [1] "Detecting V7 polyA sites"
#> [1] "Detecting V3 and V4 polyA sites"
#> [1] "Detecting V5 polyA sites"
#> [1] "Detecting V6 polyA sites"
#> [1] "C3: reads without polyA tails"
#> [1] "Identifying potential polyA sites affected by SNPs"
#> [1] "Searching for A-rich fragments surrounding polyA sites"
#> [1] "C3: detecting V8 polyA sites"
#> [1] "Completed: Identification and quantification of polyA sites"

# Display details of alignment and category of aligned reads
rmarkdown::paged_table(head(pa.tair.result$pa.table[,c("readName","cigar","seq",
                                                        "softClipFragment","trimmed_seq",
                                                        "unmapped_seq",
                                                        "reference_seq","is_Arich",
                                                        "chr","strand","coord",
                                                        "level","class","use.as.count")]),
                       options = list(rows.print = 5, cols.print = 5))
```

```
#category of aligned reads
t(table(pa.tair.result$pa.table$class))
#>
#>           C1     C2     C3  Count
#>   [1,] 630004  59850  11614  19999
#subclasses of aligned reads
t(table(pa.tair.result$pa.table$level))
#>
#>           V1     V2     V3     V4     V5     V6     V7     V8  Count
#>   [1,] 429707   9757  32559  22077 190540   5196     18  11614  19999

# Display details of PACs
rmarkdown::paged_table(head(pa.tair.result$pa.coord),
                       options = list(rows.print = 5, cols.print = 5))
```

```
#filter PACs that were supported by at least five reads
pac5.tair<- subset(pa.tair.result$pa.coord,total.count>=5)
```

4 Remove internal priming artifacts

Inspired by the **DeepPASTA** model (Arefeen, et al., 2019) that predicts polyA sites from DNA sequences, we designed a deep learning model called DeepIP to predict internal priming artifacts from A-rich polyA sites. DeepIP utilizes both convolutional neural network (CNN) and recurrent neural network (RNN). CNN extracts features from sequences, and RNN is used to combine the extracted feature effects for predicting internal priming artifacts.The corresponding DeepIP scripts are available in the PolyAseqTrap GitHub repository.

4.1 Install DeepIP

DeepIP can run on both Linux, and Windows, systems. To install and use DeepIP, you need to have Conda installed on your machine. Please follow the steps below to set up DeepIP:

- **Prerequisites**

Ensure that **Conda** is installed on your system. If not, you can download and install Miniconda or Anaconda from the following links:

- Miniconda

- Anaconda
- **Install DeepIP**

Once Conda is installed, you can create a new Conda environment and install DeepIP by running the following commands in your terminal (Linux) or command prompt (Windows):

```
# Create a new conda environment
conda create -n DeepIP python=3.7
conda env list
# Activate the environment
conda activate DeepIP
# Install additional dependencies
pip install keras
pip install tensorflow
pip install pandas
pip install sklearn
pip install scikit-klearn
```

4.2 Build the training model (optional)

Currently, we provide pre-trained models for human, mouse, and Arabidopsis species. If you would like to build a model for your own species, you can follow the steps below.

- **Prepare training data**

For model training, you need to prepare your training data. The training sequences should consist of the 100bp sequences upstream and downstream of the polyA site.

- **Build a training model**

For our pre-trained models for human, mouse, and Arabidopsis, the positive data of the model are A-rich sequences with polyA sites and the negative data are A-rich sequences without polyA sites.

```
python DeepIP_train.py \
  -trainSeq train_mini.fa \
  -trainedModel train_mini.fa.model.hdf5 \
  -epoch 10
```

Where:

- `trainSeq`: is the input training data (a FASTA file containing 200 bp geneome sequences).
- `trainedModel`: is the name of the output model (this will be saved as an HDF5 file).
- `epoch`: the number of training iterations (default is 100). You can set a different number based on your training needs.

Alternatively, you can run it in R.

```
library(reticulate)
use_condaenv("/path/miniconda3/envs/DeepIP/")
# Build your own model
py_run_string("trainSeq='train_mini.fa';
              trainedModel='train_mini.model.hdf5';
              epoch=10; seqLabel='01'")
py_run_file('DeepIP_train.py')
```

4.3 Test the model

Once your model is trained or if you are using a pre-trained model, you can proceed to test it on new sequences. The model accepts a 200 nt genomic sequence as input and predicts whether the middle position of the input sequence corresponds to an internal priming site. Here's how to perform the testing:

```
python DeepIP_test.py \
 -testSeq test_mini.fa \
 -trainedModel train_mini.fa.model.hdf5 \
 -outputFile test_result.csv
```

Alternatively, you can run it in R.

