server.R

tomis

2025-06-04

library(shiny)

## Warning: パッケージ 'shiny' はバージョン 4.4.3 の R の下で造られました

library(Seurat)

## 要求されたパッケージ SeuratObject をロード中です

## 要求されたパッケージ sp をロード中です

##   
## 次のパッケージを付け加えます: 'SeuratObject'

## 以下のオブジェクトは 'package:base' からマスクされています:  
##   
## intersect, t

library(Matrix)  
library(ggplot2)

## Warning: パッケージ 'ggplot2' はバージョン 4.4.3 の R の下で造られました

library(DT)

## Warning: パッケージ 'DT' はバージョン 4.4.3 の R の下で造られました

##   
## 次のパッケージを付け加えます: 'DT'

## 以下のオブジェクトは 'package:Seurat' からマスクされています:  
##   
## JS

## 以下のオブジェクトは 'package:SeuratObject' からマスクされています:  
##   
## JS

## 以下のオブジェクトは 'package:shiny' からマスクされています:  
##   
## dataTableOutput, renderDataTable

library(dplyr)

##   
## 次のパッケージを付け加えます: 'dplyr'

## 以下のオブジェクトは 'package:stats' からマスクされています:  
##   
## filter, lag

## 以下のオブジェクトは 'package:base' からマスクされています:  
##   
## intersect, setdiff, setequal, union

library(shinyjs)

## Warning: パッケージ 'shinyjs' はバージョン 4.4.3 の R の下で造られました

##   
## 次のパッケージを付け加えます: 'shinyjs'

## 以下のオブジェクトは 'package:Matrix' からマスクされています:  
##   
## show

## 以下のオブジェクトは 'package:SeuratObject' からマスクされています:  
##   
## show

## 以下のオブジェクトは 'package:sp' からマスクされています:  
##   
## show

## 以下のオブジェクトは 'package:shiny' からマスクされています:  
##   
## runExample

## 以下のオブジェクトは 'package:methods' からマスクされています:  
##   
## removeClass, show

library(decoupleR)  
library(OmnipathR)  
library(stringr)  
library(viridis)

## 要求されたパッケージ viridisLite をロード中です

library(tibble)  
library(tidyr)

##   
## 次のパッケージを付け加えます: 'tidyr'

## 以下のオブジェクトは 'package:Matrix' からマスクされています:  
##   
## expand, pack, unpack

library(pheatmap)  
library(RColorBrewer)  
  
options(shiny.maxRequestSize = 100 \* 1024^2)  
  
shinyServer(function(input, output, session) {  
   
 initial\_seurat\_obj <- reactiveVal(NULL)  
 seurat\_obj <- reactiveVal(NULL)  
 marker\_result <- reactiveVal(NULL)  
 heatmap\_obj <- reactiveVal(NULL)  
 tf\_activities <- reactiveVal(NULL)   
 tf\_top\_tfs <- reactiveVal(NULL)   
 tf\_pheatmap\_mat <- reactiveVal(NULL)   
 tf\_activities\_df <- reactiveVal(NULL)  
   
   
 collectri\_net <- reactive({  
 req(input$target\_spiecies)   
 net <- get\_collectri(organism = input$target\_spiecies, split\_complexes = FALSE)  
 return(net)  
 })  
   
 observe({  
   
 shinyjs::toggleState("run", !is.null(initial\_seurat\_obj()))  
 shinyjs::toggleState("calc\_marker", !is.null(seurat\_obj()))  
 shinyjs::toggleState("preview\_download\_UMAP", !is.null(seurat\_obj()))  
 shinyjs::toggleState("preview\_download\_feature", !is.null(seurat\_obj()))  
 shinyjs::toggleState("preview\_download\_dotplot", !is.null(seurat\_obj()))  
 shinyjs::toggleState("preview\_download\_vlnplot", !is.null(seurat\_obj()))  
 shinyjs::toggleState("dotsize", !is.null(seurat\_obj()))  
 shinyjs::toggleState("download\_marker", !is.null(marker\_result()))  
 shinyjs::toggleState("top\_n\_genes", !is.null(seurat\_obj()))  
 shinyjs::toggleState("draw\_heatmap", !is.null(seurat\_obj()))  
 shinyjs::toggleState("preview\_download\_heatmap", !is.null(heatmap\_obj()))  
   
