Supplemental computational methods

Table of Content

1. RAW DATA PROCESSING

Script: **process_forward_reads.sh**

Note: The script requires to have installed the following programs *cutadapt*, *bowtie2*, *bedtools*, and *seqtk* and the Perl script *bed2histogram.pl*

```
for SAMPLE in Pbat_LERE_1_S1 Pbat_LERE_2_S3 Pbat_LERE_3_S5 \
              Pbat_mut_LE88RE100_1_S2 Pbat_mut_LE88RE100_2_S4 Pbat_mut_LE88RE100_3_S6
do

  echo
  echo "############################"
  echo "#" Trimming ${SAMPLE}
  echo "############################"
  echo

  # Trim reads
  TRIMDIR=cutadapt_trim
  if [[ ! -d ./${TRIMDIR} ]]; then
      mkdir ${TRIMDIR}
  fi

  cutadapt -b "file:primer_A.fasta" \
      --overlap 6 \
      -m 27 \
      --discard-untrimmed ./raw_fastq/${SAMPLE}_L001_R1_001.fastq.gz | \
      cutadapt -b "file:primer_B.fasta" \
          --overlap 6  \
          -m 27 - | \
      cutadapt -b "file:illumina_adapters.fasta" \
          --times 2  \
```

```
        --overlap 6  \
        -m 25 \
        -o ./cutadapt_trim/${SAMPLE}_step3.fastq - 1>${SAMPLE}.cutadapt.log 2>&1

# Map reads to genome
BAMDIR=bam
if [[ ! -d ./${BAMDIR} ]]; then
    mkdir ${BAMDIR}
fi

echo
echo "############################"
echo "#" Mapping ${SAMPLE}
echo "############################"
echo

bowtie2 --end-to-end \
    --time \
    --threads 8  \
    -x ./GCA_000001405.15_GRCh38_no_alt_analysis_set.fna.bowtie_index/GCA_000001405.15_GRCh38_no_alt_a
    -U ./cutadapt_trim/${SAMPLE}_step3.fastq | \
        samtools view -hb - | \
        samtools sort -T ${SAMPLE}.tmp \
            -O BAM \
            -@ 8 \
            --write-index \
            -o ./${BAMDIR}/${SAMPLE}.cutadap_step3.R1.bam - 1>${SAMPLE}.bowtie2.log 2>&1

# Convert bam to bed with bedtools
echo
echo "######################################"
echo "#" Converting bam file to bed format
echo "#" ${SAMPLE}
echo "######################################"
echo


BEDDIR=bed

if [[ ! -d ./${BEDDIR} ]]; then
    mkdir ${BEDDIR}
fi

bedtools bamtobed -cigar -i ./${BAMDIR}/${SAMPLE}.cutadap_step3.R1.bam > ./${BEDDIR}/${SAMPLE}.cutadap


# Generate histograms from bed
echo
echo "######################################"
echo "#" Identification and quantification
echo "#" of insertion site peaks
echo "#" ${SAMPLE}
echo "######################################"
```

```
  echo

  HISTDIR=histograms

  if [[ ! -d ./${HISTDIR} ]]; then
      mkdir ${HISTDIR}
  fi

  cat ./${BEDDIR}/${SAMPLE}.cutadap_step3.R1.bed | ./bed2histogram.pl > ./${HISTDIR}/${SAMPLE}.step3.R1



  # Extract flanking DNA sequences
  echo
  echo "########################################"
  echo "#" Extracting flanking sequences
  echo "#" for motif analysis
  echo "#" ${SAMPLE}
  echo "########################################"
  echo

  FLANKDIR=flanking

  if [[ ! -d ./${FLANKDIR} ]]; then
      mkdir ${FLANKDIR}
  fi

  seqtk subseq human_GRCh38.fasta ./histograms/${SAMPLE}.step3.R1.hist > ./${FLANKDIR}/${SAMPLE}.IS_fla



  echo
  echo "########################################"
  echo "#" ${SAMPLE} Done !
  echo "########################################"
  echo ----------------------------------------------------------------------------------------------------
  echo

done
```

