BNM Project

Disease: Multiple Sclerosis

Project Overview:

This project analyzes Multiple Sclerosis using network medicine approaches including:

1. Building interactomes (BioGRID, HuRI, STRING, Reactome)
2. Disease gene prioritization (DIAMOnD and Diffusion)
3. Functional enrichment analysis
4. Drug-gene interaction analysis
5. PROCONSUL comparison

Part 1: Interactome Construction

Setup and Configuration

Libraries for our implementation.

```
ensure_packages <- function(pkgs) {
  for (p in pkgs) {
    if (!requireNamespace(p, quietly = TRUE)) {
      install.packages(p, dependencies = TRUE)
    }
    suppressPackageStartupMessages(
      library(p, character.only = TRUE)
    )
  }
}
ensure_packages(c(
  "tidyverse", "igraph", "readr", "stringr", "tidyr", "Matrix", "biomaRt",
  "ggplot2", "reshape2"
))

if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")

bioc_pkgs <- c("clusterProfiler", "org.Hs.eg.db", "enrichplot", "ReactomePA")
for (p in bioc_pkgs) {
  if (!requireNamespace(p, quietly = TRUE)) BiocManager::install(p, ask = FALSE, update = FALSE)
  library(p, character.only = TRUE)
}
```

We define the directory where we collect the main outputs of this first part.

```
# User paths
PATH_STRING_LINKS   <- "9606.protein.links.v12.0 STRING.txt.txt"
PATH_REACTOME_FI    <- "FIsInGene_04142025_with_annotations_REACTOME.txt"
PATH_BIOGRID_TAB3   <- "BIOGRID-ORGANISM-Homo_sapiens-5.0.252.tab3.txt"
PATH_HURI_2COL      <- "HuRI.tsv"
PATH_GEDIPNET_DISEASE <- "diseases_"
```

```r
OUTDIR <- "PART1_outputs"
dir.create(OUTDIR, showWarnings = FALSE, recursive = TRUE)
```

Helper Functions

Here, we define functions for constructing reactomes.

```r
# Deduplicate an undirected edge list and remove self-loops
dedup_undirected_edges <- function(df, from = "from", to = "to") {
  df %>%
    dplyr::select(from = all_of(from), to = all_of(to)) %>%
    mutate(from = as.character(from), to = as.character(to)) %>%
    filter(!is.na(from), !is.na(to), from != "", to != "") %>%
    filter(from != to) %>%
    mutate(node_min = pmin(from, to), node_max = pmax(from, to)) %>%
    distinct(node_min, node_max) %>%
    transmute(from = node_min, to = node_max)
}
```

```r
# Build graph and extract LCC
graph_lcc <- function(edge_df, directed = FALSE) {
  g <- igraph::graph_from_data_frame(edge_df, directed = directed)
  comp <- igraph::components(g)
  giant_id <- which.max(comp$csize)
  nodes_lcc <- igraph::V(g)[comp$membership == giant_id]
  igraph::induced_subgraph(g, nodes_lcc)
}
```

```r
# Robust Ensembl connection (mirrors)
connect_ensembl <- function() {
  mirrors <- c("www", "useast", "uswest", "asia")
  for (m in mirrors) {
    message("Connecting to Ensembl mirror: ", m)
    out <- try(
      biomaRt::useEnsembl(biomart="ensembl", dataset="hsapiens_gene_ensembl", mirror=m),
      silent = TRUE
    )
    if (!inherits(out, "try-error")) return(out)
  }
  stop("Could not connect to any Ensembl mirror. Try later or change network/VPN.")
}
```

```r
# Map ENSP -> HGNC (STRING)
map_ensp_to_hgnc <- function(ensp_vec, mart) {
  ensp_vec <- unique(ensp_vec[!is.na(ensp_vec) & ensp_vec != ""])
  if (length(ensp_vec) == 0) return(tibble(ensembl_peptide_id = character(), hgnc_symbol = character()))
  biomaRt::getBM(
    attributes = c("ensembl_peptide_id", "hgnc_symbol"),
    filters    = "ensembl_peptide_id",
    values     = ensp_vec,
    mart       = mart
  ) %>% as_tibble()
}
```

```r
# Map ENSG -> HGNC (HuRI)
map_ensg_to_hgnc <- function(ensg_vec, mart) {
  ensg_vec <- unique(ensg_vec[!is.na(ensg_vec) & ensg_vec != ""])
  if (length(ensg_vec) == 0) return(tibble(ensembl_gene_id = character(), hgnc_symbol = character()))
  biomaRt::getBM(
    attributes = c("ensembl_gene_id", "hgnc_symbol"),
    filters    = "ensembl_gene_id",
    values     = ensg_vec,
    mart       = mart
  ) %>% as_tibble()
}
```

```r
# Parse GeDiPNet "Associated genes" robustly
read_seed_genes_gedipnet <- function(path, col = "Associated genes") {
  df <- readr::read_delim(path, delim = "\t", show_col_types = FALSE)
  if (!(col %in% names(df))) {
    stop("Column '", col, "' not found. Columns are: ", paste(names(df), collapse = ", "))
  }

  seeds <- df %>%
    dplyr::select(all_of(col)) %>%
    mutate(across(everything(), as.character)) %>%
    tidyr::separate_rows(all_of(col), sep = "[,;\\s]+") %>%
    mutate(gene = stringr::str_trim(.data[[col]])) %>%
    filter(!is.na(gene), gene != "") %>%
    distinct(gene) %>%
    pull(gene)

  seeds
}
```

```r
hgnc_check <- function(symbols) {
  symbols <- unique(symbols)
  if (!requireNamespace("HGNChelper", quietly = TRUE)) {
    return(list(final = symbols, report = tibble(Symbol = symbols, Approved = NA)))
  }
  chk <- HGNChelper::checkGeneSymbols(symbols)
  final <- ifelse(is.na(chk$Suggested.Symbol) | chk$Suggested.Symbol == "", chk$x, chk$Suggested.Symbol
  report <- tibble(
    Symbol = chk$x,
    Approved = chk$Approved,
    Suggested = chk$Suggested.Symbol
  )
  list(final = unique(final), report = report)
}
```

```r
# Table 1
tab1_row <- function(g_lcc, seed_genes, name) {
  node_names <- igraph::V(g_lcc)$name
  seeds_present <- intersect(seed_genes, node_names)

  g_dis <- igraph::induced_subgraph(g_lcc, seeds_present)

  dis_lcc_size <- 0L
```

```
  if (igraph::vcount(g_dis) > 0 && igraph::ecount(g_dis) > 0) {
    comp <- igraph::components(g_dis)
    giant_id <- which.max(comp$csize)
    g_dis_lcc <- igraph::induced_subgraph(g_dis, igraph::V(g_dis)[comp$membership == giant_id])
    dis_lcc_size <- igraph::vcount(g_dis_lcc)
  } else if (igraph::vcount(g_dis) > 0) {
    dis_lcc_size <- 1L
  }

  tibble(
    Interactome = name,
    Nodes_LCC = igraph::vcount(g_lcc),
    Links_LCC = igraph::ecount(g_lcc),
    DiseaseGenes_Present = length(seeds_present),
    DiseaseGenes_Percent = round(100 * length(seeds_present) / length(seed_genes), 2),
    DiseaseInteractome_LCC_Size = dis_lcc_size
  )
}
```

Build Interactomes

```
mart <- connect_ensembl()
```

STRING Interactome

```
string_raw <- readr::read_delim(
  PATH_STRING_LINKS,
  delim = " ",
  trim_ws = TRUE,
  show_col_types = FALSE
)

stopifnot(all(c("protein1", "protein2") %in% names(string_raw)))

string_ensp <- string_raw %>%
  transmute(
    from_ensp = stringr::str_remove(as.character(protein1), "^9606\\."),
    to_ensp   = stringr::str_remove(as.character(protein2), "^9606\\.")
  ) %>%
  filter(from_ensp != to_ensp)

all_ensp <- unique(c(string_ensp$from_ensp, string_ensp$to_ensp))
map_string <- map_ensp_to_hgnc(all_ensp, mart)

string_mapped <- string_ensp %>%
  left_join(map_string, by = c("from_ensp" = "ensembl_peptide_id")) %>%
  dplyr::rename(from_symbol = hgnc_symbol) %>%
  left_join(map_string, by = c("to_ensp" = "ensembl_peptide_id")) %>%
  dplyr::rename(to_symbol = hgnc_symbol) %>%
  filter(!is.na(from_symbol), from_symbol != "",
         !is.na(to_symbol),   to_symbol   != "")
```

```r
string_edges <- dedup_undirected_edges(string_mapped %>% transmute(from = from_symbol, to = to_symbol))
g_string_lcc <- graph_lcc(string_edges)

cat(
  "STRING LCC nodes:", vcount(g_string_lcc),
  " | edges:", ecount(g_string_lcc),
  "\n",
  sep = " "
)
```

```
## STRING LCC nodes: 18920  | edges: 6566519
```

Reactome Interactome

```r
reactome_raw <- readr::read_tsv(PATH_REACTOME_FI, show_col_types = FALSE)

stopifnot(all(c("Gene1", "Gene2") %in% names(reactome_raw)))

reactome_edges <- dedup_undirected_edges(
  reactome_raw %>% transmute(from = Gene1, to = Gene2)
)
g_reactome_lcc <- graph_lcc(reactome_edges)

cat(
  "Reactome LCC nodes:", vcount(g_reactome_lcc),
  " | edges:", ecount(g_reactome_lcc),
  "\n",
  sep = " "
)
```

```
## Reactome LCC nodes: 13262  | edges: 272298
```

BioGRID Interactome

```r
biogrid_raw <- readr::read_tsv(PATH_BIOGRID_TAB3, show_col_types = FALSE)

required_cols_bg <- c(
  "Organism ID Interactor A", "Organism ID Interactor B",
  "Experimental System Type",
  "Official Symbol Interactor A", "Official Symbol Interactor B"
)
stopifnot(all(required_cols_bg %in% names(biogrid_raw)))

biogrid_edges <- biogrid_raw %>%
  filter(`Organism ID Interactor A` == 9606,
         `Organism ID Interactor B` == 9606) %>%
  filter(`Experimental System Type` == "physical") %>%
  transmute(from = `Official Symbol Interactor A`,
            to   = `Official Symbol Interactor B`)

biogrid_edges <- dedup_undirected_edges(biogrid_edges)
g_biogrid_lcc <- graph_lcc(biogrid_edges)

cat(
```

```
  "BioGRID LCC nodes:", vcount(g_biogrid_lcc),
  " | edges:", ecount(g_biogrid_lcc),
  "\n",
  sep = " "
)
```

```
## BioGRID LCC nodes: 20182  | edges: 907435
```

HuRI Interactome

```
huri_raw <- readr::read_tsv(PATH_HURI_2COL, col_names = c("from", "to"), show_col_types = FALSE)

huri_raw <- huri_raw %>%
  mutate(from = as.character(from), to = as.character(to)) %>%
  filter(!is.na(from), !is.na(to), from != "", to != "") %>%
  filter(str_detect(from, "^ENSG"), str_detect(to, "^ENSG"))

all_ensg <- unique(c(huri_raw$from, huri_raw$to))
map_huri <- map_ensg_to_hgnc(all_ensg, mart)

huri_mapped <- huri_raw %>%
  left_join(map_huri, by = c("from" = "ensembl_gene_id")) %>%
  dplyr::rename(from_symbol = hgnc_symbol) %>%
  left_join(map_huri, by = c("to" = "ensembl_gene_id")) %>%
  dplyr::rename(to_symbol = hgnc_symbol) %>%
  filter(!is.na(from_symbol), from_symbol != "",
         !is.na(to_symbol),   to_symbol   != "")

huri_edges <- dedup_undirected_edges(huri_mapped %>% transmute(from = from_symbol, to = to_symbol))
g_huri_lcc <- graph_lcc(huri_edges)

cat(
  "HuRI LCC nodes:", vcount(g_huri_lcc),
  " | edges:", ecount(g_huri_lcc),
  "\n",
  sep = " "
)
```

```
## HuRI LCC nodes: 8938  | edges: 62778
```

```
# Bundle interactomes
interactomes_lcc <- list(
  BioGRID  = g_biogrid_lcc,
  HuRI     = g_huri_lcc,
  STRING   = g_string_lcc,
  Reactome = g_reactome_lcc
)
```

Load Disease Genes

```
seed_genes_raw <- read_seed_genes_gedipnet(PATH_GEDIPNET_DISEASE, col = "Associated genes")
message("Parsed seed genes: ", length(seed_genes_raw))

chk <- hgnc_check(seed_genes_raw)
```

```r
seed_genes <- chk$final

if (!is.null(chk$report)) {
  write_csv(chk$report, file.path(OUTDIR, "HGNC_check_report.csv"))
  message("Saved HGNC check report")
}

cat(
  "Final seed genes after optional correction:", length(seed_genes),
  "\n",
  sep = " "
)
```

```
## Final seed genes after optional correction: 293
```

Table 1: Interactome Summary

```r
table1 <- bind_rows(lapply(names(interactomes_lcc), function(nm) {
  tab1_row(interactomes_lcc[[nm]], seed_genes, nm)
}))

write_csv(table1, file.path(OUTDIR, "TABLE1_interactome_summary.csv"))
print(table1)
```

```
## # A tibble: 4 x 6
##   Interactome Nodes_LCC Links_LCC DiseaseGenes_Present DiseaseGenes_Percent
##   <chr>           <dbl>     <dbl>                <int>                <dbl>
## 1 BioGRID         20182    907435                  247                 84.3
## 2 HuRI             8938     62778                  112                 38.2
## 3 STRING          18920   6566519                  239                 81.6
## 4 Reactome        13262    272298                  202                 68.9
## # i 1 more variable: DiseaseInteractome_LCC_Size <dbl>
```

Table 2

```r
# ------------------------------------------------------------------------
# Table 2: Disease LCC centrality metrics (ALL interactomes)
# ------------------------------------------------------------------------

OUT2B <- "PART1_outputs/table2_centrality"
dir.create(OUT2B, showWarnings = FALSE, recursive = TRUE)

table2_results <- list()
table2_top20   <- list()

for (net_name in names(interactomes_lcc)) {

  cat(
  "Computing Table 2 for interactome:", net_name,
  "\n",
  sep = " ")
```

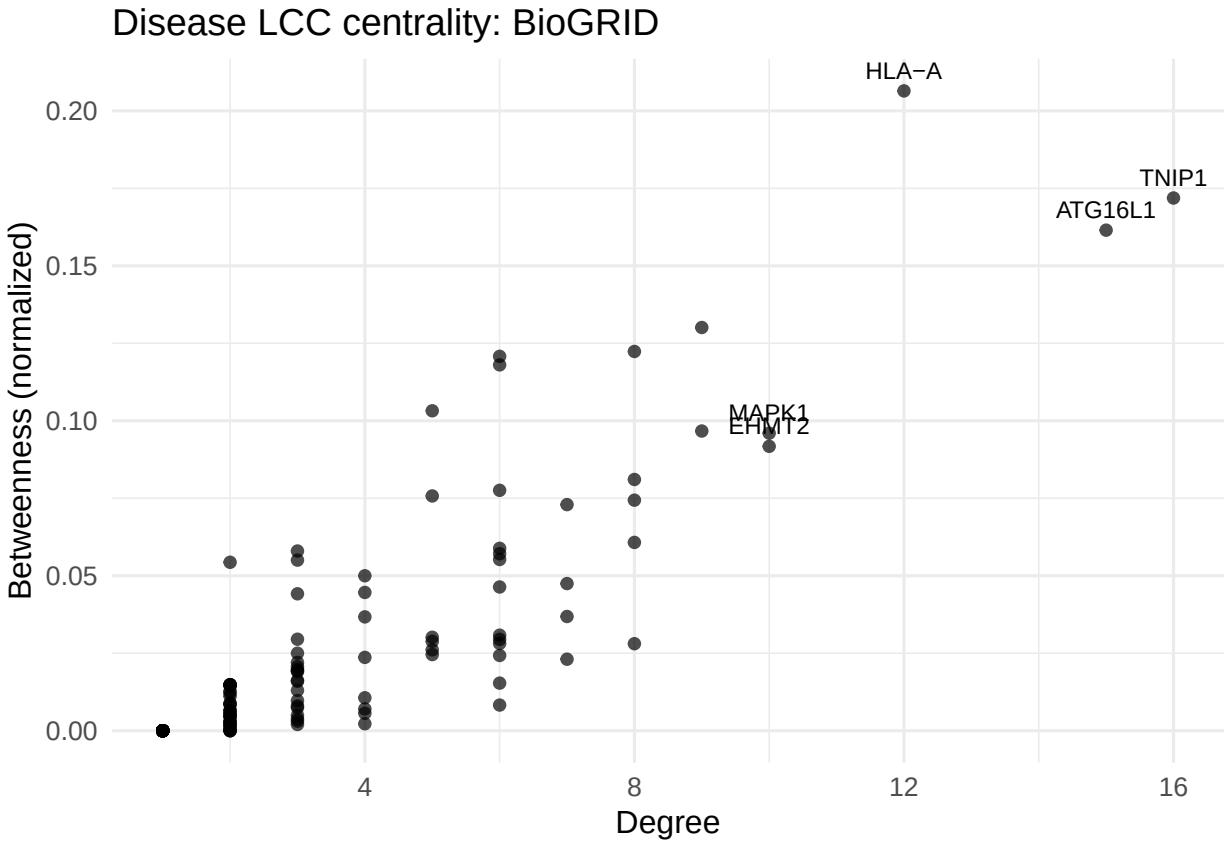
```r
g <- interactomes_lcc[[net_name]]

# Disease genes present in this interactome LCC
seeds_present <- intersect(seed_genes, V(g)$name)

if (length(seeds_present) == 0) {
  warning("No disease genes present in ", net_name, " LCC. Skipping.")
  next
}

# Disease subgraph
g_disease <- induced_subgraph(g, seeds_present)

if (vcount(g_disease) == 0 || ecount(g_disease) == 0) {
  warning("Disease subgraph empty for ", net_name, ". Skipping.")
  next
}

# Disease LCC
comp_dis <- components(g_disease)
giant_id <- which.max(comp_dis$csize)
g_disease_lcc <- induced_subgraph(
  g_disease,
  V(g_disease)[comp_dis$membership == giant_id]
)

cat(
"  Disease LCC nodes=", vcount(g_disease_lcc),
" edges=", ecount(g_disease_lcc),
"\n",
sep = "")


# Centrality metrics
centrality_tbl <- tibble(
  Interactome = net_name,
  Gene_name   = V(g_disease_lcc)$name,
  Degree      = degree(g_disease_lcc),
  Betweenness = betweenness(g_disease_lcc, directed = FALSE, normalized = TRUE),
  Eigenvector = eigen_centrality(g_disease_lcc)$vector,
  Closeness   = closeness(g_disease_lcc, normalized = TRUE)
) %>%
  mutate(
    ratio_Betw_Degree = ifelse(Degree > 0, Betweenness / Degree, NA_real_)
  ) %>%
  arrange(desc(Degree), desc(Betweenness))

# Save full Table 2
write_csv(
  centrality_tbl,
  file.path(OUT2B, paste0("TABLE2_centrality_all_", net_name, ".csv"))
)
```

```r
  # Top-20 by degree
  top20_tbl <- centrality_tbl %>% slice_head(n = 20)
  print(top20_tbl)
  write_csv(
    top20_tbl,
    file.path(OUT2B, paste0("TABLE2_top20_", net_name, ".csv"))
  )

  table2_results[[net_name]] <- centrality_tbl
  table2_top20[[net_name]]   <- top20_tbl

  # Scatter plot: Degree vs Betweenness
  p <- ggplot(centrality_tbl, aes(x = Degree, y = Betweenness)) +
    geom_point(alpha = 0.7) +
    geom_text(
      data = centrality_tbl %>% slice_max(Degree, n = 5),
      aes(label = Gene_name),
      vjust = -0.7,
      size = 3
    ) +
    theme_minimal(base_size = 12) +
    labs(
      title = paste0("Disease LCC centrality: ", net_name),
      x = "Degree",
      y = "Betweenness (normalized)"
    )
  ggplot(centrality_tbl, aes(x = Degree, y = Betweenness)) +
    geom_point(alpha = 0.7) +
    geom_text(
      data = centrality_tbl %>% slice_max(Degree, n = 5),
      aes(label = Gene_name),
      vjust = -0.7,
      size = 3
    ) +
    theme_minimal(base_size = 12) +
    labs(
      title = paste0("Disease LCC centrality: ", net_name),
      x = "Degree",
      y = "Betweenness (normalized)"
    )
  ggsave(
    file.path(OUT2B, paste0("FIG_table2_scatter_degree_betweenness_", net_name, ".png")),
    p,
    width = 9,
    height = 6,
    dpi = 300
  )

  print(p)
}
```

