The topology of software risk in scientific models

3. Global hydrological and land use models

Arnald Puy

Contents

1 Preliminary

```r
# PRELIMINARY FUNCTIONS ##########################################################
##################################################################################

sensobol::load_packages(c("data.table", "tidyverse", "openxlsx", "scales",
                          "cowplot", "readxl", "ggrepel", "tidytext", "here",
                          "tidygraph", "igraph", "foreach", "parallel", "ggraph",
                          "tools", "purrr", "sensobol", "benchmarkme"))

# Create custom theme ----------------------------------------------------------

theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent", color = NA),
          legend.key = element_rect(fill = "transparent", color = NA),
          strip.background = element_rect(fill = "white"),
          legend.text = element_text(size = 7.3),
          axis.title = element_text(size = 10),
          legend.key.width = unit(0.4, "cm"),
          legend.key.height = unit(0.4, "cm"),
          legend.key.spacing.y = unit(0, "lines"),
          legend.box.spacing = unit(0, "pt"),
          legend.title = element_text(size = 7.3),
          axis.text.x = element_text(size = 7),
          axis.text.y = element_text(size = 7),
          axis.title.x = element_text(size = 7.3),
          axis.title.y = element_text(size = 7.3),
          plot.title = element_text(size = 8),
          strip.text.x = element_text(size = 7.4),
          strip.text.y = element_text(size = 7.4))
}

# Select color palette ---------------------------------------------------------

color_languages <- c("fortran" = "steelblue", "python" = "lightgreen")

# Source all .R files in the "functions" folder --------------------------------

r_functions <- list.files(path = here("functions"),
                          pattern = "\\.R$", full.names = TRUE)

lapply(r_functions, source)

# Set seed ---------------------------------------------------------------------
```

```r
seed <- 123
```

2 Analysis

```r
# CREATE DATASET ######################################################################

# Path to folder ------------------------------------------------------------------

path <- "./datasets/call_metrics"

# List CSV files ------------------------------------------------------------------

files <- list.files(path, pattern = "\\.csv$", full.names = TRUE)

# Split by language ---------------------------------------------------------------

python_files  <- grep("python",  files, value = TRUE, ignore.case = TRUE)
fortran_files <- grep("fortran", files, value = TRUE, ignore.case = TRUE)

base_fortran <- file_path_sans_ext(basename(fortran_files))
base_python <- file_path_sans_ext(basename(python_files))

model_names_fortran <- models <- sub(".*_", "", base_fortran)
model_names_python <- models <- sub(".*_", "", base_python)

# Load and name files -------------------------------------------------------------

python_list  <- lapply(python_files, fread)
fortran_list <- lapply(fortran_files, fread)

names(python_list) <- model_names_python
names(fortran_list) <- model_names_fortran

# RBIND ---------------------------------------------------------------------------

make_callgraph <- function(lst, lang) {
  rbindlist(lst, idcol = "model") %>%
    .[, language := lang] %>%
    .[, .(file, model, language, `function`, call)] %>%
    setnames(., c("function", "call"), c("from", "to"))
}

python_callgraphs  <- make_callgraph(python_list,  "python")
fortran_callgraphs <- make_callgraph(fortran_list, "fortran")

all_callgraphs <- rbind(python_callgraphs, fortran_callgraphs)

# SOURCE CODE CLASSIFICATION BY FUNCTIONAL ROLE ###################################
```

```
# Strip leading "./models/"-----------------------------------------------------
all_callgraphs[, file_clean:= sub("^\\./models/", "", file)]

# Provenance: file must be inside ./models to be eligible at all----------------
all_callgraphs[, in_model_tree:= !is.na(file) & nchar(file) > 0L &
                 grepl("^\\./models/", file)]

# Rest = everything after first segment-----------------------------------------
all_callgraphs[, rest:= sub("^[^/]+/", "", file_clean)]

# If rest starts with model_id/ then drop it (this is the duplicate)------------
all_callgraphs[, rest:= fifelse(startsWith(rest, paste0(tolower(model), "/")),
                                sub("^[^/]+/", "", rest), rest)]

# In case of triple nesting like vic/vic/vic/...--------------------------------
all_callgraphs[, rest:= fifelse(startsWith(rest, paste0(tolower(model), "/")),
                                 sub("^[^/]+/", "", rest), rest)]

# Top_level---------------------------------------------------------------------
all_callgraphs[, top_level:= tstrsplit(rest, "/", fixed = TRUE, keep = 1L)]

# Useful generic external patterns (works across models)------------------------
all_callgraphs[, is_generic_external:= in_model_tree & (
  grepl("(^|/)(extern|external|externals|third[_-]?party|vendor|vendored)(/|$)",
        rest, ignore.case = TRUE) |
    grepl("(^|/)\\.lib(/|$)", rest, ignore.case = TRUE))]

# CTSM-specific external (framework + FoX parser under cdeps + CESM shared infra)
all_callgraphs[, is_ctsm_external:= (model == "CTSM") & in_model_tree & (
  grepl("^cime(/|$)", rest) |
    grepl("^cime_config(/|$)", rest) |
    grepl("^components/cdeps(/|$)", rest) |      # FoX XML/SAX parser
    grepl("(^|/)share_esmf(/|$)", rest) |        # shared ESMF infrastructure
    grepl("^src/unit_test_shr(/|$)", rest)       # unit tests
)]

# Extra CTSM allowlist (tightens "model core" to land-model code)---------------
CTSM_STRICT <- TRUE

all_callgraphs[, is_ctsm_allowed:= TRUE]

if (CTSM_STRICT) {
  all_callgraphs[model == "CTSM" & in_model_tree, is_ctsm_allowed :=
                   grepl("^src(/|$)", rest) |
                   grepl("^lilac(/|$)", rest) |
                   grepl("^components/cism(/|$)", rest)]
}
```

```
## Component classification (order matters)-----------------------------------
all_callgraphs[nchar(file) == 0L | is.na(file), component:= NA_character_]

all_callgraphs[!is.na(file) & nchar(file) > 0L, component:= fcase(

  # Anything not in ./models is external immediately
  !in_model_tree, "external_lib",

  # Generic external dirs (vendored/third_party/etc.)
  is_generic_external, "external_lib",

  # CI and vendored libraries
  top_level == ".github", "ci_cd",
  top_level == ".lib" | grepl("/\\.lib/", rest), "vendored_lib",

  # CIME / CESM / CDEPS infrastructure (framework)
  grepl("^cime/CIME/", rest), "framework",
  grepl("^cime_config/", rest), "framework",
  grepl("^cime/doc/", rest), "framework",
  grepl("^components/cdeps/cime_config/", rest), "framework",

  # CTSM-specific: treat cime + cdeps (incl FoX) as external/framework
  is_ctsm_external, "framework",

  # CTSM shared "shr_*" routines (framework by symbol)
  (model == "CTSM") & (grepl("^shr_", from) | grepl("^shr_", to)), "framework",

  # Tests (incl. SystemTests, case-insensitive)
  grepl("SystemTests/", rest, ignore.case = TRUE), "tests",
  grepl("/tests?/|^tests?/", rest, ignore.case = TRUE), "tests",
  grepl("(^|/)unit_test", rest, ignore.case = TRUE), "tests",

  # Drivers: Fortran code clearly in drivers directories
  grepl("(^|/)drivers(/|$)", rest) & language == "fortran", "driver",

  # Couplers: NUOPC / LILAC / CESM / cpl
  grepl("cpl_|/cesm/|/cpl/|/cpl_", rest), "coupler",

  # CLI / tools: scripts, setup, CTSM python utilities
  grepl("tools/|scripts/|cli\\.py$|setup\\.py$", rest), "cli_or_tool",
  grepl("^python/ctsm/", rest), "cli_or_tool",

  # Everything else: model core
  default = "model_core"
)]

# include flag for risk calculations ------------------------------------
```

```
# Key changes:
# - require in_model_tree
# - exclude external_lib/framework/tests/etc (already via component filter)
# - CTSM strict allowlist (optional)
all_callgraphs[, include_in_risk := (
  in_model_tree &
    language %chin% c("fortran", "python") &
    component %chin% c("model_core", "coupler") &
    (model != "CTSM" | !CTSM_STRICT | is_ctsm_allowed)
)]

# --- Sanity check: should now be FALSE for the SAX/FoX functions in CTSM
all_callgraphs[
  model == "CTSM" & include_in_risk &
    (grepl("sax|parseDTD|parsefile|parsestring|runParser", from, ignore.case = TRUE) |
       grepl("sax|parseDTD|parsefile|parsestring|runParser", to,   ignore.case = TRUE)),
  .(from, to, file, rest, component)
]
```

```
## Empty data.table (0 rows and 5 cols): from,to,file,rest,component
```

```
# SUmmarize ------------------------------------------------------------------

all_callgraphs[, .N, component]
```

```
##       component     N
##          <char> <int>
## 1:        ci_cd     5
## 2: external_lib   100
## 3:    framework  8603
## 4:        tests   194
## 5:  cli_or_tool   820
## 6:   model_core 15727
## 7:       driver    64
## 8:      coupler   299
```

```
all_callgraphs[, .N, include_in_risk]
```

```
##    include_in_risk     N
##             <lgcl> <int>
## 1:           FALSE  9786
## 2:            TRUE 16026
```

```
# Remove module calls ----------------------------------------------

all_callgraphs <- all_callgraphs[!(from %in% "<module>")] %>%
  .[include_in_risk == TRUE]

# LOAD CYCLOMATIC COMPLEXITY VALUES FOR FUNCTIONS AND SUBROUTINES ###############
```

```r
cc_unique <- fread("./datasets/cyclomatic_complexity_functions.csv")

# CREATE NETWORK FROM CALL GRAPHS ################################################

all_graphs <- all_callgraphs[, .(graph = list(as_tbl_graph(.SD, directed = TRUE))),
                             model]

# ADD NODE METRICS ###############################################################

# Define the weights to characterize risky nodes -----------------------------

alpha <- 0.6  # Weight to cyclomatic complexity
beta  <- 0.3  # Weight to in-degree (impact of bug upstream)
gamma <- 0.1  # Weight to betweenness (critical bridge)

# Add node metrics -----------------------------------------------------------

all_graphs[, graph:= Map(function(g, m) {

  comp_sub <- cc_unique[model == m]

  # mean cyclomatic complexity for this model & language ----------------------

  mean_cyclo <- mean(comp_sub$cyclomatic_complexity, na.rm = TRUE)

  g %>%
    activate(nodes) %>%

    # Left join with dataset with cyclomatic complexity values ----------------

  left_join(comp_sub, by = "name") %>%

    # replace NA cyclomatic_complexity with model-language mean ---------------

  mutate(cyclomatic_complexity = if (!is.na(mean_cyclo)) {

    ifelse(is.na(cyclomatic_complexity), mean_cyclo, cyclomatic_complexity)

  } else {

    # if even the mean is NA (all NA in comp_sub), leave as-is

    cyclomatic_complexity
  }
  ) %>%

    # Remove Python MODULE_AGG / CLASS_AGG nodes from this graph
```

```
    # because they are not callable ----------------------------------------

  filter(!(language == "python" & type %in% c("MODULE_AGG", "CLASS_AGG"))) %>%

    # Calculation of key network metrics -----------------------------------

  mutate(type = type,
         indeg = centrality_degree(mode = "in"),
         outdeg = centrality_degree(mode = "out"),
         btw = centrality_betweenness(directed = TRUE, weights = NULL),
         cyclo_sc = rescale(cyclomatic_complexity),
         indeg_sc = rescale(indeg),
         btw_sc = rescale(btw),
         risk_score = alpha * cyclo_sc + beta * indeg_sc + gamma * btw_sc)
},
graph, model)]

# EXTRACT NODE DF ##############################################################

all_graphs[, node_df:= lapply(graph, as_tibble, what = "nodes")]

# Export full node df ---------------------------------------------------------

full_node_df <- all_graphs %>%
  mutate(node_df = purrr::map(node_df, ~ select(.x, -model, -language))) %>%
  unnest(node_df) %>%
  select(-graph) %>%
  data.table()

write.xlsx(full_node_df, "full_node_df.xlsx")

# COMPUTE ALL PATHS AND THEIR RISK SCORES ######################################

all_graphs[, paths_tbl:= Map(all_paths_fun, node_df, graph)]

# Export full paths df --------------------------------------------------------

full_paths_df <- all_graphs %>%
  unnest(paths_tbl) %>%
  select(-c(graph, node_df))

write.xlsx(full_paths_df, "full_paths_df.xlsx")

# CONDUCT UNCERTAINTY AND SENSITIVITY ANALYSIS #################################

# Define sample size and order of effects -------------------------------------
```

```r
N <- 2^11
order <- "first"

# Run the function (we remove the vic and python model implementation because
# there are not paths) ----------------------------------------------------

all_graphs[!model == "VIC", uncertainty_sensitivity:= Map(full_ua_sa_risk_fun, node_df, paths_


# UNNEST APPROPRIATELY ##########################################################

unnested_df <- all_graphs %>%
  mutate(us_nodes = map(uncertainty_sensitivity, "nodes"),
         us_paths = map(uncertainty_sensitivity, "paths"))

# Create SA data frame ----------------------------------------------------

full_sa_df <- unnested_df %>%
  select(us_nodes) %>%
  unnest(cols = c(us_nodes)) %>%
  select(name, model, sensitivity_indices) %>%
  unnest(cols = c(sensitivity_indices))

# Export
fwrite(full_sa_df, "full_sa_df.csv")

# Create UA data frame ----------------------------------------------------

full_ua_df <- unnested_df %>%
  select(model, us_paths) %>%
  unnest(cols = c(us_paths)) %>%
  data.table()

# Export
fwrite(full_ua_df, "full_ua_df.csv")
```

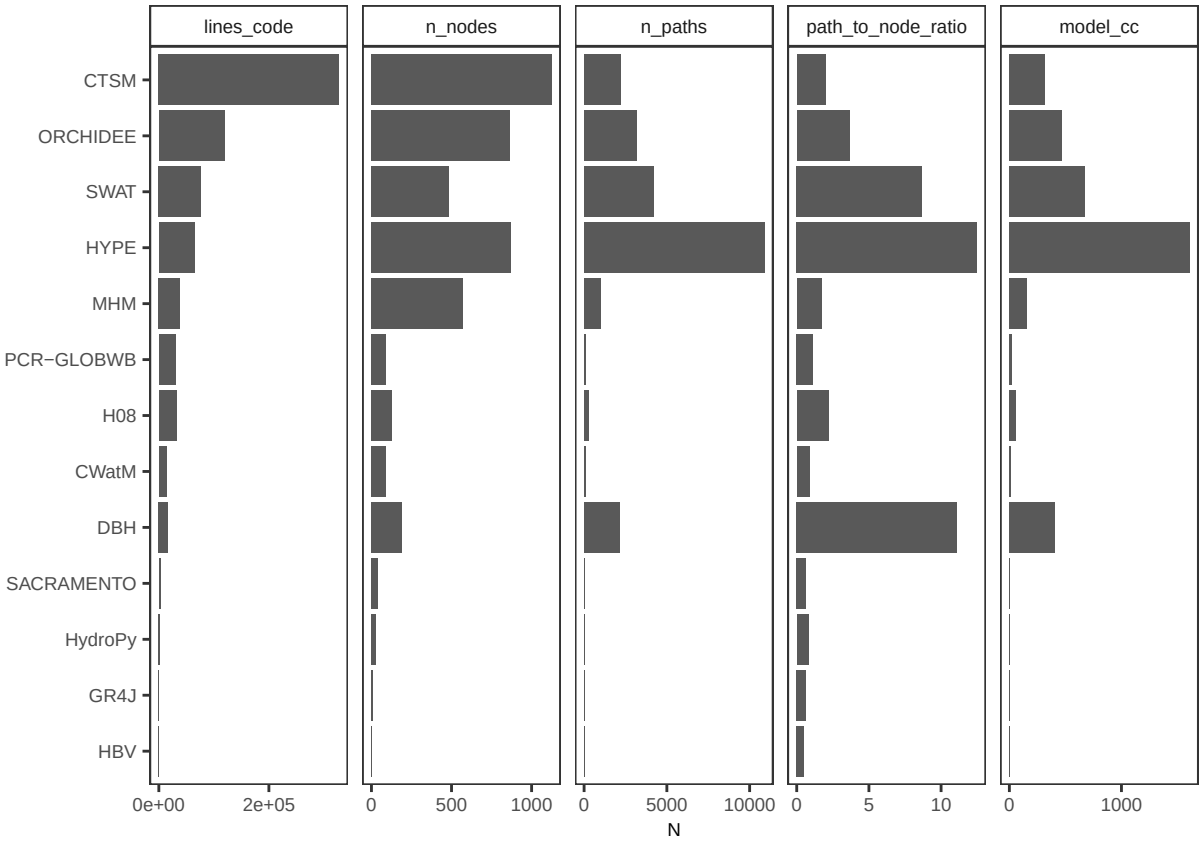
```r
# CALCULATE SOME DESCRIPTIVE METRICS ############################################

tmp <- data.table(full_paths_df)[, .(n_paths = .N), model] %>%
  .[order(-n_paths)]

tmp2 <- data.table(full_node_df)[, .(n_nodes = .N), model] %>%
  .[order(-n_nodes)]

# Path to node ratio: how interconnected the model is.
# Model_cc: Proxy for algorithmic complexity of model.
# Avg_path_length: Proxy for depth of dependency chains (risk-highway potential)
```

```r
# Model fragility: more (error) propagation routes.
models_metrics <- merge(tmp, tmp2) %>%
  .[, `:=`(path_to_node_ratio = n_paths / n_nodes,
           model_cc = n_paths / log(n_nodes),
           avg_path_length = n_nodes / log(n_paths + 1),
           model_fragility_index = n_paths / (n_nodes * (n_nodes - 1)))]

models_metrics


# Read descriptive_stats_file ----------------------------------------------------

num_cols <- c("files","functions","modules","lines","lines_code","lines_comments")

descriptive_stats <- data.table(read_xlsx("./datasets/descriptive_statistics/descriptive_statis

descriptive_stats <- dcast(melt(descriptive_stats, id.vars="model", measure.vars=num_cols),
        model ~ variable, value.var ="value", fun.aggregate = function(z) sum(z, na.rm = TRUE))
  .[, lines_function := lines_code / functions]

all_descriptive_df <- merge(models_metrics, descriptive_stats)

# Sort by model ------------------------------------------------------------------
model_ordered <- all_descriptive_df[, sum(lines), model] %>%
  .[order(V1)]

# Plot descriptive measures per model --------------------------------------------

plot_descriptive <- melt(all_descriptive_df, measure.vars = c("lines_code", "n_nodes", "n_paths
                                                              "path_to_node_ratio", "model_cc"
  .[, model:= factor(model, levels = model_ordered[, model])] %>%
  ggplot(., aes(model, value)) +
  geom_col() +
  coord_flip() +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  scale_fill_manual(values = color_languages, name = "") +
  facet_wrap(~ variable, ncol = 7, scales = "free_x") +
  labs(x = "", y = "N") +
  theme_AP() +
  theme(legend.position = c(0.1, 0.3))
```

```
## Warning in melt.data.table(all_descriptive_df, measure.vars = c("lines_code", :
## 'measure.vars' [lines_code, n_nodes, n_paths, path_to_node_ratio, ...] are not
## all of the same type. By order of hierarchy, the molten data value column will
## be of type 'double'. All measure variables not of type 'double' will be coerced
## too. Check DETAILS in ?melt.data.table for more on coercion.
```

```
plot_descriptive
```

