NanOlympicsMod – tutorial: how to add a new tool (e. g. m6Anet)

m6Anet, as many other tools, requires one step to run on each sample independently, and another step to combine results obtained for all the samples. For this reason, we are going to add two processes, named m6anet1 and m6anet2. In the following, we will show the steps required for including the tool m6Anet as part of the NanOlympicsMod pipeline.

1. In **pipeline.conf** file:
    a. Add one flag variable for each new process that you want to add, and set their status to true; in particular, we are going to add m6anet1 and m6anet2 variables (lines 41, 42). In case we would like to skip the processes execution, we will set these variables to false.
    b. Add options specific for each process; in particular, add the name of the image which should be downloaded from Docker Hub, mounting options and desired computational resources (lines 57-72).

2. In **pipeline.nf** file:
    a. Add channels to link processes, such as one channel from nanopolish1 to m6anet1 processes and one channel from m6anet2 to postprocessing processes (lines 101, 208)
    b. Add m6anet1 process, which should run on each sample independently (lines 128-150)
    c. Group by condition output from m6anet1 process and create a new channel for each condition (lines 152-156)
    d. Add m6anet2 process, which should analyse all samples assigned to test condition, after m6anet1 process is over for all the samples (lines 158-179)
    e. Add the path to m6Anet output as an argument to postprocessing.R script in postprocessing process (line 224)

3. In **postprocessing.R** file:
    a. Add a case to the switch code block for formatting m6Anet output into a genome-based bed file format. The final output should be a tab-separated file including chromosome, start, end, status (Mod/Unmod) and confidence parameter value. Since m6Anet works on the transcriptome, an additional code block for performing lift-over from transcriptome-based to genome-based coordinates is needed (lines 246-340).
    b. Add the name of the added tool and default/relaxed confidence parameter values (lines 345-352)

4. In **statistical_analysis.R** file:
    a. Add the tool to either listmax (as in case of m6Anet) or listmin vectors, depending on whether the confidence parameter should be maximised or minimised to obtain higher confidence calls, respectively (line 369-376)
    b. Add a line for renaming the tool with the correct case-sensitive spelling, useful for plot legends (line 393)
    c. Add the tool to the list of tools with a confidence parameter, for which PR curves should be plotted (lines 403, 409)

pipeline.conf

```
[…]
      nanopolish1 = true
      xpore = true
      nanocompore1 = true
      nanocompore2 = true
      m6anet1 = true
      m6anet2 = true
      yanocomp1 = true
      yanocomp2 = true
      postprocessing = true
}

[…]
      withName:xpore2{
            container = 'bproject/xpore:v1'
            containerOptions = '--bind /path/to/mount:/path/mounted'
            cpus = { params.xpore ? 3 : 1 }
            memory = { params.xpore ? 5.GB + (2.GB * (task.attempt-1)) : 1.GB }
            errorStrategy = { task.exitStatus == 130 ? 'retry' : 'terminate' }
            maxRetries = 3
      }
      withName:m6anet1{
            container = 'bproject/m6anet:v1'
            containerOptions = '--bind /path/to/mount:/path/mounted'
            cpus = { params.m6anet1 ? 3 : 1 }
            memory = { params.m6anet1 ? 5.GB + (2.GB * (task.attempt-1)) : 1.GB }
            errorStrategy = { task.exitStatus == 130 ? 'retry' : 'terminate' }
            maxRetries = 3
      }
      withName:m6anet2{
            container = 'bproject/m6anet:v1'
            containerOptions = '--bind /path/to/mount:/path/mounted'
            cpus = { params.m6anet2 ? 3 : 1 }
            memory = { params.m6anet2 ? 5.GB + (2.GB * (task.attempt-1)) : 1.GB }
            errorStrategy = { task.exitStatus == 130 ? 'retry' : 'terminate' }
            maxRetries = 3
```

```
        }
        withName:nanocompore1{
                container = 'bproject/nanocompore:v1'
                containerOptions = '--bind /path/to/mount:/path/mounted'
                cpus = { params.nanocompore1 ? 7 : 1 }
                memory = { params.nanocompore1 ? 10.GB + (2.GB * (task.attempt-1)) : 1.GB }
                errorStrategy = { task.exitStatus == 130 ? 'retry' : 'terminate' }
                maxRetries = 3
        }
[…]
```