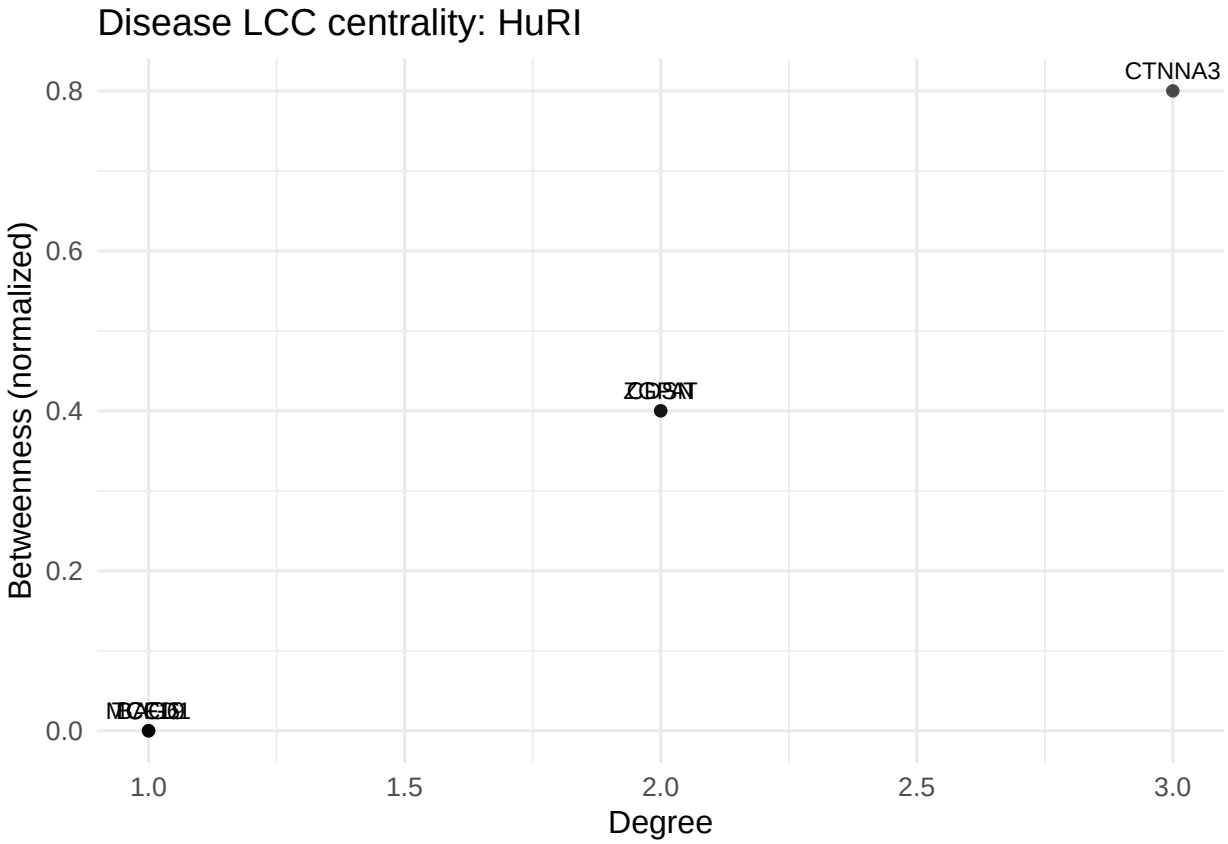
```
## Computing Table 2 for interactome: BioGRID
##   Disease LCC nodes=136 edges=227
## # A tibble: 20 x 7
```

```
##    Interactome Gene_name Degree Betweenness Eigenvector Closeness
##    <chr>       <chr>      <dbl>       <dbl>       <dbl>     <dbl>
##  1 BioGRID     TNIP1         16      0.172       1          0.347
##  2 BioGRID     ATG16L1       15      0.162       0.515      0.322
##  3 BioGRID     HLA-A         12      0.206       0.701      0.341
##  4 BioGRID     MAPK1         10      0.0960      0.627      0.324
##  5 BioGRID     EHMT2         10      0.0918      0.150      0.277
##  6 BioGRID     CDC37          9      0.130       0.514      0.324
##  7 BioGRID     STAT3          9      0.0967      0.496      0.319
##  8 BioGRID     MCAM           8      0.122       0.311      0.305
##  9 BioGRID     TRAF3          8      0.0811      0.497      0.312
## 10 BioGRID     IQGAP1         8      0.0744      0.517      0.318
## 11 BioGRID     BAG6           8      0.0608      0.323      0.299
## 12 BioGRID     HLA-DQB1       8      0.0281      0.456      0.273
## 13 BioGRID     ERBB3          7      0.0730      0.425      0.328
## 14 BioGRID     HLA-B          7      0.0475      0.559      0.307
## 15 BioGRID     NLRP3          7      0.0369      0.203      0.265
## 16 BioGRID     HLA-DRB1       7      0.0231      0.275      0.239
## 17 BioGRID     SLC30A7        6      0.121       0.215      0.289
## 18 BioGRID     RBPJ           6      0.118       0.0586     0.259
## 19 BioGRID     ITPR3          6      0.0776      0.397      0.308
## 20 BioGRID     ICAM1          6      0.0589      0.0726     0.247
## # i 1 more variable: ratio_Betw_Degree <dbl>
```

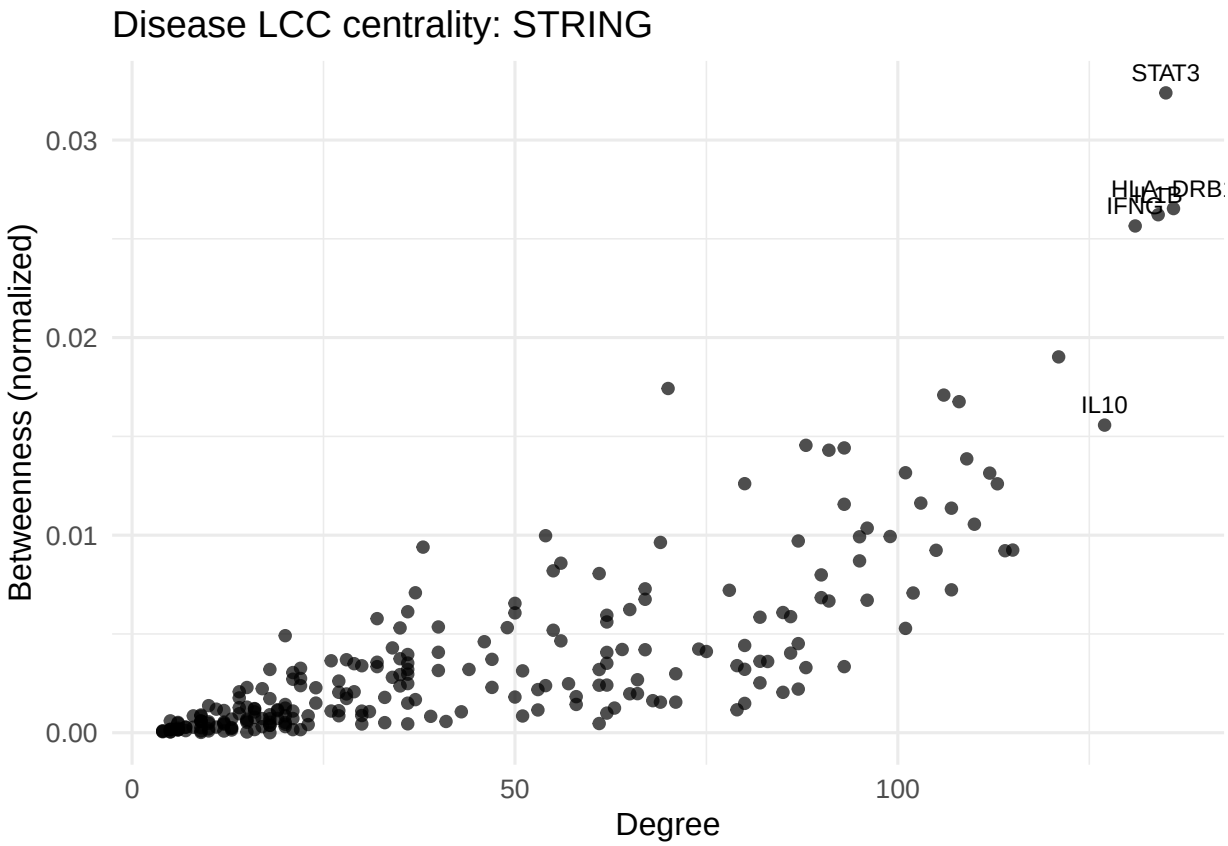
```
## Computing Table 2 for interactome: HuRI
##   Disease LCC nodes=6 edges=5
## # A tibble: 6 x 7
##   Interactome Gene_name Degree Betweenness Eigenvector Closeness
##   <chr>       <chr>      <dbl>       <dbl>       <dbl>     <dbl>
## 1 HuRI        CTNNA3         3         0.8       1         0.714
## 2 HuRI        CDSN           2         0.4       0.707     0.556
## 3 HuRI        ZGPAT          2         0.4       0.707     0.556
## 4 HuRI        BAG6           1         0         0.366     0.385
## 5 HuRI        TCF19          1         0         0.366     0.385
## 6 HuRI        MCCD1          1         0         0.518     0.455
## # i 1 more variable: ratio_Betw_Degree <dbl>
```

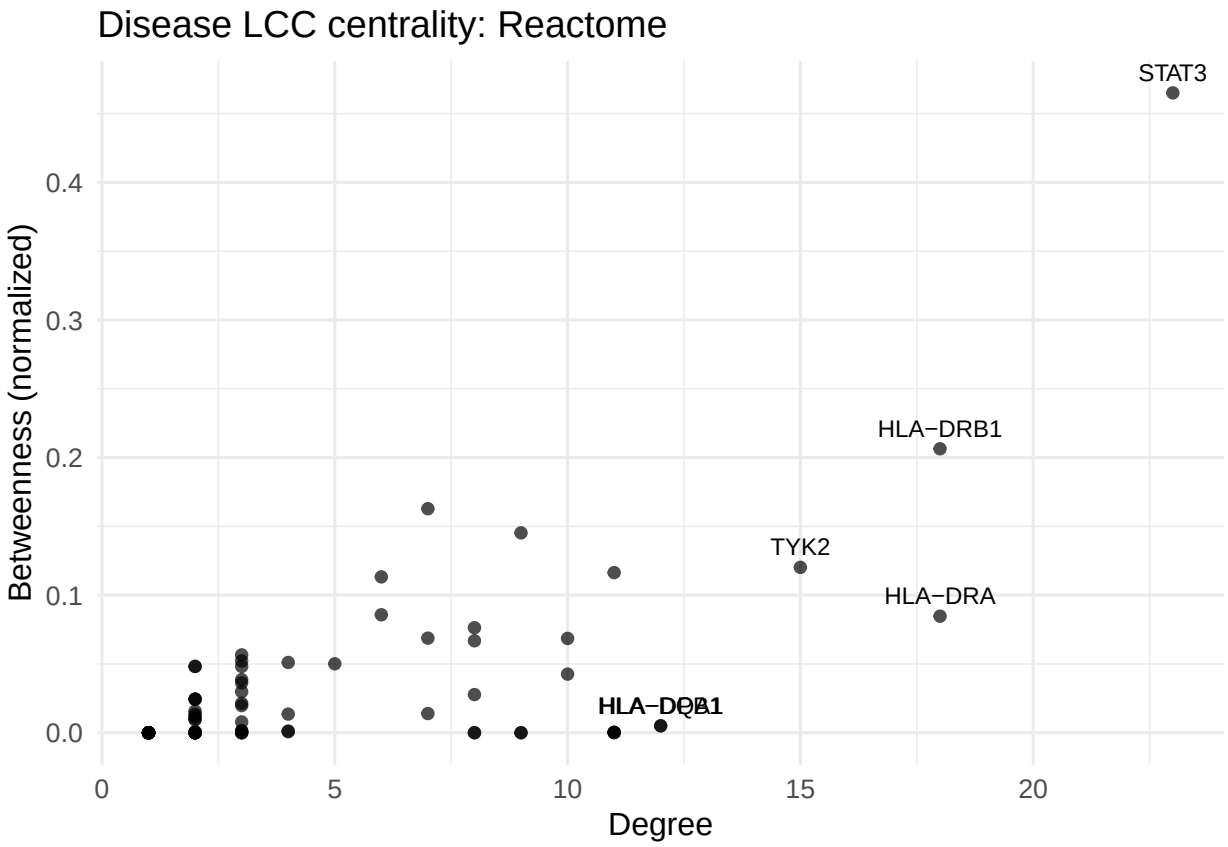



```
## Computing Table 2 for interactome: STRING
##   Disease LCC nodes=238 edges=5396
## # A tibble: 20 x 7
##    Interactome Gene_name Degree Betweenness Eigenvector Closeness
##    <chr>       <chr>      <dbl>       <dbl>       <dbl>     <dbl>
##  1 STRING      HLA-DRB1     136     0.0265        0.983     0.701
##  2 STRING      STAT3        135     0.0324        0.975     0.699
##  3 STRING      IL1B         134     0.0262        0.985     0.697
##  4 STRING      IFNG         131     0.0257        1         0.691
##  5 STRING      IL10         127     0.0156        0.996     0.683
##  6 STRING      PTPN22       121     0.0190        0.932     0.666
##  7 STRING      STAT4        115     0.00925       0.952     0.655
##  8 STRING      IL2RA        114     0.00921       0.948     0.655
##  9 STRING      IRF8         113     0.0126        0.921     0.657
## 10 STRING      TNFAIP3      112     0.0131        0.909     0.651
## 11 STRING      ITGAM        110     0.0106        0.932     0.648
## 12 STRING      HLA-DQB1     109     0.0139        0.882     0.648
## 13 STRING      TYK2         108     0.0168        0.857     0.644
## 14 STRING      ICAM1        107     0.0114        0.885     0.641
## 15 STRING      CD40         107     0.00723       0.928     0.641
## 16 STRING      IL7R         106     0.0171        0.905     0.642
## 17 STRING      HLA-B        105     0.00923       0.879     0.637
## 18 STRING      IFIH1        103     0.0116        0.862     0.634
## 19 STRING      IL17A        102     0.00707       0.883     0.634
## 20 STRING      VCAM1        101     0.0132        0.835     0.629
## # i 1 more variable: ratio_Betw_Degree <dbl>
```




```
## Computing Table 2 for interactome: Reactome
##   Disease LCC nodes=83 edges=197
## # A tibble: 20 x 7
##    Interactome Gene_name Degree Betweenness Eigenvector Closeness
##    <chr>       <chr>      <dbl>       <dbl>       <dbl>     <dbl>
##  1 Reactome    STAT3         23    0.465         0.292      0.471
##  2 Reactome    HLA-DRB1      18    0.206         1          0.412
##  3 Reactome    HLA-DRA       18    0.0847        0.984      0.363
##  4 Reactome    TYK2          15    0.120         0.369      0.427
##  5 Reactome    HLA-DPB1      12    0.00502       0.854      0.322
##  6 Reactome    HLA-DQA1      12    0.00502       0.854      0.322
##  7 Reactome    MAPK1         11    0.116         0.0870     0.392
##  8 Reactome    HLA-DQA2      11    0.000188      0.819      0.306
##  9 Reactome    HLA-DQB1      11    0.000188      0.819      0.306
## 10 Reactome    HLA-DQB2      11    0.000188      0.819      0.306
## 11 Reactome    HLA-DRB4      11    0.000188      0.819      0.306
## 12 Reactome    IL12A         10    0.0686        0.340      0.416
## 13 Reactome    STAT4         10    0.0426        0.356      0.392
## 14 Reactome    IL2RB          9    0.145         0.145      0.385
## 15 Reactome    HLA-DMA        9    0             0.710      0.304
## 16 Reactome    HLA-DMB        9    0             0.710      0.304
## 17 Reactome    HLA-A          8    0.0763        0.139      0.340
## 18 Reactome    CD86           8    0.0668        0.409      0.349
## 19 Reactome    IL2RA          8    0.0277        0.198      0.387
## 20 Reactome    PDCD1          8    0             0.645      0.303
## # i 1 more variable: ratio_Betw_Degree <dbl>
```


Disease LCC centrality: STRING
STAT3
IL10
0.03
0.02
0.01
0.00
Betweenness (normalized)
0
50
100
Degree



Part 2: Gene Prioritization

Configuration

```
set.seed(123)
N_FOLDS <- 5
TIMES <- c(0.1, 1.0, 2.0, 5.0, 10.0)
DIFF_ITERS <- 20

python_exe <- "C:/Users/Utente/AppData/Local/Programs/Python/Python314/python.exe"

dir.create("PART2_outputs", showWarnings = FALSE, recursive = TRUE)
dir.create("PART2_outputs/cv_diamond", showWarnings = FALSE, recursive = TRUE)
dir.create("PART2_outputs/cv_diffusion", showWarnings = FALSE, recursive = TRUE)

stopifnot(file.exists("DIAMOnD.py"))
stopifnot(file.exists(python_exe))
```

Performance Metrics

```
score_ranking <- function(ranked_genes, probe_set, K) {
  ranked_genes <- ranked_genes[!is.na(ranked_genes) & ranked_genes != ""]
  K_eff <- min(K, length(ranked_genes))
  if (K_eff <= 0) return(tibble(Precision = NA_real_, Recall = NA_real_, F1 = NA_real_, Hits = 0L, K_ef
```

```
  topK <- ranked_genes[seq_len(K_eff)]
  hits <- sum(topK %in% probe_set)

  precision <- hits / K_eff
  recall <- if (length(probe_set) == 0) 0 else hits / length(probe_set)
  f1 <- if ((precision + recall) == 0) 0 else 2 * precision * recall / (precision + recall)

  tibble(Precision = precision, Recall = recall, F1 = f1, Hits = hits, K_eff = K_eff)
}

make_K_values <- function(n) {
K <- unique(round(c(50, n/10, n/4, n/2, n)))
sort(K[K > 0])
}
```

Diffusion Functions

In this section, we implement a **network diffusion (heat diffusion) model** to propagate disease information from known seed genes across the interactome. The underlying idea is that genes that are *topologically close* to disease genes in the network are more likely to be functionally related to the disease.