Script **bed2histogram.pl**

```
use strict;

# This program takes as input a bed file generated by bedtools bamtobed program
# and counts the number of insertion sites as the most 3'end of the mapped read 1.

my $SCORE_CUTOFF = 40;
my %isdb;

while(<>){
    chomp;
    my @x = split /\t/;
    my $is_id;
    my ($chr, $end5, $end3, $score, $strand) = ($x[0],$x[1],$x[2],$x[4],$x[5]);
```

```
    if ($strand eq "+"){
        my $new5 = $end3 - 34;
        my $new3 = $end3 + 30;
        $is_id = "$chr:$new5-$new3-$end3:$strand";
    } else {
        my $new5 = $end5 - 30;
        my $new3 = $end5 + 34;
        $is_id = "$chr:$new5-$new3-$end5:$strand";
    }

    if($score >= $SCORE_CUTOFF){
        $isdb{$is_id}++
    }
};

# Print header
print "#Chromosome\tflank_5\tflank_3\tcounts\tIS_coord\tread_strand\n";

foreach my $is_id (keys %isdb){
    my ($ch, $co5, $co3, $hit, $strand) = ($1, $2, $3, $4, $5) if $is_id =~ m/^(\S+):(\d+)-(\d+)-(\d+):
    my $count = $isdb{$is_id};
    print "$ch\t$co5\t$co3\t$count\t$hit\t$strand\n"
}
```

Script to generate data for Figure 7A (Logos): **process_forward_reads_for_logos.sh**

```
#!/bin/bash
module load cutadapt

INPUT=$1
NAME=$(basename ${INPUT%.fastq.gz})

cutadapt  -n 2 -j 6 -m 26 \
  --discard-untrimmed \
  -g AGATGTGTATAAGAGACAG -a TGGATTGCGGGAAACGAG \
  -o ./"${NAME}"_trimmed \
  "${NAME}".fastq.gz
```

2. ANALYSIS

Identify the coordinates of TTAA motifs across the genome

Script **count_TTAA_motif_from_fasta.pl**

```
#!/Users/lorenziha/opt/anaconda3/envs/snp_caller/bin/perl
use strict;

my $usage = "$0 -f <fasta file> -c <chromosome id: e.g chr2-chr5-chr8 [def=all]>\n\n";
my %arg = @ARGV;
die $usage unless $arg{-f};
my $CHR_ID = $arg{-c} || 'all';

# Read fasta file
my (%h, $id);
open (FASTA, "<$arg{-f}") || die "ERROR, I cannot open $arg{-f}: $!\n\n";
while(<FASTA>){
```

```
        chomp;
        if(m/^>(\S+)/){
                $id=$1;
        }
        elsif(m/^[ACGTNXacgtnx]+$/){
                $h{$id}.= $_ ;
        }
};
close FASTA;

# Search TTAA motif and its position on chromosome
# and print out a bed file with their coordinates
my @chrom_list;
if ($CHR_ID eq 'all'){
        @chrom_list = keys %h;
} else {
        @chrom_list = split(/-/,$CHR_ID);
}

open (OUTFILE,">all_chromosomes_TTAA.bed");
foreach my $id (@chrom_list){
        print STDERR "Processing $id\n";
        while($h{$id} =~ m/(TTAA|AATT)/g){
                my $end5 = length($`);
                my $end3 = $end5 + 1;
                my $posid = "$id:$end5-$end3";
                print OUTFILE "$id\t$end5\t$end3\t$posid\n";
        }
}
close OUTFILE;
```

Estimate TTAA abundance and genomic span in bp of annotation types (introns, exons, intergenic, CDS and UTR)