pipeline.nf

```
[…]

// Resquigling with nanopolish for each condition
process nanopolish1 {
    input:
            tuple val(condition), val(sample) from minimap2_nanopolish1

            each file('transcriptome.fa') from transcriptome_fasta_nanopolish1
            each file('transcriptome.fa.fai') from transcriptome_fai_nanopolish1

            each file('genome.fa') from genome_fasta_nanopolish1
            each file('genome.fa.fai') from genome_fai_nanopolish1
    output:
      tuple val(condition), val(sample) into nanopolish1_xpore
      tuple val(condition), val(sample) into nanopolish1_nanocompore1
      tuple val(condition), val(sample) into nanopolish1_yanocomp1
      tuple val(condition), val(sample) into nanopolish1_m6anet1

    script:
    if(params.nanopolish1)
    """
            mkdir -p ${params.resultsDir}/${condition}/${sample}/nanopolish/
            mkdir -p ${params.resultsDir}/${condition}/${sample}/nanopolish/transcriptome/

[…]
```

```
110
111         nanocompore sampcomp --file_list1 "\${f1[*]}" --file_list2 "\${f2[*]}" \
112               --label1 ${condition1} \
113               --label2 ${condition2} \
114             --fasta transcriptome.fa \
115             --bed transcriptome.bed \
116         --outpath ${params.resultsDir}/nanocompore/ \
117         --allow_warnings \
118         --logit \
119         --nthreads ${task.cpus} \
120         --overwrite
121       """
122         else
123         """
124           echo "Skipped"
125       """
126   }
127
128   // Data formatting for m6anet for each sample
129   process m6anet1 {
130       input:
131             tuple val('condition'), val('sample') from nanopolish1_m6anet1
132
133       output:
134         tuple val(condition), val(sample), val() into m6anet1_m6anet2
135
136
137       script:
138       if(params.m6anet1)
139       """
140           mkdir -p ${params.resultsDir}/${condition}/${sample}/m6anet/
141
142           m6anet-dataprep --eventalign
143   ${params.resultsDir}/${condition}/${sample}/nanopolish/transcriptome/eventalign_readIndex.txt \
144                   --out_dir ${params.resultsDir}/${condition}/${sample}/m6anet --n_processes ${task.cpus}
145       """
146         else
147         """
```

```
            ln -sf ${params.resultsDir}/${condition}/${sample}/m6anet m6anet
    """
}

// From a single channel for all the alignments to one channel for each condition
ni_test_m6anet2=Channel.create()
ni_other_m6anet2=Channel.create()
m6anet1_m6anet2.groupTuple(by:0)
      .choice( ni_test_m6anet2, ni_other_m6anet2 ) { a -> a[0] == params.test_condition ? 0 : 1 }

// RNA modifications detection with m6anet
process m6anet2 {
    input:
         tuple val('condition1'), val('sample1') from ni_test_m6anet2

    output:
      val('flagm6anet') into m6anet_postprocessing
    script:
    if(params.m6anet2)
    """
        mkdir -p ${params.resultsDir}/m6anet
        preprocessing_dirs=\$(find ${params.resultsDir}/${condition1} -maxdepth 2 -type d | grep "m6anet\$")
        m6anet-run_inference --input_dir \$preprocessing_dirs --out_dir ${params.resultsDir}/m6anet --
infer_mod_rate --n_processes ${task.cpus}

      zcat ${params.resultsDir}/m6anet/data.result.csv.gz > ${params.resultsDir}/m6anet/data.result.csv
    """
      else
      """
        echo "Skipped"
    """
}

// Data formatting for yanocomp for each sample
process yanocomp1 {
    input:
         tuple val('condition'), val('sample') from nanopolish1_yanocomp1
         each file('genome.gtf') from gtf_yanocomp
```

```
    output:
      tuple val(condition), file('outputT.hdf5'), file('outputG.hdf5') into yanocomp1_yanocomp2

[…]

// Processing of each output to obtain bed files
process postprocessing {
    input:
            val('flagyanocomp2') from yanocomp2_postprocessing
            val('flagdena') from dena_postprocessing
            val('flagdrummer') from drummer_postprocessing
            val('flagdifferr') from differr_postprocessing
            val('flagnanom6a') from nanom6a_postprocessing
            val('flagnanocompore') from nanocompore_postprocessing
            val('flageligos') from eligos_postprocessing
            val('flagmines') from mines_postprocessing
            val('flagepinanoSVM') from epinanoSVM_postprocessing
            val('flagepinanoError') from epinanoError_postprocessing
            val('flagxpore') from xpore_postprocessing
            val('flagnanodoc') from nanodoc_postprocessing
            val('flagtombo2') from tombo2_postprocessing
            val('flagtombo3') from tombo3_postprocessing
            val('flagm6anet') from m6anet_postprocessing

    output:

    script:
    if(params.postprocessing)
    """
            mkdir -p ${params.resultsDir}/output_bed_files/
            mkdir -p ${params.resultsDir}/output_statistical/

            Rscript ${params.postprocessingScript} path=${params.resultsDir} genomebed=${params.genesbed}
genomegtf=${params.gtf} resultsFolder=${params.resultsDir}/output_bed_files/ mccores=${task.cpus}
threshold=${params.threshold} pathdena=${params.test_condition}/dena pathdrummer=drummer pathdifferr=differr
pathyanocomp=yanocomp pathmines=${params.test_condition}/mines pathnanocompore=nanocompore
patheligos=eligos/merged pathepinanoError=epinanoError pathepinanoSVM=${params.test_condition}/epinanoSVM
```

```
pathxpore=xpore pathnanodoc=nanodoc pathnanom6a=${params.test_condition}/nanom6a/result_final
pathtomboComparison=tomboComparison pathm6anet=m6anet

                Rscript ${params.statisticalAnalysis} bed_folder=${params.resultsDir}/output_bed_files
genomegtf=${params.gtf} genesbed=${params.genesbed} resultsFolder=${params.resultsDir}/output_statistical/
mccores=${task.cpus} peaks=${params.peaksfile} binLength=${params.binLength} genomefile=${params.genome_fasta}

    """
      else
      """
        echo "Skipped"
    """
}
```