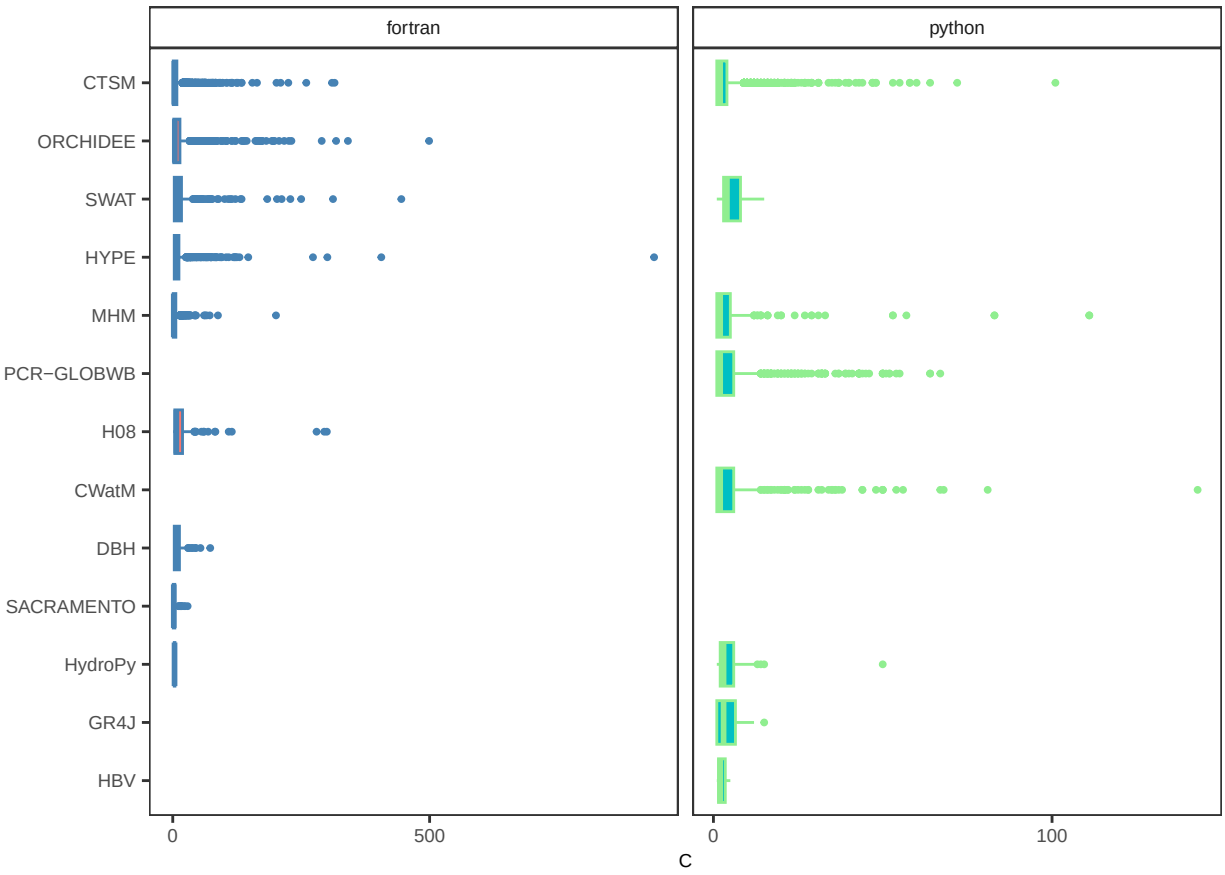
```
## Warning: No shared levels found between `names(values)` of the manual scale and the
## data's fill values.
```




```
# METRICS AT THE FILE AND FUNCTION LEVEL ########################################

folder <- "./datasets/results_per_function"

# Get names of files ------------------------------------------------------------

csv_files <- list.files(path = folder, pattern = "\\.csv$", full.names = TRUE)

# Split into file_metrics and func_metrics --------------------------------------

file_metric_files <- grep("file_metrics", csv_files, value = TRUE)
func_metric_files <- grep("func_metrics", csv_files, value = TRUE)

# Build one named list ----------------------------------------------------------

list_metrics <- list(file_metrics = setNames(lapply(file_metric_files, fread),
                                             basename(file_metric_files)),
                     func_metrics = setNames(lapply(func_metric_files, fread),
                                             basename(func_metric_files)))
```

```r
# Create function to combine files ----------------------------------------

make_combined <- function(subset_list, pattern) {
  rbindlist(subset_list[grep(pattern, names(subset_list))], idcol = "source_file")
}

# Combine files -----------------------------------------------------------

metrics_combined <- list(file_fortran = make_combined(list_metrics$file_metrics, "fortran"),
                         file_python = make_combined(list_metrics$file_metrics, "python"),
                         func_fortran = make_combined(list_metrics$func_metrics, "fortran"),
                         func_python = make_combined(list_metrics$func_metrics, "python"))


# Functions to extract name of model and language from file ---------------

extract_model <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\2", x)

extract_lang  <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\3", x)

# Extract name of model and language --------------------------------------

metrics_combined <- lapply(metrics_combined, function(dt) {
  dt[, source_file:= sub("\\.csv$", "", basename(source_file))]
  dt[, model:= extract_model(source_file)]
  dt[, language:= extract_lang(source_file)]
  dt
})

# Add column of complexity category ---------------------------------------

metrics_combined <- lapply(names(metrics_combined), function(nm) {
  dt <- as.data.table(metrics_combined[[nm]])
  if (grepl("^func_", nm) && "cyclomatic_complexity" %in% names(dt)) {
    dt[, complexity_category := cut(
      cyclomatic_complexity,
      breaks = c(-Inf, 10, 20, 50, Inf),
      labels = c("b1","b2","b3","b4")
    )]
  }
  dt
}) |> setNames(names(metrics_combined))

# Define labels -----------------------------------------------------------
```

```r
lab_expr <- c(b1 = expression(C %in% "(" * 0 * ", 10" * "]"),
              b2 = expression(C %in% "(" * 10 * ", 20" * "]"),
              b3 = expression(C %in% "(" * 20 * ", 50" * "]"),
              b4 = expression(C %in% "(" * 50 * ", " * infinity * ")"))

# Define vector to exclude classes that are not functions ----------------------

excluded_classes_vec <- c("MODULE_AGG", "CLASS_AGG")

# PLOT ########################################################################

## ----plot_c_model, dependson="read_metrics_function_data", fig.height=2.2, fig.width=3.1----

plot_c_model <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, `function`, cyclomatic_complexity, loc, bugs, type)]) %>%
  rbindlist() %>%
  .[!type %in% excluded_classes_vec] %>%
  .[, model:= factor(model, levels = model_ordered[, model])] %>%
  na.omit() %>%
  ggplot(., aes(model, cyclomatic_complexity, fill = language, color = language)) +
  geom_boxplot(outlier.size = 0.7) +
  coord_flip() +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 2)) +
  facet_wrap(~language, scales = "free_x") +
  labs(x = "", y = "C") +
  theme_AP() +
  scale_color_manual(values = color_languages) +
  theme(legend.position = "none",
        plot.margin = margin(0, 2, 0, 0))

plot_c_model
```




```
## ----plot_scatter_and_bar, dependson="read_metrics_function_data", fig.height=2.5, fig.width=
```

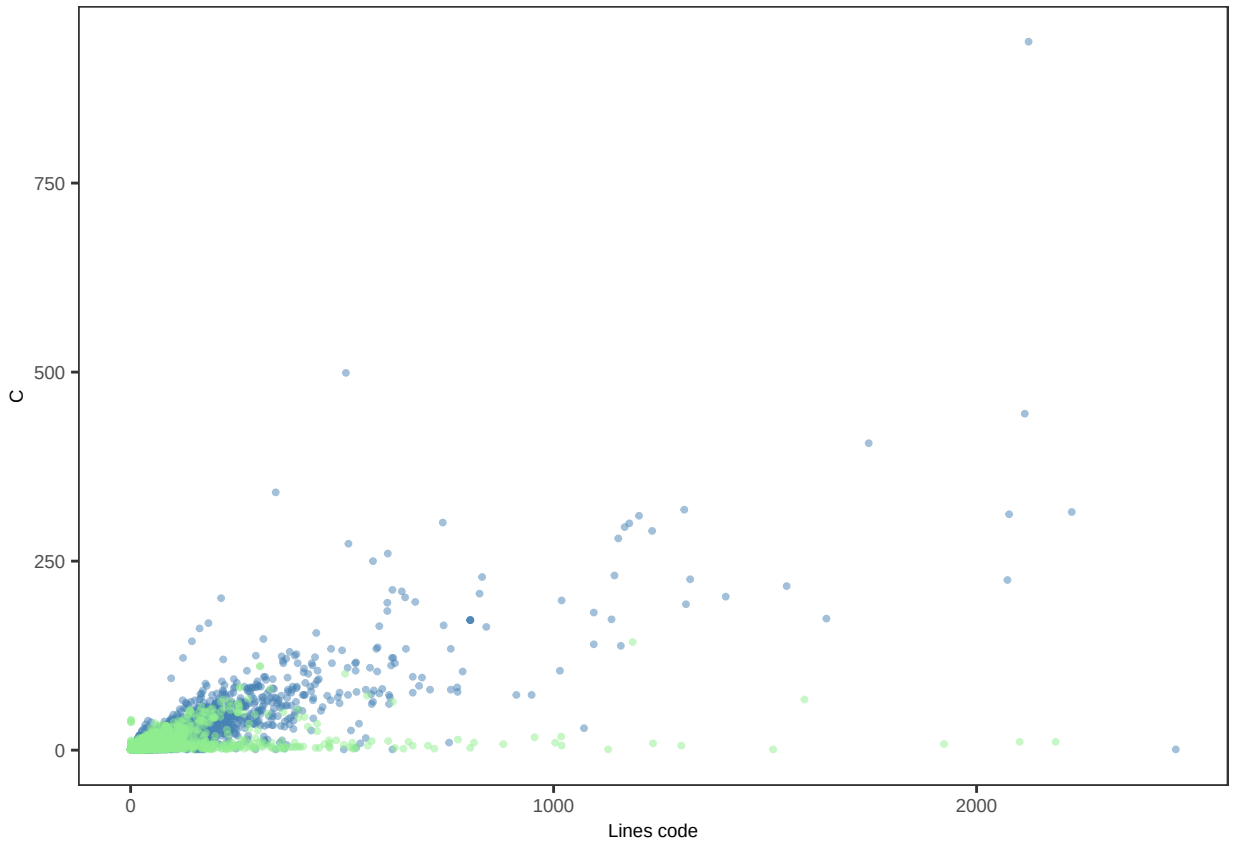
```
# Scatterplot cyclomatic vs lines of code ------------------------------------
```

```
plot_c_vs_loc <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(loc, cyclomatic_complexity, language)]) %>%
  rbindlist() %>%
  ggplot(., aes(loc, cyclomatic_complexity, color = language)) +
  geom_point(alpha = 0.5, size = 0.7) +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = "Lines code", y = "C") +
  scale_color_manual(values = color_languages) +
  theme_AP() +
  scale_x_continuous(breaks = breaks_pretty(n = 2)) +
  theme(legend.position = "none")
```

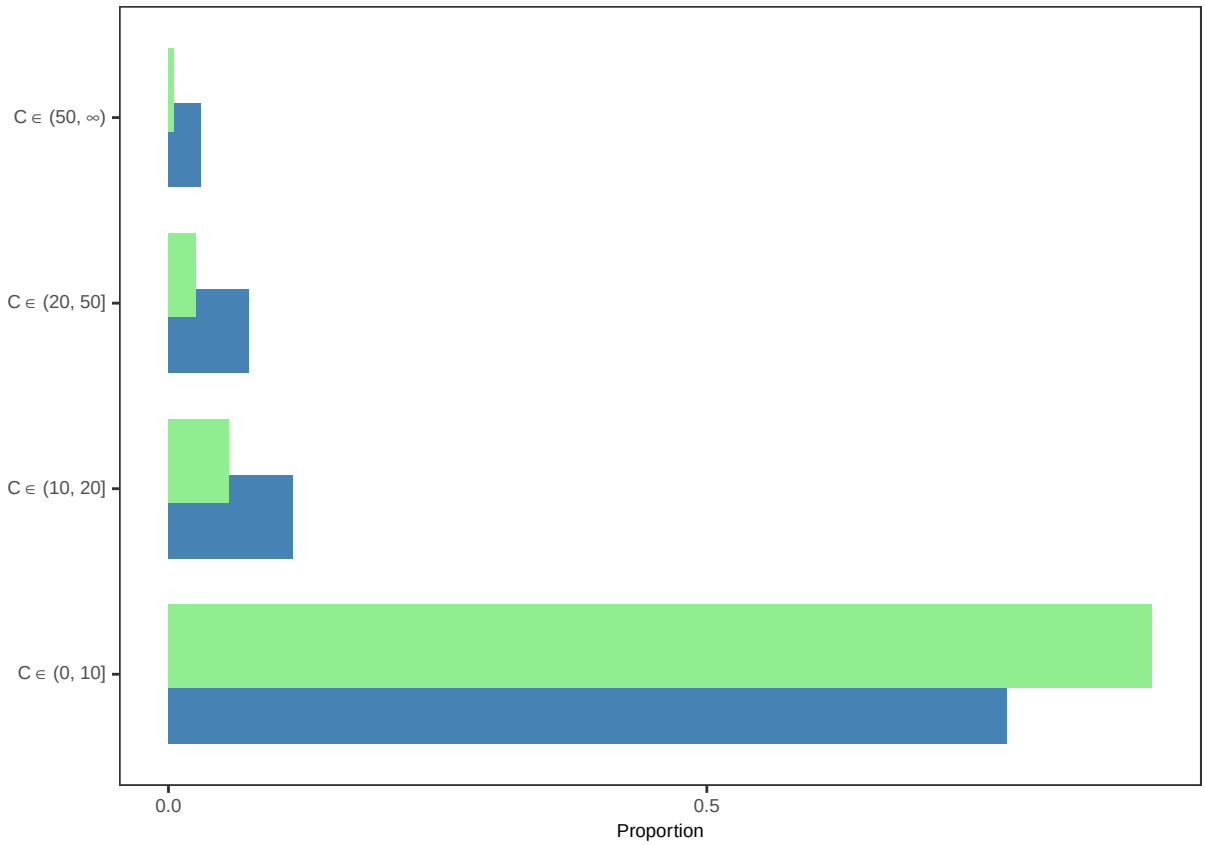
```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```

```
plot_c_vs_loc
```

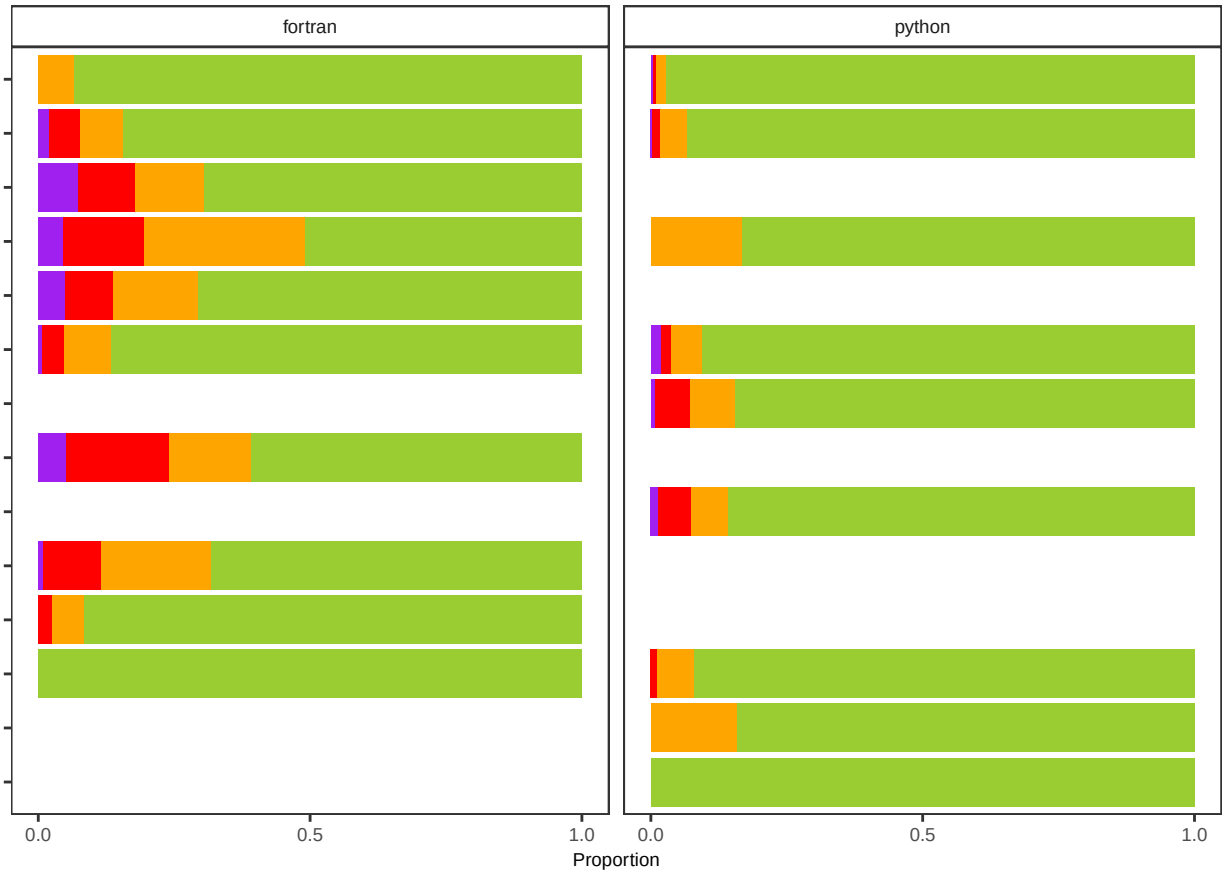
```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```
# Count & proportion ------------------------------------------------------------

plot_bar_cyclomatic <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(complexity_category, language, type)]) %>%
  rbindlist() %>%
  .[!type %in% excluded_classes_vec] %>%
  .[, .N, .(complexity_category, language)] %>%
  .[, proportion := N / sum(N), language] %>%
  ggplot(., aes(complexity_category, proportion, fill = language)) +
  geom_bar(stat = "identity", position = position_dodge(0.6)) +
  scale_fill_manual(values = color_languages) +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 3)) +
  scale_x_discrete(labels = lab_expr) +
  labs(x = "", y = "Proportion") +
  coord_flip() +
  theme_AP() +
  theme(legend.position = "none")

plot_bar_cyclomatic
```




```
plot_bar_category <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, complexity_category, type)]) %>%
  rbindlist() %>%
  .[!type %in% excluded_classes_vec] %>%
  .[, model:= factor(model, levels = model_ordered[, model])] %>%
  .[, .N, .(model, language, complexity_category)] %>%
  .[, proportion := N / sum(N), .(language, model)] %>%
  ggplot(., aes(model, proportion, fill = complexity_category)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("yellowgreen", "orange", "red", "purple"),
                    labels = lab_expr,
                    name = "") +
  facet_wrap(~language) +
  labs(x = "", y = "Proportion") +
  coord_flip() +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 3)) +
  theme_AP() +
  theme(legend.position = "none") +
  theme(axis.text.y = element_blank(),
        legend.text = element_text(size = 7),
        plot.margin = margin(0, 0, 0, 2))

plot_bar_category
```



3 Descriptive plots

```
# MERGE FIGURES ######################################################################

legend2 <- get_legend_fun(plot_bar_category + theme(legend.position = "top"))
```