We model diffusion using the **random-walk normalized Laplacian**, $L_{rw} = I - D^{-1}A$, where A is the adjacency matrix and D the degree matrix. This formulation corresponds to a **random walk on the network**, which is well suited for biological networks with heterogeneous node degrees.

The continuous diffusion process

$$\frac{dp}{dt} = -L_{rw}p$$

is approximated numerically via an **explicit Euler scheme**, leading to the update rule

$$p_{k+1} = (1 - \Delta t)\, p_k + \Delta t\, W p_k,$$

where $W = D^{-1}A$ is the row-stochastic transition matrix.

The function `prepare_diffusion()` precomputes this matrix once for efficiency, while `run_diffusion()` initializes heat on the seed genes, iteratively propagates it for a total diffusion time t, and returns a ranked list of genes based on their final heat scores.

We normalize the initial heat vector to make it probability-like and explicitly **remove seed genes from the final ranking**, ensuring a fair evaluation of candidate genes.

```
## Diffusion Functions (HEAT DIFFUSION / NETWORK PROPAGATION)

# We implement heat diffusion using the random-walk normalized Laplacian:
#   L_rw = I - D^{-1} A  where W = D^{-1}A is the row-stochastic walk matrix.
# Heat diffusion solves:  dp/dt = - L_rw p
# Discretized (explicit Euler): p_{k+1} = p_k - dt * L_rw p_k
#                             = (1 - dt) p_k + dt * W p_k
#
# Here, the diffusion "time" t controls how far we propagate:
# total simulated time = n_steps * dt   t

prepare_diffusion <- function(g) {
  A <- igraph::as_adjacency_matrix(g, sparse = TRUE)
  d <- igraph::degree(g)
```

```
  d_inv <- ifelse(d > 0, 1 / d, 0)
  W <- Matrix::Diagonal(x = d_inv) %*% A  # row-stochastic random-walk matrix
  list(W = W, nodes = igraph::V(g)$name)
}

run_diffusion <- function(prep, seeds, t, dt = 0.01) {
  nodes <- prep$nodes
  W <- prep$W
  n <- length(nodes)

  # seeds restricted to graph
  seeds <- intersect(seeds, nodes)

  # initial heat: 1 on seeds, 0 elsewhere; normalize to sum to 1 (probability-like)
  p <- numeric(n); names(p) <- nodes
  p[seeds] <- 1
  if (sum(p) > 0) p <- p / sum(p)

  # number of Euler steps so that total diffusion time approx equals t
  t <- as.numeric(t)
  if (is.na(t) || t < 0) stop("t must be a non-negative number")
  dt <- as.numeric(dt)
  if (is.na(dt) || dt <= 0 || dt > 1) stop("dt must be in (0, 1] for stability")

  n_steps <- max(1L, as.integer(ceiling(t / dt)))

  # Heat diffusion iterations
  p_mat <- as.matrix(p)
  for (k in seq_len(n_steps)) {
    p_mat <- (1 - dt) * p_mat + dt * (W %*% p_mat)
  }

  scores <- as.vector(p_mat); names(scores) <- nodes


  scores[seeds] <- -Inf

  sort(scores, decreasing = TRUE)
}
```

DIAMOnD Functions

```
run_diamond <- function(netfile, seeds, max_needed, outpath, python_exe) {
  write.table(seeds, "current_seeds.txt", row.names = FALSE, col.names = FALSE, quote = FALSE)

  cmd <- paste(
    shQuote(python_exe),
    shQuote("DIAMOnD.py"),
    shQuote(netfile),
    shQuote("current_seeds.txt"),
    as.character(max_needed),
    "1",
    shQuote(outpath)
```

```
  )

  system(cmd, intern = FALSE, ignore.stdout = TRUE, ignore.stderr = FALSE)

  if (!file.exists(outpath)) {
    stop("DIAMOnD failed: output not created at ", outpath)
  }
}

read_diamond_rank <- function(path) {
  lines <- readLines(path, warn = FALSE)
  lines <- lines[!grepl("^\\s*#", lines)]
  lines <- lines[nzchar(lines)]
  parts <- strsplit(lines, "\\s+")
  genes <- vapply(parts, function(x) if (length(x) >= 2) x[[2]] else NA_character_, character(1))
  genes <- genes[!is.na(genes) & genes != ""]
  genes
}
```

Cross-Validation for Gene Prioritization (DIAMOnD vs Diffusion)

This chunk implements a K-fold cross-validation (CV) framework to evaluate and compare our two gene prioritization methods—DIAMOnD and network diffusion—across multiple interactomes (BioGRID, HuRI, STRING, Reactome). It is marked `eval=FALSE` because it is computationally expensive; the corresponding results are stored on disk and can be reloaded without rerunning the full CV.

For each interactome LCC, we first restrict the disease seed list to genes that actually appear in that network (`seeds_present`). If too few seeds are available (here, `< 10`), we skip the network because CV estimates would be unstable and uninformative.

We then define a set of Top-K thresholds (`K_values`) as a function of the available seed count. This avoids using an arbitrary single K and allows us to study performance at multiple operating points. For DIAMOnD, we ensure the Python implementation outputs at least `max(K_values)` candidates (`diamond_iters_needed`), otherwise precision/recall at large K would be artificially capped by output length.

Within each of the `N_FOLDS` folds, we split the seeds into: - `train`: the seeds given to the algorithm (known disease genes), - `probe`: held-out seeds used as ground truth for evaluation.

For each fold we:

1. Run DIAMOnD (or reuse the cached output file), read the ranked list, and remove training seeds from the ranking to prevent leakage into evaluation.
2. Run diffusion for each diffusion time `t` in `TIMES`, producing a score-based ranking; we similarly remove training seeds and optionally save ranked scores to disk for reproducibility.

Finally, at each `K` we compute Precision, Recall, F1, and Hits using `score_ranking()` and store one row per `(network, method, fold, parameter, K)`. The final dataset `results_all` is obtained by row-binding all collected results and is later summarized (mean ± SD over folds) to compare methods fairly and consistently.

```
# Results are saved in PART2_outputs/cv_all_folds_results.csv
all_rows <- list()
row_i <- 1

for (net_name in names(interactomes_lcc)) {

  message("Processing: ", net_name)
```

```r
g <- interactomes_lcc[[net_name]]
universe <- V(g)$name

seeds_present <- intersect(seed_genes, universe)
n <- length(seeds_present)

if (n < 10) {
  warning("Too few disease genes in LCC for ", net_name)
  next
}

K_values = make_K_values(n)

# DIAMOnD must output at least n candidates (module expansion length)
diamond_iters_needed <- max(K_values)


prep <- prepare_diffusion(g)

shuffled <- sample(seeds_present)
fold_id <- cut(seq_along(shuffled), breaks = N_FOLDS, labels = FALSE)

netfile <- paste0(net_name, "_network.txt")
if (!file.exists(netfile)) stop("Missing DIAMOnD network file: ", netfile)

for (f in seq_len(N_FOLDS)) {

  probe <- shuffled[fold_id == f]
  train <- shuffled[fold_id != f]

  # DIAMOnD
  diamond_out <- file.path("PART2_outputs/cv_diamond", paste0("diamond_", net_name, "_fold", f, ".txt

  if (!file.exists(diamond_out)) {
    run_diamond(netfile, train, diamond_iters_needed, diamond_out, python_exe)
  }

  diamond_rank <- read_diamond_rank(diamond_out)
  diamond_rank <- setdiff(diamond_rank, train)

  if (length(diamond_rank) < max(K_values)) {
    warning("DIAMOnD produced only ", length(diamond_rank),
            " candidates but max(K)=", max(K_values),
            " for net=", net_name, " fold=", f,
            ". Consider increasing diamond_iters_needed.")
  }

  for (K in K_values) {
    perf <- score_ranking(diamond_rank, probe, K)
    all_rows[[row_i]] <- tibble(
      Network = net_name,
      Algo = "DIAMOnD",
      Parameter = NA_real_,
```

```
        Fold = f,
        TopK = K,
        Precision = perf$Precision,
        Recall = perf$Recall,
        F1 = perf$F1,
        Hits = perf$Hits
      )
      row_i <- row_i + 1
}

    # Diffusion
    for (t_val in TIMES) {

      diff_scores <- run_diffusion(prep, train, t = t_val, dt = 0.01)
      diff_rank <- names(diff_scores)
      diff_rank <- diff_rank[!(diff_rank %in% train)]
      t_tag <- gsub("\\.", "p", as.character(t_val))
      diff_out <- file.path("PART2_outputs/cv_diffusion", paste0("diff_", net_name, "_fold", f, "_t", t_
      if (!file.exists(diff_out)) {
        write_csv(tibble(Gene = diff_rank, Score = as.numeric(diff_scores[diff_rank])), diff_out)
      }

      for (K in K_values) {
        perf <- score_ranking(diff_rank, probe, K)
        all_rows[[row_i]] <- tibble(
          Network = net_name,
          Algo = "Diffusion",
          Parameter = t_val,
          Fold = f,
          TopK = K,
          Precision = perf$Precision,
          Recall = perf$Recall,
          F1 = perf$F1,
          Hits = perf$Hits
        )

        row_i <- row_i + 1
      }
    }
  }
}

results_all <- bind_rows(all_rows)
```

Save Results

```
if (!exists("results_all")) {
  results_all <- read_csv("PART2_outputs/cv_all_folds_results.csv", show_col_types = FALSE)
}

write_csv(results_all, "PART2_outputs/cv_all_folds_results.csv")

summary_results <- results_all %>%
```

```
  group_by(Network, Algo, Parameter, TopK) %>%
  summarise(
    Precision_mean = mean(Precision, na.rm = TRUE),
    Precision_sd   = sd(Precision, na.rm = TRUE),
    Recall_mean    = mean(Recall, na.rm = TRUE),
    Recall_sd      = sd(Recall, na.rm = TRUE),
    F1_mean        = mean(F1, na.rm = TRUE),
    F1_sd          = sd(F1, na.rm = TRUE),
    Hits_mean      = mean(Hits, na.rm = TRUE),
    .groups = "drop"
  )

write_csv(summary_results, "PART2_outputs/cv_summary_mean_sd.csv")
```

```
# 1) Make sure summary_results exists (load from disk if needed)
if (!exists("summary_results")) {
  if (file.exists("PART2_outputs/cv_summary_mean_sd.csv")) {
    summary_results <- readr::read_csv("PART2_outputs/cv_summary_mean_sd.csv", show_col_types = FALSE)
  } else {
    stop("Missing summary_results and PART2_outputs/cv_summary_mean_sd.csv. Run CV once or provide the r
  }
}

# 2) Pick TopK=50
target_topk <- 50
if (!(target_topk %in% summary_results$TopK)) {
  target_topk <- max(summary_results$TopK, na.rm = TRUE)
  message("TopK=50 not available; using TopK=", target_topk)
}

# 3) Compute the best combination
best_combo <- summary_results %>%
  dplyr::filter(TopK == target_topk) %>%
  dplyr::arrange(
    dplyr::desc(F1_mean),
    dplyr::desc(Precision_mean),
    dplyr::desc(Recall_mean),
    F1_sd
  ) %>%
  dplyr::slice(1)


best_network <- best_combo$Network[[1]]
best_algo    <- best_combo$Algo[[1]]
best_param   <- best_combo$Parameter[[1]]

cat(
  "Best combo @TopK=", target_topk, ": ", best_algo, " on ", best_network,
  ifelse(is.na(best_param), "", paste0(" (t=", best_param, ")")),
  "\n",
  sep = ""
)
```

```
## Best combo @TopK=50: DIAMOnD on STRING
```

Identify Best Performer

```
print(best_combo)
```

```
## # A tibble: 1 x 11
##   Network Algo    Parameter  TopK Precision_mean Precision_sd Recall_mean
##   <chr>   <chr>       <dbl> <dbl>          <dbl>        <dbl>       <dbl>
## 1 STRING  DIAMOnD        NA    50          0.136       0.0219       0.142
## # i 4 more variables: Recall_sd <dbl>, F1_mean <dbl>, F1_sd <dbl>,
## #   Hits_mean <dbl>
```

The best combination is DIAMOnD - STRING.

Visualizations

```
dir.create("PART2_outputs/figures", showWarnings = FALSE, recursive = TRUE)

p1 <- ggplot(summary_results %>% filter(TopK == 50)
, aes(x = Network, y = Precision_mean, fill = Algo)) +
  geom_col(position = position_dodge(width = 0.8)) +
  geom_errorbar(aes(ymin = Precision_mean - Precision_sd,
                    ymax = Precision_mean + Precision_sd),
                width = 0.2, position = position_dodge(width = 0.8)) +
  facet_wrap(~ Parameter, labeller = label_both) +
  theme_minimal(base_size = 13) +
  labs(title = "Top-50 Precision across interactomes",
       subtitle = "Mean ± SD over 5-fold CV",
       y = "Precision", x = "Interactome")

ggsave("PART2_outputs/figures/PART2_precision_top50_bar.png", p1, width = 12, height = 6, dpi = 300)
print(p1)
```

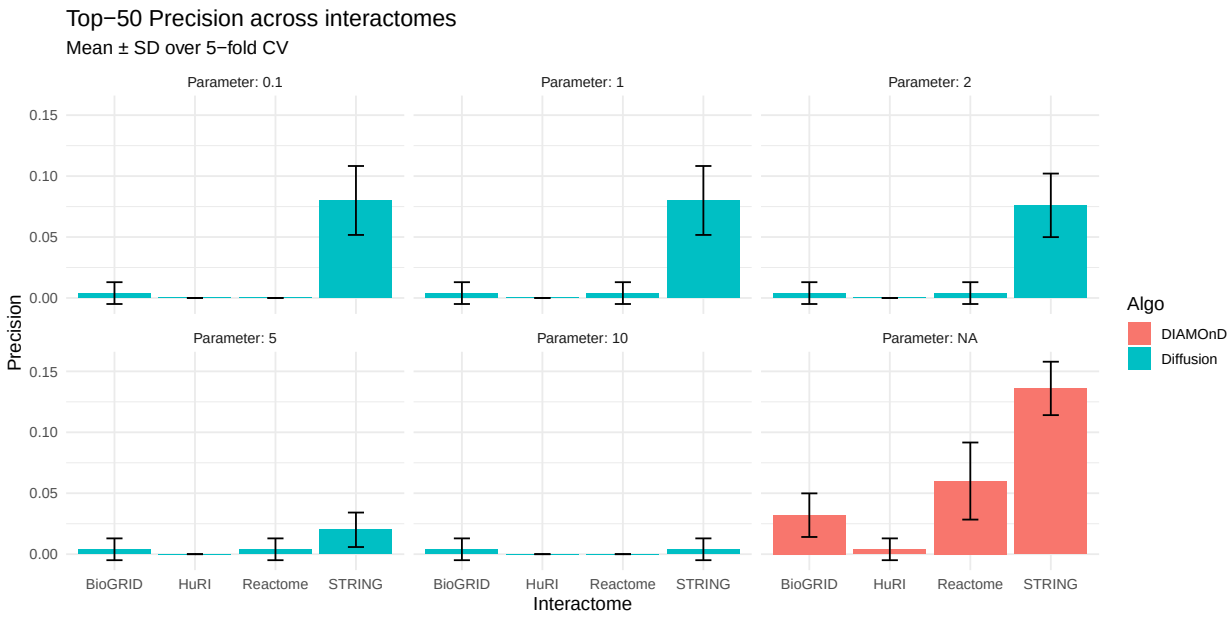


Figure 1: Top-50 Precision across interactomes (Mean ± SD over 5-fold CV)

```r
p2 <- ggplot(summary_results %>% filter(TopK == 50),
             aes(x = Recall_mean, y = Precision_mean, color = Algo, shape = Network)) +
  geom_point(size = 3) +
  facet_wrap(~ Parameter) +
  theme_minimal(base_size = 13) +
  labs(title = "Precision-Recall tradeoff (Top-50)",
       x = "Recall", y = "Precision")

ggsave("PART2_outputs/figures/PART2_precision_recall_top50.png", p2, width = 12, height = 6, dpi = 300)
print(p2)
```