Postprocessing.R

```
[…]
              else {message(paste0(tool,"'s output files don't exist."))}
            }
            if (!file.exists(output_file)) {
              output_tomboComparison()
            }
            },
            m6anet = {output_m6anet <- function(){if (file.exists(paste0(path_folder, "/", "data.result.csv"))
                                                   && file.info(paste0(path_folder, "/",
"data.result.csv"))$size != 0){
            #read results from new_tool and save relevant columns to m6anet df
            data_m6anet <- read.table(paste0(path_folder, "/", "data.result.csv"), header = TRUE, sep=",")

            m6anet <- data.frame("TranscriptID" = data_m6anet[,1],
                                 "Start" = data_m6anet[,2] + 2,
                                 "End" = data_m6anet[,2] + 2,
                                 "Status" = data_m6anet[,4],
                                 "Prob_mod" = data_m6anet[,4]
              )

            #use m6anet$Status < filtering_parameter instead of m6anet$Status > filtering_parameter if ConfPar
      parameter needs to be minimised
```

```
      m6anet$Status <- ifelse(!is.nan(m6anet$Status) & !is.na(m6anet$Status) & m6anet$Status >
filtering_parameter, "Mod", "Unmod")
#retain only Mod sites
      m6anet <- m6anet[which(m6anet$Status == "Mod"), ]
      #Creation Edb Database from genome GTF
      EnsDb <- suppressWarnings(suppressMessages(ensDbFromGtf(gtf = genome_gtf)))
      edb <- EnsDb(EnsDb)
      # Lift-over + output bed
      test_m6anet <- IRanges(start = m6anet[,2], end = m6anet[,3], names = c(m6anet[,1]))
      #process num_rows_chunk at a time
      num_rows_chunk <- 1
      mc.cores <- as.numeric(mccores)
      #split the reads in chunks of size num_rows_chunk
      if (length(test_m6anet) < num_rows_chunk) {
            test_m6anet_split <- list(test_m6anet)
            } else {
            test_m6anet_split <- split(test_m6anet, rep(seq(from = 1, to =
      ceiling(length(test_m6anet)/num_rows_chunk)), each = num_rows_chunk)[1:length(test_m6anet)])
            }
      #initialize vectors
      tmp1 <- vector(mode = "list", length = length(test_m6anet_split))
      names(tmp1) <- 1:length(test_m6anet_split)
      tmp <- vector(mode = "list", length = length(test_m6anet_split))
      names(tmp) <- 1:length(test_m6anet_split)
      ind_retry <- 1:length(test_m6anet_split)
      #keep retrying on chunks which failed unexpectedly
      while(any(unlist(lapply(tmp, is.null)))) {
            cat(sprintf("Starting new iteration for m6Anet; %d sites missing\n",
      length(which(unlist(lapply(tmp, is.null))))))
            tmp1 <- tmp1[ind_retry]
            #if lift-over fails, print "Warning" or "Error"
            tmp1 <- mclapply(test_m6anet_split[ind_retry], function(x) {
            tryCatch({
                  coordinate_m6anet_unlisted <- unlist(transcriptToGenome(x, edb))
                  return(coordinate_m6anet_unlisted)
            }, warning = function(w) {
            print("Warning")
            return(NULL)
```

```
                }, error = function(e) {
                      print("Error")
