**FASTQC**

# FASTQC quality control with 32 threads

fastqc -r patch-fastqc -t 32 /your/path/to/input\_file.fastq.gz -o /your/path/to/output\_directory/

**ALIGNMENT**

# Alignment, sorting, and indexing with minimap2 and samtools

# Align reads with minimap2 and sort with samtools

minimap2 -ax map-ont -t 32 /your/path/to/reference\_genome.fa /your/path/to/reads.fastq.gz | samtools sort -@32 -o /your/path/to/output.bam

# Index the sorted BAM file

samtools index /your/path/to/output.bam

**BAMQC**

# Quality control report with Qualimap

# Generate a BAM QC report using Qualimap

qualimap bamqc -bam /your/path/to/output.bam -nt 32 -outdir /your/path/to/qualimap\_report/ --java-mem-size=64G

**VARIANT CALLING**

**cuteSV**  
*cuteSV --min\_size 50 test.bam reference\_genome.fa test.vcf output\_directory/*

**DeBreak**  
*debreak --min\_size 50 --bam test.bam -o output\_directory/ --ref reference\_genome.fa*

**Delly**

*delly lr --minrefsep 50 -g reference\_genome.fa test.bam > test.vcf*

**Dysgu**

*dysgu run --min-size 50 reference\_genome.fa dysgu\_output\_directory/ test.bam > test.vcf -x*

**NanoVar**  
*nanovar -t 24 --minlen 50 test.bam reference\_genome.fa output\_directory/*

**Severus**

*severus --target-bam tumor.bam --control-bam normal.bam --out-dir output\_directory/ -t 16 --vntr-bed reference\_genome.trf.bed --min-sv-size 50*

Unlike the other tools, Severus simplifies the workflow by directly outputting somatic VCF files, eliminating the need for merging tumor and normal results separately.

**Sniffles**

*sniffles --report\_BND --input test.bam --vcf output\_directory/test.vcf --reference reference\_genome.fa*

**SVIM**

*svim alignment svim\_output\_directory/ test.bam reference\_genome.fa --min\_sv\_size 50*

**MERGING TUMOR AND NORMAL**

*echo "tumor\_sniffles.vcf" > list\_vcf*

*echo "normal\_sniffles.vcf" >> list\_vcf*

*SURVIVOR merge list\_vcf 1000 1 1 0 0 50 sniffles\_tumornormal\_merged.vcf*

**FINDING SOMATICS**

*# Set working directory*

*setwd("C:/Path/To/Your/Directory/")*

*# Read annotated and merged VCF files into R*

*merged <- read.vcfR("merged\_file.vcf")*

*# Remove headers*

*merged <- data.frame(getFIX(merged, getINFO = TRUE), merged@gt)*

*# Separate format columns*

*merged[c('T\_GT','T\_PSV','T\_LN','T\_DR','T\_ST','T\_QV','T\_TY','T\_ID','T\_RAL','T\_AAL','T\_CO')] <- str\_split\_fixed(merged[,10], ':', 11)*

*merged[c('N\_GT','N\_PSV','N\_LN','N\_DR','N\_ST','N\_QV','N\_TY','N\_ID','N\_RAL','N\_AAL','N\_CO')] <- str\_split\_fixed(merged[,11], ':', 11)*

*# If T\_GT is ./. and T\_LN is 0, it is not found in tumor, remove these*

*merged\_filt <- merged[!(merged$T\_GT == "./." & merged$T\_LN == 0), ]*

*# Compare T\_GT and N\_GT, if N\_GT is ./. and N\_LN == 0, take these*

*merged\_filt <- merged\_filt[which(merged\_filt$N\_GT == "./." & merged\_filt$N\_LN == 0), ]*