```
library(reticulate)
use_condaenv("/path/miniconda3/envs/DeepIP/")
# to identify whether PACs are internal priming artifacts
py_run_string("testSeq='test_mini.fa';
              trainedModel='train.mini.model.hdf5';
              outputFile='train.mini.predicted.csv';
              seqLabel='0/1'")
py_run_file('DeepIP_test.py')
```

4.4 Remove internal priming and regroup nearby cleavage sites

3'seq techniques based on oligo(dT), such as PAC-seq, PAS-seq, polyA-seq, and WTTS-seq, are prone to internal priming artifacts, which can lead to inaccurate identification of PACs. To mitigate this issue, `PolyAseqTrap` integrates a deep learning-based model, `DeepIP`, to accurately identify whether A-rich PACs are internal priming artifacts. To utilize `DeepIP`, first extract the 100 bp upstream and downstream sequences of A-rich polyA sites from the PACs results generated by the `FindPTA` function. These sequences are then classified by `DeepIP` to identify potential internal priming artifacts.

- **Extract a sequence of 200 nt surrounding A-rich polyA sites**

```
library(PolyAseqTrap, warn.conflicts = FALSE, quietly=TRUE)
library("BSgenome.Hsapiens.UCSC.hg38", quietly = TRUE)
bsgenome <-BSgenome.Hsapiens.UCSC.hg38
## load identified PACs in human genome generated by FindPTA function
data(PACs_human)

## extract 200 bp genome sequences surrounding A-rich polyA sites
generateFASTA(reads=PACs_human$pa.table,
              bsgenome=bsgenome,
              output.name="human_Arich_PACs.fasta")
```

- **Use DeepIP to classify A-rich polyA sites**

```
## use DeepIP to classify A-rich polyA sites
library(reticulate)
use_condaenv("/path/miniconda3/envs/DeepIP/")
py_run_string("testSeq='human_Arich_PACs.fasta';
              trainedModel='human.train.model.hdf5';
outputFile='DeepIP_result_hg.csv'")
py_run_file("DeepIP_test.py")

# Alternatively, run the following Python command outside R:
# python DeepIP_test.py -testSeq human_Arich_PACs.fasta  \
  -trainedModel human.train.model.hdf5 -outputFile DeepIP_result_hg.csv
```

- **Remove internal priming artifacts and regroup nearby cleavage sites**

```
### remove internal priming artifacts
ip.table <- read.csv("DeepIP_result_hg.csv")
head(ip.table)
#title                score     true_label predict_label res
#>chr22_19354941_-_0 0.4143335          0             0  TN
#>chr22_19382772_-_0 0.4143623          0             0  TN
#>chr22_19849373_-_0 0.4143335          0             0  TN

## extract and remove internal priming artifacts
ip.table <- subset(  ip.table,predict_label==0)
ip.table$title <- gsub("^>","",  ip.table$title)
PACs_human$pa.table$label <- paste0(
  PACs_human$pa.table$chr,"_",
  PACs_human$pa.table$coord,"_",
  PACs_human$pa.table$strand,"_0")
PACs_human$pa.table$level <- as.character(PACs_human$pa.table$level)
index <- which(PACs_human$pa.table$label %in% ip.table$title)

# remove internal priming artifacts
PACs_human$pa.table$level[index] <- "Count"
PACs_human$pa.table$level <- factor( PACs_human$pa.table$level,
                             levels=c("V1","V2","V3","V4","V5","V6","V7","V8","Count"))