   
 shinyjs::toggleState("draw\_tf\_heatmap", !is.null(seurat\_obj()) && !is.null(input$target\_spiecies))  
   
 shinyjs::toggleState("top\_n\_active\_TF", !is.null(seurat\_obj()))  
 shinyjs::toggleState("preview\_download\_tf", !is.null(tf\_pheatmap\_mat()))  
 })  
   
 #Upload data & Create VlnPlot for QC  
 observeEvent(input$run\_initial\_processing, {  
 req(input$zipfile)  
 withProgress(message = "Reading data and Preparing for QC...", {  
 zip\_path <- input$zipfile$datapath  
 extract\_dir <- tempfile()  
 unzip(zip\_path, exdir = extract\_dir)  
   
 subdirs <- list.dirs(extract\_dir, full.names = TRUE, recursive = FALSE)  
 data\_dir <- if (length(subdirs) == 1) subdirs[1] else extract\_dir  
   
 expr <- Read10X(data.dir = data\_dir)  
 seu <- CreateSeuratObject(counts = expr)  
   
 if (ncol(seu) > 1000) {   
 seu <- subset(seu, cells = sample(colnames(seu), size = 500, replace = FALSE))  
 }  
   
 seu[["percent.mt"]] <- PercentageFeatureSet(seu, pattern = "^MT-|^mt-")  
   
 initial\_seurat\_obj(seu)  
 })  
 })  
   
 #QC Plot  
 output$qc\_plot\_mt <- renderPlot({  
 req(initial\_seurat\_obj())  
 VlnPlot(initial\_seurat\_obj(), features = c("nFeature\_RNA", "nCount\_RNA", "percent.mt"), ncol = 3, pt.size =0.1)  
 })  
   
 #apply QC Check & Run Seurat  
 observeEvent(input$run, {  
 req(initial\_seurat\_obj())  
 req(input$min\_features, input$max\_features, input$max\_mt\_percent)  
   
 withProgress(message = "Running Seurat Analysis...", value = 0, {  
 seu <- initial\_seurat\_obj()  
   
 #QC filtering  
 seu\_filtered <- subset(seu,  
 subset = nFeature\_RNA > input$min\_features &  
 nFeature\_RNA < input$max\_features &  
 percent.mt < input$max\_mt\_percent)  
   
   
 if (ncol(seu\_filtered) == 0) {  
 showModal(modalDialog(  
 title = "Error",  
 "No cells remain after quality filtering. Please adjust your QC parameters.",  
 footer = modalButton("Close")  
 ))  
 seurat\_obj(NULL)   
 return(NULL)   
 }  
   
 incProgress(0.1, message = "Normalizing data...")  
 seu <- NormalizeData(seu\_filtered)  
   
 incProgress(0.2, message = "Finding Variable Features...")  
 seu <- FindVariableFeatures(seu)  
   
 incProgress(0.3, message = "Scaling Data...")  
 seu <- ScaleData(seu, features = VariableFeatures(seu))  
 seu <- RunPCA(seu)  
   
 incProgress(0.4, message = "Applying Parameters")  
 dims\_use <- 1:input$dims\_pca  
 resolution\_use <- input$resolution  
   
 seu <- FindNeighbors(seu, dims = dims\_use)  
 seu <- FindClusters(seu, resolution = resolution\_use)  
   
 incProgress(0.5, message = "Running UMAP")  
 seu <- RunUMAP(seu, dims = dims\_use)  
   
 incProgress(0.6, message = "Depicting UMAP")  
 seurat\_obj(seu)  
 })  
 })  
   