*# Write output to files*

*write.xlsx(merged\_filt, "somatic\_data.xlsx", row.names = FALSE)*

*write.xlsx(merged\_filt, "somatic\_data.vcf", row.names = FALSE)*

**MERGING ALL SOMATIC OUTPUTS**

*echo "cutesv\_somatic.vcf" > list\_vcf*

*echo "svim\_somatic.vcf" >> list\_vcf*

*echo "sniffles\_somatic.vcf" >> list\_vcf*

*echo "nanovar\_somatic.vcf" >> list\_vcf*

*echo "delly\_somatic.vcf" >> list\_vcf*

*echo "debreak\_somatic.vcf" >> list\_vcf*

*echo "dysgu\_somatic.vcf" >> list\_vcf*

*echo "severus\_somatic.vcf" >> list\_vcf*

*SURVIVOR merge list\_vcf 1000 1 1 0 0 50 all\_somatic\_variants.vcf*

**FINDING OVERLAPPING VARIANTS SUPPORTED BY AT LEAST 3 TOOLS (OUT OF 8 TOOLS)**

# Set the working directory

# Replace "/your/path/to/working\_directory/" with your actual working directory path

setwd("/your/path/to/working\_directory/")

# Read the VCF file

# Replace "annotated\_allsomatic\_COLO829.vcf" with the path to your VCF file

tumor\_merged\_all <- read.table("annotated\_allsomatic\_COLO829.vcf", sep = "\t", skip = 323, header = FALSE)

# Define column names for the VCF file

# These are the standard VCF columns followed by tool-specific columns

colnames(tumor\_merged\_all) <- c("CHROM", "POS", "ID", "REF", "ALT", "QUAL", "FILTER", "INFO", "FORMAT",

"CuteSV", "Svim", "Sniffles", "Nanovar", "Delly", "Debreak", "Dysgu", "Severus")

# Function to split columns based on the ':' delimiter

# The function creates multiple new columns with prefixes based on the tool name

split\_columns <- function(df, colname, prefix) {

split\_data <- str\_split\_fixed(df[[colname]], ':', 11) # Split the column into 11 parts

df[paste0(prefix, '\_GT')] <- split\_data[,1] # Genotype

df[paste0(prefix, '\_PSV')] <- split\_data[,2] # Phase set value

df[paste0(prefix, '\_LN')] <- split\_data[,3] # Length

df[paste0(prefix, '\_DR')] <- split\_data[,4] # Depth ratio

df[paste0(prefix, '\_ST')] <- split\_data[,5] # Strand

df[paste0(prefix, '\_QV')] <- split\_data[,6] # Quality value

df[paste0(prefix, '\_TY')] <- split\_data[,7] # Type

df[paste0(prefix, '\_ID')] <- split\_data[,8] # Identifier

df[paste0(prefix, '\_RAL')] <- split\_data[,9] # Reference allele

df[paste0(prefix, '\_AAL')] <- split\_data[,10] # Alternate allele

df[paste0(prefix, '\_CO')] <- split\_data[,11] # Confidence

return(df)

}

# Apply the splitting function to each tool column

# Tools are defined as a list of tool names

tools <- c("CuteSV", "Svim", "Sniffles", "Nanovar", "Delly", "Debreak", "Dysgu", "Severus")

for (tool in tools) {

tumor\_merged\_all <- split\_columns(tumor\_merged\_all, tool, tool)

}

# Filter out entries where PSV is not NaN

# Create a list of filtered data for each tool

tool\_data <- lapply(tools, function(tool) filter(tumor\_merged\_all, !!sym(paste0(tool, "\_PSV")) != "NaN"))

# Select necessary columns for each tool

# Keep only key columns for further analysis

tool\_data <- lapply(seq\_along(tools), function(i) select(tool\_data[[i]], CHROM, POS, ID, REF, ALT, QUAL, FILTER, INFO, FORMAT, tools[i]))

# Create a list for the UpSet plot

# The list contains the IDs detected by each tool

x <- setNames(lapply(tool\_data, function(df) df$ID), tools)

# Find variants detected by at least three tools

# Use combinations of three tools and find the intersection of their variant IDs

common\_variants <- Reduce(union, lapply(combn(x, 3, simplify = FALSE), function(pair) Reduce(intersect, pair)))