## regroup nearby cleavage sites
PACs_human <- resut.PA(aln.result=PACs_human$pa.table,d=24)
```

Note: When running the analysis for other species, ensure that both the `bsgenome` object and the corresponding `training model` are updated to reflect the appropriate species. For example, for mouse, use `BSgenome.Mmusculus.UCSC.mm10` as the `bsgenome` object and `mouse.train.model.hdf5` as the training model. However, if a species-specific model is not available, the human model (`human.train.model.hdf5`) can still be used and should provide reliable results.

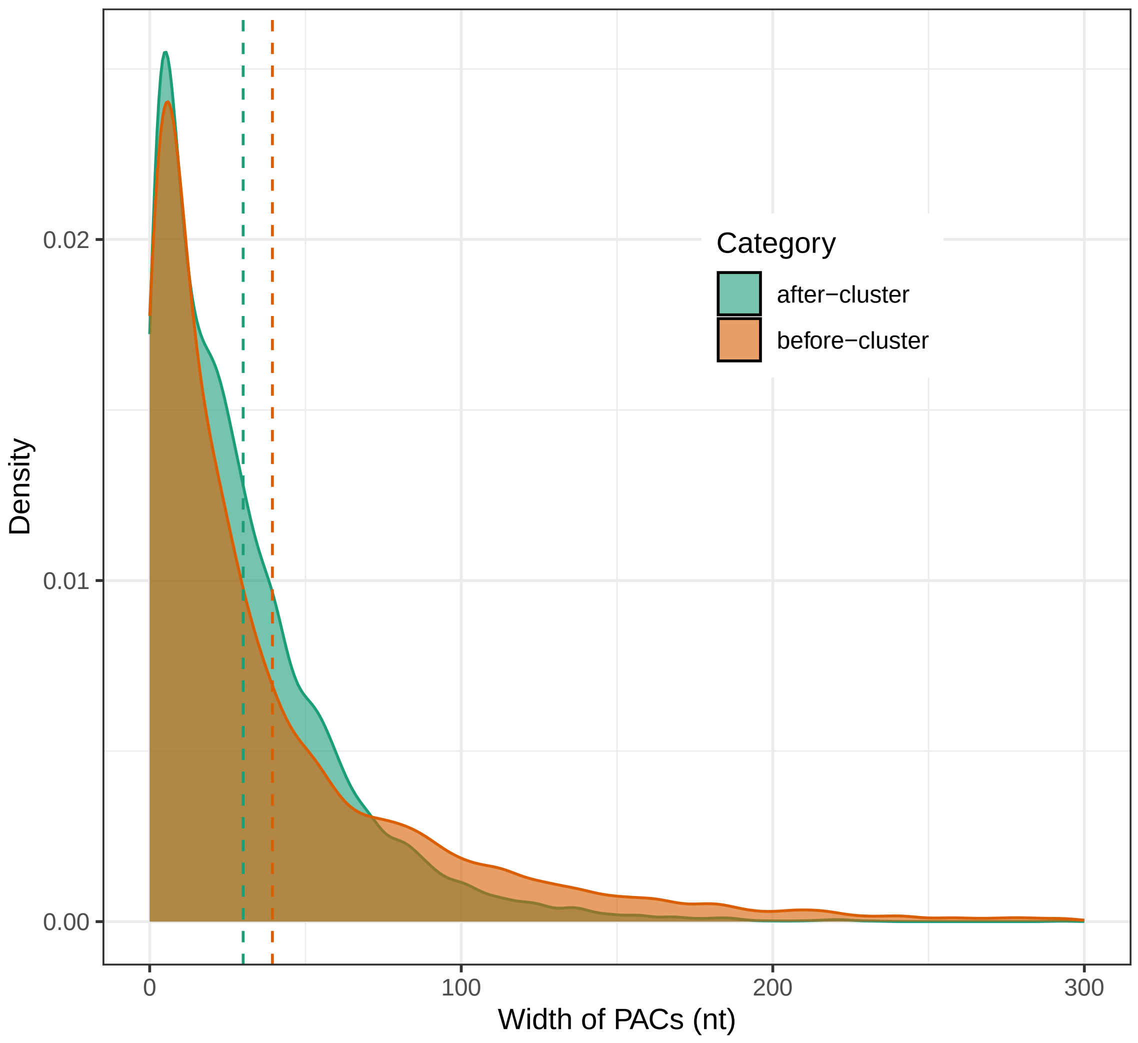
5 Mitigating Microheterogeneity in PACs

Eukaryotic cleavage and polyadenylation processes exhibit considerable microheterogeneity, with plants showing higher variability than animals (Ye, et al., 2021). Traditional methods often struggle to accurately identify distinct PACs due to overlapping regions. To address this, PolyAseqTrap integrates a density peak clustering algorithm, inspired by the QuantifyPoly(A) method, to mitigate microheterogeneity and improve the accuracy of PAC identification.

Here, we demonstrate this approach using Arabidopsis as an example, showcasing how it refines PAC determination and reduces the impact of microheterogeneity in plant species.