Script to generate data for figure 7B: **estimate_TTAA_distribution.sh**

```
#!/bin/zsh

GENOME_FASTA=GCA_000001405.15_GRCh38_no_alt_analysis_set.fna
GENOME_ANNOTATION=GCA_000001405.15_GRCh38_full_analysis_set.refseq_annotation.gtf

# Export gene coords from genome annotation gtf file as a bed file
echo "################################"
echo Exporting gene annotations
echo "################################"

egrep  --color "\tgene\t" ${GENOME_ANNOTATION} | \
    cut -f 1,4,5 | \
    perl -lane '$F[1]--;print "$F[0]\t$F[1]\t$F[2]"' | \
    egrep "^chr\d+\t" > gene_unmerged.bed

# Merge overlapping exons
bedtools merge -i gene_unmerged.bed > gene_merged.bed
```

```
# Generate bed file with intergenic regions
echo "#################################"
echo Exporting intergenic annotations
echo "#################################"

seqtk comp ${GENOME_FASTA} | \
    egrep "^chr\d+\t" | \
    perl -lane 'print "$F[0]\t0\t$F[1]"' > chromosomes.bed

cut -f 1,3 chromosomes.bed > my.genome

bedtools complement -i  gene_merged.bed -g my.genome > intergenic.bed

# Centromeric coords were extracted from UCSC Table browser
# https://genome.ucsc.edu/cgi-bin/hgTables
# on human assembly Dec 2012 (GRCh38/hg38)
# Group = "Mapping and Sequencing"
# Track = Centromeres
# Output format = BED
# Output file centromeres.bed

echo "#################################"
echo Sorting centromeres.bed
echo "#################################"

bedtools sort -i centromeres.bed > centromeres.bed

# Remove centromeric regions from intergenic regions
bedtools subtract -a intergenic.bed -b centromeres.bed > intergenic_no_centromere.bed

# Remove centromeric regions from genes
bedtools subtract -a gene_merged.bed -b centromeres.bed > gene_merged_no_centromere.bed

# Export exon coords from genome annotation gtf file as a bed file
echo "#################################"
echo Exporting exonic annotations
echo "#################################"

egrep  --color "\texon\t" ${GENOME_ANNOTATION} | \
    cut -f 1,4,5 | \
    perl -lane '$F[1]--;print "$F[0]\t$F[1]\t$F[2]"' | \
    egrep "^chr\d+\t" | \
    sort -k1,1 -k2,2n > exon_unmerged.bed

# Merge overlaping exons
bedtools merge -i exon_unmerged.bed > exon_merged.bed

# Export CDS coords from genome annotation gtf file as a bed file
echo "#################################"
echo Exporting CDS annotations
echo "#################################"

egrep  --color "\tCDS\t" ${GENOME_ANNOTATION} | \
```

```
    cut -f 1,4,5 | \
    perl -lane '$F[1]--;print "$F[0]\t$F[1]\t$F[2]"' | \
    egrep "^chr\d+\t" | \
    sort -k1,1 -k2,2n > cds_unmerged.bed

# Merge overlaping CDSs
bedtools merge -i cds_unmerged.bed > cds_merged.bed

# Generate bed file with intron coords
bedtools subtract -a gene_merged.bed -b exon_merged.bed > introns_unmerged.bed

# Merge overlaping introns
bedtools merge -i introns_unmerged.bed > introns_merged.bed

# Generate bed file with UTR coords
bedtools subtract -a exon_merged.bed -b cds_merged.bed > utr_unmerged.bed

# Merge overlaping UTRs
bedtools merge -i utr_unmerged.bed > utr_merged.bed

# Remove centromeric regions from CDS regions
bedtools subtract -a cds_merged.bed -b centromeres.bed > cds_merged_no_centromere.bed

# Remove centromeric regions from intronic regions
bedtools subtract -a introns_merged.bed -b centromeres.bed > introns_merged_no_centromere.bed

# Remove centromeric regions from UTR regions
bedtools subtract -a utr_merged.bed -b centromeres.bed > utr_merged_no_centromere.bed


# Count total length for each annotation type with and without regions overlaping centromeres
intron_counts=$(./count_total_length_from_bed.pl < introns_merged.bed)
intron_counts_no_centromere=$(./count_total_length_from_bed.pl < introns_merged_no_centromere.bed)

intergenic_counts=$(./count_total_length_from_bed.pl < intergenic.bed)
intergenic_counts_no_centromere=$(./count_total_length_from_bed.pl < intergenic_no_centromere.bed)

cds_counts=$(./count_total_length_from_bed.pl < cds_merged.bed)
cds_counts_no_centromere=$(./count_total_length_from_bed.pl < cds_merged_no_centromere.bed)

utr_counts=$(./count_total_length_from_bed.pl < utr_merged.bed)
utr_counts_no_centromere=$(./count_total_length_from_bed.pl < utr_merged_no_centromere.bed)

echo
echo Intron counts = $intron_counts
echo Intron counts without centromeres = $intron_counts_no_centromere
echo
echo Intergenic counts = $intergenic_counts
echo Intergenic counts without centromeres = $intergenic_counts_no_centromere
echo
echo CDS counts = $cds_counts
echo CDS counts without centromeres = $cds_counts_no_centromere
echo
```

```
echo UTR counts = $utr_counts
echo UTR counts without centromeres = $utr_counts_no_centromere
echo

# Extract TTAA motif coords across the human genome
#./count_TTAA_motif_from_fasta.pl -f ${GENOME_FASTA} > all_chromosomes_TTAA.bed

# Estimate number of TTA motifs per annotation type excluding centromeric regions
echo "##########################################"
echo Estimating TTAA counts per annotation type
echo "##########################################"

intergenic_ttaa=$(bedtools intersect -a all_chromosomes_TTAA.bed -b intergenic_no_centromere.bed | wc -l
intronic_ttaa=$(bedtools intersect -a all_chromosomes_TTAA.bed -b introns_merged_no_centromere.bed | wc
utr_ttaa=$(bedtools intersect -a all_chromosomes_TTAA.bed -b utr_merged_no_centromere.bed | wc -l)
cds_ttaa=$(bedtools intersect -a all_chromosomes_TTAA.bed -b cds_merged_no_centromere.bed | wc -l)

echo
echo Intergenic TTAAs = $intergenic_ttaa
echo Intronic TTAAs = $intronic_ttaa
echo CDS TTAAs = $cds_ttaa
echo UTR TTAAs = $utr_ttaa
echo

echo "################################"
echo Done!
echo "################################"
```