# Filter the original data to keep only the common variants

# This filters the main data frame to include only the common variants

tumor\_merged\_common <- filter(tumor\_merged\_all, ID %in% common\_variants)

# Write the common variants to a new VCF file

# Replace "least3common\_COLO829.vcf" with your desired output file name

write.table(tumor\_merged\_common, "least3common\_COLO829.vcf", sep = "\t", row.names = FALSE, quote = FALSE)

**COMPARING FINDINGS AGAINST THE TRUTH SET**

# Set working directory

setwd("/Path/To/Your/Directory/") # Adjust the path to the folder where your files are located

# Read truthset into R (VCF file format)

truthset <- read.table("truthset\_file.vcf", sep = "\t", skip = 651, header = FALSE) # Skipping first 651 lines (metadata)

# Extract INFO column and split it into separate columns

split\_data <- str\_split\_fixed(truthset$V8, ';', 6) # Split the INFO column (V8) by ';' into 6 parts

# Extract specific key-value information from each part of the INFO column

truthset$SVLEN <- sub(".\*=", "", split\_data[, 1]) # Extract SVLEN (size of structural variant)

truthset$SVTYPE <- sub(".\*=", "", split\_data[, 2]) # Extract SVTYPE (type of structural variant)

truthset$SUPP\_SEQ <- sub(".\*=", "", split\_data[, 3]) # Extract SUPP\_SEQ (supporting sequences)

truthset$SUPP\_VAL <- sub(".\*=", "", split\_data[, 4]) # Extract SUPP\_VAL (supporting value)

truthset$GENE <- sub(".\*=", "", split\_data[, 5]) # Extract GENE (gene information)

truthset$CLUSTER <- sub(".\*=", "", split\_data[, 6]) # Extract CLUSTER (cluster information)

# Rename columns for better clarity

colnames(truthset) <- c("CHROM","POS","ID","REF","ALT","QUAL","FILTER","INFO","V9","V10","SVLEN","SVTYPE","SUPP\_SEQ","SUPP\_VAL","GENE","CLUSTER")

# Read merged somatic variant data to be compared (VCF format)

common\_variants <- read.table("least5\_comb.vcf", sep = "\t", header = TRUE) # Read the common variants file

# Function to process each row of the INFO column into key-value pairs

process\_row <- function(row) {

# Split the row by ';' into multiple elements (key-value pairs)

split\_data <- str\_split\_fixed(row, ';', n = Inf)

# Initialize an empty list to store key-value pairs

key\_value\_list <- list()

# Loop through each element in the split data

for (i in 1:length(split\_data)) {

key\_value <- str\_split\_fixed(split\_data[i], '=', 2) # Split by '=' into key and value

key <- key\_value[1, 1] # Extract the key

value <- key\_value[1, 2] # Extract the value

# Add the key-value pair to the list

key\_value\_list[[key]] <- value

}

# Convert the key-value list into a data frame and return it

return(as.data.frame(key\_value\_list, stringsAsFactors = FALSE))

}

# Apply the function to each row in the INFO column of common\_variants

result <- do.call(rbind, lapply(common\_variants$INFO, process\_row))

# Combine the result with the original common\_variants data

common\_variants <- cbind(common\_variants, result)

# Convert SVLEN to absolute values for both datasets

common\_variants$SVLEN <- abs(as.numeric(common\_variants$SVLEN))

truthset$SVLEN <- abs(as.numeric(truthset$SVLEN))

# Function to compare rows from two data frames based on specific criteria

compare\_sv <- function(df1, df2) {

results <- data.frame() # Initialize an empty data frame to store results

# Loop through each row of the first data frame (df1)

for (i in 1:nrow(df1)) {

row1 <- df1[i, ] # Extract the current row of df1

# Find matching rows in df2 based on the following criteria:

matches <- df2 %>%

filter(CHROM == row1$CHROM) %>% # Match the same chromosome

filter(abs(POS - row1$POS) < 1000) %>% # Match positions within 1000 bases

filter(SVTYPE == row1$SVTYPE) %>% # Match the same structural variant type

filter(abs(SVLEN - row1$SVLEN) < 100) # Match SVLEN within a 100 base pair difference