```
library(PolyAseqTrap, warn.conflicts = FALSE, quietly=TRUE)
## load identified PACs in Arabidopsis genome generated by FindPTA function
data("PACs_tair")

#filter PACs that were supported by at least five reads
PACs_tair$pa.coord <- subset(PACs_tair$pa.coord,total.count>=5)

# reduce the impact of microheterogeneity
PACs_tair <-split_pac(pa.data=PACs_tair,d=24,mc.cores=1)
#> [1] "Re-clustering by weighted density peak clustering algorithm!"
```

```r
#> [1] "Detail refer to R package-QuantifyPolyA"

#filter refined PACs that were supported by at least five reads
PACs_tair$split.clusters<- subset(PACs_tair$split.clusters,total.count>=5)


## compare and visualize the difference before and after reducing microheterogeneity
library(dplyr)
library(ggplot2)
width.density<- data.frame(pac.width=c(PACs_tair$pa.coord$width,
                                       PACs_tair$split.clusters$width) ,
                           type=rep(c("before-cluster","after-cluster"),
        time=c(length(PACs_tair$pa.coord$width),length(PACs_tair$split.clusters$width)))
)
width.density %>% dplyr::group_by(type) %>%
       dplyr::summarise(mean=mean(pac.width),median=median(pac.width ))
#> # A tibble: 2 x 3

#>   type             mean median
#>   <chr>           <dbl>  <dbl>
#> 1 after-cluster   30.0     22
#> 2 before-cluster  39.4     21



# density plot
mu <- plyr::ddply(width.density, "type", summarise, grp.mean=mean(pac.width))
ggplot(width.density, aes(x=pac.width,color=type,fill=type)) +
  geom_density(alpha=0.6)+
  geom_vline(data=mu, aes(xintercept=grp.mean, color=type),
             linetype="dashed")+theme_bw()+
  scale_fill_brewer(palette="Dark2")+
  scale_color_brewer(palette="Dark2")+xlim(0,300)+
  labs(x="Width of PACs (nt)",y="Density")+
  guides(fill = guide_legend(title="Category"),color="none")+
  theme(legend.position = c(0.7,0.7))
```



6 Annotate PACs

movAPA is a powerful R package developed by our team for modeling and visualizing the dynamics of PACs usage across biological samples. It provides comprehensive tools for preprocessing, annotating, and analyzing polyA sites, identifying polyA signals, profiling alternative polyadenylation (APA) dynamics, and generating visualizations. Here we briefly demonstrate how to seamlessly integrate `PolyAseqTrap` with `movAPA` for annotating the PACs identified by `PolyAseqTrap`. For detailed usage instructions and additional functionalities, please refer to the official movAPA documentation.

```r
library(PolyAseqTrap, warn.conflicts = FALSE, quietly=TRUE)
library("BSgenome.Hsapiens.UCSC.hg38", quietly = TRUE)
library(movAPA, warn.conflicts = FALSE, quietly=TRUE)
bsgenome <-BSgenome.Hsapiens.UCSC.hg38

athGFF <- "Homo_sapiens.GRCh38.110.gtf"
annotation <- parseGff(athGFF)