Script to format histogram files to bedGraph: **histogram2bedGraph.pl**

```
use strict;

# This script convert a histogram file generated by process_forward_reads.sh
# with the format specified below to a bedGraph file using as score the
# "counts" column and as coordinates the
# IS_coord that corresponds to the mapped insertion site.

# The script also normalizes the scores (peaks) to peaks per million peaks.

# Format example:
# #Chromosome   read_5_end  read_3_end  counts  IS_coord    read_strand
# chr22 24310207    24310271    1   24310241    +
# chr12 68572516    68572580    1   68572546    -

my %peaks;
my $total_peaks = 0;

while(<>){
    next if m/^#/;
    chomp;
    my ($chr, $end5, $end3, $counts, $is_coord, $strand) = split /\t/;
    $total_peaks += $counts;
    if ($strand eq "+"){
        my $new5 = $is_coord - 1;
```

```
        my $name = "$chr:$end5-$end3";
        $peaks{$name} = "$chr\t$new5\t$is_coord\t$counts";
    } else {
        my $new3 = $is_coord + 1;
        my $name = "$chr:$end5-$end3";
        $peaks{$name} = "$chr\t$is_coord\t$new3\t$counts";
    }
}

# Print out normalized peaks
foreach my $name (keys %peaks){
    my ($chr, $end5, $end3, $counts) = split("\t", $peaks{$name});
    my $norm_counts = ($counts / $total_peaks) * 1000000;
    print "$chr\t$end5\t$end3\t$norm_counts\n";
}
```

3. PLOTS

Figure 7C:

Load required R libraries

```
library(tidyverse)
library(cowplot)
```

Load bedGraph data

```
bedgraph_files <- dir(path = "./histograms/", pattern = "*.bedGraph")
is.bedgraph <- bedgraph_files %>% gsub(pattern = "_S.*.step3.R1.hist.bedGraph", replacement = "") %>%
                   gsub(pattern = "Pbat_LERE", replacement = "Pbat_WT") %>%
                   gsub(pattern = "Pbat_mut_LE88RE100", replacement = "Pbat_Mut")

is.list <- list()
idx <- 1
for (file in bedgraph_files){
  flag <- is.bedgraph[idx] %>% gsub(pattern = "_.$", replacement = "")
  is.list[[is.bedgraph[idx]]] <- read_delim(file = paste0("./histograms/",file), delim = "\t", col_names
  is.list[[is.bedgraph[idx]]]$Genotype <- flag
  is.list[[is.bedgraph[idx]]] <- is.list[[is.bedgraph[idx]]][grep("_|chrM",is.list[[is.bedgraph[idx]]]$
  idx <- idx + 1
}
```

Generate plot for Fig 7C

```
# Summarize bedGraph and concatenate into a single object
is.counts.list <- lapply(is.list, function(x){
      x %>% group_by(Chr) %>%
    summarize( Counts = n() ) %>%
    mutate( Count_total = sum(Counts)) %>%
    mutate( Perc_peaks = Counts * 100 /Count_total) %>%
    mutate( Group = x$Genotype[1])
    }
)

# append all tibbles into a single tibble
all.tbl <- tibble(Chr = as.character(), Counts = as.numeric(), Count_total = as.numeric(), Perc_peaks =
for (tbl in is.counts.list){
```

```
    all.tbl <- rbind(all.tbl, tbl)
}

# Estimate mean counts per chromosome per Group
all.tbl.stat <- all.tbl %>% group_by(Group,Chr) %>%
            summarize( Mean_perc_peaks = mean(Perc_peaks)) %>%
  mutate(Chr_name = gsub("chr", "", x = as.character(Chr)))


all.tbl.stat$Chr_name <- factor(all.tbl.stat$Chr_name, levels = c(as.character(1:22),"X") )

## Plot heatmap
genTile <- ggplot() +
  geom_tile(data = all.tbl.stat, aes(y = Group, x = Chr_name, fill=Mean_perc_peaks),
            colour = "white", size = 1) +
  scale_fill_distiller(palette = "PuBu", direction = 1) +
  coord_fixed() + xlab("") + ylab("") +
  theme(axis.ticks = element_blank(),
        panel.grid.minor.x = element_blank(),panel.grid.minor.y = element_blank(),
        panel.grid.major.x = element_blank(),panel.grid.major.y = element_blank(),
        axis.text.x = element_text(angle = 0, hjust = 1, vjust = 0.5)) +
  scale_x_discrete(position = "top")

print(genTile)
ggsave2(filename = "figX_panel_C_sorted_by_chr.pdf", plot = genTile, path = "./Plots")
```

pBat plots

```
# Load required libraries

suppressPackageStartupMessages({library(GenomicRanges); library(data.table);
  library(ggplot2); library(ggpubr); library(Biostrings); library(rtracklayer); library(scales); library
```