# If any matches are found, combine the results (df1 row + matched df2 rows)

if (nrow(matches) > 0) {

matched\_rows <- cbind(row1, matches) # Combine row1 with the matching rows from df2

results <- rbind(results, matched\_rows) # Add to the results data frame

}

}

# Return the final results data frame

return(results)

}

# Compare the two datasets (common\_variants and truthset) using the compare\_sv function

results <- compare\_sv(common\_variants, truthset)

# Ensure unique column names in the results data frame (in case of overlapping column names)

colnames(results) <- make.unique(colnames(results))

# Save the results to a file for further analysis

write.table(results, "least5\_comb\_truthset\_file.txt", sep = "\t", row.names = FALSE) # Save as tab-delimited file

**PLOTTING OVERLAPPING VARIANTS USING ALL SOMATIC MERGED VCF FILE**

# Set working directory to the path where the data files are located

setwd("/Path/To/Your/Directory/")

# Read the merged somatic variant data (VCF file format)

tumor\_merged\_all <- read.table("all\_somatic\_variants.vcf", sep="\t", skip=323) # Skipping metadata from the first 323 lines

# Assign column names for better understanding of the data structure

colnames(tumor\_merged\_all) <- c("CHROM", "POS", "ID", "REF", "ALT", "QUAL", "FILTER", "INFO", "FORMAT",

"CuteSV", "Svim", "Sniffles", "Nanovar", "Delly", "Dbreak", "Dysgu", "Severus")

# Separate the values in each column corresponding to different variant callers by splitting based on ":"

# Each variant caller's data is split into 11 attributes (e.g., genotype, support, quality, etc.)

tumor\_merged\_all[c('cutesv\_GT', 'cutesv\_PSV', 'cutesv\_LN', 'cutesv\_DR', 'cutesv\_ST', 'cutesv\_QV', 'cutesv\_TY',

'cutesv\_ID', 'cutesv\_RAL', 'cutesv\_AAL', 'cutesv\_CO')] <- str\_split\_fixed(tumor\_merged\_all$CuteSV, ':', 11)

tumor\_merged\_all[c('svim\_GT', 'svim\_PSV', 'svim\_LN', 'svim\_DR', 'svim\_ST', 'svim\_QV', 'svim\_TY', 'svim\_ID',

'svim\_RAL', 'svim\_AAL', 'svim\_CO')] <- str\_split\_fixed(tumor\_merged\_all$Svim, ':', 11)

tumor\_merged\_all[c('sniffles\_GT', 'sniffles\_PSV', 'sniffles\_LN', 'sniffles\_DR', 'sniffles\_ST', 'sniffles\_QV',

'sniffles\_TY', 'sniffles\_ID', 'sniffles\_RAL', 'sniffles\_AAL', 'sniffles\_CO')] <- str\_split\_fixed(tumor\_merged\_all$Sniffles, ':', 11)

tumor\_merged\_all[c('nanovar\_GT', 'nanovar\_PSV', 'nanovar\_LN', 'nanovar\_DR', 'nanovar\_ST', 'nanovar\_QV', 'nanovar\_TY',

'nanovars\_ID', 'nanovar\_RAL', 'nanovar\_AAL', 'nanovar\_CO')] <- str\_split\_fixed(tumor\_merged\_all$Nanovar, ':', 11)

tumor\_merged\_all[c('delly\_GT', 'delly\_PSV', 'delly\_LN', 'delly\_DR', 'delly\_ST', 'delly\_QV', 'delly\_TY', 'delly\_ID',

'delly\_RAL', 'nanovar\_AAL', 'delly\_CO')] <- str\_split\_fixed(tumor\_merged\_all$Delly, ':', 11)

tumor\_merged\_all[c('dbreak\_GT', 'dbreak\_PSV', 'dbreak\_LN', 'dbreak\_DR', 'delly\_ST', 'dbreak\_QV', 'dbreak\_TY', 'dbreak\_ID',