   
 observe({  
 req(seurat\_obj())   
 current\_obj <- seurat\_obj()  
 assay\_choices <- Assays(current\_obj)  
   
 current\_selected\_assay <- input$selected\_assay   
   
 if ("tfsulm" %in% assay\_choices) {  
 choices\_to\_use <- c("RNA", "tfsulm")  
 selected\_to\_use <- ifelse(current\_selected\_assay %in% choices\_to\_use, current\_selected\_assay, "RNA")  
   
 updateSelectInput(session, "selected\_assay",  
 choices = choices\_to\_use,  
 selected = selected\_to\_use)  
 } else {  
 choices\_to\_use <- "RNA"  
 selected\_to\_use <- "RNA"  
   
 updateSelectInput(session, "selected\_assay",  
 choices = choices\_to\_use,  
 selected = selected\_to\_use)  
 }  
 })  
   
   
 observe({  
   
 req(seurat\_obj(), input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 genes <- rownames(temp\_obj)  
   
 updateSelectizeInput(session, 'selected\_gene', choices = genes, server = TRUE)  
 })  
   
   
 observeEvent(input$calc\_marker, {  
 req(seurat\_obj())  
 showModal(modalDialog("Calculating marker genes, please wait...", footer = NULL))  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- "RNA"  
 markers <- FindAllMarkers(temp\_obj, only.pos = TRUE, min.pct = 0.3, logfc.threshold = 0.4, test.use = "MAST")  
 marker\_result(markers)  
 removeModal()  
 })  
   
 output$cluster\_selector <- renderUI({  
 req(marker\_result())  
 clusters <- sort(unique(marker\_result()$cluster))  
 selectInput("selected\_cluster", "Select Cluster", choices = clusters, selected = clusters[1])  
 })  
   
 output$marker\_table <- DT::renderDataTable({  
 req(marker\_result())  
 df <- marker\_result()  
 datatable(df, filter = "top", options = list(pageLength = 10))  
 })  
   
 output$umapPlot <- renderPlot({  
 req(seurat\_obj())  
 DimPlot(seurat\_obj(), reduction = "umap", label = TRUE,  
 label.size = 8, pt.size = input$dotsize) +  
 ggtitle("UMAP (clusters)") +  
 theme(axis.text.y = element\_text(size = 15, color = "black"),  
 axis.text.x = element\_text(size = 15, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 20)) +  
 guides(color = guide\_legend(override.aes = list(size = 8, alpha = 1)))  
 })  
   
   
 observeEvent(input$preview\_download\_UMAP, {  
 req(seurat\_obj())  
 showModal(modalDialog(  
 title = "Download UMAP Plot",  
 fluidRow(  
 column(9,  
 h4("Preview"),  
 plotOutput("umap\_preview\_plot", height = "auto")  
 ),  
 column(3,  
 h4("Settings"),  
 numericInput("umap\_download\_width", "Width (pixel)", value = 500, min = 1),  
 numericInput("umap\_download\_height", "Height (pixel)", value = 500, min = 1),  
 textInput("umap\_download\_filename", "File Name", value = "UMAP\_plot.png"),  
 downloadButton("do\_download\_UMAP", "Download Plot")  
 )  
 ),  
 size = "l",  
 footer = modalButton("Close")  
 ))  
 })  
   
   
 output$umap\_preview\_plot <- renderPlot({  
 req(seurat\_obj())  
 plt <- DimPlot(seurat\_obj(), reduction = "umap", label = TRUE,  
 label.size = 8, pt.size = input$dotsize) +  
 ggtitle("UMAP (clusters)") +  
 theme(axis.text.y = element\_text(size = 15, color = "black"),  
 axis.text.x = element\_text(size = 15, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 20)) +  
 guides(color = guide\_legend(override.aes = list(size = 8, alpha = 1)))  
   
 plt  
 }, width = function() {  
 req(input$umap\_download\_width)  
 return(input$umap\_download\_width)  
 }, height = function() {  
 req(input$umap\_download\_height)  
 return(input$umap\_download\_height)  
 })  
   