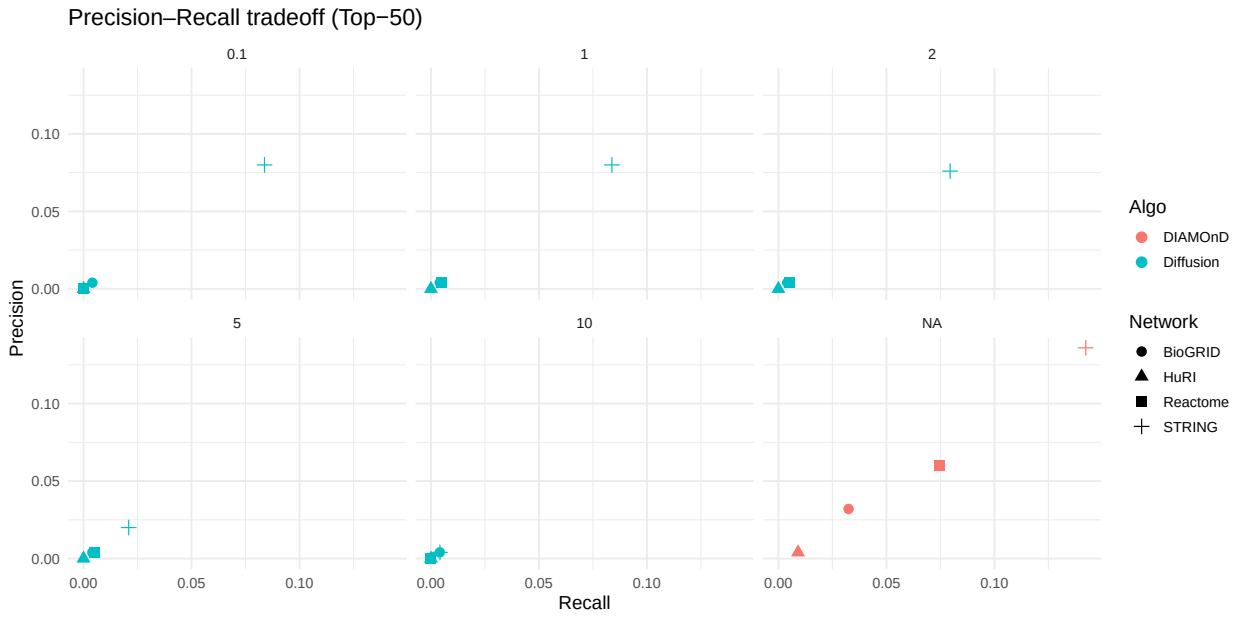


Figure 2: Precision-Recall tradeoff at Top-50

```r
p3 <- ggplot(summary_results,
             aes(x = factor(TopK), y = F1_mean,
                 group = interaction(Algo, Parameter),
                 color = Algo)) +
  geom_line() + geom_point() +
  facet_grid(Network ~ Parameter) +
  theme_minimal(base_size = 12) +
  labs(title = "F1-score across Top-K thresholds",
       x = "Top-K", y = "Mean F1")

ggsave("PART2_outputs/figures/PART2_f1_across_topk.png", p3, width = 12, height = 10, dpi = 300)
print(p3)
```

```r
# Ensure viridis is available
if (!requireNamespace("viridis", quietly = TRUE)) install.packages("viridis")
library(viridis)

# Build heatmap data at TopK = 50
heat_data <- summary_results %>%
  filter(TopK == 50) %>%
```

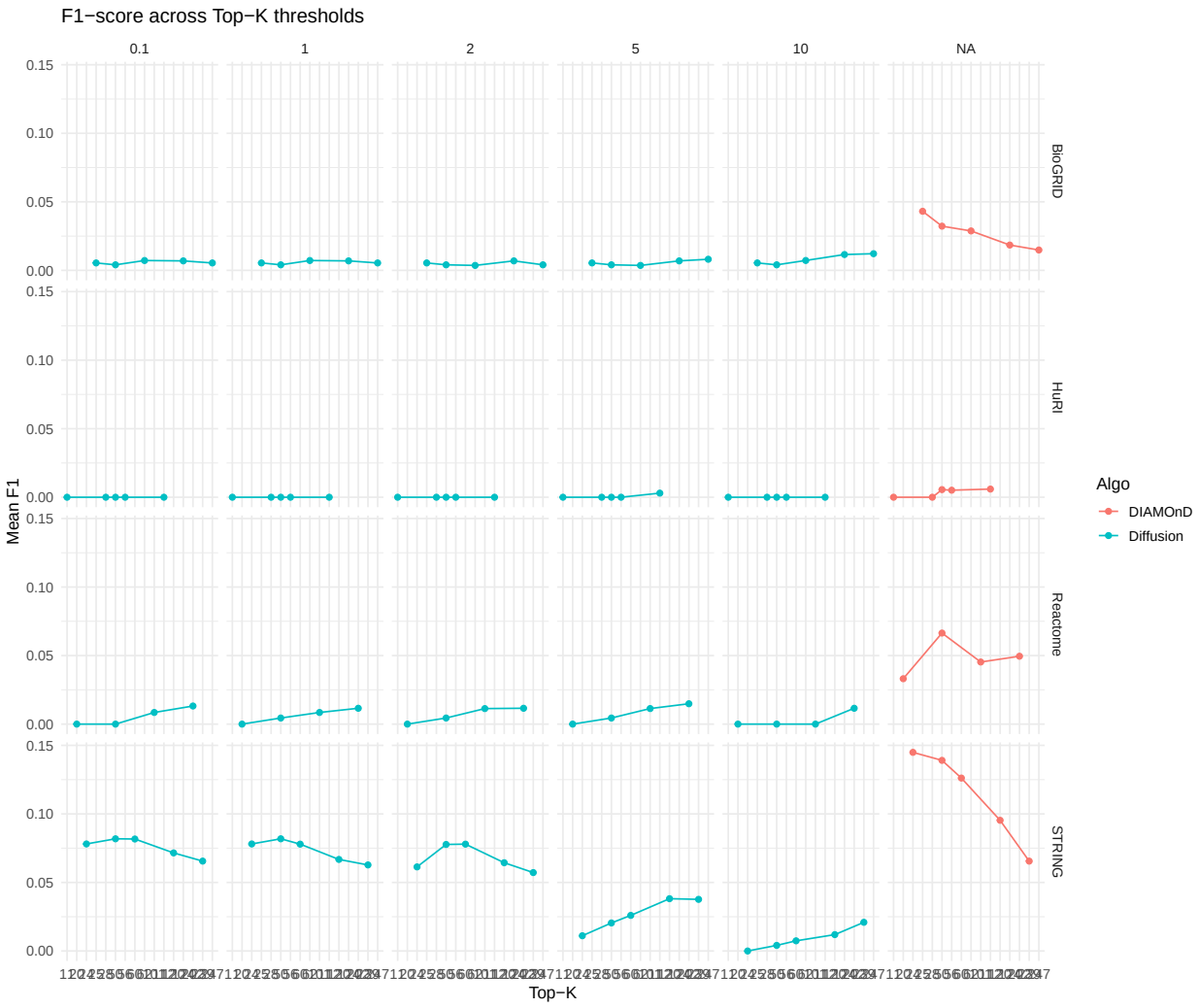


Figure 3: F1-score across Top-K thresholds

```
  mutate(
    Method = ifelse(is.na(Parameter),
                    Algo,
                    paste0(Algo, "_t=", Parameter))
  ) %>%
  dplyr::select(Network, Method, Precision_mean)

# Plot
p_heat <- ggplot(heat_data, aes(x = Method, y = Network, fill = Precision_mean)) +
  geom_tile(color = "white") +
  scale_fill_viridis_c(option = "C") +
  theme_minimal(base_size = 12) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(
    title = "Precision heatmap (Top-50)",
    x = "Method",
    y = "Interactome",
    fill = "Mean Precision"
  )

ggsave("PART2_outputs/figures/PART2_precision_top50_heatmap.png",
       p_heat, width = 12, height = 5.5, dpi = 300)

print(p_heat)
```

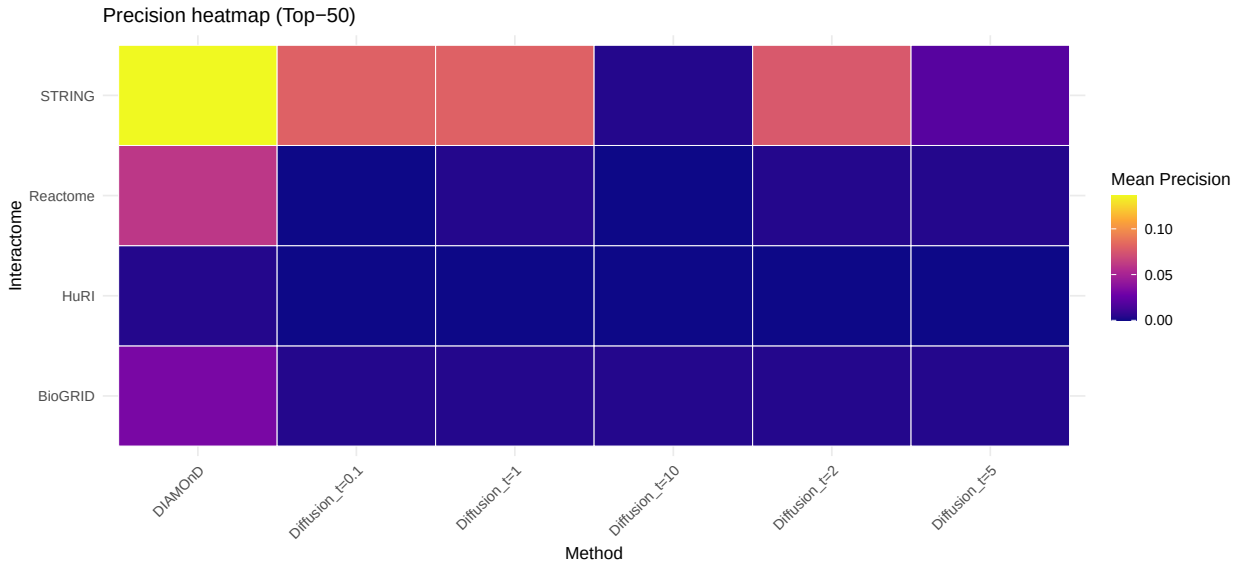


Figure 4: Precision heatmap (Top-50) across interactomes and methods

Part 3: Best Algorithm and Enrichment Analysis

Setup

```
dir.create("PART3_outputs", showWarnings = FALSE, recursive = TRUE)
dir.create("PART3_outputs/final_run", showWarnings = FALSE, recursive = TRUE)
```

```r
dir.create("PART3_outputs/enrichment", showWarnings = FALSE, recursive = TRUE)
dir.create("PART3_outputs/figures", showWarnings = FALSE, recursive = TRUE)
```

Generate Putative Genes

```r
TOP_N <- 100
DIFF_ITERS <- 20

g_best <- interactomes_lcc[[best_network]]
universe <- V(g_best)$name

final_seeds <- intersect(seed_genes, universe)
writeLines(final_seeds, "PART3_outputs/final_run/final_seeds_in_best_interactome.txt")

cat(
  "Universe size:", length(universe),
  "\n",
  sep = " "
)
```

```
## Universe size: 18920
```

```r
cat(
  "Seeds in universe:", length(final_seeds),
  "\n",
  sep = " "
)
```

```
## Seeds in universe: 239
```

```r
best_algo_norm <- tolower(trimws(best_algo))

if (best_algo_norm == "diamond") {

  netfile <- paste0(best_network, "_network.txt")
  diamond_out <- "PART3_outputs/final_run/diamond_final_results.txt"
  run_diamond(netfile, final_seeds, TOP_N, diamond_out, python_exe)

  diamond_rank <- read_diamond_rank(diamond_out)
  diamond_rank <- setdiff(diamond_rank, final_seeds)

  putative_genes <- diamond_rank[seq_len(TOP_N)]
  putative_tbl <- tibble(
    Rank = seq_len(TOP_N),
    Gene = putative_genes,
    Score = NA_real_,
    Method = "DIAMOnD",
    Network = best_network,
    Parameter = NA_real_
  )

} else if (best_algo_norm == "diffusion") {

  prep <- prepare_diffusion(g_best)
```

```
  scores <- run_diffusion(prep, final_seeds, t = as.numeric(best_param), dt = 0.01)

  ranked_genes <- names(scores)
  ranked_genes <- ranked_genes[!(ranked_genes %in% final_seeds)]

  putative_genes <- ranked_genes[seq_len(TOP_N)]
  putative_tbl <- tibble(
    Rank = seq_len(TOP_N),
    Gene = putative_genes,
    Score = as.numeric(scores[putative_genes]),
    Method = "Diffusion",
    Network = best_network,
    Parameter = as.numeric(best_param)
  )
}

write_csv(putative_tbl, "PART3_outputs/final_run/putative_genes_top100.csv")
writeLines(putative_genes, "PART3_outputs/final_run/putative_genes_top100.txt")
```

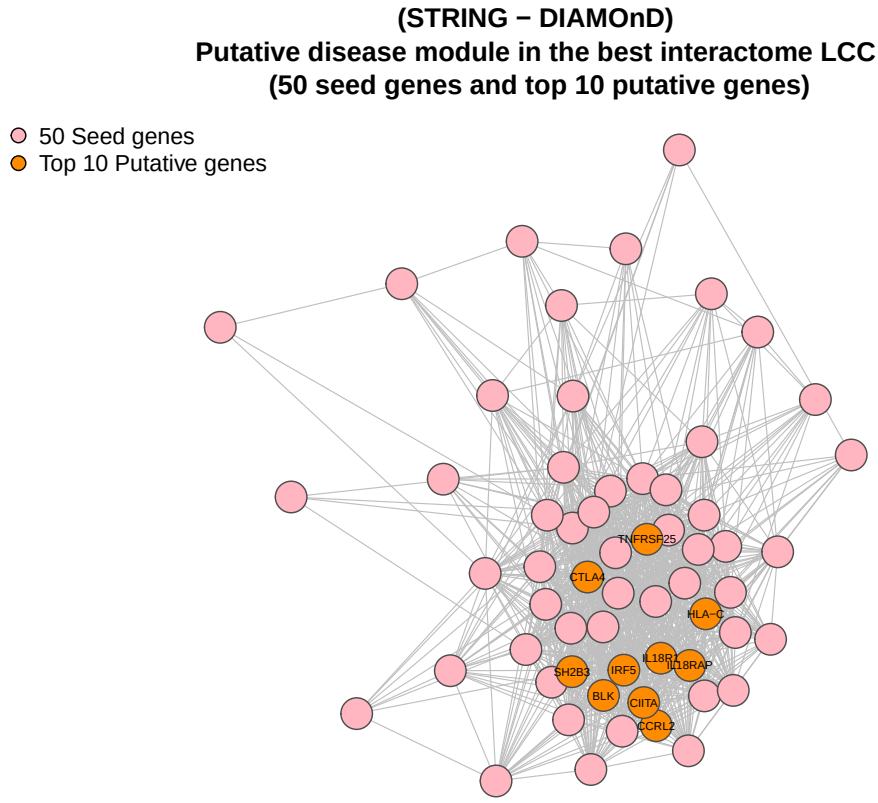
```
if (!exists("putative_tbl")) {
  putative_tbl <- read_csv("PART3_outputs/final_run/putative_genes_top100.csv", show_col_types = FALSE)
  putative_genes <- putative_tbl$Gene
}

print(head(putative_tbl, 20))
```

```
## # A tibble: 20 x 6
##     Rank Gene     Score Method  Network Parameter
##    <dbl> <chr>    <lgl> <chr>   <chr>   <lgl>
##  1     1 TNFRSF25 NA    DIAMOnD STRING  NA
##  2     2 IRF5     NA    DIAMOnD STRING  NA
##  3     3 IL18R1   NA    DIAMOnD STRING  NA
##  4     4 SH2B3    NA    DIAMOnD STRING  NA
##  5     5 BLK      NA    DIAMOnD STRING  NA
##  6     6 CTLA4    NA    DIAMOnD STRING  NA
##  7     7 HLA-C    NA    DIAMOnD STRING  NA
##  8     8 IL18RAP  NA    DIAMOnD STRING  NA
##  9     9 CCRL2    NA    DIAMOnD STRING  NA
## 10    10 CIITA    NA    DIAMOnD STRING  NA
## 11    11 TNFSF4   NA    DIAMOnD STRING  NA
## 12    12 TBX21    NA    DIAMOnD STRING  NA
## 13    13 TNFRSF14 NA    DIAMOnD STRING  NA
## 14    14 CD28     NA    DIAMOnD STRING  NA
## 15    15 IL21R    NA    DIAMOnD STRING  NA
## 16    16 FCGR2A   NA    DIAMOnD STRING  NA
## 17    17 BTLA     NA    DIAMOnD STRING  NA
## 18    18 TNFRSF4  NA    DIAMOnD STRING  NA
## 19    19 CD247    NA    DIAMOnD STRING  NA
## 20    20 CD83     NA    DIAMOnD STRING  NA
```

```
# ------------------------------------------------------------------------
# Putative disease module visualization:
# induced subgraph of (final_seeds  putative_genes) on the best interactome
# ------------------------------------------------------------------------
```

```r
# Safety checks
stopifnot(exists("g_best"))
stopifnot(exists("final_seeds"))
stopifnot(exists("putative_genes"))

dir.create("PART3_outputs/figures", showWarnings = FALSE, recursive = TRUE)

# --- Limit number of seeds to keep the plot readable ---
TOP_SEEDS <- 50
plot_seeds <- head(intersect(final_seeds, V(g_best)$name), TOP_SEEDS)

# Use a manageable number of putatives to keep the plot readable
TOP_PLOT <- 10
plot_putatives <- head(intersect(putative_genes, V(g_best)$name), TOP_PLOT)

module_nodes <- unique(c(plot_seeds, plot_putatives))

if (length(module_nodes) < 5) {
  warning("Too few nodes to plot the putative module. Check seeds/putatives present in g_best.")
} else {

  g_module <- induced_subgraph(g_best, vids = module_nodes)

  # If induced subgraph is disconnected, keep the LCC for clearer visualization
  if (vcount(g_module) > 0 && ecount(g_module) > 0) {
    comp <- components(g_module)
    giant_id <- which.max(comp$csize)
    g_module <- induced_subgraph(g_module, V(g_module)[comp$membership == giant_id])
  }

  # Vertex annotations
  V(g_module)$type <- ifelse(V(g_module)$name %in% plot_seeds, "Seed", "Putative")

  V(g_module)$color <- ifelse(V(g_module)$type == "Seed", "lightpink", "darkorange")
  V(g_module)$size  <- ifelse(V(g_module)$type == "Seed", 10, 10)
  V(g_module)$frame.color <- "grey30"

  # Label only putative genes
  V(g_module)$label <- ifelse(V(g_module)$type == "Putative", V(g_module)$name, NA)
  V(g_module)$label.cex <- 0.5
  V(g_module)$label.color <- "black"

  # Layout (Fruchterman-Reingold is standard for small biological subnetworks)
  set.seed(123)
  lay <- layout_with_fr(g_module)

  # Plot
  plot(
    g_module,
    layout = lay,
    vertex.label.family = "sans",
    edge.color = "grey75",
    edge.width = 0.8,
```

```
    main = paste0(" (", best_network, " - ", best_algo, ") \n Putative disease module in the best intera
  )

  legend(
    "topleft",
    legend = c("50 Seed genes", "Top 10 Putative genes"),
    pt.bg = c("lightpink", "darkorange"),
    pch = 21,
    pt.cex = 1.4,
    bty = "n"
  )

  # Save a high-resolution PNG version as an external artifact
  png_out <- file.path("PART3_outputs/figures", "PART3_putative_disease_module.png")
  png(png_out, width = 1800, height = 1300, res = 200)
  plot(
    g_module,
    layout = lay,
    vertex.label.family = "sans",
    edge.color = "grey75",
    edge.width = 0.8,
    main = paste0(" (", best_network, " - ", best_algo, ") \n Putative disease module in the best intera
  )
  legend(
    "topleft",
    legend = c("50 Seed genes", "Top 10 Putative genes"),
    pt.bg = c("lightpink", "darkorange"),
    pch = 21,
    pt.cex = 1.4,
    bty = "n"
  )
  dev.off()
}
```




```
## pdf
##   2
```