'dbreak\_RAL', 'dbreak\_AAL', 'dbreak\_CO')] <- str\_split\_fixed(tumor\_merged\_all$Dbreak, ':', 11)

tumor\_merged\_all[c('dysgu\_GT', 'dysgu\_PSV', 'dysgu\_LN', 'dysgu\_DR', 'dysgu\_ST', 'dysgu\_QV', 'dysgu\_TY', 'dysgu\_ID',

'dysgu\_RAL', 'dysgu\_AAL', 'dysgu\_CO')] <- str\_split\_fixed(tumor\_merged\_all$Dysgu, ':', 11)

tumor\_merged\_all[c('severus\_GT', 'severus\_PSV', 'severus\_LN', 'severus\_DR', 'severus\_ST', 'severus\_QV', 'severus\_TY',

'severus\_ID', 'severus\_RAL', 'severus\_AAL', 'severus\_CO')] <- str\_split\_fixed(tumor\_merged\_all$Severus, ':', 11)

# Filter the data to only include rows where the variant is not "NaN" for each variant caller

cutesv <- tumor\_merged\_all[tumor\_merged\_all$cutesv\_PSV != "NaN",]

svim <- tumor\_merged\_all[tumor\_merged\_all$svim\_PSV != "NaN",]

sniffles <- tumor\_merged\_all[tumor\_merged\_all$sniffles\_PSV != "NaN",]

nanovar <- tumor\_merged\_all[tumor\_merged\_all$nanovar\_PSV != "NaN",]

delly <- tumor\_merged\_all[tumor\_merged\_all$delly\_PSV != "NaN",]

dbreak <- tumor\_merged\_all[tumor\_merged\_all$dbreak\_PSV != "NaN",]

dysgu <- tumor\_merged\_all[tumor\_merged\_all$dysgu\_PSV != "NaN",]

severus <- tumor\_merged\_all[tumor\_merged\_all$severus\_PSV != "NaN",]

# Select only the "ID" and corresponding type (TY) for each variant caller

cutesv <- dplyr::select(cutesv, ID, cutesv\_TY)

svim <- dplyr::select(svim, ID, svim\_TY)

sniffles <- dplyr::select(sniffles, ID, sniffles\_TY)

nanovar <- dplyr::select(nanovar, ID, nanovar\_TY)

delly <- dplyr::select(delly, ID, delly\_TY)

dbreak <- dplyr::select(dbreak, ID, dbreak\_TY)

dysgu <- dplyr::select(dysgu, ID, dysgu\_TY)

severus <- dplyr::select(severus, ID, severus\_TY)

# Create a list with the IDs for each variant caller, this will be used to create the upset plot

x <- list(

cuteSV = cutesv$ID, # IDs from CuteSV caller

Svim = svim$ID, # IDs from Svim caller

Sniffles = sniffles$ID, # IDs from Sniffles caller

Nanovar = nanovar$ID, # IDs from Nanovar caller

Delly = delly$ID, # IDs from Delly caller

DeBreak = dbreak$ID, # IDs from DeBreak caller

Dysgu = dysgu$ID, # IDs from Dysgu caller

Severus = severus$ID # IDs from Severus caller

)

# Create an upset plot using the data, which shows the intersections between variant caller sets

# 'fromList' converts the list into the required format for the 'upset' function

upset(fromList(x),

nsets = 8, # The number of sets to plot (8 variant callers)

order.by = "freq", # Order the intersections by frequency

sets.x.label = "Total Somatic SV Number", # Label for the x-axis

mainbar.y.label = "Intersection", # Label for the main intersection bar

number.angles = 30, # Adjust angle for text labels in the plot

point.size = 2.5, # Size of the points in the plot

line.size = 0.75, # Line thickness

empty.intersections = "on", # Display empty intersections

text.scale = c(1.5, 1.5, 1.5, 1.5, 1.5, 1.5)) # Scale text size for readability

**PERFORMANCE METRICS AND CALCULATIONS**