   
 output$do\_download\_UMAP <- downloadHandler(  
 filename = function() {  
 input$umap\_download\_filename  
 },  
 content = function(file) {  
 req(seurat\_obj())  
 plt <- DimPlot(seurat\_obj(), reduction = "umap", label = TRUE,  
 label.size = 8, pt.size = input$dotsize) +  
 ggtitle("UMAP (clusters)") +  
 theme(axis.text.y = element\_text(size = 15, color = "black"),  
 axis.text.x = element\_text(size = 15, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 20)) +  
 guides(color = guide\_legend(override.aes = list(size = 8, alpha = 1)))  
   
 ggsave(file, plot = plt,  
 width = input$umap\_download\_width,  
 height = input$umap\_download\_height,  
 units = "px",  
 dpi = 96)  
 }  
 )  
   
 output$featurePlot <- renderPlot({  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 FeaturePlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 15, color = "black"),  
 axis.text.x = element\_text(size = 15, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 20),  
 title = element\_text(size = 20, color = "black"),  
 legend.key.height = unit(20, "pt")) &  
 scale\_color\_gradientn(colours = rev(brewer.pal(n = 11, name = "RdBu")))  
 })  
   
   
 observeEvent(input$preview\_download\_feature, {  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 showModal(modalDialog(  
 title = paste0("Download Feature Plot (", input$selected\_gene, ")"),  
 fluidRow(  
 column(9,  
 h4("Preview"),  
 plotOutput("feature\_preview\_plot", height = "auto")  
 ),  
 column(3,  
 h4("Settings"),  
 numericInput("feature\_download\_width", "Width (pixel)", value = 500, min = 1),  
 numericInput("feature\_download\_height", "Height (pixel)", value = 500, min = 1),  
 textInput("feature\_download\_filename", "File Name", value = paste0("FeaturePlot\_", input$selected\_gene, ".png")),  
 downloadButton("do\_download\_feature", "Download Plot")  
 )  
 ),  
 size = "l",  
 footer = modalButton("Close")  
 ))  
 })  
   
   
 output$feature\_preview\_plot <- renderPlot({  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 plt <- FeaturePlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 15, color = "black"),  
 axis.text.x = element\_text(size = 15, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 20),  
 title = element\_text(size = 20, color = "black"),  
 legend.key.height = unit(20, "pt"))  
 plt  
 }, width = function() {  
 req(input$feature\_download\_width)  
 return(input$feature\_download\_width)  
 }, height = function() {  
 req(input$feature\_download\_height)  
 return(input$feature\_download\_height)  
 })  
   
   
 output$do\_download\_feature <- downloadHandler(  
 filename = function() {  
 input$feature\_download\_filename  
 },  
 content = function(file) {  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 plt <- FeaturePlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 15, color = "black"),  
 axis.text.x = element\_text(size = 15, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 20),  
 title = element\_text(size = 20, color = "black"))  
 ggsave(file, plot = plt,  
 width = input$feature\_download\_width,  
 height = input$feature\_download\_height,  
 units = "px",  
 dpi = 96)  
 }  
 )  
   
 output$dotPlot <- renderPlot({  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 DotPlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 20, color = "black", angle = 0),  
 axis.text.x = element\_text(size = 20, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 15),  
 title = element\_text(size = 20, color = "black"))  
 })  
   
   
 observeEvent(input$preview\_download\_dotplot, {  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 showModal(modalDialog(  
 title = paste0("Download Dot Plot (", input$selected\_gene, ")"),  
 fluidRow(  
 column(9,  
 h4("Preview"),  
 plotOutput("dot\_preview\_plot", height = "auto")  
 ),  
 column(3,  
 h4("Settings"),  
 numericInput("dot\_download\_width", "Width (pixel)", value = 300, min = 1),  
 numericInput("dot\_download\_height", "Height (pixel)", value = 500, min = 1),  
 textInput("dot\_download\_filename", "File Name", value = paste0("DotPlot\_", input$selected\_gene, ".png")),  
 downloadButton("do\_download\_dotplot", "Download Plot")  
 )  
 ),  
 size = "l",  
 footer = modalButton("Close")  
 ))  
 })  
   