```
## Warning: `is.ggplot()` was deprecated in ggplot2 3.5.2.
## i Please use `is_ggplot()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

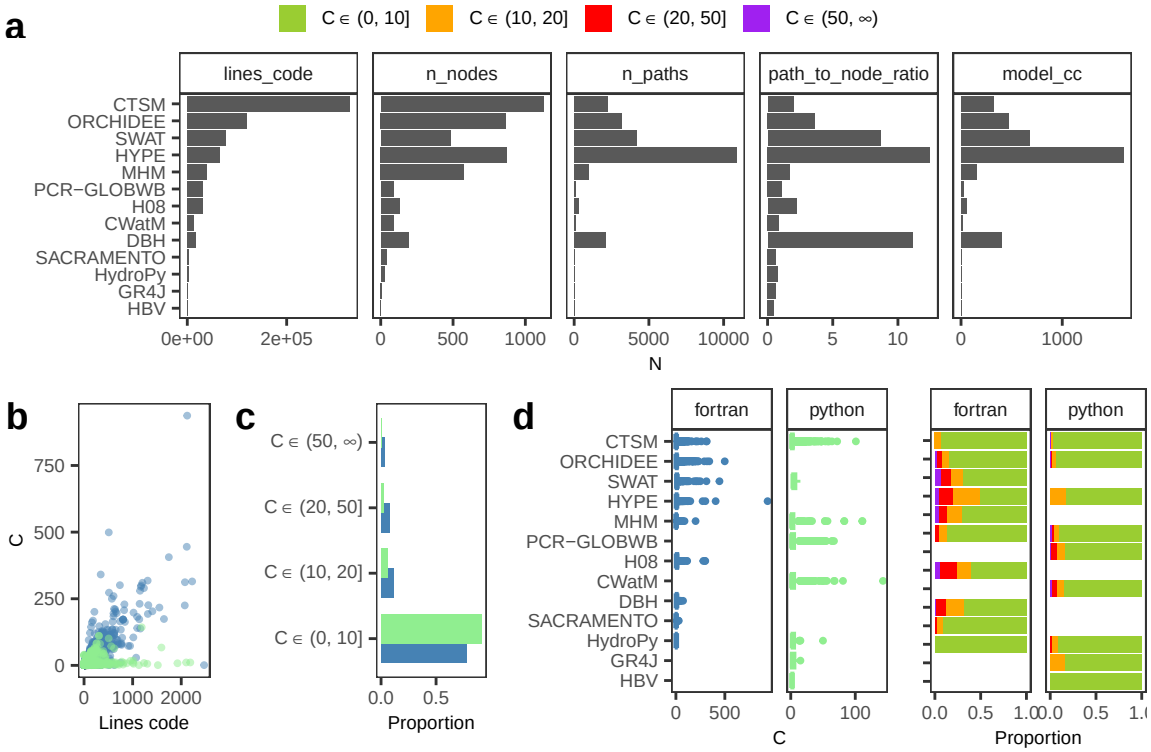
```
top_plot <- plot_grid(legend2, plot_descriptive, rel_heights = c(0.1, 0.9), ncol = 1,
                      labels = "a")
```

```
## Warning: No shared levels found between `names(values)` of the manual scale and the
## data's fill values.
```

```
bottom <- plot_grid(plot_c_vs_loc, plot_bar_cyclomatic, plot_c_model,
                    plot_bar_category, ncol = 4, rel_widths = c(0.2, 0.24, 0.34, 0.22),
                    labels = c("b", "c", "d"))
```

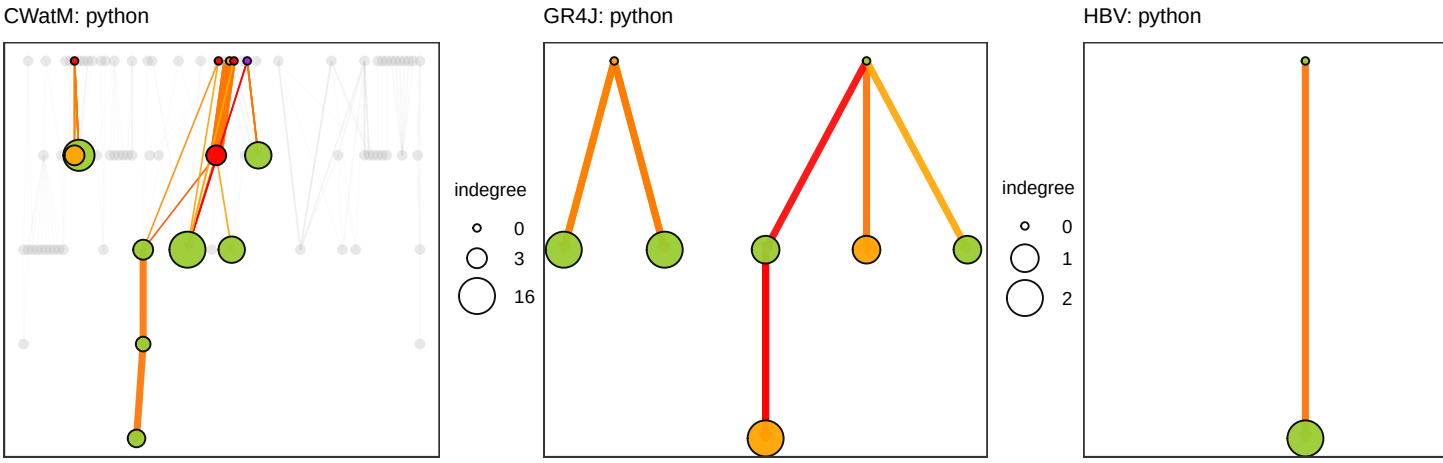
```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

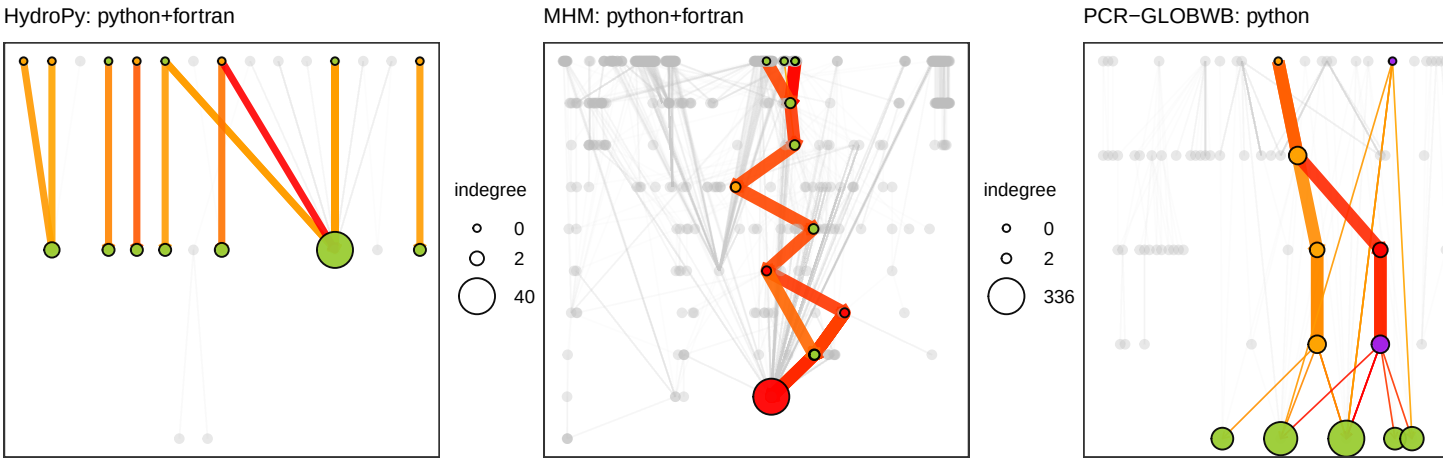
```
plot_grid(top_plot, bottom, ncol = 1, rel_heights = c(0.52, 0.48), align = "h",
          axis = "tb")
```



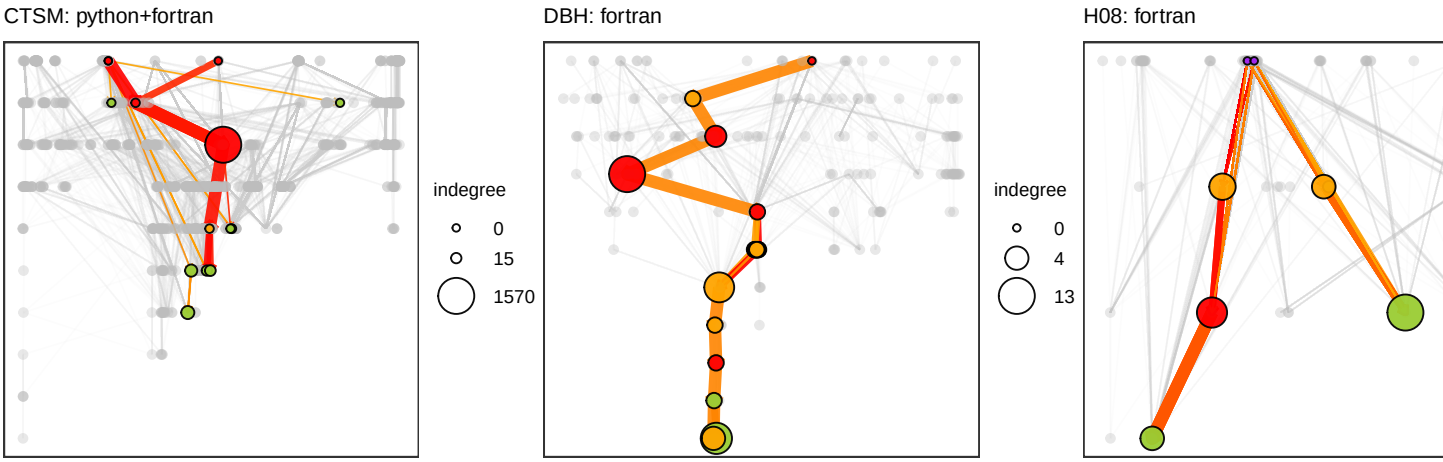
4 Figures

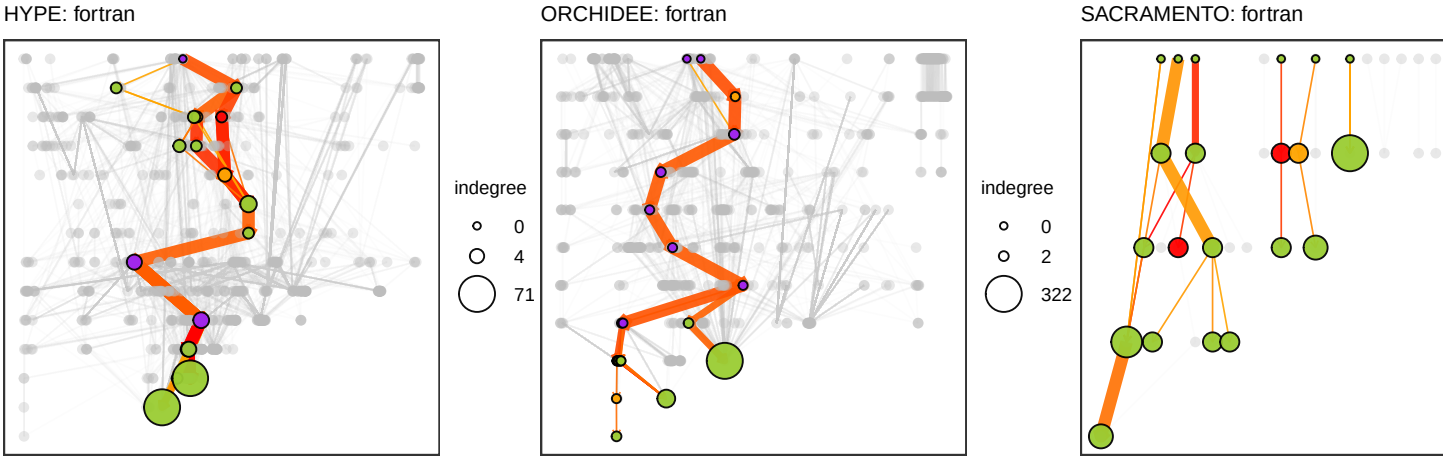
```
# PLOT FIGURES ###################################################################

# Define language of models ------------------------------------------------

python_models <- c("CWatM", "GR4J", "HBV", "PCR-GLOBWB")
fortran_models <- c("ORCHIDEE", "H08", "HYPE", "DBH", "SACRAMENTO")
python_and_fortran <- c("CTSM", "SWAT", "MHM", "HydroPy", "VIC")

all_graphs[, language := fcase(model %chin% python_models, "python",
                               model %chin% fortran_models, "fortran",
                               model %chin% python_and_fortran, "python+fortran",
                               default = NA_character_)]

# Plot graphs --------------------------------------------------------------

set.seed(seed)


# Thickness of edge:frequency across top-10 riski paths
# Color of edge: mean risk of paths using that edge

all_graphs <- all_graphs[, plot_obj:= mapply(plot_top_paths_fun, call_g = graph,
                                             paths_tbl = paths_tbl, model.name = model,
                                             language = language, SIMPLIFY = FALSE)]
```



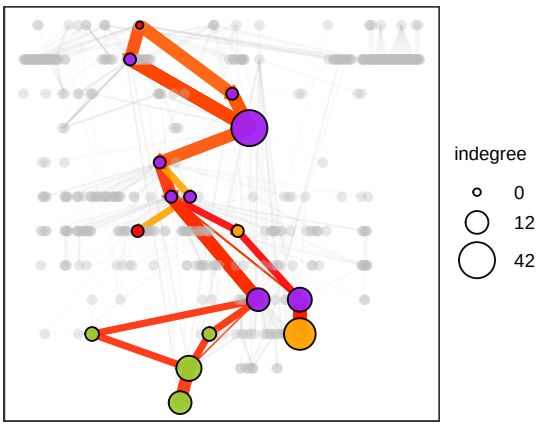



```
## No paths in paths_tbl; skipping plot for: VIC
```





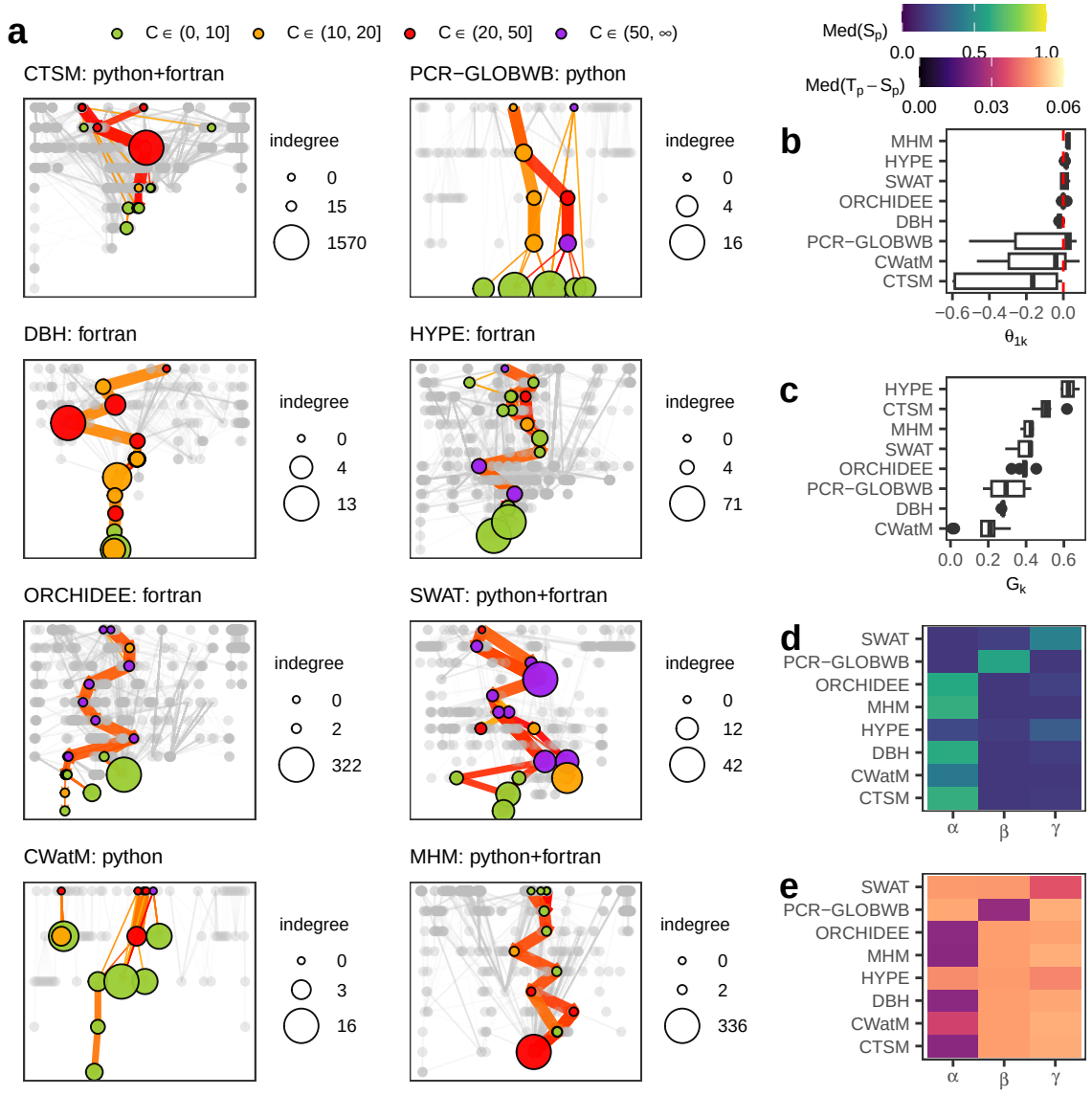
SWAT: python+fortran




```
# PLOT OTHER FIGURES ##############################################################

selected_models <- data.table(model = c("CTSM", "PCR-GLOBWB", "DBH", "HYPE",
                                        "ORCHIDEE", "SWAT", "CWatM", "MHM"))

# Plot call graphs ------------------------------------------------------------

tmp <- all_graphs[selected_models, on = .(model)]
plot_all_risky_paths <- plot_grid(plotlist = tmp$plot_obj, ncol = 2, align = "hv")