Figure 7A

Figure 7A: Identification of nucleotide frequencies immediately downstream of the pBat

```
# Plot Logos of the trimmed reads


dir="/Users/konstantinidop2/Documents/Projects/Piggy_Bat/Trimmed__R1/"

# File pBat_WT_1_S1.fasta

Pbat_WT_1<-Biostrings::readDNAStringSet(paste0(dir,"pBat_WT_1_S1.fasta"))

Pbat_WT_1_RC<-Biostrings::reverseComplement(Pbat_WT_1)

Pbat_WT_1_RC2<-Pbat_WT_1_RC[grepl("^AGTG", Pbat_WT_1_RC)]

Pbat_WT_1_RC.DT<-as.data.table(Pbat_WT_1_RC2)

Pbat_WT_1_RC_first15nt<-substr(Pbat_WT_1_RC.DT$x, 1, 15)
```

```
ggplot() + geom_logo(Pbat_WT_1_RC_first15nt, seq_type='DNA') + theme_logo() + ylim(0,2) + ggtitle(paste(

# File pBat_WT_2_S3.fasta

pBat_WT_2<-Biostrings::readDNAStringSet(paste0(dir,"pBat_WT_2_S3.fasta"))

pBat_WT_2_RC<-Biostrings::reverseComplement(pBat_WT_2)

pBat_WT_2_RC2<-pBat_WT_2_RC[grepl("^AGTG", pBat_WT_2_RC)]

pBat_WT_2_RC.DT<-as.data.table(pBat_WT_2_RC2)

pBat_WT_2_RC_first15nt<-substr(pBat_WT_2_RC.DT$x, 1, 15)

ggplot() + geom_logo(pBat_WT_2_RC_first15nt, seq_type='DNA') + theme_logo() + ylim(0,2) + ggtitle(paste(

# File pBat_WT_3_S5.fasta

pBat_WT_3<-Biostrings::readDNAStringSet(paste0(dir,"pBat_WT_3_S5.fasta"))

pBat_WT_3_RC<-Biostrings::reverseComplement(pBat_WT_3)

pBat_WT_3_RC2<-pBat_WT_3_RC[grepl("^AGTG", pBat_WT_3_RC)]

pBat_WT_3_RC.DT<-as.data.table(pBat_WT_3_RC2)

pBat_WT_3_RC_first15nt<-substr(pBat_WT_3_RC.DT$x, 1, 15)

ggplot() + geom_logo(pBat_WT_3_RC_first15nt, seq_type='DNA') + theme_logo() + ylim(0,2) + ggtitle(paste(


# File pBat_mut_1_S2.fasta

pBat_mut_1<-Biostrings::readDNAStringSet(paste0(dir,"pBat_mut_1_S2.fasta"))

pBat_mut_1_RC<-Biostrings::reverseComplement(pBat_mut_1)

pBat_mut_1_RC2<-pBat_mut_1_RC[grepl("^AGTG", pBat_mut_1_RC)]

pBat_mut_1_RC.DT<-as.data.table(pBat_mut_1_RC2)

pBat_mut_1_RC_first15nt<-substr(pBat_mut_1_RC.DT$x, 1, 15)

ggplot() + geom_logo(pBat_mut_1_RC_first15nt, seq_type='DNA') + theme_logo() + ylim(0,2) + ggtitle(paste


# File pBat_mut_2_S4.fasta

pBat_mut_2<-Biostrings::readDNAStringSet(paste0(dir,"pBat_mut_2_S4.fasta"))

pBat_mut_2_RC<-Biostrings::reverseComplement(pBat_mut_2)

pBat_mut_2_RC2<-pBat_mut_2_RC[grepl("^AGTG", pBat_mut_2_RC)]
```

```
pBat_mut_2_RC.DT<-as.data.table(pBat_mut_2_RC2)

pBat_mut_2_RC_first15nt<-substr(pBat_mut_2_RC.DT$x, 1, 15)

ggplot() + geom_logo(pBat_mut_2_RC_first15nt, seq_type='DNA') + theme_logo() + ylim(0,2) + ggtitle(paste


# File pBat_mut_3_S6.fasta

pBat_mut_3<-Biostrings::readDNAStringSet(paste0(dir,"pBat_mut_3_S6.fasta"))

pBat_mut_3_RC<-Biostrings::reverseComplement(pBat_mut_3)

pBat_mut_3_RC2<-pBat_mut_3_RC[grepl("^AGTG", pBat_mut_3_RC)]

pBat_mut_3_RC.DT<-as.data.table(pBat_mut_3_RC2)

pBat_mut_3_RC_first15nt<-substr(pBat_mut_3_RC.DT$x, 1, 15)

# Generate logos plot

ggplot() + geom_logo(pBat_mut_3_RC_first15nt, seq_type='DNA') + theme_logo() + ylim(0,2) + ggtitle(paste
```