   
 output$dot\_preview\_plot <- renderPlot({  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 plt <- DotPlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 20, color = "black", angle = 0),  
 axis.text.x = element\_text(size = 20, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 15),  
 title = element\_text(size = 20, color = "black"))  
 plt  
 }, width = function() {  
 req(input$dot\_download\_width)  
 return(input$dot\_download\_width)  
 }, height = function() {  
 req(input$dot\_download\_height)  
 return(input$dot\_download\_height)  
 })  
   
   
 output$do\_download\_dotplot <- downloadHandler(  
 filename = function() {  
 input$dot\_download\_filename  
 },  
 content = function(file) {  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 plt <- DotPlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 20, color = "black", angle = 0),  
 axis.text.x = element\_text(size = 20, color = "black"),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 20),  
 axis.title.x = element\_text(size = 20),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 legend.text = element\_text(size = 15),  
 title = element\_text(size = 20, color = "black"))  
 ggsave(file, plot = plt,  
 width = input$dot\_download\_width,  
 height = input$dot\_download\_height,  
 units = "px",  
 dpi = 96)  
 }  
 )  
   
 output$vlnPlot <- renderPlot({  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 VlnPlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 25, color = "black", angle = 0),  
 axis.text.x = element\_text(size = 25, color = "black", angle = 0),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 25, color = "black"),  
 axis.title.x = element\_text(size = 25, color = "black"),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 axis.ticks.y = element\_line(linewidth = 1),  
 axis.ticks.x = element\_line(linewidth = 1),  
 title = element\_text(size = 20, color = "black")) +  
 NoLegend()  
 })  
   
   
 observeEvent(input$preview\_download\_vlnplot, {  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 showModal(modalDialog(  
 title = paste0("Download Violin Plot (", input$selected\_gene, ")"),  
 fluidRow(  
 column(9,  
 h4("Preview"),  
 plotOutput("vln\_preview\_plot", height = "auto")  
 ),  
 column(3,  
 h4("Settings"),  
 numericInput("vln\_download\_width", "Width (pixel)", value = 1000, min = 1),  
 numericInput("vln\_download\_height", "Height (pixel)", value = 500, min = 1),  
 textInput("vln\_download\_filename", "File Name", value = paste0("VlnPlot\_", input$selected\_gene, ".png")),  
 downloadButton("do\_download\_vlnplot", "Download Plot")  
 )  
 ),  
 size = "l",  
 footer = modalButton("Close")  
 ))  
 })  
   
   
 output$vln\_preview\_plot <- renderPlot({  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 plt <- VlnPlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 25, color = "black", angle = 0),  
 axis.text.x = element\_text(size = 25, color = "black", angle = 0),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 25, color = "black"),  
 axis.title.x = element\_text(size = 25, color = "black"),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 axis.ticks.y = element\_line(linewidth = 1),  
 axis.ticks.x = element\_line(linewidth = 1),  
 title = element\_text(size = 20, color = "black")) +  
 NoLegend()  
 plt  
 }, width = function() {  
 req(input$vln\_download\_width)  
 return(input$vln\_download\_width)  
 }, height = function() {  
 req(input$vln\_download\_height)  
 return(input$vln\_download\_height)  
 })  
   