EnrichR Analysis

In this section, we perform a functional enrichment analysis to biologically interpret both the known disease genes (seeds) and the top-ranked putative genes identified by our network-based prioritization methods. The goal is:

(i) to characterize the biological processes and pathways associated with each gene set, and

(ii) to assess whether the predicted genes recover functional signals consistent with established disease biology.

We first rely on EnrichR, querying a curated and complementary set of annotation libraries (GO Biological Process, Molecular Function, Cellular Component, KEGG, and Reactome). These resources jointly capture molecular roles, cellular localization, and pathway-level mechanisms, providing a broad functional view. A standard adjusted p-value cutoff of 0.05 is used to control for multiple testing.

To ensure robustness and reproducibility, we clean gene lists explicitly (removing missing or duplicated symbols), standardize EnrichR outputs into a consistent tabular format, and save: - the full enrichment results, - statistically significant terms, - and the top-ranked significant terms per database.

We then directly compare enrichment results between seeds and putative genes. After normalizing term names (to avoid database-specific suffix artifacts), we compute both: - the number of overlapping significant

terms, and - the Jaccard similarity between term sets.

This overlap analysis provides a quantitative measure of biological consistency: high overlap suggests that the network-based predictions recover pathways already implicated in the disease, supporting their plausibility.

Finally, we complement EnrichR with clusterProfiler and ReactomePA dotplot visualizations, converting gene symbols to Entrez IDs to ensure compatibility with these tools. Dotplots for GO, KEGG, and Reactome offer an intuitive summary of the most enriched categories and allow a direct visual comparison between seed-driven and prediction-driven functional landscapes.

Overall, this enrichment workflow serves as a biological validation layer for our network medicine pipeline, linking topological prioritization back to interpretable molecular mechanisms.

```
if (!requireNamespace("enrichR", quietly = TRUE)) install.packages("enrichR")
library(enrichR)

ENRICHR_LIBS <- c(
  "GO_Biological_Process_2021",
  "GO_Molecular_Function_2021",
  "GO_Cellular_Component_2021",
  "KEGG_2021_Human",
  "Reactome_2022"
)

PADJ_CUTOFF <- 0.05

clean_gene_list <- function(x) {
  x <- as.character(x)
  x <- x[!is.na(x)]
  x <- trimws(x)
  x <- x[x != ""]
  unique(x)
}

seeds_for_enrichment <- clean_gene_list(final_seeds)
putative_for_enrichment <- clean_gene_list(putative_genes)

cat(
  "Enrichment seeds (n=", length(seeds_for_enrichment),
  ") | putative (n=", length(putative_for_enrichment),
  ")\n",
  sep = " "
)
```

```
## Enrichment seeds (n= 239 ) | putative (n= 100 )
```

```
enr_put  <- enrichr(putative_for_enrichment, ENRICHR_LIBS)
```

```
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2021... Done.
##   Querying GO_Molecular_Function_2021... Done.
##   Querying GO_Cellular_Component_2021... Done.
##   Querying KEGG_2021_Human... Done.
##   Querying Reactome_2022... Done.
## Parsing results... Done.
```

```
enr_seed <- enrichr(seeds_for_enrichment,   ENRICHR_LIBS)
```

```
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2021... Done.
##   Querying GO_Molecular_Function_2021... Done.
##   Querying GO_Cellular_Component_2021... Done.
##   Querying KEGG_2021_Human... Done.
##   Querying Reactome_2022... Done.
## Parsing results... Done.
```

```
standardize_enrichr <- function(df, db_name) {
  if (is.null(df) || nrow(df) == 0) return(tibble::tibble())

  df <- tibble::as_tibble(df)
  df <- dplyr::mutate(df, Database = db_name)

  for (nm in c("Adjusted.P.value", "P.value", "Combined.Score", "Odds.Ratio")) {
    if (nm %in% names(df)) {
      df[[nm]] <- suppressWarnings(as.numeric(df[[nm]]))
    }
  }

  if ("Overlap" %in% names(df)) {
    df <- dplyr::mutate(
      df,
      Overlap = as.character(.data$Overlap),
      Overlap_k = suppressWarnings(as.integer(sub("/.*", "", .data$Overlap))),
      Overlap_n = suppressWarnings(as.integer(sub(".*/", "", .data$Overlap)))
    )
  }

  if ("Genes" %in% names(df)) df$Genes <- as.character(df$Genes)
  if ("Term"  %in% names(df)) df$Term  <- as.character(df$Term)

  preferred <- c(
    "Database", "Term", "Adjusted.P.value", "P.value",
    "Overlap", "Overlap_k", "Overlap_n",
    "Odds.Ratio", "Combined.Score", "Genes"
  )
  keep_front <- preferred[preferred %in% names(df)]

  dplyr::select(df, dplyr::all_of(keep_front), dplyr::everything())
}

save_enrichr_outputs <- function(enr_list, label, padj = 0.05, top_show = 20) {
  all_tbls <- list()
  sig_tbls <- list()
  top_tbls <- list()

  for (db in names(enr_list)) {
    df <- standardize_enrichr(enr_list[[db]], db)

    out_all <- file.path("PART3_outputs/enrichment", paste0(label, "_", db, "_ALL.csv"))
    readr::write_csv(df, out_all)
```

```
    all_tbls[[db]] <- df

    df_sig <- if ("Adjusted.P.value" %in% names(df)) {
      dplyr::filter(df, !is.na(.data$Adjusted.P.value) & .data$Adjusted.P.value < padj)
    } else {
      tibble::tibble()
    }
    out_sig <- file.path("PART3_outputs/enrichment", paste0(label, "_", db, "_SIG_padj", padj, ".csv"))
    readr::write_csv(df_sig, out_sig)
    sig_tbls[[db]] <- df_sig

    df_top <- if (nrow(df_sig) > 0) {
      if ("Combined.Score" %in% names(df_sig)) {
        dplyr::arrange(df_sig, .data$Adjusted.P.value, dplyr::desc(.data$Combined.Score)) %>%
          dplyr::slice_head(n = top_show)
      } else {
        dplyr::arrange(df_sig, .data$Adjusted.P.value) %>%
          dplyr::slice_head(n = top_show)
      }
    } else {
      tibble::tibble()
    }

    out_top <- file.path("PART3_outputs/enrichment", paste0(label, "_", db, "_TOP", top_show, "_SIG.csv
    readr::write_csv(df_top, out_top)
    top_tbls[[db]] <- df_top
  }

  list(all = all_tbls, sig = sig_tbls, top = top_tbls)
}

out_put  <- save_enrichr_outputs(enr_put,  "PUTATIVE", padj = PADJ_CUTOFF, top_show = 20)
out_seed <- save_enrichr_outputs(enr_seed, "SEEDS",   padj = PADJ_CUTOFF, top_show = 20)
```

```
normalize_term <- function(term_vec, db) {
  x <- as.character(term_vec)
  x <- trimws(x)
  x <- x[x != "" & !is.na(x)]

  if (grepl("^Reactome", db)) {
    x <- sub("\\s*\\(R-HSA-[0-9]+\\)\\s*$", "", x)
  }
  if (grepl("^KEGG", db)) {
    x <- sub("\\s*-\\s*Homo\\s+sapiens\\s*\\(human\\)\\s*$", "", x)
  }
  unique(x)
}

compute_overlap <- function(sig_put, sig_seed, libs, padj = 0.05) {
  rows <- list()

  for (db in libs) {
    put_df  <- sig_put[[db]]
    seed_df <- sig_seed[[db]]
```

```
    put_terms_raw  <- if (!is.null(put_df)  && nrow(put_df)  > 0 && "Term" %in% names(put_df))  put_df$
    seed_terms_raw <- if (!is.null(seed_df) && nrow(seed_df) > 0 && "Term" %in% names(seed_df)) seed_df$

    put_terms  <- normalize_term(put_terms_raw,  db)
    seed_terms <- normalize_term(seed_terms_raw, db)

    overlap_terms <- intersect(put_terms, seed_terms)
    union_terms <- union(put_terms, seed_terms)
    jaccard <- if (length(union_terms) == 0) NA_real_ else length(overlap_terms) / length(union_terms)

    writeLines(overlap_terms, file.path("PART3_outputs/enrichment", paste0("OVERLAP_terms_", db, "_padj

    rows[[db]] <- tibble(
      Database = db,
      Putative_sig_terms = length(put_terms),
      Seeds_sig_terms    = length(seed_terms),
      Overlap_terms      = length(overlap_terms),
      Jaccard            = jaccard
    )
  }

  bind_rows(rows)
}

overlap_summary <- compute_overlap(out_put$sig, out_seed$sig, ENRICHR_LIBS, padj = PADJ_CUTOFF)
write_csv(overlap_summary, "PART3_outputs/enrichment/OVERLAP_summary.csv")

print(overlap_summary)
```

```
## # A tibble: 5 x 5
##   Database              Putative_sig_terms Seeds_sig_terms Overlap_terms Jaccard
##   <chr>                              <int>           <int>         <int>   <dbl>
## 1 GO_Biological_Proces~                386             352           147  0.249
## 2 GO_Molecular_Functio~                 28               6             3  0.0968
## 3 GO_Cellular_Componen~                 28              36            16  0.333
## 4 KEGG_2021_Human                       46              71            45  0.625
## 5 Reactome_2022                         62              52            26  0.295
```

```
# ------------------------------------------------------------------------
#  clusterProfiler / ReactomePA enrichment dotplots
# ------------------------------------------------------------------------


dir.create("PART3_outputs/figures", showWarnings = FALSE, recursive = TRUE)

#  SYMBOL -> ENTREZID robust conversion
symbol_to_entrez <- function(symbols) {
  symbols <- unique(trimws(as.character(symbols)))
  symbols <- symbols[!is.na(symbols) & symbols != ""]
  conv <- suppressMessages(
    clusterProfiler::bitr(
      symbols,
      fromType = "SYMBOL",
      toType   = "ENTREZID",
```

```
      OrgDb    = org.Hs.eg.db
    )
  )
  unique(conv$ENTREZID)
}

# Prepare Entrez IDs for both sets
entrez_putative <- symbol_to_entrez(putative_genes)
entrez_seeds    <- symbol_to_entrez(seed_genes)

cat(
  "clusterProfiler IDs: putative Entrez n=", length(entrez_putative),
  " | seeds Entrez n=", length(entrez_seeds),
  "\n",
  sep = " "
)
```

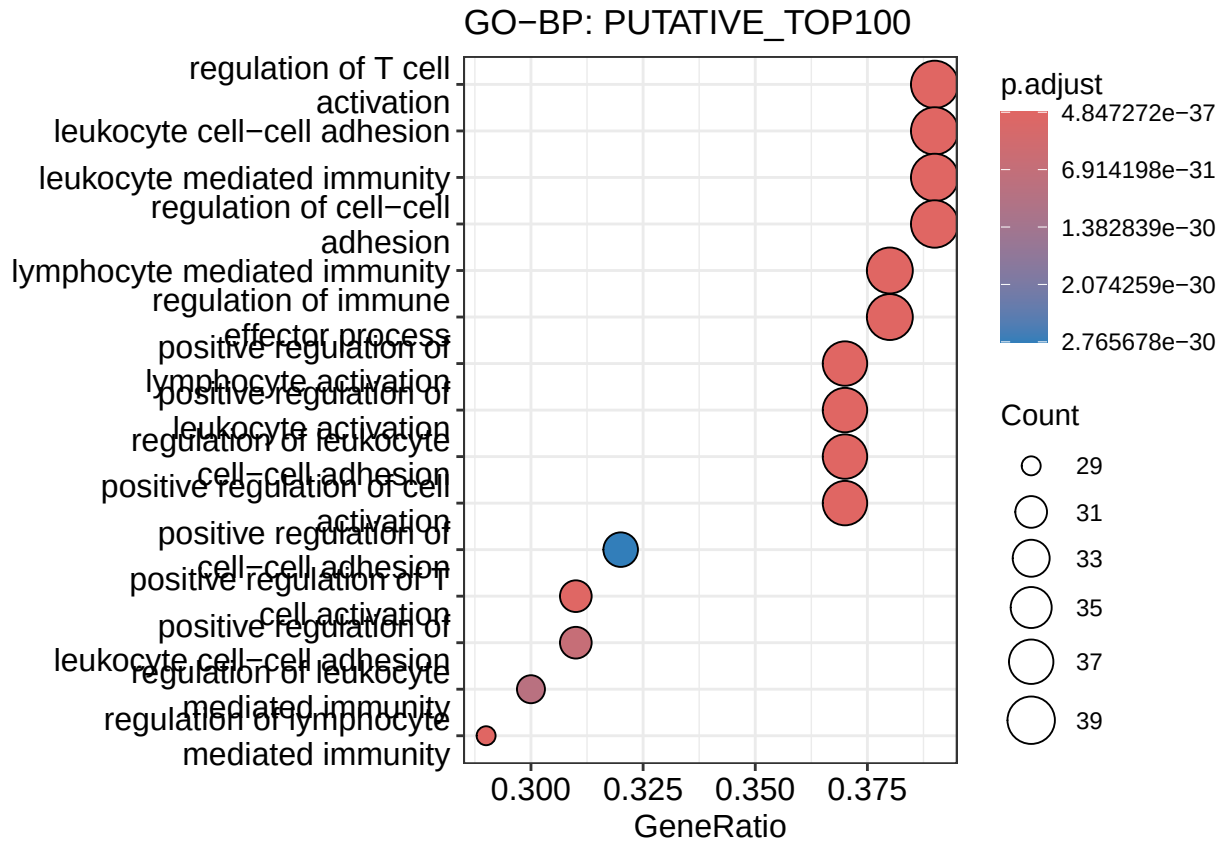
```
## clusterProfiler IDs: putative Entrez n= 100  | seeds Entrez n= 291
```

```
# Safety: stop early if conversion fails
if (length(entrez_putative) < 5 || length(entrez_seeds) < 5) {
  warning("Too few Entrez IDs after conversion. Dotplots may be uninformative. Check gene symbols / mapp
}

# ---------- GO dotplots (BP / MF / CC) ----------
run_go_dotplots <- function(entrez_vec, label, showCategory = 15, pcut = 0.05) {

  for (ont in c("BP", "MF", "CC")) {
    ego <- suppressMessages(
      clusterProfiler::enrichGO(
        gene          = entrez_vec,
        OrgDb         = org.Hs.eg.db,
        ont           = ont,
        pvalueCutoff  = pcut,
        readable      = TRUE
      )
    )

    # Print table preview in the report
    cat(
    "GO-", ont, " terms for ", label, ": ", nrow(as.data.frame(ego)),
    "\n",
    sep = " ")

    p <- enrichplot::dotplot(ego, showCategory = showCategory) +
      ggplot2::ggtitle(paste0("GO-", ont, ": ", label))

    out_png <- file.path("PART3_outputs/figures", paste0("PART3_dotplot_GO_", ont, "_", label, ".png"))
    ggsave(out_png, p, width = 10, height = 6, dpi = 300)
    print(p)
  }
}

# Putative GO
```