# load identified PACs in human genome using FindPTA function
data(PACs_human)
colnames(PACs_human$pa.coord)[1:3] <- c("chr","UPA_start","UPA_end")
```

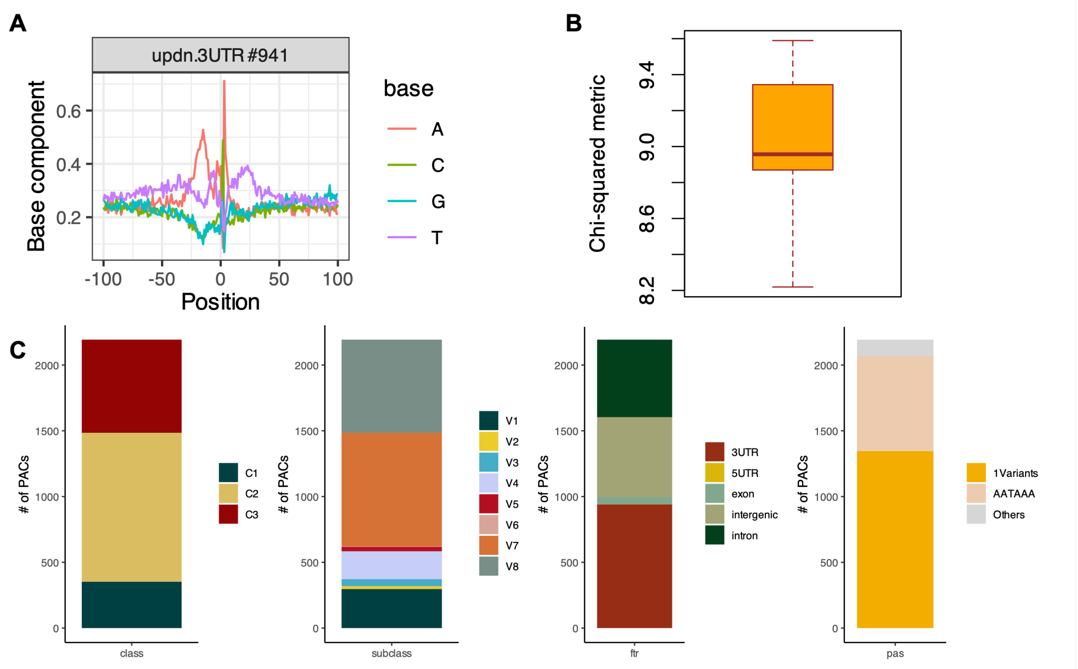
```
data.PACds <- readPACds(PACs_human$pa.coord, colDataFile=NULL)
# annotate PAC
data.PACds <- annotatePAC(data.PACds, aGFF = annotation)
# extend 3'UTR region
data.PACds <- ext3UTRPACds(data.PACds,ext3UTRlen = 1000)
# identify polyA signals
data.PACds<- annotateByPAS(data.PACds, bsgenome, grams='AATAAA', from=-50, to=25, label=NULL)
data.PACds <- annotateByPAS(data.PACds, bsgenome, grams='V1', from=-50, to=25, label=NULL)
data.PACds@anno$pA.signal <- "Others"
data.PACds@anno$pA.signal[which(!is.na(data.PACds@anno$V1_dist))] <- "1Variants"
data.PACds@anno$pA.signal[which(!is.na(data.PACds@anno$AATAAA_dist))] <- "AATAAA"

table(data.PACds@anno$pA.signal)
#1Variants    AATAAA    Others
#1346       721       126

table(data.PACds@anno$ftr)
#3UTR       5UTR       exon  intergenic      intron
#941           1          54         608         589

save(data.PACds,file="data.PACds.rda")
```

7 Summary report

The summary report provides an overview of PACs at different levels, including their genomics region and length distributions, signal distribution, and nucleotide frequency distribution. It helps users quickly understand the predicted results by displaying the classification of PACs into categories (C1, C2, C3) and further subdividing them into subclasses (V1 to V8) based on polyA tail length, base composition, and alignment results. Additionally, chi-square values are provided to assess the similarity between identified PACs and reference sequences.

```
library(ggplot2)
library("BSgenome.Hsapiens.UCSC.hg38", quietly = TRUE)
library(movAPA, warn.conflicts = FALSE, quietly=TRUE)
library(ggpubr)
bsgenome <-BSgenome.Hsapiens.UCSC.hg38
data(data.PACds)
faFiles=faFromPACds(data.PACds, bsgenome, what='updn', fapre='updn',
                    up=-100, dn=100,byGrp='ftr')
#608 >>> updn.intergenic.fa
#589 >>> updn.intron.fa
#941 >>> updn.3UTR.fa
#54 >>> updn.exon.fa
#1 >>> updn.5UTR.fa

##plot single nucleotide profiles for 3'UTR PACs
plotATCGforFAfile("updn.3UTR.fa", ofreq=FALSE, opdf=FALSE,
                  refPos=101, mergePlots = TRUE)


##calculate a chi-square metric to assess the similarity between the single nucleotide
profile of identified polyA sites and the reference profile
#load reference PACs profile
data(ref.data)
```

```
#This demonstrates functionality with 10 iterations and 200 random PACs. Larger parameters
are recommended (e.g., iteration = 100, use.size = 5000)
chisq.result <- cal.chisq(fafile="updn.3UTR.fa",ref.data=ref.data,
                          iteration=10,use.size=200)
statistic <- as.numeric(unlist(chisq.result$statistic))
summary(statistic )
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#8.219   8.878   8.957   8.997   9.261   9.589

boxplot(statistic ,col="orange",

        ylab = "Chi-squared metric",
        border = "brown")
##plot distributions of PACs across different categories, subclasses, genomic features,
and polyA signals
data.PACds@anno$coord_class <- factor(data.PACds@anno$coord_level,
                              levels=c("V1" ,"V2","V3","V4","V5","V6" ,"V7","V8","Count"),
                         labels= c("C1","C1","C2","C2","C1","C2","C2","C3","Count")

p<- plot_summary(data=data.PACds@anno)
ggarrange(p$p1,p$p2,p$p3,p$p4,    labels = c("A", "B", "C","D"))
```



8 Session Information

The session information records the versions of all the packages used in the generation of the present document.