   
 output$do\_download\_vlnplot <- downloadHandler(  
 filename = function() {  
 input$vln\_download\_filename  
 },  
 content = function(file) {  
 req(seurat\_obj(), input$selected\_gene, input$selected\_assay)  
 temp\_obj <- seurat\_obj()  
 DefaultAssay(temp\_obj) <- input$selected\_assay  
 plt <- VlnPlot(temp\_obj, features = input$selected\_gene) +  
 ggtitle(input$selected\_gene) +  
 theme(axis.text.y = element\_text(size = 25, color = "black", angle = 0),  
 axis.text.x = element\_text(size = 25, color = "black", angle = 0),  
 axis.line.y = element\_line(linewidth = 1),  
 axis.line.x = element\_line(linewidth = 1),  
 axis.title.y = element\_text(size = 25, color = "black"),  
 axis.title.x = element\_text(size = 25, color = "black"),  
 axis.ticks.length.y = unit(.0, "cm"),  
 axis.ticks.length.x = unit(.0, "cm"),  
 axis.ticks.y = element\_line(linewidth = 1),  
 axis.ticks.x = element\_line(linewidth = 1),  
 title = element\_text(size = 20, color = "black")) +  
 NoLegend()  
 ggsave(file, plot = plt,  
 width = input$vln\_download\_width,  
 height = input$vln\_download\_height,  
 units = "px",  
 dpi = 96)  
 }  
 )  
   
 output$download\_marker <- downloadHandler(  
 filename = function() {  
 paste0("markers\_cluster", ".csv")  
 },  
 content = function(file) {  
 df <- marker\_result()  
 write.csv(df, file, row.names = FALSE)  
 }  
 )  
   
 observeEvent(input$draw\_heatmap, {  
 req(seurat\_obj(), marker\_result())  
 withProgress(message = "Generating heatmap...", {  
 top\_genes <- marker\_result() %>%  
 group\_by(cluster) %>%  
 slice\_max(order\_by = avg\_log2FC, n = input$top\_n\_genes, with\_ties = FALSE) %>%  
 pull(gene) %>%  
 unique()  
   
 obj <- seurat\_obj()  
 DefaultAssay(obj) <- "RNA"   
 obj <- ScaleData(obj, features = top\_genes)  
   
 plt <- DoHeatmap(obj, features = top\_genes,  
 group.colors = NULL)+  
 theme(axis.text.y = element\_text(size = 20, colour = "black")) +  
 NoLegend()  
 heatmap\_obj(plt)  
 })  
 })  
   
 output$heatmapPlot <- renderPlot({  
 req(heatmap\_obj())  
 heatmap\_obj()  
 })  
   
   
 observeEvent(input$preview\_download\_heatmap, {  
 req(heatmap\_obj())  
 showModal(modalDialog(  
 title = "Download Heatmap",  
 fluidRow(  
 column(10,  
 h4("Preview"),  
 plotOutput("Heatmap\_preview\_plot", height = "auto")  
 ),  
 column(2,  
 h4("Settings"),  
 numericInput("Heatmap\_download\_width", "Width (pixel)", value = 600, min = 10),  
 numericInput("Heatmap\_download\_height", "height (pixel)", value = 1000, min = 10),  
 textInput("Heatmap\_download\_filename", "File name", value = "Heatmap\_plot.png"),  
 downloadButton("do\_download\_Heatmap", "Download Heatmap")  
 )  
 ),  
 size = "l",  
 footer = modalButton("Close")  
 ))  
 })  
   
   
 output$Heatmap\_preview\_plot <- renderPlot({  
 req(heatmap\_obj())  
 heatmap\_obj()  
 }, width = function() {  
 req(input$Heatmap\_download\_width)  
 return(input$Heatmap\_download\_width)  
 }, height = function() {  
 req(input$Heatmap\_download\_height)  
 return(input$Heatmap\_download\_height)  
 })  
   
 output$do\_download\_Heatmap <- downloadHandler(  
 filename = function() {  
 input$Heatmap\_download\_filename  
 },  
 content = function(file) {  
 req(heatmap\_obj())  
 plt <- heatmap\_obj()  
   
 ggsave(file, plot = plt,  
 width = input$Heatmap\_download\_width,  
 height = input$Heatmap\_download\_height,  
 units = "px",  
 dpi = 96)  
 }  
 )  
   