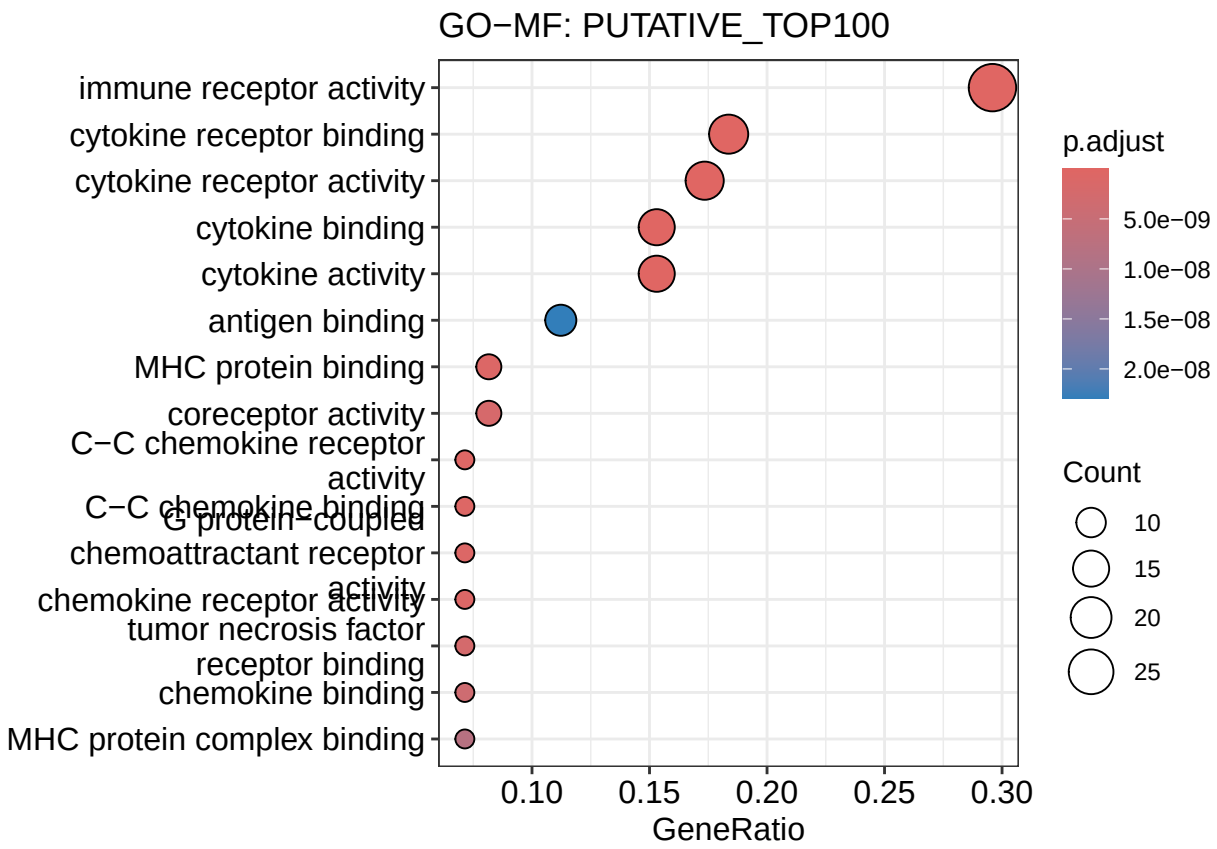
```
run_go_dotplots(entrez_putative, label = "PUTATIVE_TOP100", showCategory = 15, pcut = 0.05)
```

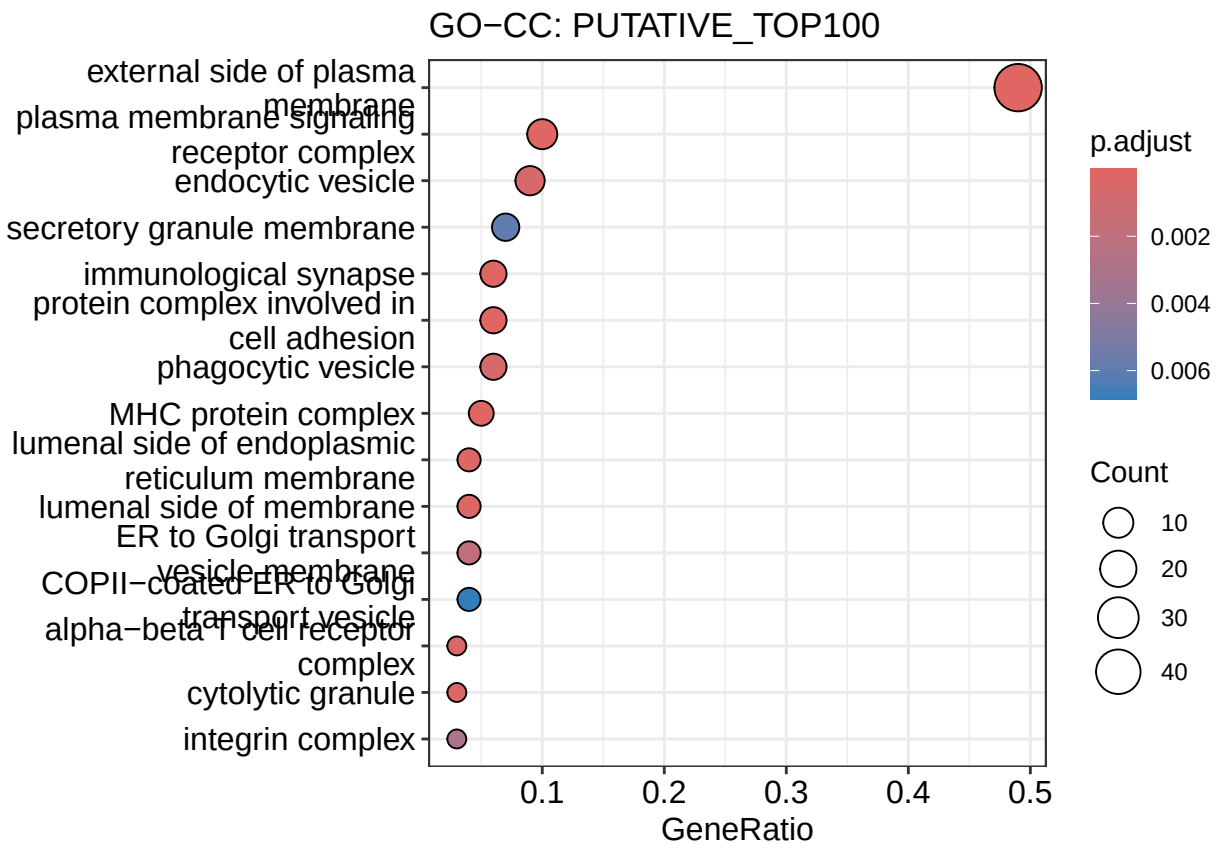
```
## GO- BP  terms for  PUTATIVE_TOP100 :  642
```

```
## GO- MF  terms for  PUTATIVE_TOP100 :  47
```




```
## GO- CC  terms for  PUTATIVE_TOP100 :  23
```

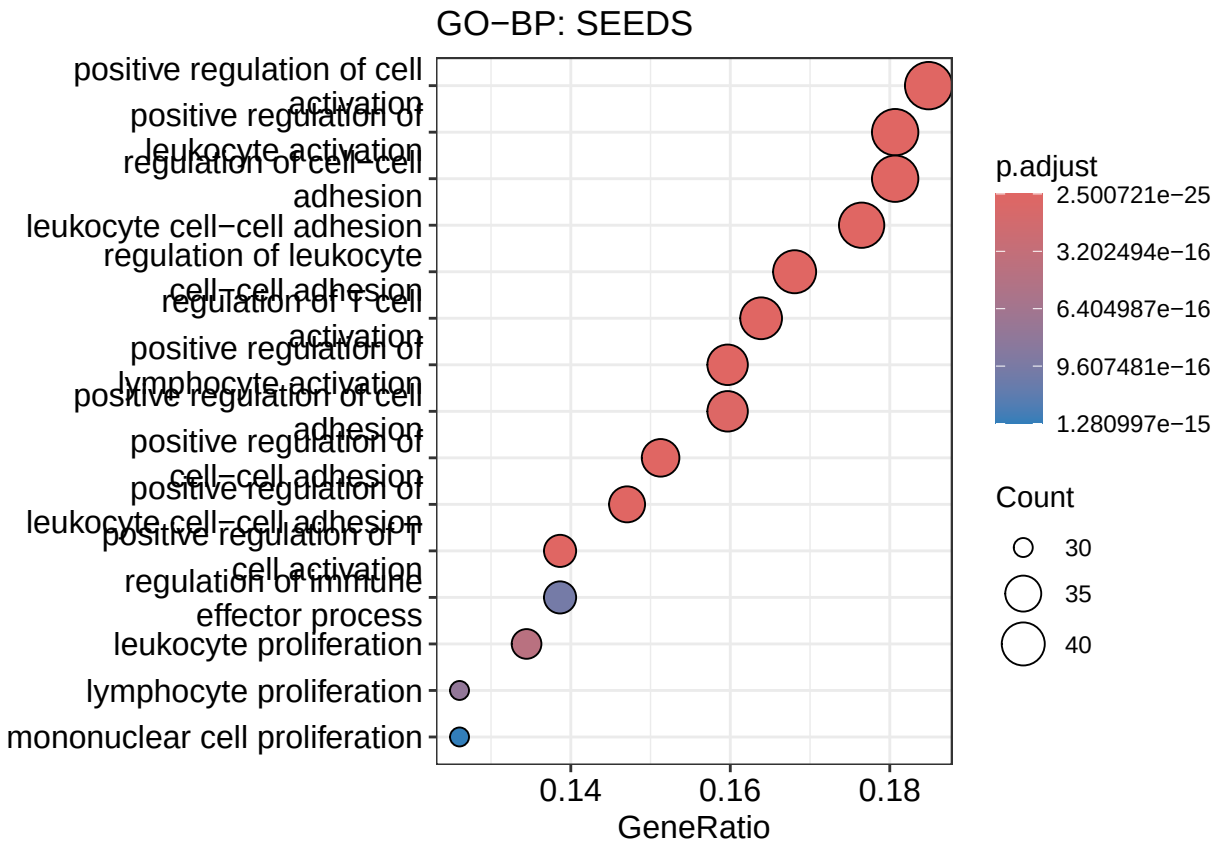

GO−MF: PUTATIVE_TOP100
immune receptor activity
cytokine receptor binding
cytokine receptor activity
cytokine binding
cytokine activity
antigen binding
MHC protein binding
coreceptor activity
C−C chemokine receptor activity
C−C chemokine binding
G protein−coupled chemoattractant receptor activity
chemokine receptor activity
tumor necrosis factor receptor binding
chemokine binding
MHC protein complex binding
0.10
0.15
0.20
0.25
0.30
GeneRatio
p.adjust
5.0e−09
1.0e−08
1.5e−08
2.0e−08
Count
10
15
20
25



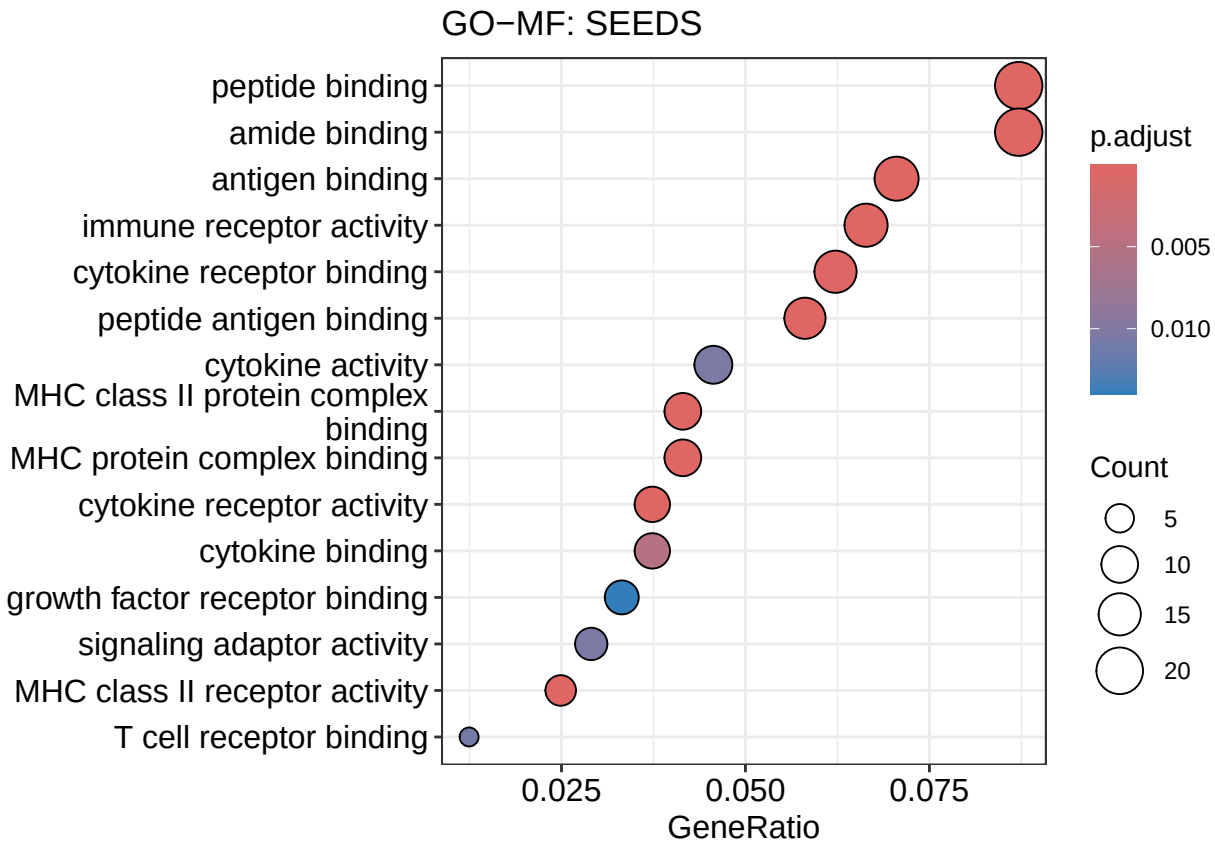

```
# Seeds GO
run_go_dotplots(entrez_seeds, label = "SEEDS", showCategory = 15, pcut = 0.05)
```

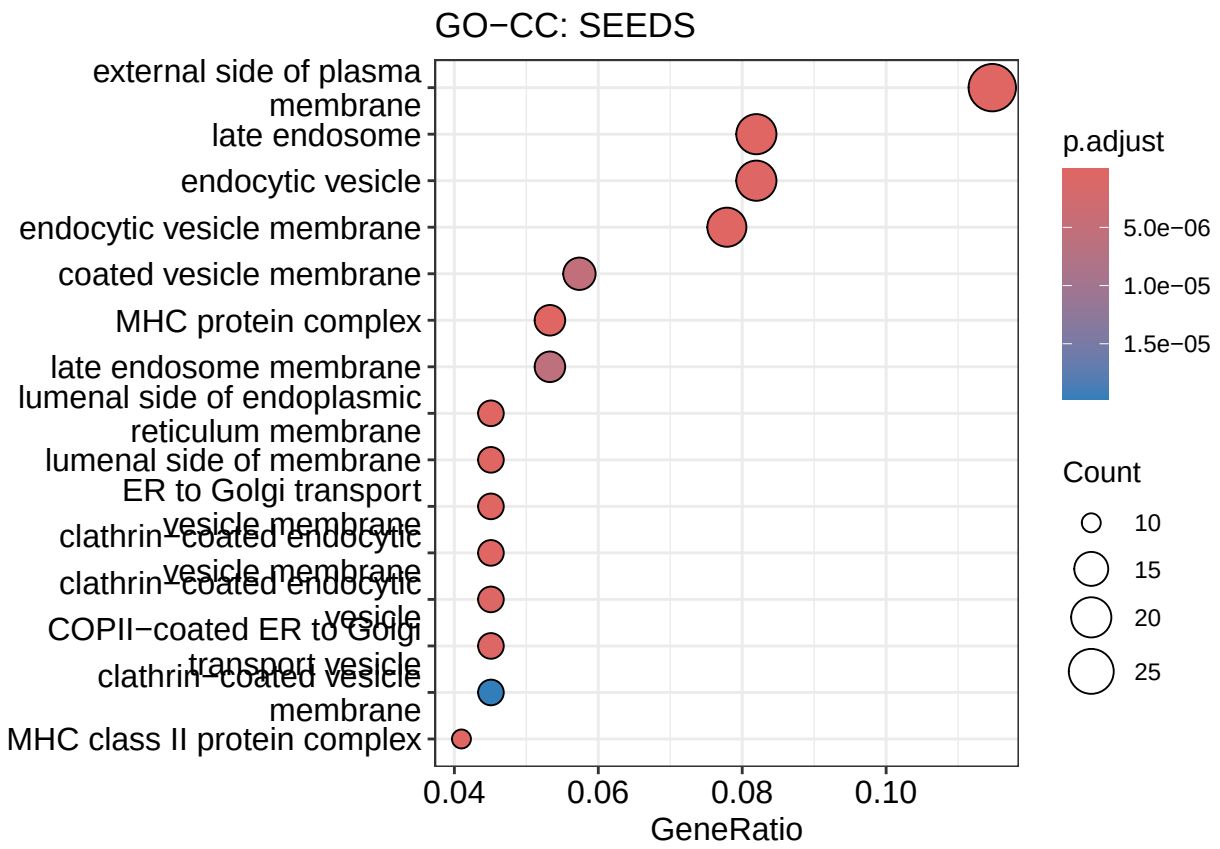
```
## GO- BP  terms for  SEEDS :  615
```

```
## GO- MF  terms for  SEEDS :  16
```