# Plot risk_slope -------------------------------------------------------------

a <- full_paths_df %>%
  data.table() %>%
  .[selected_models, on = .(model)] %>%
  .[order(-p_path_fail), .SD[1:10], model] %>%
  ggplot(., aes(reorder(model, risk_slope), risk_slope)) +
  geom_boxplot() +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  geom_hline(yintercept = 0, lty = 2, color = "red") +
  coord_flip() +
  labs(x = "", y = expression(theta[1*k])) +
  theme_AP()

# Plot Gini metric ------------------------------------------------------------

b <- full_paths_df %>%
  data.table() %>%
  .[selected_models, on = .(model)] %>%
  .[order(-p_path_fail), .SD[1:10], model] %>%
  ggplot(., aes(reorder(model, gini_node_risk), gini_node_risk)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "", y = expression(G[k])) +
```

```
  theme_AP()

# Plot Si values ------------------------------------------------------------

c <- full_sa_df %>%
  data.table() %>%
  .[selected_models, on = .(model)] %>%
  .[sensitivity == "Si", .(median = median(original, na.rm = TRUE)), .(model, parameters)] %>%
  ggplot(., aes(x = parameters, y = model, fill = median)) +
  geom_tile() +
  scale_fill_viridis_c(name = expression("Med(" * S[p] * ")"),
                       limits = c(0, 1),
                       breaks = c(0, 0.5, 1)) +
  scale_x_discrete(labels = c(a_raw = expression(alpha),
                              b_raw = expression(beta),
                              c_raw = expression(gamma))) +
  labs(x = NULL, y = NULL) +
  theme_AP() +
  theme(legend.position = "none")

# Plot interaction strength -------------------------------------------------

tmp <- full_sa_df %>%
  data.table() %>%
  dcast(., name + model  + parameters ~ sensitivity, value.var = "original",
        fun.aggregate= mean) %>%
  .[, interaction:= Ti - Si]

d <- tmp %>%
  .[selected_models, on = .(model)] %>%
  .[, .(median = median(interaction, na.rm = TRUE)), .(parameters, model)] %>%
  ggplot(., aes(x = parameters, y = model, fill = median)) +
  geom_tile() +
  scale_x_discrete(labels = c(a_raw = expression(alpha),
                              b_raw = expression(beta),
                              c_raw = expression(gamma))) +
  scale_fill_viridis_c(name   = expression("Med(" * T[p] - S[p] * ")"),
                       limits = c(0, 0.06),
                       breaks = c(0, 0.03, 0.06),
                       option = "magma") +
  labs(x = NULL, y = NULL) +
  theme_AP() +
  theme(legend.position = "none")


# MERGE ########################################################################
```

```
p_for_fill_legend <- all_graphs$plot_obj[[6]] +
  guides(size = "none", fill = guide_legend(title = ""))
fill_legend <- get_legend_fun(p_for_fill_legend + theme(legend.position = "top"))
plot_top_paths <- plot_grid(fill_legend, plot_all_risky_paths, ncol = 1, rel_heights = c(0.05
                            labels = "a")
heatmap_legend <- get_legend(c + theme(legend.position = "top"))
ti_legend <- get_legend(d + theme(legend.position = "top"))
dada <- plot_grid(a, b, c, d, ncol = 1, labels = c("b", "c", "d", "e"))
all_legends <- plot_grid(heatmap_legend, ti_legend, ncol = 1)
right_plot <- plot_grid(all_legends, dada, ncol = 1, rel_heights = c(0.1, 0.9))
plot_grid(plot_top_paths, right_plot, ncol = 2, rel_widths = c(0.7, 0.3))
```



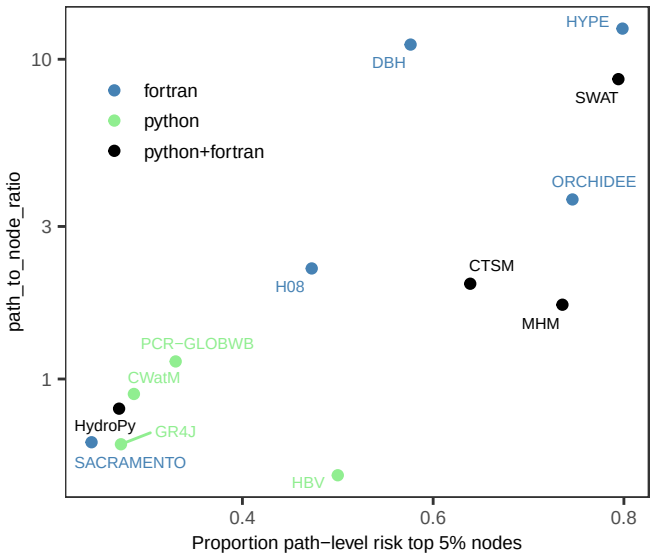

```
# PATH-LEVEL RISK ACCOUNTED FOR THE TOP 5% NODES ################################

setDT(full_paths_df)
```

```r
# To long format ------------------------------------------------------------

paths_long <- full_paths_df[, .(node = unlist(path_nodes),
                                p_path_fail = p_path_fail,
                                gini_node_risk = gini_node_risk,
                                risk_slope = risk_slope,
                                risk_mean = risk_mean,
                                risk_sum  = risk_sum),
                            .(model, path_id)]


# Aggregate at function level -----------------------------------------------

node_from_paths <- paths_long[, .(n_paths = .N,
                                  mean_p_path = mean(p_path_fail, na.rm = TRUE),
                                  max_p_path = max(p_path_fail,  na.rm = TRUE),
                                  sum_p_path = sum(p_path_fail,  na.rm = TRUE),
                                  mean_gini = mean(gini_node_risk, na.rm = TRUE),
                                  mean_slope = mean(risk_slope, na.rm = TRUE),
                                  mean_risksum = mean(risk_sum, na.rm = TRUE)),
                              .(model, node)]

# Join with nodes -----------------------------------------------------------

node_summary <- merge(node_from_paths, full_node_df, by.x = c("model", "node"),
                      by.y = c("model", "name"), all.x = TRUE)

# Calculate risk mass -------------------------------------------------------

node_summary[, risk_mass:= mean_p_path * n_paths]

# share of risk mass in top X% nodes, per model .----------------------------


top_share <- function(X = 0.05) {

  node_summary[!is.na(risk_mass) & risk_mass >= 0, {
    dt <- .SD[order(-risk_mass)]
    n_top <- max(1L, ceiling(.N * X))
    .(X = X, n_nodes = .N, n_top = n_top,
      share_risk_mass_topX = sum(dt$risk_mass[1:n_top]) / sum(dt$risk_mass))
  },
  model
  ]
}

# Run function --------------------------------------------------------------
```

```r
tmp <- top_share(0.05) %>%
  .[order(-share_risk_mass_topX)]

tmp
```

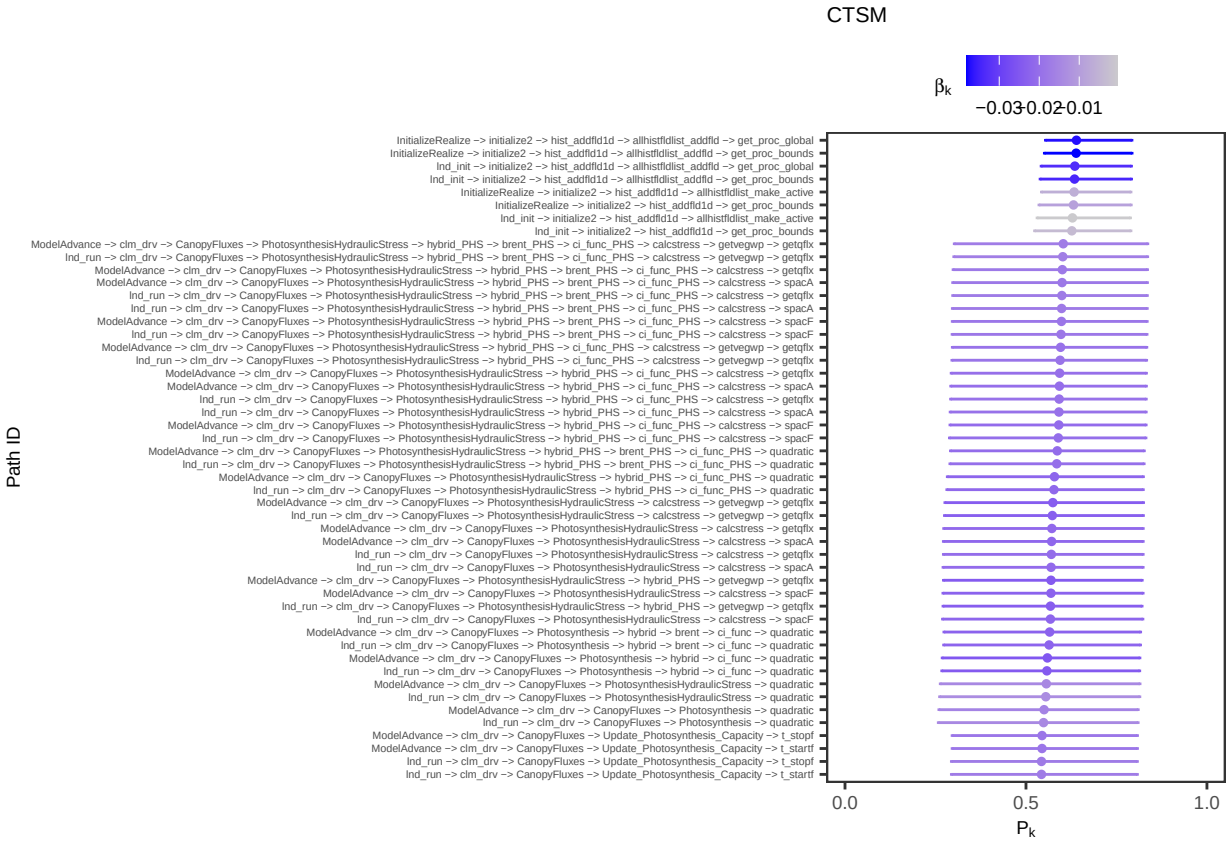
```
##          model     X n_nodes n_top share_risk_mass_topX
##         <char> <num>   <int> <num>                <num>
##  1:       HYPE  0.05     872    44            0.7985366
##  2:       SWAT  0.05     481    25            0.7942209
##  3:   ORCHIDEE  0.05     865    44            0.7460796
##  4:        MHM  0.05     565    29            0.7355383
##  5:       CTSM  0.05    1124    57            0.6388660
##  6:        DBH  0.05     191    10            0.5763146
##  7:        HBV  0.05       2     1            0.5000000
##  8:        H08  0.05     129     7            0.4726677
##  9: PCR-GLOBWB  0.05      89     5            0.3300054
## 10:      CWatM  0.05      84     5            0.2862374
## 11:       GR4J  0.05       8     1            0.2726486
## 12:    HydroPy  0.05      26     2            0.2706942
## 13: SACRAMENTO  0.05      41     3            0.2418015
```

```r
# Plot--------------------------------------------------------------------------

color_languages_2 <- c("fortran" = "steelblue", "python" = "lightgreen",
                       "python+fortran" = "black")

merge(all_descriptive_df, tmp, by = "model") %>%
  .[, language := fcase(model %chin% python_models, "python",
                        model %chin% fortran_models, "fortran",
                        model %chin% python_and_fortran, "python+fortran",
                        default = NA_character_)] %>%
  ggplot(., aes(share_risk_mass_topX, path_to_node_ratio, color = language)) +
  geom_point() +
  scale_color_manual(values = color_languages_2, name = "") +
  geom_text_repel(aes(label = model), size = 2, max.overlaps = Inf, show.legend = FALSE) +
  scale_y_log10() +
  labs(x = "Proportion path-level risk top 5% nodes", y = "path_to_node_ratio") +
  theme_AP() +
  theme(legend.position = c(0.2, 0.8))
```

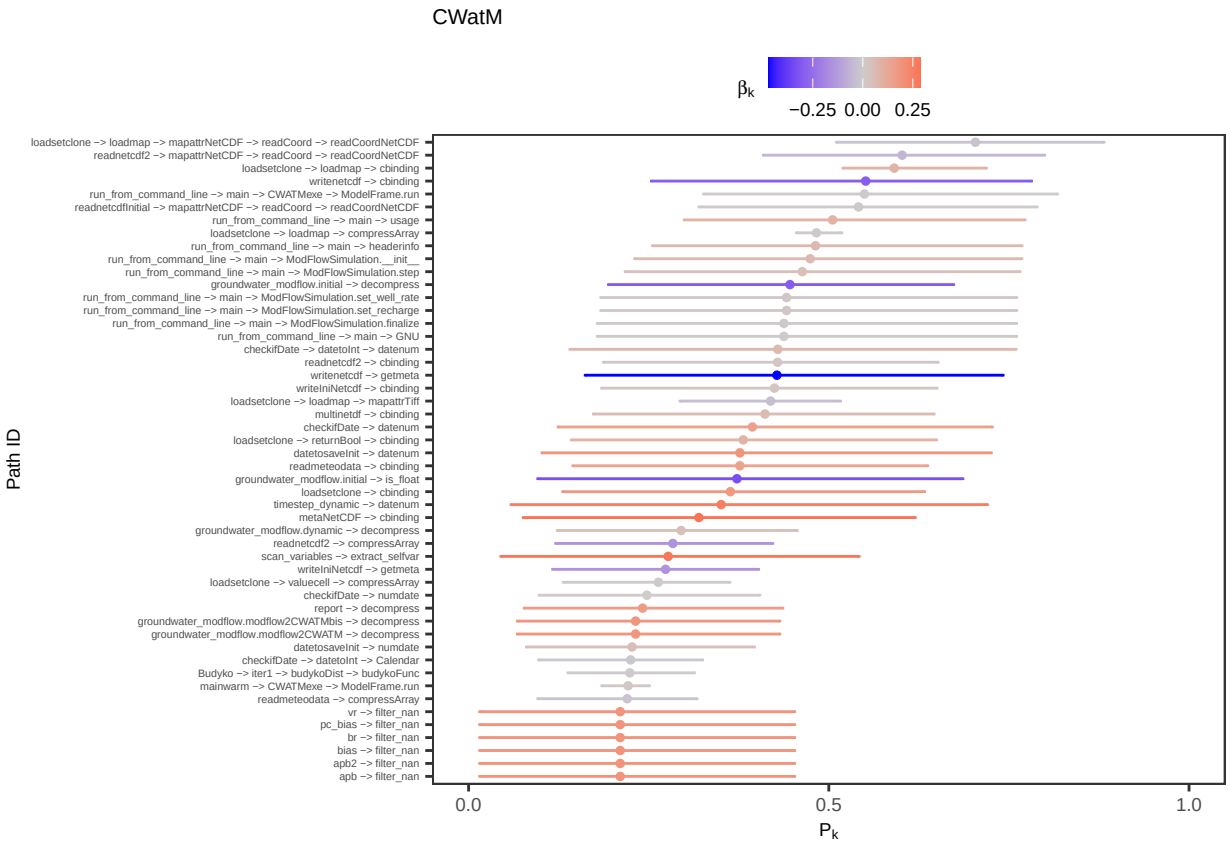



```
# PLOT THE TOP 50 PATHS PER MODEL ################################################

tmp <- full_ua_df %>%
  .[order(-P_k_mean), .SD[1:50], model] %>%
  na.omit() %>%
  split(., .$model)

# Plot in a for loop ------------------------------------------------------------

out <- list()

for ( i in 1:length(tmp)) {

  out[[i]] <- ggplot(tmp[[i]], aes(P_k_mean, reorder(path_str, P_k_mean), color = risk_slope))
    geom_point(size = 1) +
    geom_errorbar(aes(xmin = P_k_q025, xmax = P_k_q975), height = 0.2) +
    scale_color_gradient2(low = "blue", mid = "grey80", high = "red", midpoint = 0,
                          name = expression(beta[k])) +
    labs(y = "Path ID", x = expression(P[k])) +
    theme_AP() +
    scale_x_continuous(breaks = breaks_pretty(n = 3),
                       limits = c(0, 1)) +
    theme(axis.text.y = element_text(size = 4),
          legend.position = "top") +
    ggtitle(names(tmp[i]))

}

out
```

```
## [[1]]

## `height` was translated to `width`.
```

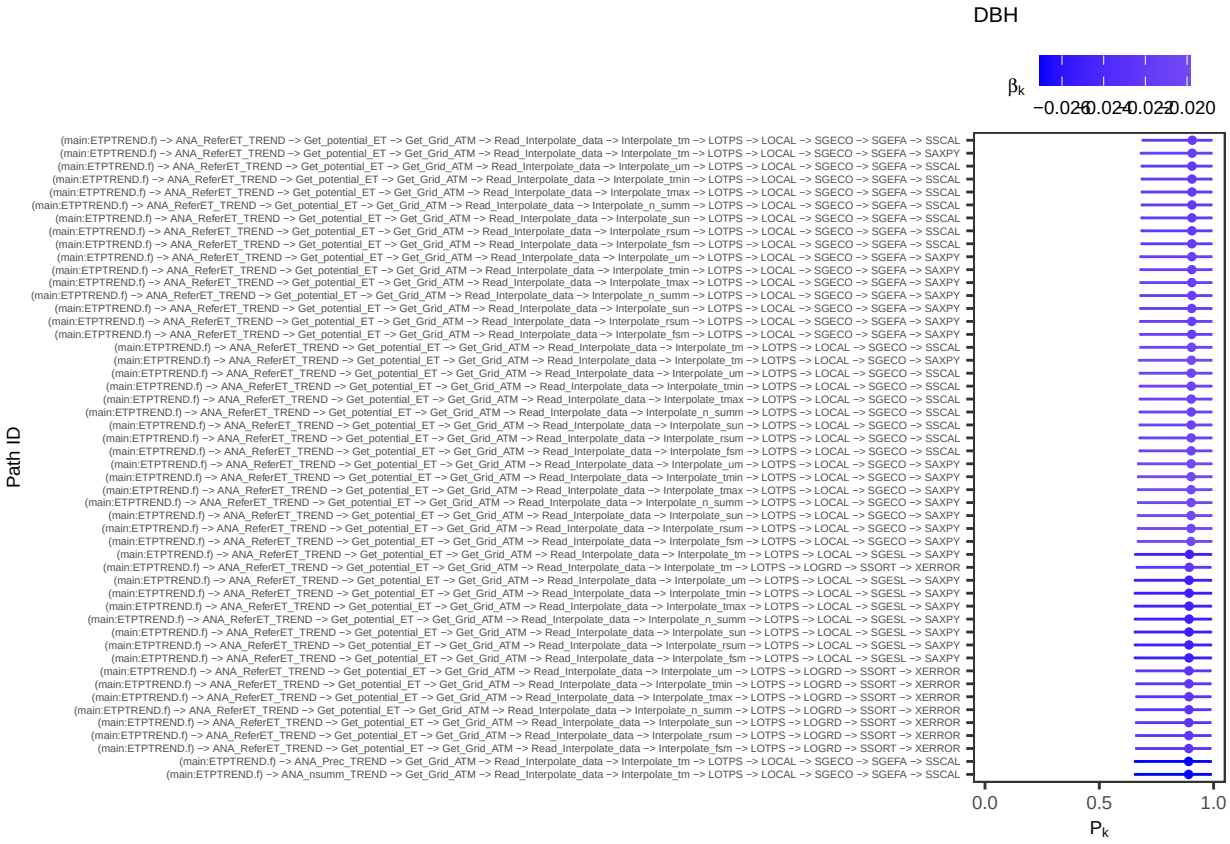



```
##
## [[2]]