 合  
 observeEvent(input$draw\_tf\_heatmap, {  
 req(seurat\_obj(), input$target\_spiecies)  
   
 withProgress(message = "Running TF activity prediction...", value = 0, {  
   
 incProgress(0.1, message = "Loading TF network...")  
 net <- collectri\_net()  
   
   
 incProgress(0.2, message = "Extracting expression data...")  
   
 mat <- as.matrix(GetAssayData(seurat\_obj(), assay = "RNA", slot = "data"))  
   
   
 incProgress(0.5, message = "Inferring TF activities with decoupleR...")  
 acts <- run\_ulm(mat = mat, net = net, .source = 'source', .target = 'target',  
 .mor = 'mor', minsize = 5) # verbose = FALSE 引数を削除  
   
   
 incProgress(0.8, message = "Storing TF activities in Seurat object...")  
 tf\_assay\_data <- acts %>%  
 pivot\_wider(id\_cols = 'source', names\_from = 'condition',  
 values\_from = 'score') %>%  
 column\_to\_rownames('source') %>%  
 as.matrix()  
   
 temp\_seu <- seurat\_obj()  
 temp\_seu[['tfsulm']] <- CreateAssayObject(counts = tf\_assay\_data)  
   
 DefaultAssay(temp\_seu) <- "tfsulm"  
 temp\_seu <- Seurat::ScaleData(temp\_seu, features = rownames(tf\_assay\_data), verbose = FALSE)  
 temp\_seu@assays$tfsulm@data <- temp\_seu@assays$tfsulm@scale.data  
   
 seurat\_obj(temp\_seu)   
 tf\_activities(acts)   
   
   
 df\_long <- t(as.matrix(temp\_seu@assays$tfsulm@data)) %>%  
 as.data.frame() %>%  
 mutate(cluster = Idents(temp\_seu)) %>%  
 pivot\_longer(cols = -cluster, names\_to = "source", values\_to = "score") %>%  
 group\_by(cluster, source) %>%  
 summarise(mean = mean(score), .groups = 'drop')  
   
 tf\_activities\_df(df\_long)   
   
 incProgress(1, message = "TF Analysis Complete and Heatmap Generating...")  
 })  
 })  
   
   
 output$tfActivityHeatmap <- renderPlot({  
 req(tf\_activities\_df(), input$top\_n\_active\_TF)  
   
 n\_tfs\_val <- input$top\_n\_active\_TF  
 df\_long <- tf\_activities\_df()  
   
 tfs <- df\_long %>%  
 group\_by(source) %>%  
 summarise(std = sd(mean), .groups = 'drop') %>%  
 arrange(-abs(std)) %>%  
 head(n\_tfs\_val) %>%  
 pull(source)  
   
 top\_acts\_mat\_local <- df\_long %>%  
 filter(source %in% tfs) %>%  
 pivot\_wider(id\_cols = 'cluster', names\_from = 'source',  
 values\_from = 'mean') %>%  
 column\_to\_rownames('cluster') %>%  
 as.matrix()  
   
 if (!is.null(seurat\_obj())) {  
 ordered\_clusters <- sort(as.numeric(as.character(rownames(top\_acts\_mat\_local))))  
 top\_acts\_mat\_local <- top\_acts\_mat\_local[as.character(ordered\_clusters), , drop = FALSE]  
 }  
   
 tf\_pheatmap\_mat(top\_acts\_mat\_local)  
   
 palette\_length <- 100  
 my\_color <- colorRampPalette(c("Darkblue", "white","red"))(palette\_length)  
 max\_abs\_score <- max(abs(top\_acts\_mat\_local), na.rm = TRUE)  
 my\_breaks <- c(seq(-max\_abs\_score, 0, length.out = ceiling(palette\_length/2) + 1),  
 seq(0.001, max\_abs\_score, length.out = floor(palette\_length/2)))  
   
 pheatmap(top\_acts\_mat\_local, border\_color = NA,  
 color = my\_color,  
 breaks = my\_breaks,  
 angle\_col = 45,  
 cluster\_cols = TRUE,  
 cluster\_rows = FALSE,  
 cellwidth = 18,  
 cellheight = 18,  
 fontsize = 15,  
 fontsize\_row = 15,  
 main = paste0("TF Activity Heatmap (Top ", n\_tfs\_val, " TFs)")  
 )  
 })  
   