```
sessionInfo()
#> R version 4.3.1 (2023-06-16)
#> Platform: aarch64-apple-darwin20 (64-bit)
#> Running under: macOS 15.1
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> time zone: America/Los_Angeles
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats4    stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#>  [1] ggplot2_3.5.1
#>  [2] dplyr_1.1.4
#>  [3] BSgenome.Athaliana.ENSEMBL.TAIR10_1.4.2
#>  [4] BSgenome.Mmusculus.UCSC.mm39_1.4.3
#>  [5] BSgenome.Hsapiens.UCSC.hg38_1.4.5
#>  [6] BSgenome_1.70.1
#>  [7] rtracklayer_1.62.0
#>  [8] BiocIO_1.12.0
#>  [9] Biostrings_2.70.1
#> [10] XVector_0.42.0
#> [11] GenomicRanges_1.54.1
#> [12] GenomeInfoDb_1.38.1
#> [13] IRanges_2.36.0
#> [14] S4Vectors_0.40.2
#> [15] BiocGenerics_0.48.1
#> [16] PolyAseqTrap_0.1.0
#>
#> loaded via a namespace (and not attached):
#>  [1] SummarizedExperiment_1.32.0 gtable_0.3.4
#>  [3] rjson_0.2.21                xfun_0.41
#>  [5] Biobase_2.62.0              lattice_0.22-5
#>  [7] vctrs_0.6.5                 tools_4.3.1
#>  [9] bitops_1.0-7                generics_0.1.3
#> [11] parallel_4.3.1              tibble_3.2.1
#> [13] pbmcapply_1.5.1             fansi_1.0.6
#> [15] highr_0.10                  pkgconfig_2.0.3
#> [17] Matrix_1.6-4                RColorBrewer_1.1-3
#> [19] lifecycle_1.0.4             GenomeInfoDbData_1.2.11
#> [21] farver_2.1.1                compiler_4.3.1
#> [23] stringr_1.5.1               Rsamtools_2.18.0
#> [25] munsell_0.5.0               statmod_1.5.0
#> [27] codetools_0.2-19            htmltools_0.5.7
#> [29] RCurl_1.98-1.13             yaml_2.3.7
#> [31] pillar_1.9.0                crayon_1.5.2
#> [33] BiocParallel_1.36.0         DelayedArray_0.28.0
#> [35] limma_3.58.1                abind_1.4-5
#> [37] tidyselect_1.2.0            digest_0.6.33
#> [39] stringi_1.8.2               restfulr_0.0.15
#> [41] labeling_0.4.3              fastmap_1.1.1
#> [43] grid_4.3.1                  colorspace_2.1-0
#> [45] cli_3.6.2                    SparseArray_1.2.2
#> [47] magrittr_2.0.3               S4Arrays_1.2.0
#> [49] XML_3.99-0.16                utf8_1.2.4
#> [51] withr_3.0.1                  scales_1.3.0
#> [53] rmarkdown_2.25               matrixStats_1.1.0
#> [55] evaluate_0.23                knitr_1.45
#> [57] rlang_1.1.4                  Rcpp_1.0.11
#> [59] glue_1.8.0                   jsonlite_1.8.8
#> [61] rstudioapi_0.15.0            R6_2.5.1
#> [63] plyr_1.8.9                   MatrixGenerics_1.14.0
#> [65] GenomicAlignments_1.38.0     zlibbioc_1.48.0
#> [67] outliers_0.15
```

9 References

[1] Arefeen A, et al. DeepPASTA: deep neural network based polyadenylation site analysis. Bioinformatics 2019;35(22):4577–4585.

[2] Ye C, et al. QuantifyPoly(A): reshaping alternative polyadenylation landscapes of eukaryotes with weighted density peak clustering. Briefings Bioinformatics 2021;22(6)

[3] Ye W, et al. movAPA: modeling and visualization of dynamics of alternative polyadenylation across biological samples. Bioinformatics 2021;37(16):2470–2472.