## `height` was translated to `width`.
```

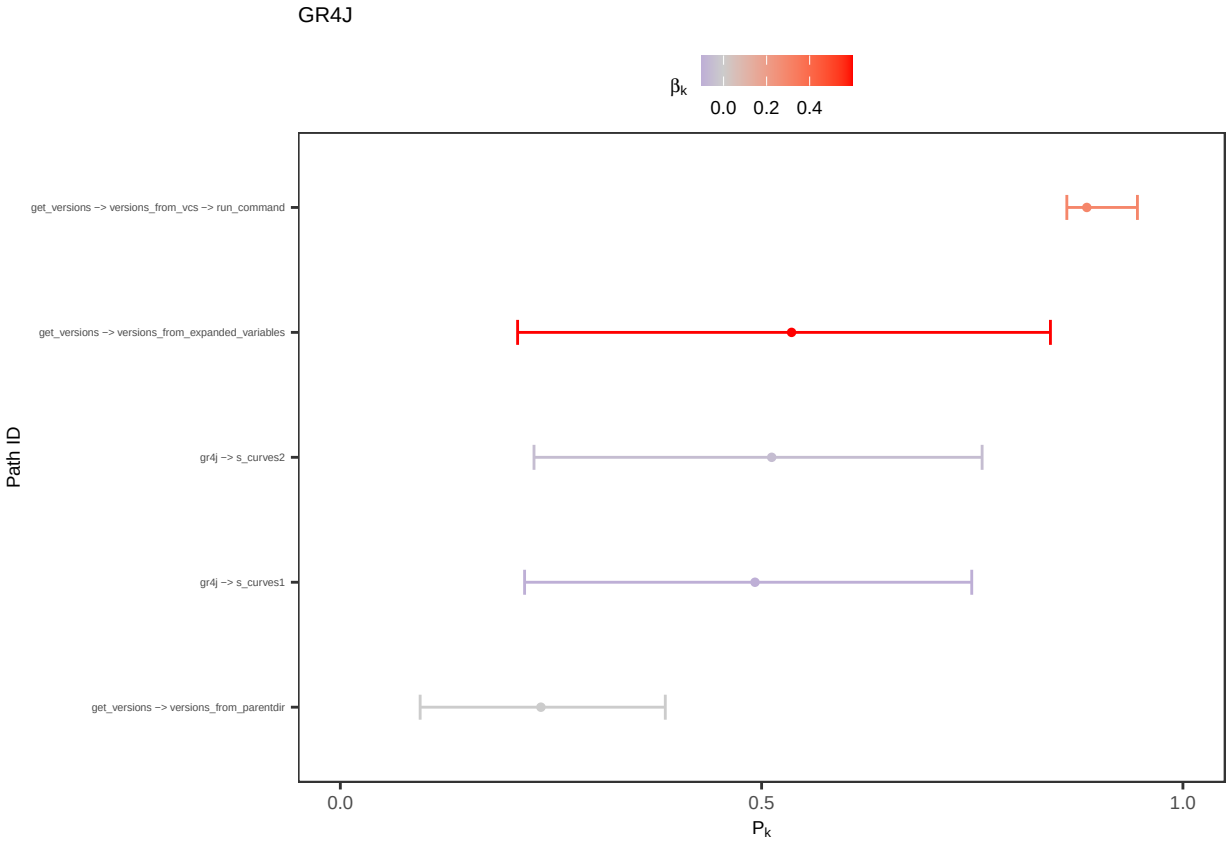



```
##
## [[3]]
```

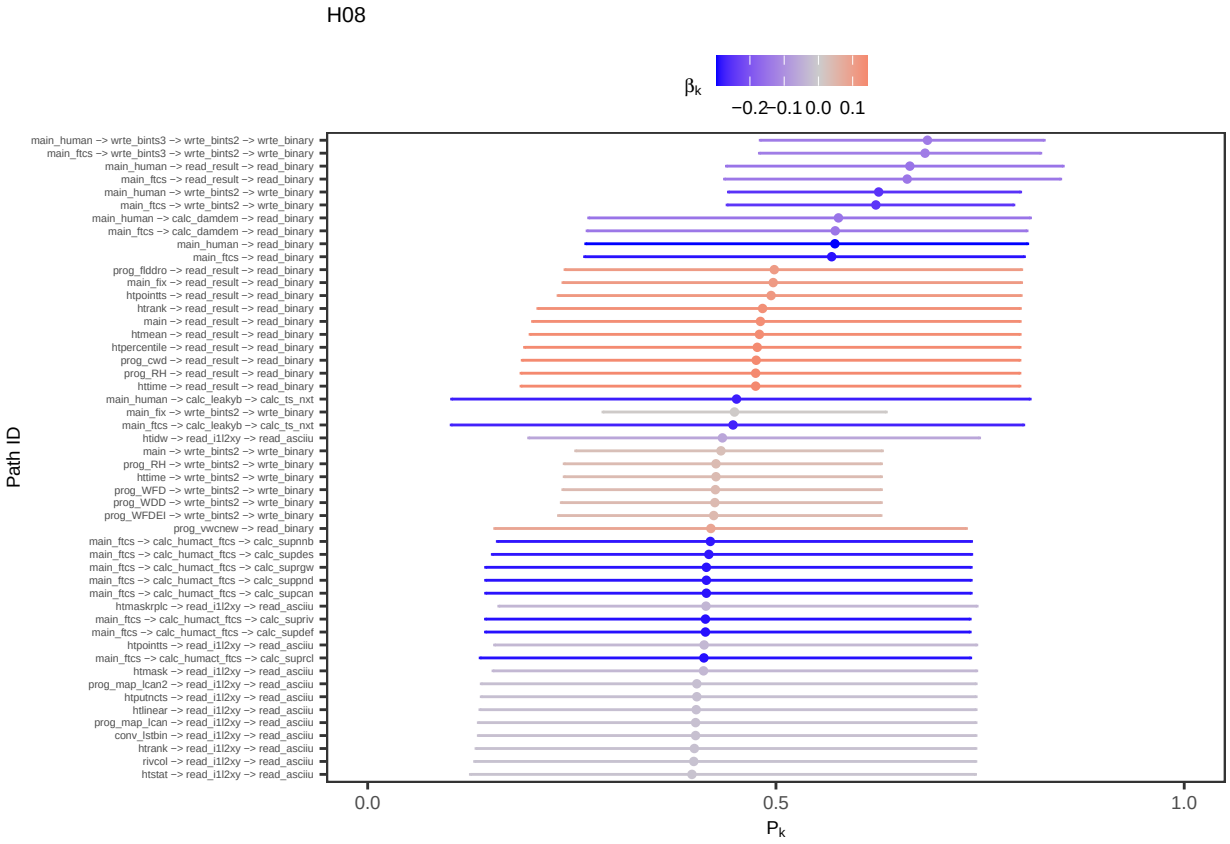
```
## `height` was translated to `width`.
```



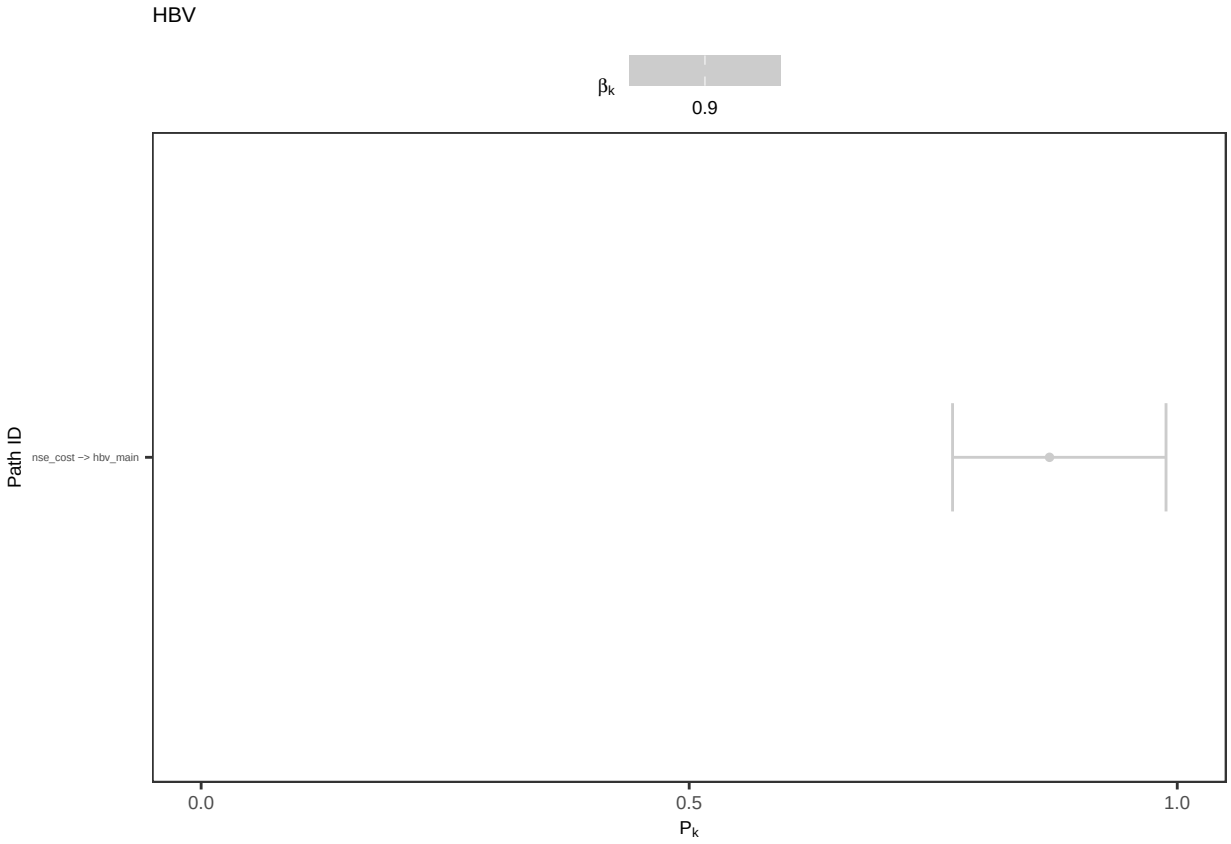

```
##
## [[4]]

## `height` was translated to `width`.
```



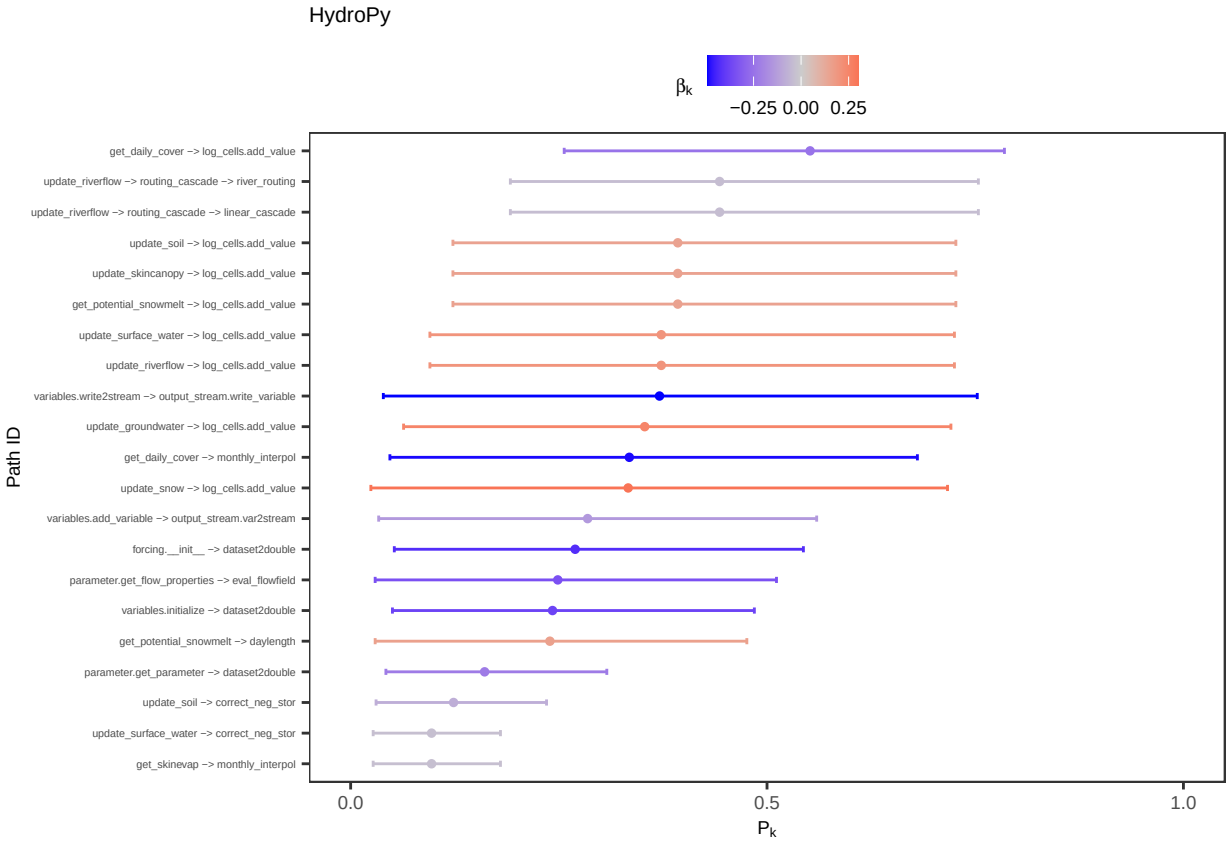

```
##
## [[5]]

## `height` was translated to `width`.
```




```
##
## [[6]]

## `height` was translated to `width`.
```



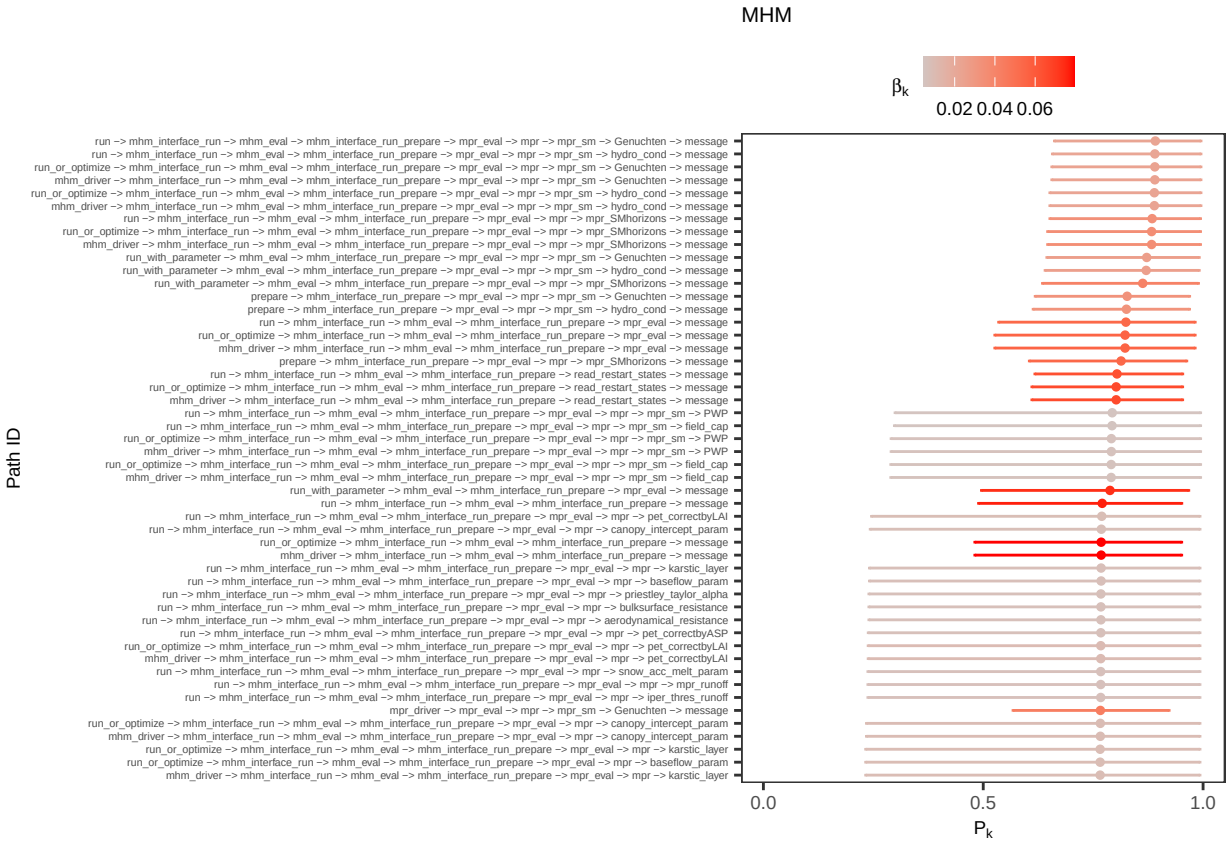

```
##
## [[7]]

## `height` was translated to `width`.
```




```
##
## [[8]]