   
 observeEvent(input$preview\_download\_tf, {  
 req(tf\_pheatmap\_mat())  
 showModal(modalDialog(  
 title = "Download TF Activity Heatmap",  
 fluidRow(  
 column(9,  
 h4("Preview"),  
 plotOutput("tf\_preview\_heatmap\_plot", height = "auto")  
 ),  
 column(3,  
 h4("Settings"),  
 numericInput("tf\_download\_width\_px", "Width (pixels)", value = 600, min = 100),  
 numericInput("tf\_download\_height\_px", "Height (pixels)", value = 600, min = 100),  
 textInput("tf\_download\_filename", "File name", value = "TF\_Activity\_Heatmap.png"),  
 downloadButton("do\_download\_tf\_heatmap", "Download Heatmap")  
 )  
 ),  
 size = "l",  
 footer = modalButton("Close")  
 ))  
 })  
   
   
 output$tf\_preview\_heatmap\_plot <- renderPlot({  
 req(tf\_pheatmap\_mat())  
   
 mat\_to\_plot\_preview <- tf\_pheatmap\_mat()  
   
 palette\_length <- 100  
 my\_color <- colorRampPalette(c("Darkblue", "white","red"))(palette\_length)  
 max\_abs\_score <- max(abs(mat\_to\_plot\_preview), na.rm = TRUE)  
 my\_breaks <- c(seq(-max\_abs\_score, 0, length.out = ceiling(palette\_length/2) + 1),  
 seq(0.001, max\_abs\_score, length.out = floor(palette\_length/2)))  
   
 pheatmap(mat\_to\_plot\_preview, border\_color = NA,  
 color = my\_color,  
 breaks = my\_breaks,  
 angle\_col = 45,  
 cluster\_cols = TRUE,  
 cluster\_rows = FALSE,  
 cellwidth = 18,  
 cellheight = 18,  
 fontsize = 15,  
 fontsize\_row = 15,  
 main = paste0("TF Activity Heatmap (Top ", input$top\_n\_active\_TF, " TFs)")  
 )  
 }, width = function() {  
 req(input$tf\_download\_width\_px)  
 return(min(input$tf\_download\_width\_px, 800))  
 }, height = function() {  
 req(input$tf\_download\_height\_px)  
 return(min(input$tf\_download\_height\_px, 800))  
 })  
   
   
 output$do\_download\_tf\_heatmap <- downloadHandler(  
 filename = function() {  
 input$tf\_download\_filename  
 },  
 content = function(file) {  
 req(tf\_pheatmap\_mat())  
   
 mat\_to\_plot\_download <- tf\_pheatmap\_mat()  
   
 palette\_length <- 100  
 my\_color <- colorRampPalette(c("Darkblue", "white","red"))(palette\_length)  
 max\_abs\_score <- max(abs(mat\_to\_plot\_download), na.rm = TRUE)  
 my\_breaks <- c(seq(-max\_abs\_score, 0, length.out = ceiling(palette\_length/2) + 1),  
 seq(0.001, max\_abs\_score, length.out = floor(palette\_length/2)))  
   
 png(file, width = input$tf\_download\_width\_px, height = input$tf\_download\_height\_px, units = "px", res = 96)  
 pheatmap(mat\_to\_plot\_download, border\_color = NA,  
 color = my\_color,  
 breaks = my\_breaks,  
 angle\_col = 45,  
 cluster\_cols = TRUE,  
 cluster\_rows = FALSE,  
 cellwidth = 18,  
 cellheight = 18,  
 fontsize = 15,  
 fontsize\_row = 15,  
 main = paste0("TF Activity Heatmap (Top ", input$top\_n\_active\_TF, " TFs)")  
 )  
 dev.off()  
 }  
 )  
})