Supplementary Figure (Circos):

Genome-wide distribution of insertion sites depicted in circular plot (Supplementary Figure -Circos-)

```
# Import coordinates of insertion sites

dir2="/Users/konstantinidop2/Documents/Projects/Piggy_Bat/Insertion_peaks/"

pBat_WT_1<-import(paste0(dir2,"Bed_files/Pbat_LERE_1_S1.step3.R1.bed"))
pBat_WT_2<-import(paste0(dir2,"Bed_files/Pbat_LERE_2_S3.step3.R1.bed"))
pBat_WT_3<-import(paste0(dir2,"Bed_files/Pbat_LERE_3_S5.step3.R1.bed"))
pBat_mut_1<-import(paste0(dir2,"Bed_files/Pbat_mut_LE88RE100_1_S2.step3.R1.bed"))
pBat_mut_2<-import(paste0(dir2,"Bed_files/Pbat_mut_LE88RE100_2_S4.step3.R1.bed"))
pBat_mut_3<-import(paste0(dir2,"Bed_files/Pbat_mut_LE88RE100_3_S6.step3.R1.bed"))


# Prepare data.frames for circ plot

GR_list<-list("pBat_WT_1"=pBat_WT_1, "pBat_WT_2"=pBat_WT_2, "pBat_WT_3"=pBat_WT_3,
              "pBat_mut_1"=pBat_mut_1, "pBat_mut_2"=pBat_mut_2, "pBat_mut_3"=pBat_mut_3)

Bed_list<-lapply(names(GR_list), function(i){
  GR<-GR_list[[i]]
  DT<-data.frame(chr=seqnames(GR), start=start(GR), end=end(GR), value1=1, strand=strand(GR))
  DT[DT$strand == "-",]$value1<--1
  DT$counts<-GR$name
  # rescale counts (so that they always fall within the range of 0-1)
  DT$rescaled_counts<-rescale(as.numeric(DT$counts), c(0,1))
  DT[DT$strand == "-",]$rescaled_counts<--DT[DT$strand == "-",]$rescaled_counts
  return(DT)
})
```

```
names(Bed_list)<-names(GR_list)

# Initialize circos of human chromosomes with ideogram
library("RCircos")
library("circlize")

circos.par("track.height" = 0.05, "start.degree" = 90, "gap.degree" = rep(c(2, 4), 12))
circos.initializeWithIdeogram(species = "hg19")


# Add tracks with all the predicted human insertion sites retaining strand information

circos.genomicTrack(Bed_list$pBat_WT_1, numeric.column=4,
    panel.fun = function(region, value, ...) {
        circos.genomicRect(region, value, ytop.column = 1, ybottom = 0,
            col = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"),
            border = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"), ...)
        circos.lines(CELL_META$cell.xlim, c(0, 0), lty = 2, col = "#606060")
})

circos.genomicTrack(Bed_list$pBat_WT_2, numeric.column=4,
    panel.fun = function(region, value, ...) {
        circos.genomicRect(region, value, ytop.column = 1, ybottom = 0,
            col = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"),
            border = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"), ...)
        circos.lines(CELL_META$cell.xlim, c(0, 0), lty = 2, col = "#606060")
})

circos.genomicTrack(Bed_list$pBat_WT_3, numeric.column=4,
    panel.fun = function(region, value, ...) {
        circos.genomicRect(region, value, ytop.column = 1, ybottom = 0,
            col = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"),
            border = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"), ...)
        circos.lines(CELL_META$cell.xlim, c(0, 0), lty = 2, col = "#606060")
})

circos.genomicTrack(Bed_list$pBat_mut_1, numeric.column=4,
    panel.fun = function(region, value, ...) {
        circos.genomicRect(region, value, ytop.column = 1, ybottom = 0,
            col = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"),
            border = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"), ...)
        circos.lines(CELL_META$cell.xlim, c(0, 0), lty = 2, col = "#606060")
})

circos.genomicTrack(Bed_list$pBat_mut_2, numeric.column=4,
    panel.fun = function(region, value, ...) {
        circos.genomicRect(region, value, ytop.column = 1, ybottom = 0,
            col = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"),
            border = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"), ...)
        circos.lines(CELL_META$cell.xlim, c(0, 0), lty = 2, col = "#606060")
})

circos.genomicTrack(Bed_list$pBat_mut_3, numeric.column=4,
```

```
    panel.fun = function(region, value, ...) {
        circos.genomicRect(region, value, ytop.column = 1, ybottom = 0,
            col = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"),
            border = ifelse(value[[1]] > 0, "#FF6666", "#9999FF"), ...)
        circos.lines(CELL_META$cell.xlim, c(0, 0), lty = 2, col = "#606060")
})

title(main = "pBat insertion sites throughout the genome")
legend("bottomright", pch = "-", lwd = 2, col = c("#FF6666", "#9999FF"),
       legend = c("plus strand", "minus strand"))
legend("bottomleft", legend = c("out->inside:", "WT_1", "WT_2", "WT_3", "mut_1", "mut_2", "mut_3"))
```

```
sessionInfo()
```