## `height` was translated to `width`.
```

```
##
## [[9]]
```

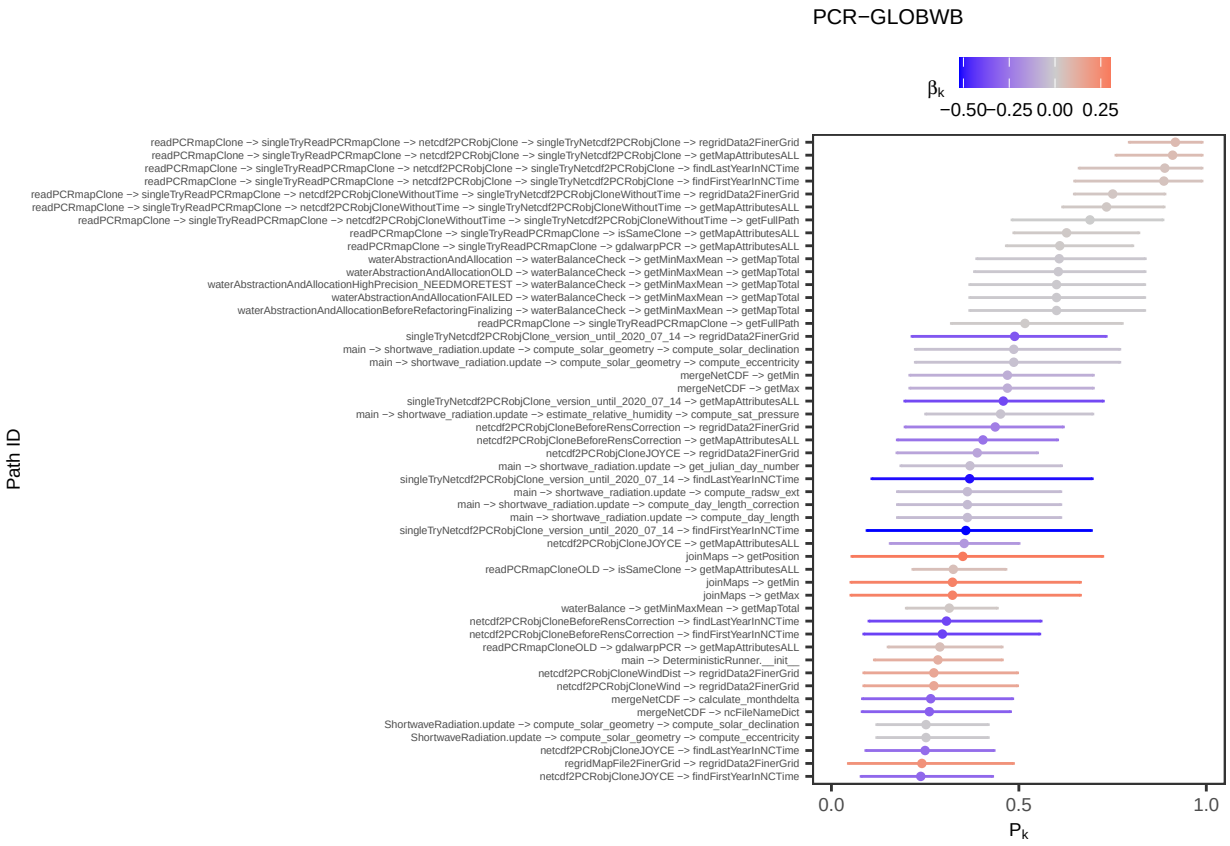
```
## `height` was translated to `width`.
```



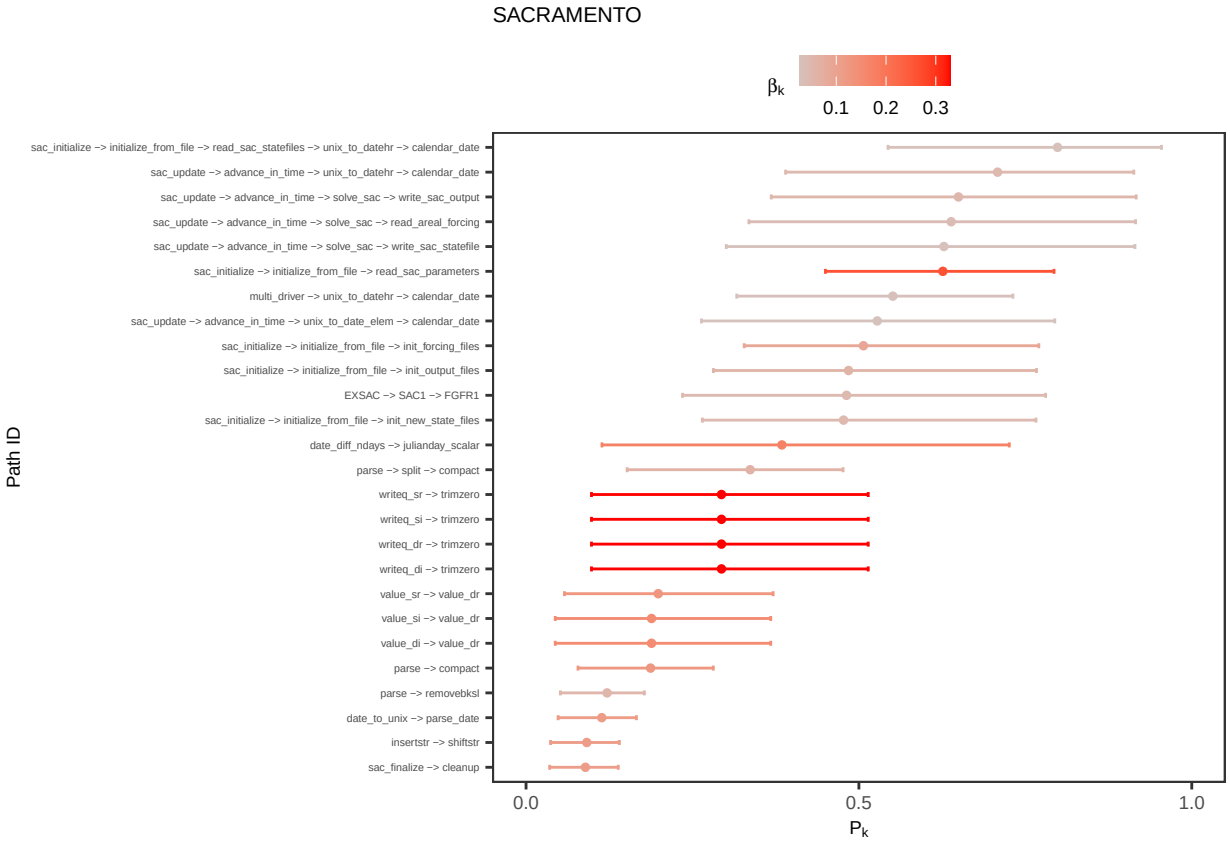

```
##
## [[10]]
```

```
## `height` was translated to `width`.
```

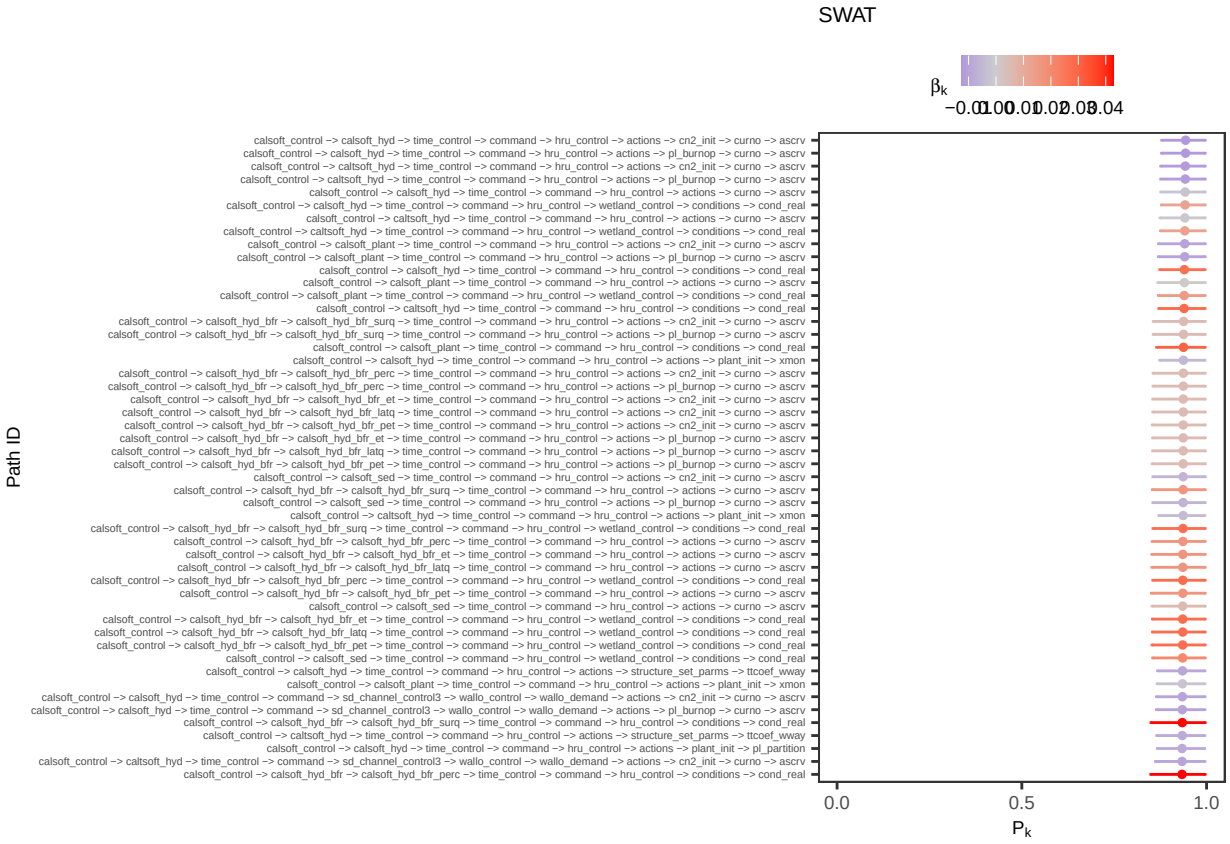

```
##
## [[11]]

## `height` was translated to `width`.
```




```
##
## [[12]]

## `height` was translated to `width`.
```

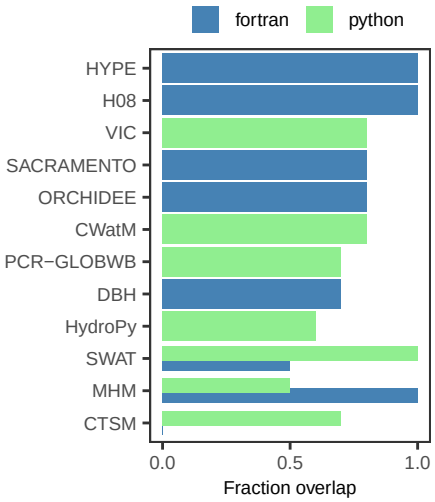



```
##
## [[13]]

## `height` was translated to `width`.
```




```
# READ RANKING OF THE UA / SA DATASET ##########################################

top_ten_overlap <- readRDS("top_ten_overlap.rds")


# ORDER AND FILTER OUT MODELS WITH LESS THAN 10 PATHS ###########################

model_names_ordered <- top_ten_overlap %>%
  .[n_paths > 10] %>%
  .[order(overlap_fraction)] %>%
  .[, model] %>%
  unique()

# PLOT ###########################################################################

top_ten_overlap %>%
  .[n_paths > 10] %>%
  .[, model:= factor(model, levels = model_names_ordered)] %>%
  ggplot(., aes(model, overlap_fraction, fill = language)) +
  geom_bar(stat = "identity", position = position_dodge(0.6)) +
  scale_fill_manual(values = color_languages, name = "") +
  coord_flip() +
  scale_y_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = "", y = "Fraction overlap") +
```

```
  theme_AP() +
  theme(legend.position = "top")
```

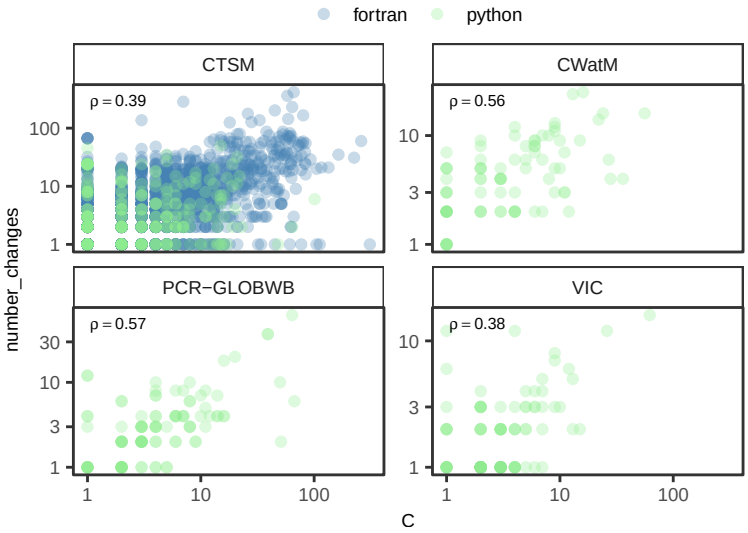



```
# METRICS AT THE FILE AND FUNCTION LEVEL ########################################

folder <- "./datasets/git_logs"

# Get names of files ------------------------------------------------------------

csv_files <- list.files(path = folder, pattern = "\\.csv$", full.names = TRUE)

# Merge and flatten -------------------------------------------------------------

logs_dt <- lapply(csv_files, fread) %>%
  lapply(., function(x) x[, .(model, language, `function`, cyclomatic_complexity, number_change
                              change_per_days, age_days)]) %>%
  rbindlist() %>%
  na.omit() %>%
  setnames(., "function", "node")

# Create and run function -------------------------------------------------------

spearman_fun <- function(dt) {

  cor.test(dt[, cyclomatic_complexity],
           dt[, number_changes], method = "spearman")$estimate[[1]]
}

rho_values <- logs_dt[, .(rho = spearman_fun(.SD)), model]
```

```
## Warning in cor.test.default(dt[, cyclomatic_complexity], dt[, number_changes],
## : Cannot compute exact p-value with ties
## Warning in cor.test.default(dt[, cyclomatic_complexity], dt[, number_changes],
```

```
## : Cannot compute exact p-value with ties
## Warning in cor.test.default(dt[, cyclomatic_complexity], dt[, number_changes],
## : Cannot compute exact p-value with ties
## Warning in cor.test.default(dt[, cyclomatic_complexity], dt[, number_changes],
## : Cannot compute exact p-value with ties
```

```
# Create dt to position labels ------------------------------------------------

pos_df <- logs_dt %>%
  filter(is.finite(cyclomatic_complexity), is.finite(number_changes)) %>%
  group_by(model) %>%
  summarise(x = min(cyclomatic_complexity, na.rm = TRUE),
            y = max(number_changes, na.rm = TRUE), .groups = "drop")

# join rho and make label text ------------------------------------------------

lab_df <- rho_values %>%
  left_join(pos_df, by = "model") %>%
  mutate(label = ifelse(is.na(rho), "rho==NA", paste0("rho==", round(rho, 2))))

# Plot ------------------------------------------------------------------------

plot_logs <-  logs_dt %>%
  ggplot(aes(cyclomatic_complexity, number_changes, color = language)) +
  geom_point(alpha = 0.3) +
  scale_color_manual(values = color_languages, name = "") +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = expression(C), y = "number_changes") +
  facet_wrap(~model, scales = "free_y") +
  geom_text(data = lab_df %>% na.omit(),
            aes(x = x, y = y, label = label), inherit.aes = FALSE,
            hjust = -0.05, vjust = 1.2, size = 2, parse = TRUE) +
  coord_cartesian(clip = "off") +
  theme_AP() +
  theme(legend.position = "top")

plot_logs
```




```
# ARE REFACTORING ACTIONS MORE COMMON IN RISKY PATHS? ############################

models_to_check <- unique(logs_dt$model)

# Extract name of functions in top ten risky paths per model --------------------

list_nodes <- full_paths_df %>%
  data.table() %>%
  .[order(-p_path_fail), .SD[1:10], model] %>%
  .[model %in% models_to_check] %>%
  na.omit() %>%
  .[, .(node = unlist(path_nodes)), model]

# keep only unique risk nodes
risk_nodes <- unique(list_nodes[, .(model, node)])
risk_nodes[, high_risk_path := TRUE]

# Filter out and keep only models that have log data ----------------------------

logs_dt <- logs_dt[model %in% models_to_check]

# left join into logs_dt --------------------------------------------------------

logs_dt[risk_nodes, high_risk_path:= i.high_risk_path,
        on = .(model, node)]

# Turn NAs into FALSE -----------------------------------------------------------

logs_dt[is.na(high_risk_path), high_risk_path := FALSE]

# We now quantify effect size using the common-language statistic A,
# defined as the probability that a randomly selected high-risk function
```

```
# has higher churn than a randomly selected non-risk function. -----------------

wilcox_dt <- logs_dt[is.finite(number_changes),
                     {
                       x <- number_changes[high_risk_path]
                       y <- number_changes[!high_risk_path]

                       n_x <- length(x)
                       n_y <- length(y)

                       if (n_x >= 3L && n_y >= 3L) {

                         wt <- wilcox.test(x, y, alternative = "greater", exact = FALSE)

                         # Wilcoxon rank-sum statistic (sum of ranks for x)-------------------
                         W <- as.numeric(wt$statistic)

                         # Convert to Mann-Whitney U ----------------------------------------
                         U <- W - n_x * (n_x + 1) / 2

                         # Common-language effect size (A statistic) ------------------------
                         A <- U / (n_x * n_y)

                         .(
                           p_value = wt$p.value,
                           A_common_language = A,
                           n_risk = n_x,
                           n_nonrisk = n_y,
                           median_risk = median(x),
                           median_nonrisk = median(y)
                         )

                       } else {

                         .(
                           p_value = NA_real_,
                           A_common_language = NA_real_,
                           n_risk = n_x,
                           n_nonrisk = n_y,
                           median_risk = if (n_x) median(x) else NA_real_,
                           median_nonrisk = if (n_y) median(y) else NA_real_
                         )
                       }
                     },
                     model
]
```

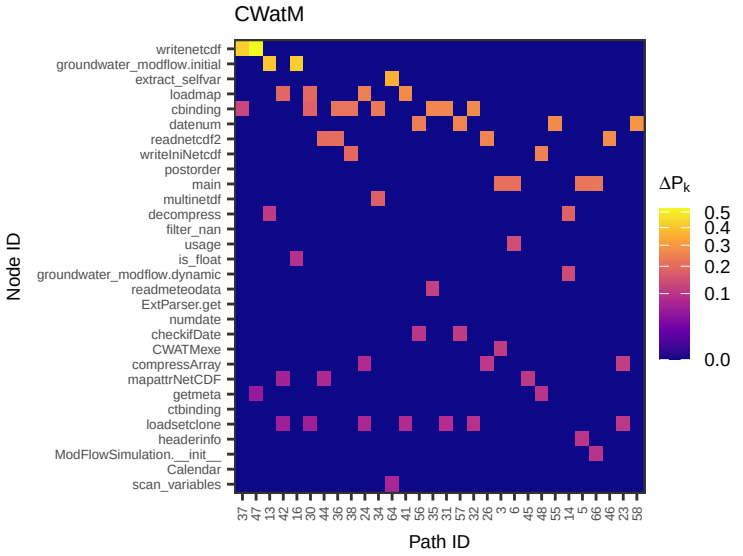
```
wilcox_dt
```

```
##          model      p_value A_common_language n_risk n_nonrisk median_risk
##         <char>        <num>              <num>  <int>     <int>       <num>
## 1:        CTSM 0.9999999998          0.2263026     48      2044         1.0
## 2:      CWatM 0.0013898071          0.6284314     15        68         6.0
## 3: PCR-GLOBWB 0.0006109416          0.5789773     22        80         7.5
## 4:         VIC           NA                 NA      0       101          NA
##    median_nonrisk
##             <num>
## 1:            4.5
## 2:            4.0
## 3:            3.0
## 4:            2.0
```

```
# WHICH PATHS IMPROVE IF A NODE'S RISK IS FIXED TO 0? ############################

# how many nodes and paths to show in the heatmap ------------------------------

number_of_interest <- 30
N_nodes <- number_of_interest
K_paths <- number_of_interest

list_plots <- list_nodes <- list()

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

  if (i == 7) next

  model.name <- all_graphs$node_df[[i]]$model

  node_df <- all_graphs$node_df[[i]]
  paths_tbl <- all_graphs$paths_tbl[[i]]

  # top-N nodes by failure probability -----------------------------------------

  top_nodes <- node_df %>%
    arrange(desc(risk_score)) %>%
    slice_head(n = N_nodes)