                      return(NULL)
                }
                )}, mc.cores = mc.cores)
                #retrieve chunks which failed
                ind_retry <- names(which(unlist(lapply(tmp1, function(x) is.null(x)))))
                ind_ok <- names(which(unlist(lapply(tmp1, function(x) !is.null(x)))))
                tmp[ind_ok] <- tmp1[ind_ok]
                if (length(ind_retry) > 0) {
                      tmp1 <- tmp1[ind_retry]
                }
          }
          #unlist the output of lift-over
          coordinate_m6anet_unlisted <- unlist(as(tmp, "GRangesList"))
          #convert the lifted-over hits to dataframe
          df_m6anet <- as.data.frame(unname(coordinate_m6anet_unlisted[,c(0,2,4,5)]))[,c(1:3,5,6,7,8)]
          #assign rownames and delete duplicate hits (may be due to hits falling on exon-intron junctions)
          tmp_rownames <- paste0(df_m6anet[, 6], "_", df_m6anet[, 7], "_", df_m6anet[, 5])
          dup_names <- names(which(table(tmp_rownames) > 1))
          ind_dup <- which(tmp_rownames %in% dup_names)
          ind_dup_rm <- ind_dup[which(duplicated(df_m6anet[ind_dup, c(5,6,7)]))]
          if (length(ind_dup_rm) > 0) {
                df_m6anet <- df_m6anet[-ind_dup_rm, ]
          }
          rownames(df_m6anet) <- paste0(df_m6anet[, 6], "_", df_m6anet[, 7], "_", df_m6anet[, 5]
          rownames(m6anet) <- paste0(m6anet[, 2], "_", m6anet[, 3], "_", m6anet[, 1])
          #assign colnames and write to file
          df_m6anet$Status <- m6anet[rownames(df_m6anet), 4]
          df_m6anet$Prob_Mod <- m6anet[rownames(df_m6anet), 5]
          df_m6anet_final <- df_m6anet[,c(1,2,3,4,8,9)]
          colnames(df_m6anet_final) <- c("Chr", "Start", "End", "Strand", "Status", "Prob_mod")
          write.table(df_m6anet_final, file = output_file, quote = F, sep = "\t", row.names = F)
          }
                else {message(paste0(tool,"'s output files don't exist."))}
}
      if (!file.exists(output_file)) {
            output_m6anet()
```

```
            }},
             stop("Enter a valid tool as input!")
    )}
}

# Defininining a set of parameters used to filter the results for those tools which give as output all the sites
rrach <- c("AAACA","AAACT","AAACC","GAACA","GAACT","GAACC","GGACA","GGACT","GGACC","GAACA","GAACT","GAACC")
tools <- c("dena", "drummer", "differr", "yanocomp", "nanocompore", "eligos", "mines"
           , "epinanoErr", "epinanoSvm", "xpore", "nanodoc", "nanom6a", "tomboComparison", "m6anet")

pathTools <- c(pathdena, pathdrummer, pathdifferr, pathyanocomp, pathnanocompore, patheligos
               , pathmines, pathepinanoError, pathepinanoSVM, pathxpore, pathnanodoc, pathnanom6a
               , pathtomboComparison, pathm6anet)

default <- c(0.1, 0.05, 0.05, 0.05, 0.01, 0.0001, NA, 0.1, 0.5, 0.05, 0.02, NA, 0.05, 0.9)
relaxed <- c(0, NA, NA, NA, 1, 1, NA, NA, 0, 1, 0, NA, 1, 0)
value <- rep(threshold, length(default))

names(pathTools) <- names(default) <- names(relaxed) <- names(value) <- tools

parameters_list <- list("default" = unname(default), "relaxed" = unname(relaxed), "value" = unname(value))

[…]
```