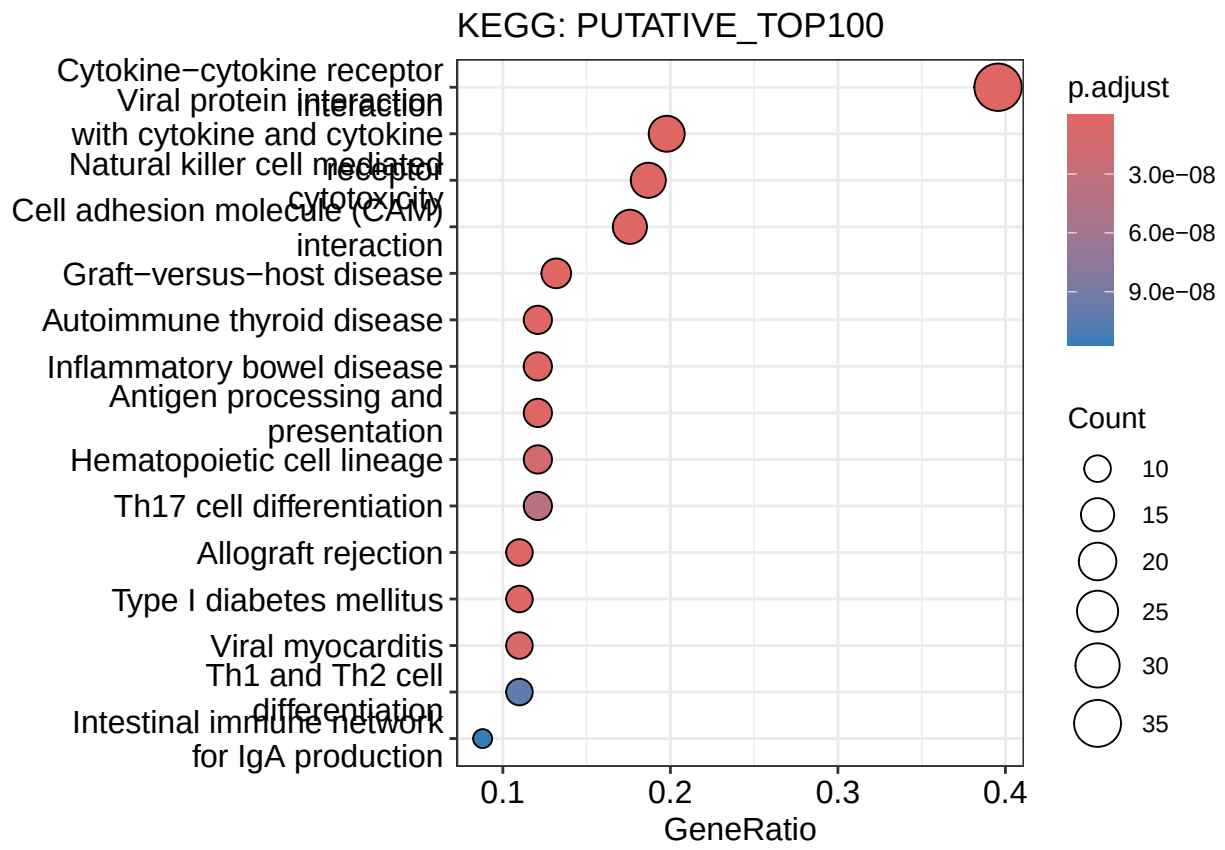

```
## GO- CC  terms for  SEEDS :  35
```


GO−MF: SEEDS
peptide binding
amide binding
antigen binding
immune receptor activity
cytokine receptor binding
peptide antigen binding
cytokine activity
MHC class II protein complex binding
MHC protein complex binding
cytokine receptor activity
cytokine binding
growth factor receptor binding
signaling adaptor activity
MHC class II receptor activity
T cell receptor binding
0.025
0.050
0.075
GeneRatio
p.adjust
0.005
0.010
Count
5
10
15
20



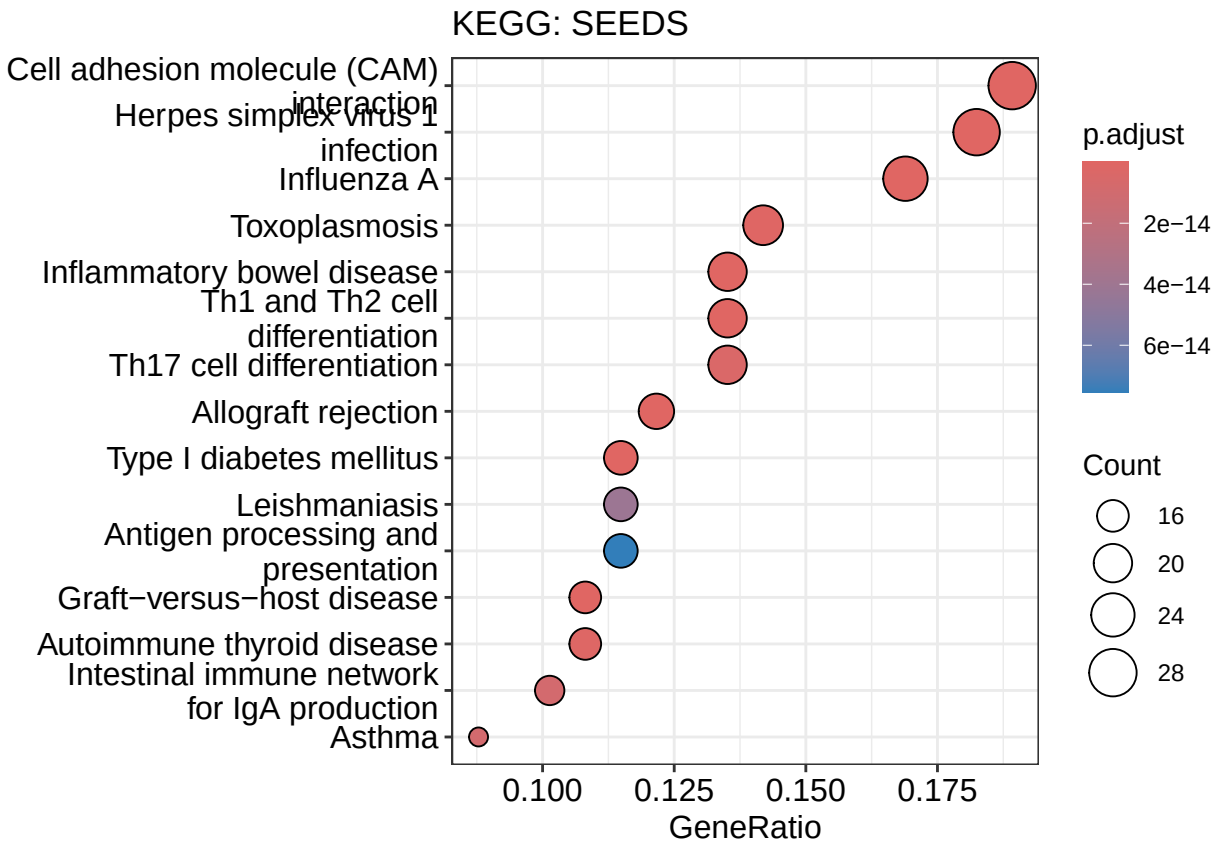

```r
# ---------- KEGG dotplots ----------
run_kegg_dotplot <- function(entrez_vec, label, showCategory = 15, pcut = 0.05) {
  ekegg <- suppressMessages(
    clusterProfiler::enrichKEGG(
      gene         = entrez_vec,
      organism     = "hsa",
      pvalueCutoff = pcut
    )
  )

  cat(
  "KEGG terms for ", label, ": ", nrow(as.data.frame(ekegg)),
  "\n",
  sep = " ")


  p <- enrichplot::dotplot(ekegg, showCategory = showCategory) +
    ggplot2::ggtitle(paste0("KEGG: ", label))

  out_png <- file.path("PART3_outputs/figures", paste0("PART3_dotplot_KEGG_", label, ".png"))
  ggsave(out_png, p, width = 10, height = 6, dpi = 300)
  print(p)
}

run_kegg_dotplot(entrez_putative, "PUTATIVE_TOP100", showCategory = 15, pcut = 0.05)
```

```
## KEGG terms for  PUTATIVE_TOP100 :  43
```



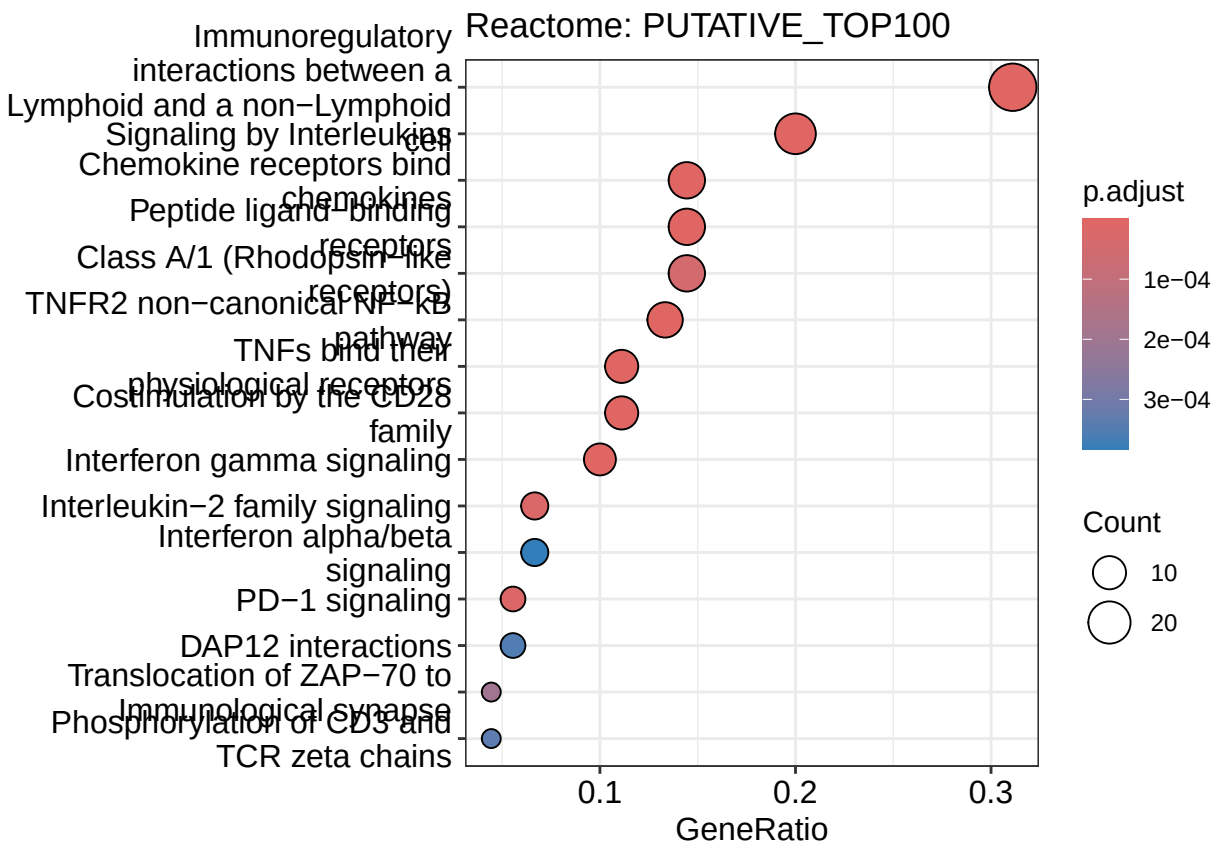

```
run_kegg_dotplot(entrez_seeds,    "SEEDS",          showCategory = 15, pcut = 0.05)
```

```
## KEGG terms for  SEEDS :  63
```




```r
# ---------- ReactomePA dotplots ----------
run_reactome_dotplot <- function(entrez_vec, label, showCategory = 15, pcut = 0.05) {
  er <- suppressMessages(
    ReactomePA::enrichPathway(
      gene         = entrez_vec,
      organism     = "human",
      pvalueCutoff = pcut,
      readable     = TRUE
    )
  )

  cat(
  "Reactome terms for ", label, ": ", nrow(as.data.frame(er)),
  "\n",
  sep = " ")


  p <- enrichplot::dotplot(er, showCategory = showCategory) +
    ggplot2::ggtitle(paste0("Reactome: ", label))

  out_png <- file.path("PART3_outputs/figures", paste0("PART3_dotplot_REACTOME_", label, ".png"))
  ggsave(out_png, p, width = 10, height = 6, dpi = 300)
  print(p)
}

run_reactome_dotplot(entrez_putative, "PUTATIVE_TOP100", showCategory = 15, pcut = 0.05)
```

```
## Reactome terms for  PUTATIVE_TOP100 :  36
```




```
run_reactome_dotplot(entrez_seeds,    "SEEDS",          showCategory = 15, pcut = 0.05)
```

```
## Reactome terms for  SEEDS :  37
```

Visualization

```
p_overlap <- ggplot(overlap_summary, aes(x = Database, y = Overlap_terms)) +
  geom_col(fill = "steelblue", alpha = 0.8) +
  geom_text(aes(label = Overlap_terms), vjust = -0.5, size = 3.5) +
  theme_minimal(base_size = 12) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
  labs(
    title = "Overlap of significant EnrichR terms (adjusted p < 0.05)",
    x = "EnrichR library",
    y = "Number of overlapping terms"
  )

ggsave("PART3_outputs/figures/PART3_overlap_counts.png", p_overlap, width = 10, height = 4.5, dpi = 300)
print(p_overlap)
```

```
p_jacc <- ggplot(overlap_summary, aes(x = Database, y = Jaccard)) +
  geom_col(fill = "coral", alpha = 0.8) +
  geom_text(aes(label = round(Jaccard, 3)), vjust = -0.5, size = 3.5) +
  theme_minimal(base_size = 12) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
  labs(
    title = "Jaccard similarity of significant term sets (padj < 0.05)",
    subtitle = "Jaccard = |Overlap| / |Union| after term normalization",
    x = "EnrichR library",
```

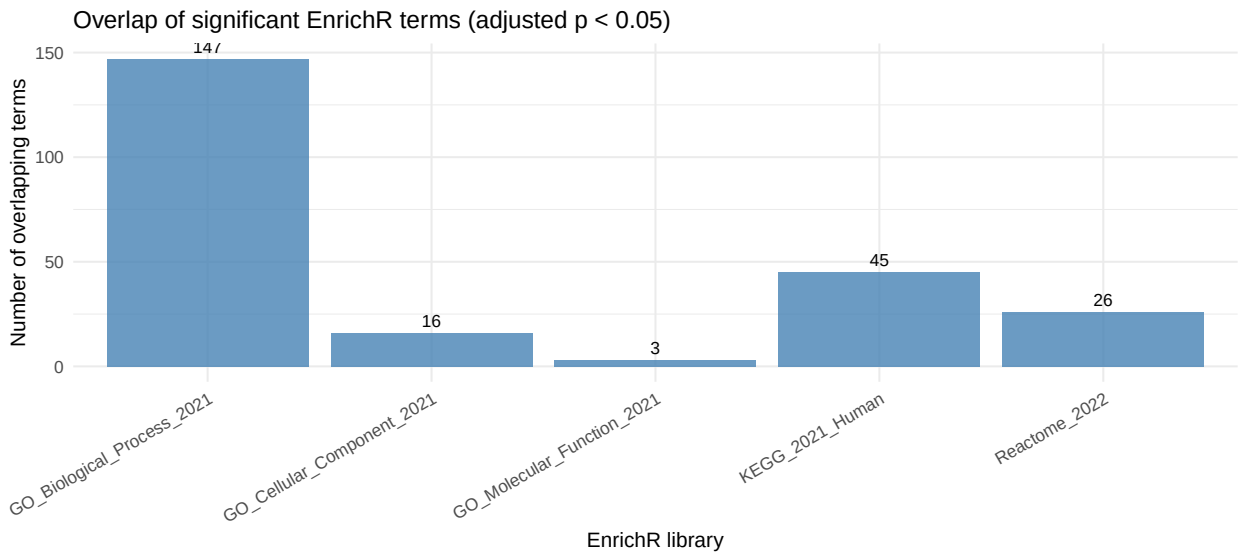


Figure 5: Overlap of significant EnrichR terms between seeds and putative genes

```
    y = "Jaccard index"
  )

ggsave("PART3_outputs/figures/PART3_overlap_jaccard.png", p_jacc, width = 10, height = 4.5, dpi = 300)
print(p_jacc)
```

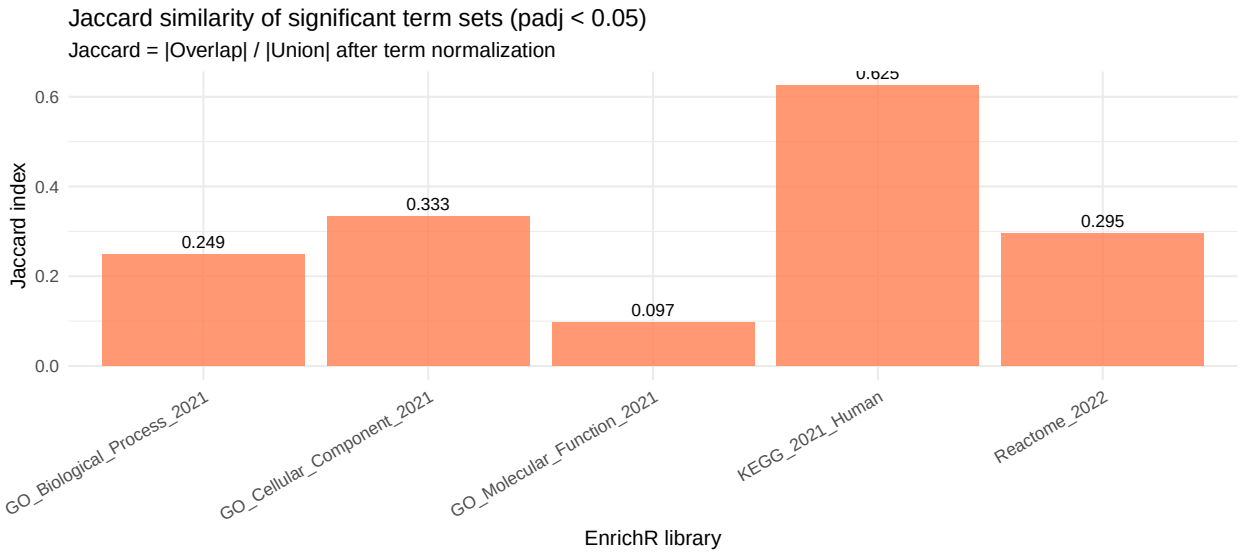


Figure 6: Jaccard similarity of significant term sets

Part 4: Drug Analysis

Setup

```
DISEASE_NAME <- "Multiple Sclerosis"
TOP_GENES_N  <- 20
TOP_DRUGS_N  <- 3

DGIDB_TSV_PATH <- "interactions.tsv"

OUT4 <- "PART4_outputs"
dir.create(OUT4, showWarnings = FALSE, recursive = TRUE)
dir.create(file.path(OUT4, "dgidb"), showWarnings = FALSE, recursive = TRUE)
dir.create(file.path(OUT4, "clinicaltrials"), showWarnings = FALSE, recursive = TRUE)
```