  # Return list of nodes -------------------------------------------------------

  list_nodes[[i]] <- top_nodes %>%
    transmute(model = model,
              name = name,
              risk_score = risk_score,
              cyclomatic_complexity = cyclomatic_complexity,
```

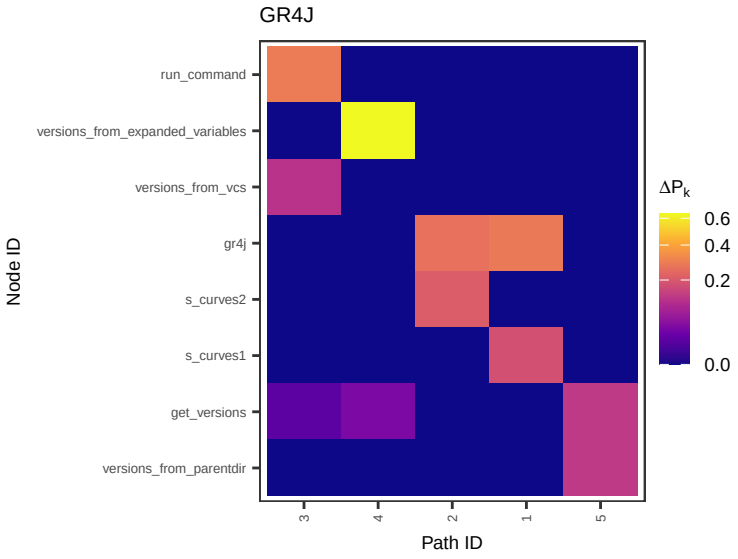
```
            indegree = indeg,
            outdegree = outdeg,
            betweenness = btw) %>%
  data.table::as.data.table()

# top-K paths by path failure probability -------------------------------------

top_paths <- paths_tbl %>%
  arrange(desc(p_path_fail)) %>%
  slice_head(n = K_paths)

# COMPUTE ##############################################################################

# make a named vector for fast lookup of risk_score by node name ------------------

p_map <- setNames(node_df$risk_score, node_df$name)

# Compute --------------------------------------------------------------------

delta_tbl <- map_dfr(seq_len(nrow(top_nodes)), function(i) {
  node_name <- top_nodes$name[i]

  map_dfr(seq_len(nrow(top_paths)), function(j) {
    pid <- top_paths$path_id[j]
    nodes_vec <- top_paths$path_nodes[[j]]
    P_orig <- top_paths$p_path_fail[j]

    # if node not in this path, improvement is zero ----------------------------

    if (!node_name %in% nodes_vec) {
      tibble(node = node_name,
             path_id = pid,
             deltaP = 0)
    } else {

      # failure probs along the path ---------------------------------------

      p_vec <- p_map[nodes_vec]

      # if node appears multiple times, fixing ANY of its appearances is enough:

      idxs <- which(nodes_vec == node_name)

      # we fix all occurrences (set all to zero) ------------

      p_vec_fix <- p_vec
      p_vec_fix[idxs] <- 0
```

```
        P_fix <- if (length(p_vec_fix) == 0) 0 else 1 - prod(1 - p_vec_fix)
        tibble(node = node_name,
               path_id = pid,
               deltaP = P_orig - P_fix)
      }
    })
  })

  # order factors so tha highest risk nodes are at the top and the highest
  # risk paths are on the left ----------------------------------------------

  # Bright cells: nodes that are chokepoints for a given path.
  # Rows with many bright cells: nodes whose refactoring will improve many
  # risk paths (global chokepoints)
  # Columns with few very bright cells: paths dominated by a single risky node
  # Rows/columns with dark colors: diffuse risk.

  delta_tbl <- delta_tbl %>%
    mutate(node = factor(node, levels = rev(top_nodes$name)),
           path_id = factor(path_id, levels = top_paths$path_id))


  # Plote the tile figure ---------------------------------------------------

  minP <- min(delta_tbl$deltaP, na.rm = TRUE)
  medP <- median(delta_tbl$deltaP, na.rm = TRUE)
  maxP <- max(delta_tbl$deltaP, na.rm = TRUE)

  list_plots[[i]] <- ggplot(delta_tbl, aes(x = path_id, y = node, fill = deltaP)) +
    geom_tile() +
    scale_fill_viridis_c(option = "C",
                         name = expression(Delta * P[k]),
                         trans  = "sqrt") +
    theme_AP() +
    theme(axis.text.x  = element_text(angle = 90, vjust = 0.5, hjust = 1, size = 5),
          axis.text.y = element_text(size = 5),
          axis.title.x = element_text(margin = margin(t = 5)),
          axis.title.y = element_text(margin = margin(r = 5))) +
    labs(x = "Path ID", y = "Node ID") +
    ggtitle(model.name)

}

for (i in 1:length(list_plots)) {
  print(list_plots[[i]])
  print(list_nodes[[i]])
}
```

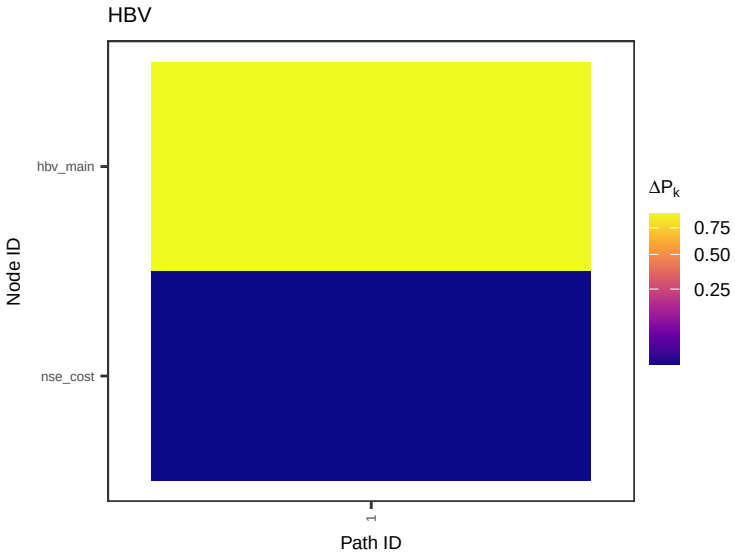



```
##      model                        name risk_score cyclomatic_complexity
##     <char>                      <char>      <num>                 <num>
##  1:  CWatM                  writenetcdf  0.6000000               56.0000
##  2:  CWatM groundwater_modflow.initial  0.5127273               48.0000
##  3:  CWatM              extract_selfvar  0.3976077               36.0000
##  4:  CWatM                     loadmap  0.3264593               22.0000
##  5:  CWatM                   cbinding  0.3071770                6.0000
##  6:  CWatM                    datenum  0.3000000                1.0000
##  7:  CWatM                 readnetcdf2  0.2945455               28.0000
##  8:  CWatM               writeIniNetcdf  0.2836364               27.0000
##  9:  CWatM                    postorder  0.2609569                9.0000
## 10:   <NA>                         main  0.2520201               13.4878
## 11:  CWatM                   multinetdf  0.2509091               24.0000
## 12:  CWatM                   decompress  0.2173206                5.0000
## 13:  CWatM                   filter_nan  0.1894737                1.0000
## 14:   <NA>                        usage  0.1835990               13.4878
## 15:   <NA>                     is_float  0.1835990               13.4878
## 16:  CWatM groundwater_modflow.dynamic  0.1745455               17.0000
## 17:  CWatM                readmeteodata  0.1636364               16.0000
## 18:  CWatM                ExtParser.get  0.1613397               10.0000
## 19:  CWatM                     numdate  0.1578947                1.0000
## 20:  CWatM                  checkifDate  0.1527273               15.0000
## 21:  CWatM                     CWATMexe  0.1500478                8.0000
## 22:  CWatM                compressArray  0.1432536                4.0000
## 23:  CWatM                mapattrNetCDF  0.1424880                2.0000
## 24:  CWatM                      getmeta  0.1323445                3.0000
## 25:  CWatM                    ctbinding  0.1323445                3.0000
## 26:  CWatM                 loadsetclone  0.1309091               13.0000
## 27:  CWatM                   headerinfo  0.1297608               10.0000
## 28:  CWatM  ModFlowSimulation.__init__  0.1248804               11.0000
```

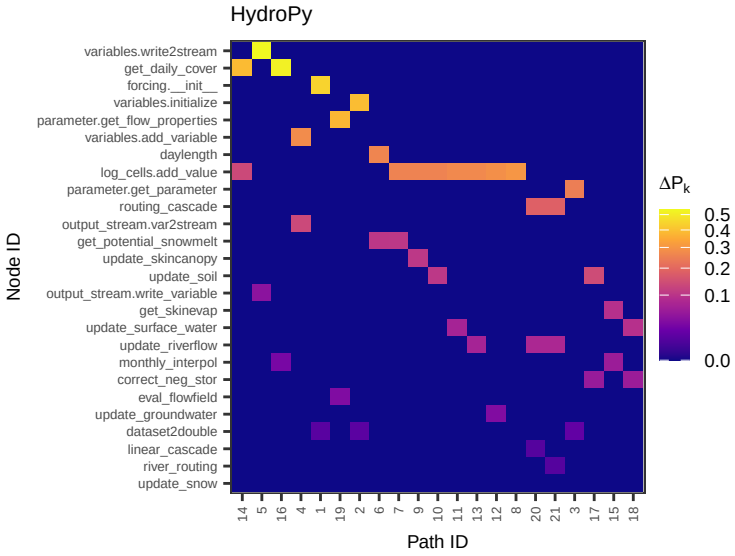
```
## 29:  CWatM                   Calendar  0.1237321                8.0000
## 30:  CWatM              scan_variables  0.1200000               12.0000
##      model                        name risk_score cyclomatic_complexity
##     <char>                      <char>      <num>                 <num>
##     indegree outdegree betweenness
##        <num>     <num>       <num>
##  1:        0         6        0.00
##  2:        0         8        0.00
##  3:        1         0        0.00
##  4:        3         5        4.75
##  5:       16         0        0.00
##  6:       19         0        0.00
##  7:        0         4        0.00
##  8:        0         5        0.00
##  9:       11         8        0.00
## 10:        1        10        9.50
## 11:        0         1        0.00
## 12:       11         0        0.00
## 13:       12         0        0.00
## 14:        3         0        0.00
## 15:        3         0        0.00
## 16:        0         3        0.00
## 17:        0         3        0.00
## 18:        4         0        0.00
## 19:       10         0        0.00
## 20:        0        23        0.00
## 21:        3         1        2.50
## 22:        7         0        0.00
## 23:        3         1        8.00
## 24:        7         0        0.00
## 25:        7         0        0.00
## 26:        0        12        0.00
## 27:        2         0        0.00
## 28:        1         0        0.00
## 29:        3         0        0.00
## 30:        0         1        0.00
##     indegree outdegree betweenness
##        <num>     <num>       <num>
```




```
##     model                           name risk_score cyclomatic_complexity
##    <char>                         <char>      <num>                 <int>
## 1:   GR4J                    run_command       0.75                    12
## 2:   GR4J versions_from_expanded_variables       0.75                    15
## 3:   GR4J              versions_from_vcs       0.55                     9
## 4:   GR4J                            gr4j       0.40                    11
## 5:   GR4J                       s_curves2       0.35                     4
## 6:   GR4J                       s_curves1       0.30                     3
## 7:   GR4J                     get_versions       0.15                     6
## 8:   GR4J          versions_from_parentdir       0.15                     3
##    indegree outdegree betweenness
##       <num>     <num>       <num>
## 1:        2         0           0
## 2:        1         0           0
## 3:        1         2           1
## 4:        0         4           0
## 5:        2         0           0
## 6:        2         0           0
## 7:        0         3           0
## 8:        1         0           0
```



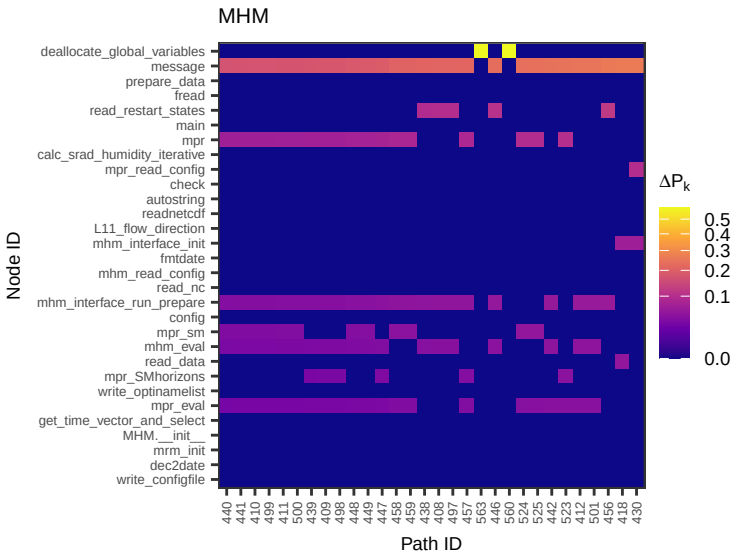

```
##     model     name risk_score cyclomatic_complexity indegree outdegree
##    <char>   <char>      <num>                 <int>     <num>      <num>
## 1:    HBV hbv_main       0.95                     5         1          0
## 2:    HBV nse_cost       0.05                     1         0          1
##    betweenness
##          <num>
## 1:           0
## 2:           0
```




```
##       model                          name risk_score cyclomatic_complexity
##      <char>                        <char>      <num>                 <num>
##  1: HydroPy        variables.write2stream  0.6000000              15.000000
##  2: HydroPy             get_daily_cover  0.5500000              14.000000
##  3: HydroPy           forcing.__init__  0.4500000              12.000000
##  4: HydroPy        variables.initialize  0.4000000              11.000000
##  5: HydroPy parameter.get_flow_properties  0.4000000              11.000000
```

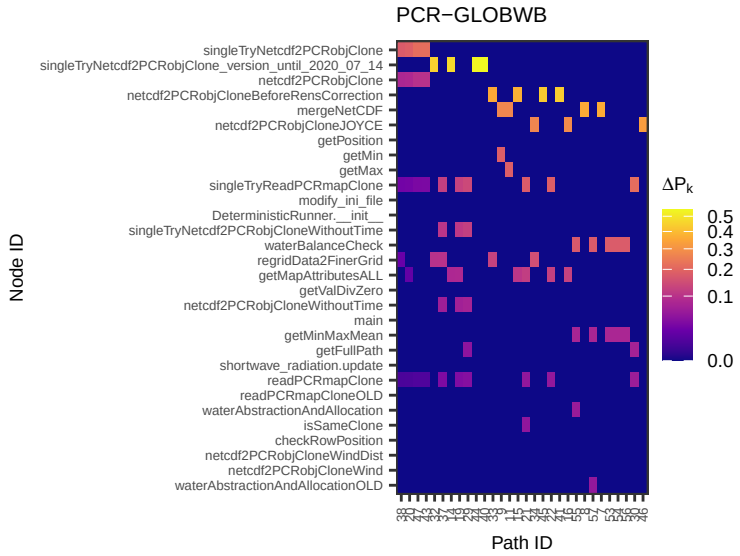
```
##  6: HydroPy        variables.add_variable  0.3500000             10.000000
##  7:    <NA>                     daylength  0.3064583              8.979167
##  8: HydroPy           log_cells.add_value  0.3000000              3.000000
##  9: HydroPy       parameter.get_parameter  0.2500000              8.000000
## 10: HydroPy               routing_cascade  0.2075000              5.000000
## 11: HydroPy      output_stream.var2stream  0.2075000              7.000000
## 12: HydroPy        get_potential_snowmelt  0.1500000              6.000000
## 13: HydroPy             update_skincanopy  0.1500000              6.000000
## 14: HydroPy                   update_soil  0.1500000              6.000000
## 15: HydroPy  output_stream.write_variable  0.1075000              5.000000
## 16: HydroPy                  get_skinevap  0.1000000              5.000000
## 17: HydroPy          update_surface_water  0.1000000              5.000000
## 18: HydroPy              update_riverflow  0.1000000              5.000000
## 19: HydroPy              monthly_interpol  0.0650000              4.000000
## 20: HydroPy              correct_neg_stor  0.0650000              4.000000
## 21: HydroPy                eval_flowfield  0.0575000              4.000000
## 22: HydroPy            update_groundwater  0.0500000              4.000000
## 23: HydroPy                dataset2double  0.0225000              3.000000
## 24: HydroPy                linear_cascade  0.0150000              3.000000
## 25: HydroPy                 river_routing  0.0150000              3.000000
## 26: HydroPy                   update_snow  0.0000000              3.000000
##       model                          name risk_score cyclomatic_complexity
##      <char>                        <char>      <num>                 <num>
##     indegree outdegree betweenness
##        <num>     <num>       <num>
##  1:        0         1           0
##  2:        0         2           0
##  3:        0         1           0
##  4:        0         1           0
##  5:        0         1           0
##  6:        0         1           0
##  7:        1         0           0
##  8:       40         0           0
##  9:        0         1           0
## 10:        1         4           2
## 11:        1         0           0
## 12:        0         3           0
## 13:        0        10           0
## 14:        0         7           0
## 15:        1         0           0
## 16:        0         1           0
## 17:        0         7           0
## 18:        0         5           0
## 19:        2         0           0
## 20:        2         0           0
## 21:        1         0           0
## 22:        0         3           0
## 23:        3         0           0
```

```
## 24:        2          0           0
## 25:        2          0           0
## 26:        0          8           0
##     indegree outdegree betweenness
##        <num>     <num>       <num>
```



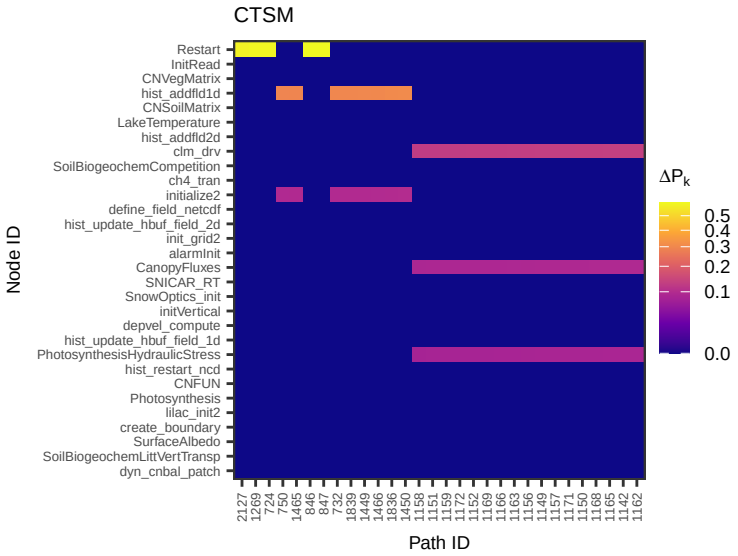