R session information

```
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/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/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices datasets  utils     methods
## [8] base
##
## other attached packages:
##  [1] formatR_1.14         knitr_1.44           ggseqlogo_0.1
##  [4] scales_1.2.1         rtracklayer_1.60.1   Biostrings_2.68.1
##  [7] XVector_0.40.0       ggpubr_0.6.0         ggplot2_3.4.3
## [10] data.table_1.14.8    GenomicRanges_1.52.1 GenomeInfoDb_1.36.4
## [13] IRanges_2.34.1       S4Vectors_0.38.2     BiocGenerics_0.46.0
##
## loaded via a namespace (and not attached):
##  [1] SummarizedExperiment_1.30.2 gtable_0.3.4
##  [3] rjson_0.2.21                xfun_0.40
##  [5] rstatix_0.7.2               lattice_0.21-8
##  [7] Biobase_2.60.0              vctrs_0.6.3
##  [9] tools_4.3.1                 bitops_1.0-7
## [11] generics_0.1.3              parallel_4.3.1
## [13] tibble_3.2.1                fansi_1.0.4
## [15] pkgconfig_2.0.3             Matrix_1.5-4.1
## [17] lifecycle_1.0.3             GenomeInfoDbData_1.2.10
## [19] compiler_4.3.1              Rsamtools_2.16.0
## [21] munsell_0.5.0               codetools_0.2-19
```

```
## [23] carData_3.0-5              htmltools_0.5.6.1
## [25] RCurl_1.98-1.12            yaml_2.3.7
## [27] pillar_1.9.0               car_3.1-2
## [29] crayon_1.5.2               tidyr_1.3.0
## [31] BiocParallel_1.34.2        DelayedArray_0.26.7
## [33] abind_1.4-5                tidyselect_1.2.0
## [35] digest_0.6.33              dplyr_1.1.3
## [37] purrr_1.0.2                restfulr_0.0.15
## [39] fastmap_1.1.1              grid_4.3.1
## [41] colorspace_2.1-0           cli_3.6.1
## [43] magrittr_2.0.3             S4Arrays_1.0.6
## [45] XML_3.99-0.14              utf8_1.2.3
## [47] broom_1.0.5                withr_2.5.1
## [49] backports_1.4.1            rmarkdown_2.25
## [51] matrixStats_1.0.0          ggsignif_0.6.4
## [53] evaluate_0.22              BiocIO_1.10.0
## [55] rlang_1.1.1                glue_1.6.2
## [57] renv_1.0.3                 rstudioapi_0.15.0
## [59] R6_2.5.1                   MatrixGenerics_1.12.3
## [61] GenomicAlignments_1.36.0   zlibbioc_1.46.0
```