Load Top Genes

```
get_top_putative_genes <- function(n = 20) {
  if (exists("putative_tbl") && is.data.frame(putative_tbl) && all(c("Rank", "Gene") %in% names(putativ
    putative_tbl %>%
      arrange(.data$Rank) %>%
      slice_head(n = n) %>%
      pull(.data$Gene) %>%
      as.character()
  } else if (exists("putative_genes") && is.character(putative_genes)) {
    putative_genes[seq_len(min(n, length(putative_genes)))]
  } else {
    stop("Missing ranked putative genes")
  }
}

top_genes <- unique(str_trim(get_top_putative_genes(TOP_GENES_N)))
top_genes <- top_genes[!is.na(top_genes) & top_genes != ""]

writeLines(top_genes, file.path(OUT4, "top20_putative_genes.txt"))
write_csv(tibble(Rank = seq_along(top_genes), Gene = top_genes),
          file.path(OUT4, "top20_putative_genes.csv"))

cat(
  "Top putative genes loaded: n=", length(top_genes),
  "\n",
  sep = ""
)
```

```
## Top putative genes loaded: n=20
```

DGIdb Analysis

```
as_logical_robust <- function(x) {
  if (is.logical(x)) return(x)
  x <- as.character(x)
  x <- str_to_lower(str_trim(x))
  x %in% c("true", "t", "1", "yes", "y")
```

```
}

load_dgidb_interactions <- function(path) {
  if (!file.exists(path)) stop("DGIdb interactions.tsv not found at: ", path)

  d <- readr::read_tsv(path, show_col_types = FALSE, progress = FALSE)

  required <- c("gene_name", "drug_name", "approved")
  missing  <- setdiff(required, names(d))
  if (length(missing) > 0) {
    stop("DGIdb file is missing required columns: ", paste(missing, collapse = ", "))
  }

  d %>%
    mutate(
      gene_name = as.character(.data$gene_name),
      drug_name = as.character(.data$drug_name),
      approved_logical = as_logical_robust(.data$approved)
    ) %>%
    filter(!is.na(.data$gene_name) & .data$gene_name != "",
           !is.na(.data$drug_name) & .data$drug_name != "")
}

dg <- load_dgidb_interactions(DGIDB_TSV_PATH)
dg_approved <- dg %>% filter(.data$approved_logical)
dg_top <- dg_approved %>% filter(.data$gene_name %in% top_genes)

write_csv(dg_top, file.path(OUT4, "dgidb", "dgidb_approved_interactions_top20genes.csv"))

cat(
  "DGIdb approved interactions found: n_rows=", nrow(dg_top),
  "\n",
  sep = " "
)
```

```
## DGIdb approved interactions found: n_rows= 70
```

Drug Ranking

```
rank_drugs_by_gene_coverage <- function(dg_top_tbl) {
  if (nrow(dg_top_tbl) == 0) {
    return(tibble(
      drug_name = character(),
      n_genes_top20 = integer(),
      n_interactions = integer(),
      genes_top20 = character()
    ))
  }

  dg_top_tbl %>%
    group_by(.data$drug_name) %>%
    summarise(
      n_genes_top20  = n_distinct(.data$gene_name),
```

```
      n_interactions = n(),
      genes_top20    = paste(sort(unique(.data$gene_name)), collapse = ";"),
      .groups = "drop"
    ) %>%
    arrange(desc(.data$n_interactions), desc(.data$n_genes_top20), .data$drug_name)
}

drug_ranking <- rank_drugs_by_gene_coverage(dg_top)

write_csv(drug_ranking, file.path(OUT4, "dgidb", "drug_ranking_by_top20gene_coverage.csv"))

cat(
  "Drugs ranked: n=", nrow(drug_ranking),
  "\n",
  sep = " "
)
```

```
## Drugs ranked: n= 52
```

```
print(drug_ranking)
```

```
## # A tibble: 52 x 4
##    drug_name                n_genes_top20 n_interactions genes_top20
##    <chr>                            <int>          <int> <chr>
##  1 IPILIMUMAB                           1              9 CTLA4
##  2 TREMELIMUMAB                         1              6 CTLA4
##  3 PEGINTERFERON ALFA-2B                3              3 CTLA4;HLA-C;IL21R
##  4 RIBAVIRIN                            3              3 CTLA4;HLA-C;IL21R
##  5 RITUXIMAB                            1              2 FCGR2A
##  6 ABATACEPT                            1              1 CTLA4
##  7 ALDESLEUKIN                          1              1 CD28
##  8 AMOXICILLIN ANHYDROUS                1              1 HLA-C
##  9 ANTI-THYMOCYTE GLOBULIN              1              1 CTLA4
## 10 ATEZOLIZUMAB                         1              1 CTLA4
## # i 42 more rows
```

```
top_drugs <- drug_ranking %>% slice_head(n = TOP_DRUGS_N) %>% pull(.data$drug_name)
```

Part 5: PROCONSUL Comparison

In this final section we benchmark our **best-performing gene prioritization pipeline** (DIAMOnD, selected in Part 2 via cross-validation) against **PROCONSUL**, a probabilistic extension of DIAMOnD-style module expansion. The goal is not to prove one method is universally better, but to check whether an alternative, theory-consistent network medicine algorithm yields a **similar set of high-priority candidates** when starting from the same seed genes in the best interactome (STRING).

PROCONSUL follows the same network medicine intuition as DIAMOnD: *true disease genes tend to cluster in the interactome*, so genes that are unusually well-connected to the current disease module are plausible candidates.
The difference is that DIAMOnD is **deterministic** (always picks the single best candidate by smallest p-value), whereas PROCONSUL is **stochastic**:

1. At each iteration, we consider the **boundary nodes** (neighbors of the current seed/module set not yet in the module).

2. For each candidate we compute a **connectivity significance p-value** (hypergeometric tail) analogous to DIAMOnD: "how surprising is it to have k links to the module given the node's degree?"
3. We convert significance to an **information score** (Shannon information): `I = -log(p)` so that smaller p-values become larger "evidence".
4. We transform information scores into a **sampling distribution** using a softmax with temperature `T`:
   - low `T` → more greedy / DIAMOnD-like behavior (mass concentrated on top candidates),
   - high `T` → more exploratory behavior (flatter probabilities).
5. We **sample one gene** according to these probabilities, add it to the module, and repeat until we have `m` predictions.
6. We repeat this process for `n_runs` runs and aggregate results by **position-based scoring** (genes selected earlier receive higher scores), producing a stable final ranking.

This probabilistic design is motivated by the fact that interactomes are noisy and incomplete: when many candidates have similar significance, allowing controlled exploration can reduce sensitivity to small topological artifacts and can reveal **multiple plausible module expansions**.

- **Setup:** we create a dedicated output folder (`PART5_outputs`) and ensure `PROCONSUL.py` is present, enforcing reproducibility and early failure if the dependency is missing.
- **Wrapper functions:**
  - `run_proconsul()` writes the seed list to disk, constructs a command-line call to the Python implementation (passing `m`, `n_runs`, `temp`, and a fixed RNG seed), and checks that an output file is actually produced.
  - `read_proconsul_rank()` loads the TSV output and extracts the ranked gene list, with a strict format check to avoid silent parsing errors.
- **Run step:** we define the comparison target as the **Top-20 genes** from our best performer and save them. The actual PROCONSUL execution chunk is set to `eval=FALSE` because it can be time-consuming; we rely on cached results already saved in `PART5_outputs/proconsul_*_temp1.tsv`.
- **Load cached results:** if the expected PROCONSUL output is missing, we search for an existing file (same naming pattern) to keep the pipeline robust across machines/reruns; otherwise we stop with a clear error.
- **Comparison:** we compute:
  - `overlap` = intersection between the two Top-20 lists,
  - `jaccard` = `|A  B| / |A  B|` as a normalized similarity measure, and we save both the overlapping genes and a summary table to disk. Printing the set differences highlights which candidates are unique to each method.

Overall, this section provides a **sanity-check validation**: if PROCONSUL recovers many of the same candidates as our best performer, it strengthens confidence that our predictions reflect robust disease-module topology rather than a method-specific artifact.

Setup

```
OUT5 <- "PART5_outputs"
dir.create(OUT5, showWarnings = FALSE, recursive = TRUE)

stopifnot(file.exists("PROCONSUL.py"))
```

PROCONSUL Functions

```
run_proconsul <- function(netfile, seeds, m, n_runs, temp, outpath, python_exe, seed = 123) {
  seed_file <- file.path(OUT5, "proconsul_seeds.txt")
  write.table(seeds, seed_file, row.names = FALSE, col.names = FALSE, quote = FALSE)

  cmd <- paste(
```

```
    shQuote(python_exe),
    shQuote("PROCONSUL.py"),
    shQuote(netfile),
    shQuote(seed_file),
    "--m", as.integer(m),
    "--n_runs", as.integer(n_runs),
    "--temp", as.numeric(temp),
    "--seed", as.integer(seed),
    "--out", shQuote(outpath)
  )

  system(cmd, intern = FALSE, ignore.stdout = TRUE, ignore.stderr = FALSE)

  if (!file.exists(outpath)) {
    stop("PROCONSUL failed: output not created at ", outpath)
  }
}

read_proconsul_rank <- function(path) {
  d <- readr::read_tsv(path, show_col_types = FALSE)
  if (!all(c("rank", "gene") %in% names(d))) {
    stop("Unexpected PROCONSUL output format")
  }
  d %>% arrange(.data$rank) %>% pull(.data$gene) %>% as.character()
}
```

Run PROCONSUL

```
best_top20 <- get_top_putative_genes(20)
best_top20 <- unique(stringr::str_trim(best_top20))
best_top20 <- best_top20[!is.na(best_top20) & best_top20 != ""]

writeLines(best_top20, file.path(OUT5, "bestperformer_top20.txt"))
```

```
# NOTA: Questo chunk è impostato con eval=FALSE perché PROCONSUL richiede molto tempo.
# I risultati sono già salvati in PART5_outputs/proconsul_*_temp1.tsv
netfile_best <- paste0(best_network, "_network.txt")

M_PRO <- 20
N_RUNS_PRO <- 50
TEMP_PRO <- 1.0

pro_out <- file.path(OUT5, paste0("proconsul_", best_network, "_temp1.tsv"))

run_proconsul(
  netfile = netfile_best,
  seeds = final_seeds,
  m = M_PRO,
  n_runs = N_RUNS_PRO,
  temp = TEMP_PRO,
  outpath = pro_out,
  python_exe = python_exe,
  seed = 123
```

```
)
```

```
pro_out <- file.path(OUT5, paste0("proconsul_", best_network, "_temp1.tsv"))

if (!file.exists(pro_out)) {
  # Cerca file PROCONSUL esistenti
  pro_files <- list.files(OUT5, pattern = "^proconsul_.*_temp1\\.tsv$", full.names = TRUE)
  if (length(pro_files) > 0) {
    pro_out <- pro_files[1]
    message("Using existing PROCONSUL output: ", pro_out)
  } else {
    stop("PROCONSUL output not found. Run the algorithm first or set eval=TRUE in the previous chunk.")
  }
}

pro_rank <- read_proconsul_rank(pro_out)
pro_top20 <- pro_rank[seq_len(min(20, length(pro_rank)))]
pro_top20 <- unique(stringr::str_trim(pro_top20))
pro_top20 <- pro_top20[!is.na(pro_top20) & pro_top20 != ""]

writeLines(pro_top20, file.path(OUT5, "proconsul_temp1_top20.txt"))
```

Compare Results

```
overlap <- intersect(best_top20, pro_top20)
jaccard <- if (length(union(best_top20, pro_top20)) == 0) NA_real_ else
  length(overlap) / length(union(best_top20, pro_top20))

writeLines(overlap, file.path(OUT5, "overlap_genes_top20.txt"))

cmp_tbl <- tibble::tibble(
  Method_A = paste0("BestPerformer_", best_algo, "_", best_network),
  Method_B = paste0("PROCONSUL_temp1_", best_network),
  TopK = 20,
  BestTop20_n = length(best_top20),
  ProconsulTop20_n = length(pro_top20),
  Overlap_n = length(overlap),
  Jaccard = jaccard,
  Overlap_genes = paste(overlap, collapse = ";")
)

readr::write_csv(cmp_tbl, file.path(OUT5, "TOP20_overlap_best_vs_proconsul_temp1.csv"))


print(cmp_tbl)
```

```
## # A tibble: 1 x 8
##   Method_A         Method_B  TopK BestTop20_n ProconsulTop20_n Overlap_n Jaccard
##   <chr>            <chr>    <dbl>       <int>            <int>     <int>   <dbl>
## 1 BestPerformer_D~ PROCONS~    20          20               20        19   0.905
## # i 1 more variable: Overlap_genes <chr>
```

```
cat(
  "DIAMOnD (best performer) Top-20 genes include:\n  ",
```

```
  paste(best_top20, collapse = ", "),
  "\n",
  "Non-overlapping DIAMOnD genes (vs PROCONSUL):\n  ",
  paste(setdiff(best_top20, overlap), collapse = ", "),
  "\n",
  sep = ""
)
```

```
## DIAMOnD (best performer) Top-20 genes include:
##   TNFRSF25, IRF5, IL18R1, SH2B3, BLK, CTLA4, HLA-C, IL18RAP, CCRL2, CIITA, TNFSF4, TBX21, TNFRSF14, C
## Non-overlapping DIAMOnD genes (vs PROCONSUL):
##   CD83
```

```
cat(
  "PROCONSUL (temp=1) Top-20 genes include:\n  ",
  paste(pro_top20, collapse = ", "),
  "\n",
  "Non-overlapping PROCONSUL genes (vs DIAMOnD):\n  ",
  paste(setdiff(pro_top20, overlap), collapse = ", "),
  "\n",
  sep = ""
)
```

```
## PROCONSUL (temp=1) Top-20 genes include:
##   TNFRSF25, IRF5, IL18R1, SH2B3, BLK, CTLA4, HLA-C, IL18RAP, CCRL2, CIITA, TNFSF4, TBX21, TNFRSF14, C
## Non-overlapping PROCONSUL genes (vs DIAMOnD):
##   ORMDL3
```

Project Summary

In this project we applied a full network medicine pipeline to Multiple Sclerosis, starting from known disease genes (seeds) and moving toward
(i) robust interactome selection,
(ii) validated gene prioritization,
(iii) functional interpretation,
(iv) drug repurposing signals, and
(v) external-method sanity checking.

Part 1 — Interactomes and seed mapping

We constructed four human interactomes (BioGRID, HuRI, STRING, Reactome) and worked on each network's largest connected component (LCC) to ensure that diffusion and module-expansion methods are well defined and comparable.

Gene identifiers were mapped to HGNC symbols using Ensembl mirrors, and we quantified how many Multiple Sclerosis seed genes were present in each interactome LCC. Seed coverage was high in BioGRID (247 seeds; 84.3%) and STRING (239; 81.6%), intermediate in Reactome (202; 68.9%), and substantially lower in HuRI (112; 38.2%). This motivated a preference for interactomes that combine strong connectivity with substantial disease-gene presence.

Part 2 — Gene prioritization with cross-validation

We benchmarked two complementary families of methods across interactomes using 5-fold cross-validation:
(i) DIAMOnD (local module expansion), and
(ii) heat diffusion / network propagation (global smoothing) across multiple diffusion times.

To avoid information leakage, training seeds were removed from candidate rankings before evaluation. Performance was assessed at multiple TopK thresholds adapted to the number of available seeds.

Across networks and parameters, the best-performing configuration was DIAMOnD on STRING, justifying the selection of STRING + DIAMOnD as the final "best performer" for downstream analysis.

Part 3 — Functional enrichment of predicted genes

Using the selected configuration, we generated the Top-100 putative genes (e.g., TNFRSF25, IRF5, IL18R1, SH2B3, BLK, CTLA4, CIITA, TBX21, CD28, IL21R).

Functional enrichment was then performed on both the seed genes and the putative genes using EnrichR libraries, complemented by clusterProfiler and ReactomePA dotplots. The putative gene set strongly recapitulated immune and inflammatory biology consistent with Multiple Sclerosis, highlighting pathways such as T-cell activation, leukocyte adhesion, cytokine and chemokine signaling, antigen processing and presentation, Th1/Th2/Th17 differentiation, interferon signaling, and PD-1 / CD28 co-stimulation.

Overlap analysis between seed and putative enrichments showed substantial agreement, particularly for pathway-level databases. KEGG displayed a Jaccard similarity of 0.625 (45 overlapping significant terms), while Reactome and GO Biological Process showed moderate but meaningful overlap (e.g., Reactome Jaccard = 0.295; GO-BP Jaccard = 0.249). This indicates that the prioritization recovers disease-relevant biology rather than generic network hubs.

Part 4 — Drug–gene interactions (DGIdb)

We queried approved DGIdb interactions for the Top-20 predicted genes and ranked drugs by the number of these genes they target. The strongest signals were PEGinterferon alfa-2b and ribavirin, each covering three of the Top-20 genes (CTLA4, HLA-C, IL21R), with additional high-frequency interactions centered on immune checkpoint targets such as CTLA4.

This step was treated as a hypothesis-generation layer, emphasizing drugs that act on multiple top-ranked genes rather than isolated single-gene interactions.

Part 5 — PROCONSUL comparison (sanity check)

As a final robustness check, we ran PROCONSUL on the same best interactome and seed set (m = 20, n_runs = 50, temp = 1.0) and compared its Top-20 list with the Top-20 genes from our best-performing configuration. The agreement was extremely high, with 19 out of 20 genes overlapping and a Jaccard similarity of 0.905. This demonstrates that our final candidate set is not an artifact of a single algorithmic choice.

Overall, our methodological choices were driven by:
(i) enforcing graph connectivity via the LCC,
(ii) prioritizing interactomes with strong disease-seed coverage,
(iii) selecting methods based on cross-validated performance rather than intuition, and
(iv) validating biological plausibility through enrichment concordance and methodological robustness through PROCONSUL agreement.