statistical_analysis.R

```
[…]

## 1) Load all the output files from the output directory
files <- list.files(bed_folder, full.names = TRUE, pattern = "\\.bed") # output_directory parameter from
outside

listmax <- paste0(c("DENA", "EpiNano-Error", "EpiNano-SVM", "NanoDoc", "m6Anet"), collapse = "|") # for these
tools we need to maximize the filtering paramenter when there are more than 1 in a bin
listmin <- paste0(c("DiffErr", "DRUMMER", "Yanocomp", "Nanocompore", "ELIGOS", "xPore", "Tombo"), collapse =
"|") # for these tools we need to minimize the filtering parameter

threshold_default <- c(0.1, 0.05, 0.05, 0.05, 0.01, 0.0001, 0.1, 0.5, 0.05, 0.02, 0.05, 0.9)
```

```
names(threshold_default) <- c("DENA", "DiffErr", "DRUMMER", "Yanocomp", "Nanocompore", "ELIGOS",
                              "EpiNano-Error", "EpiNano-SVM", "xPore", "NanoDoc", "Tombo", "m6Anet")

chrs <- readDNAStringSet(genomefile, format="fasta")
RRACH_plus <- GRanges(vmatchPattern(pattern = "RRACH", subject = chrs, fixed = "subject"), strand = "+")
RRACH_minus <- GRanges(vmatchPattern(pattern = "DGTYY", subject = chrs, fixed = "subject"), strand = "-")
RRACH <- c(RRACH_plus, RRACH_minus)

[…]

Run_statistical_analysis <- function(genesBins_par, peaks_par, files_par, notes = "", w) {
  tools <- basename(files_par)
  tools[grep(pattern = "dena", x = tools)] <- "DENA"
  tools[grep(pattern = "drummer", x = tools)] <- "DRUMMER"
  tools[grep(pattern = "differr", x = tools)] <- "DiffErr"
  tools[grep(pattern = "eligos", x = tools)] <- "ELIGOS"
  tools[grep(pattern = "epinanoErr", x = tools)] <- "EpiNano-Error"
  tools[grep(pattern = "epinanoSvm", x = tools)] <- "EpiNano-SVM"
  tools[grep(pattern = "m6anet", x = tools)] <- "m6Anet"
  tools[grep(pattern = "mines", x = tools)] <- "MINES"
  tools[grep(pattern = "nanocompore", x = tools)] <- "Nanocompore"

[…]

    # Overlap between m6A detected site of each tool and genome binned
    overlap <- as.matrix(findOverlaps(query = granges, subject = genesBins_par, minoverlap = 1, type = "any"))
    if (grepl(x, pattern = paste0(c("DENA","DRUMMER","DiffErr","Yanocomp","Nanocompore","ELIGOS","EpiNano-
Error",
                                    "EpiNano-SVM","xPore","NanoDoc","Tombo", "m6Anet"), collapse = "|"))) {
      # Add column of filtering parameter
      filtering_parameter <- bed_file[overlap[,"queryHits"], 6]
      overlap_w_parameter <- cbind(overlap, filtering_parameter)
      default <- unname(threshold_default[grep(x, pattern =
paste0(c("DENA","DRUMMER","DiffErr","Yanocomp","Nanocompore","ELIGOS","EpiNano-Error","EpiNano-SVM",
                                                                "xPore","NanoDoc","Tombo", "m6Anet"),
collapse = "|"), value = T)])
 […]
```