```
##      model                          name risk_score cyclomatic_complexity
##     <char>                        <char>      <num>                 <num>
##  1:    MHM  deallocate_global_variables 0.60196813                   201
##  2:    MHM                       message 0.37200000                    25
##  3:    MHM                  prepare_data 0.26100000                    88
##  4:    MHM                         fread 0.24600000                    83
##  5:    MHM           read_restart_states 0.21389286                    72
##  6:    MHM                          main 0.19500000                    66
##  7:    MHM                           mpr 0.19429071                    35
##  8:    MHM calc_srad_humidity_iterative 0.18689286                    63
##  9:    MHM               mpr_read_config 0.18178571                    61
## 10:    MHM                         check 0.17360714                     5
## 11:    MHM                     autostring 0.16889286                    57
## 12:    MHM                     readnetcdf 0.16297005                    53
## 13:    MHM           L11_flow_direction 0.13298246                    45
## 14:    MHM           mhm_interface_init 0.13245104                    21
## 15:    MHM                       fmtdate 0.13200000                    45
## 16:    MHM                mhm_read_config 0.12850576                    43
## 17:    MHM                       read_nc 0.12834562                    27
## 18:    MHM    mhm_interface_run_prepare 0.12465668                    14
## 19:    MHM                        config 0.12300000                    42
## 20:    MHM                        mpr_sm 0.11754877                    32
## 21:    MHM                      mhm_eval 0.10778571                     3
## 22:    MHM                     read_data 0.10685023                    25
## 23:    MHM                 mpr_SMhorizons 0.10043049                    34
## 24:    MHM           write_optinamelist 0.09989286                    34
```

```
## 25:    MHM                    mpr_eval 0.09788249                    4
## 26:    MHM   get_time_vector_and_select 0.09705415                   18
## 27:    MHM                MHM.__init__ 0.09000000                   31
## 28:    MHM                    mrm_init 0.09000000                   31
## 29:    MHM                    dec2date 0.08941475                    4
## 30:    MHM             write_configfile 0.08489286                   29
##      model                        name risk_score cyclomatic_complexity
##     <char>                      <char>      <num>                 <num>
##     indegree outdegree betweenness
##        <num>     <num>       <num>
##  1:        1         1   2.0000000
##  2:      336         0   0.0000000
##  3:        0        18   0.0000000
##  4:        0        16   0.0000000
##  5:        1         1   0.0000000
##  6:        0         4   0.0000000
##  7:        1        14 170.0000000
##  8:        1         0   0.0000000
##  9:        2         4   0.0000000
## 10:      181         0   0.0000000
## 11:        1         0   0.0000000
## 12:        6        34   3.0000000
## 13:        1         5   0.1666667
## 14:        2        23 131.4375000
## 15:        0        12   0.0000000
## 16:        1        35   3.0000000
## 17:        4         1  87.0000000
## 18:        2         5 156.0000000
## 19:        0         1   0.0000000
## 20:        1        14  44.0000000
## 21:        2         9 186.0000000
## 22:        2        25  61.5000000
## 23:        1         2   1.0000000
## 24:        1         2   0.0000000
## 25:        2         3 162.0000000
## 26:        1         8  84.0000000
## 27:        0         3   0.0000000
## 28:        0        38   0.0000000
## 29:       16         3 123.0000000
## 30:        1         4   0.0000000
##     indegree outdegree betweenness
##        <num>     <num>       <num>
```



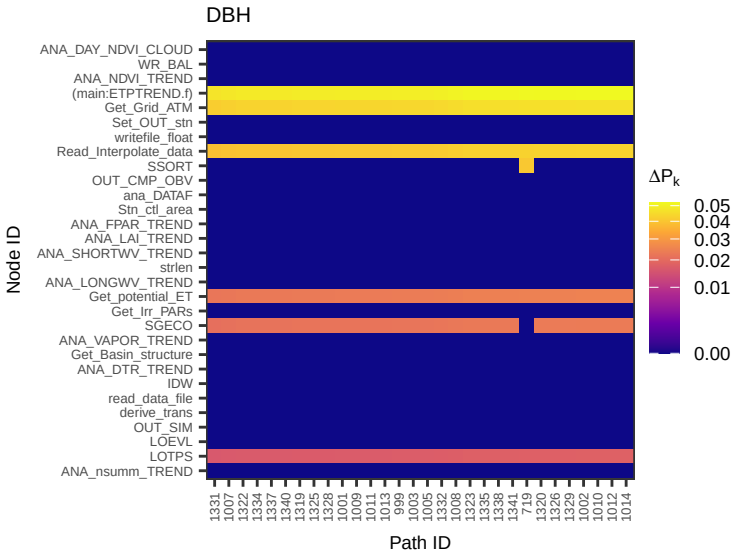

```
##          model                                                 name risk_score
##         <char>                                               <char>      <num>
##  1: PCR-GLOBWB                                singleTryNetcdf2PCRobjClone 0.70681818
##  2: PCR-GLOBWB  singleTryNetcdf2PCRobjClone_version_until_2020_07_14 0.57272727
##  3: PCR-GLOBWB                                   netcdf2PCRobjClone 0.52159091
##  4: PCR-GLOBWB                netcdf2PCRobjCloneBeforeRensCorrection 0.45454545
##  5: PCR-GLOBWB                                         mergeNetCDF 0.38181818
##  6: PCR-GLOBWB                              netcdf2PCRobjCloneJOYCE 0.34545455
##  7: PCR-GLOBWB                                         getPosition 0.31818182
##  8: PCR-GLOBWB                                              getMin 0.29318182
##  9: PCR-GLOBWB                                              getMax 0.29318182
## 10: PCR-GLOBWB                             singleTryReadPCRmapClone 0.26136364
## 11: PCR-GLOBWB                                     modify_ini_file 0.24090909
## 12: PCR-GLOBWB                          DeterministicRunner.__init__ 0.23863636
## 13: PCR-GLOBWB                singleTryNetcdf2PCRobjCloneWithoutTime 0.23409091
## 14: PCR-GLOBWB                                   waterBalanceCheck 0.22954545
## 15: PCR-GLOBWB                                 regridData2FinerGrid 0.22727273
## 16: PCR-GLOBWB                                  getMapAttributesALL 0.18977273
## 17: PCR-GLOBWB                                       getValDivZero 0.17500000
## 18: PCR-GLOBWB                         netcdf2PCRobjCloneWithoutTime 0.15795455
## 19:       <NA>                                                 main 0.12413163
## 20: PCR-GLOBWB                                       getMinMaxMean 0.11590909
## 21: PCR-GLOBWB                                         getFullPath 0.10454545
## 22: PCR-GLOBWB                           shortwave_radiation.update 0.10340909
## 23: PCR-GLOBWB                                     readPCRmapClone 0.09090909
## 24: PCR-GLOBWB                                  readPCRmapCloneOLD 0.09090909
## 25: PCR-GLOBWB                        waterAbstractionAndAllocation 0.09090909
## 26: PCR-GLOBWB                                       isSameClone 0.08863636
## 27: PCR-GLOBWB                                    checkRowPosition 0.08409091
## 28: PCR-GLOBWB                          netcdf2PCRobjCloneWindDist 0.08181818
## 29: PCR-GLOBWB                              netcdf2PCRobjCloneWind 0.08181818
## 30: PCR-GLOBWB                         waterAbstractionAndAllocationOLD 0.08181818
```

```
##          model                                                 name risk_score
##         <char>                                               <char>      <num>
##     cyclomatic_complexity indegree outdegree betweenness
##                     <num>    <num>     <num>       <num>
##  1:              67.00000        2         6           9
##  2:              64.00000        0         6           0
##  3:              50.00000        1         2           7
##  4:              51.00000        0         5           0
##  5:              43.00000        0        40           0
##  6:              39.00000        0         5           0
##  7:               3.00000       24         0           0
##  8:               3.00000       22         0           0
##  9:               3.00000       22         0           0
## 10:              16.00000        2         5          11
## 11:              22.00000        4         0           0
## 12:               8.00000       14         0           0
## 13:              20.00000        2         5           4
## 14:               8.00000        6         1          10
## 15:               4.00000       16         0           0
## 16:               4.00000       13         0           0
## 17:               1.00000       14         0           0
## 18:              14.00000        1         2           3
## 19:              14.65448        0        28           0
## 20:               5.00000        2         1           6
## 21:               7.00000        4         0           0
## 22:               2.00000        1         6           9
## 23:              11.00000        0         2           0
## 24:              11.00000        0         3           0
## 25:              11.00000        0         3           0
## 26:               6.00000        2         2           2
## 27:               2.00000        6         0           0
## 28:              10.00000        0         2           0
## 29:              10.00000        0         2           0
## 30:              10.00000        0         5           0
##     cyclomatic_complexity indegree outdegree betweenness
##                     <num>    <num>     <num>       <num>
## NULL
## NULL
```



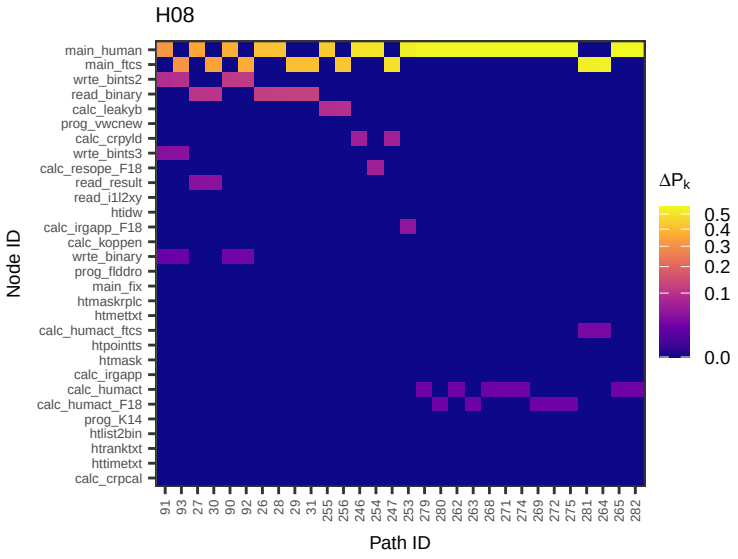
	model	name	risk_score	cyclomatic_complexity
	<char>	<char>	<num>	<int>
1:	CTSM	Restart	0.6000000	310
2:	CTSM	InitRead	0.5029126	260
3:	CTSM	CNVegMatrix	0.4355878	225
4:	CTSM	hist_addfld1d	0.4237577	63
5:	CTSM	CNSoilMatrix	0.3168444	164
6:	CTSM	LakeTemperature	0.2597794	134
7:	CTSM	hist_addfld2d	0.2307528	105
8:	CTSM	clm_drv	0.2265958	66
9:	CTSM	SoilBiogeochemCompetition	0.2240858	116
10:	CTSM	ch4_tran	0.2024723	105
11:	CTSM	initialize2	0.1914262	64
12:	CTSM	define_field_netcdf	0.1868026	97
13:	CTSM	hist_update_hbuf_field_2d	0.1795740	93
14:	CTSM	init_grid2	0.1747573	91
15:	CTSM	alarmInit	0.1712560	89
16:	CTSM	CanopyFluxes	0.1707223	80
17:	CTSM	SNICAR_RT	0.1617279	83
18:	CTSM	SnowOptics_init	0.1613561	84
19:	CTSM	initVertical	0.1613561	84
20:	CTSM	depvel_compute	0.1613561	84
21:	CTSM	hist_update_hbuf_field_1d	0.1595628	83
22:	CTSM	PhotosynthesisHydraulicStress	0.1586358	80
23:	CTSM	hist_restart_ncd	0.1573533	79
24:	CTSM	CNFUN	0.1484003	77
25:	CTSM	Photosynthesis	0.1480913	76
26:	CTSM	lilac_init2	0.1447588	74
27:	CTSM	create_boundary	0.1436893	75
28:	CTSM	SurfaceAlbedo	0.1398363	72
29:	CTSM	SoilBiogeochemLittVertTransp	0.1344685	70
30:	CTSM	dyn_cnbal_patch	0.1341717	70

```
##      model                          name risk_score cyclomatic_complexity
##     <char>                        <char>      <num>                 <int>
##     indegree outdegree betweenness
##        <num>     <num>       <num>
##  1:        0         8    0.000000
##  2:        0         2    0.000000
##  3:        1        23    3.000000
##  4:     1570         3   22.700061
##  5:        1        30    1.000000
##  6:        1         4    9.000000
##  7:      142         3   11.299939
##  8:        2       184  673.714286
##  9:        1         6    4.000000
## 10:        2         2    1.000000
## 11:        2        93  462.935829
## 12:        1        35    1.372881
## 13:        1         2    5.000000
## 14:        0        18    0.000000
## 15:        2         0    0.000000
## 16:        1        32  115.428571
## 17:       10         4    4.000000
## 18:        1         1    0.000000
## 19:        1         2    0.000000
## 20:        1         0    0.000000
## 21:        1         1    1.000000
## 22:        1         4   34.000000
## 23:        3        15   35.866667
## 24:        2         9    3.000000
## 25:        2         2   14.000000
## 26:        1         9   19.000000
## 27:        0         4    0.000000
## 28:        1        14   12.000000
## 29:        1         1    2.000000
## 30:        1         0    0.000000
##     indegree outdegree betweenness
##        <num>     <num>       <num>
```



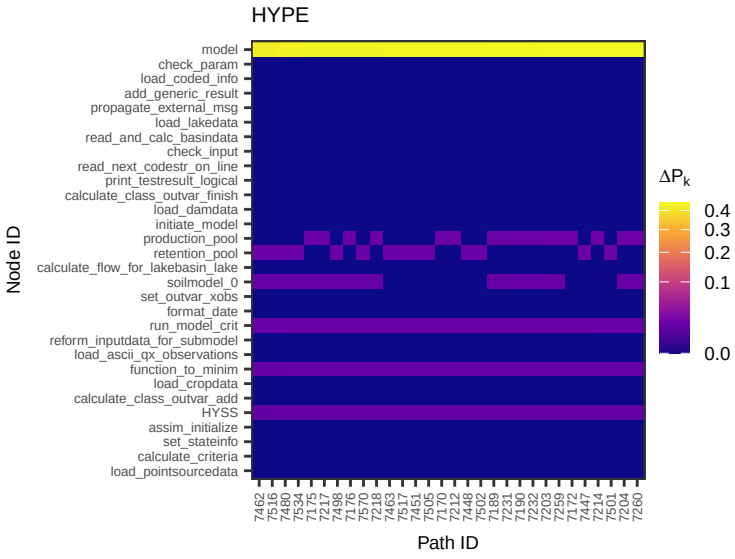

```
##      model                  name risk_score cyclomatic_complexity indegree
##     <char>                <char>      <num>                  <num>    <num>
##  1:    DBH    ANA_DAY_NDVI_CLOUD  0.6000000                     73        0
##  2:    DBH                WR_BAL  0.4446982                     54        1
##  3:    DBH        ANA_NDVI_TREND  0.3666667                     45        0
##  4:    DBH     (main:ETPTREND.f)  0.3583333                     44        0
##  5:    DBH          Get_Grid_ATM  0.3312000                     25       13
##  6:    DBH           Set_OUT_stn  0.3193265                     39        1
##  7:    DBH      writefile_float  0.3166667                      3      125
##  8:    DBH Read_Interpolate_data  0.3151339                     27        1
##  9:    DBH                SSORT  0.3025789                     37        1
## 10:    DBH          OUT_CMP_OBV  0.2666667                     33        0
## 11:    DBH            ana_DATAF  0.2421333                      5       87
## 12:    DBH         Stn_ctl_area  0.2258489                     26        7
## 13:    DBH        ANA_FPAR_TREND  0.2201400                     27        1
## 14:    DBH         ANA_LAI_TREND  0.2201400                     27        1
## 15:    DBH     ANA_SHORTWV_TREND  0.2120155                     25        2
## 16:    DBH               strlen  0.2098000                      4       77
## 17:    DBH      ANA_LONGWV_TREND  0.2096302                     25        2
## 18:    DBH      Get_potential_ET  0.2087837                     23        3
## 19:    DBH          Get_Irr_PARs  0.2026309                     25        1
## 20:    DBH                 SGECO  0.1990909                     23        1
## 21:    DBH       ANA_VAPOR_TREND  0.1921883                     23        2
## 22:    DBH   Get_Basin_structure  0.1778857                     22        1
## 23:    DBH        ANA_DTR_TREND  0.1746124                     21        1
## 24:    DBH                  IDW  0.1745669                     19        8
## 25:    DBH        read_data_file  0.1731813                     21        1
## 26:    DBH          derive_trans  0.1693052                     21        1
## 27:    DBH               OUT_SIM  0.1692559                     21        1
## 28:    DBH                 LOEVL  0.1690667                     21        1
## 29:    DBH                 LOTPS  0.1615691                     11        8
## 30:    DBH       ANA_nsumm_TREND  0.1614932                     20        1
```

```
##      model                 name risk_score cyclomatic_complexity indegree
##     <char>               <char>      <num>                 <num>    <num>
##     outdegree betweenness
##         <num>       <num>
##  1:        36    0.000000
##  2:        55    3.530030
##  3:         6    0.000000
##  4:        50    0.000000
##  5:        24  558.983471
##  6:         8    1.452381
##  7:         0    0.000000
##  8:        11  537.000000
##  9:         2    1.000000
## 10:        20    0.000000
## 11:         0    0.000000
## 12:         2    4.000000
## 13:        11    6.000000
## 14:        11    6.000000
## 15:        15   40.333333
## 16:         0    0.000000
## 17:        14   27.000000
## 18:         9  102.016529
## 19:        34    1.290909
## 20:        10   74.666667
## 21:        11   22.666667
## 22:         8    2.715152
## 23:         8   31.000000
## 24:         1   30.000000
## 25:         3   23.000000
## 26:         1    1.333333
## 27:         5    1.057692
## 28:         0    0.000000
## 29:         9  330.000000
## 30:         8    4.247639
##     outdegree betweenness
##         <num>       <num>
```



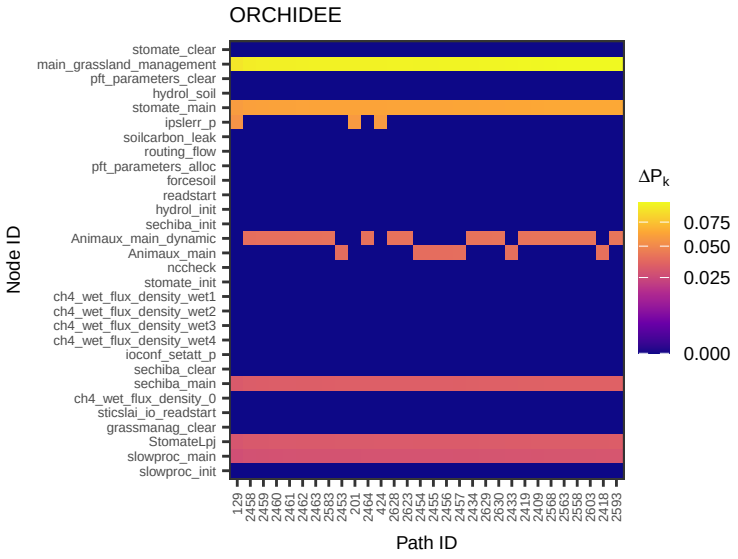
	model	name	risk_score	cyclomatic_complexity	indegree	outdegree
	<char>	<char>	<num>	<num>	<num>	<num>
1:	H08	main_human	0.60000000	300	0	377
2:	H08	main_ftcs	0.58996656	295	0	198
3:	H08	wrte_bints2	0.30693331	45	225	4
4:	H08	read_binary	0.30200669	2	356	0
5:	H08	calc_leakyb	0.24587925	115	5	1
6:	H08	prog_vwcnew	0.21672241	109	0	20
7:	H08	calc_crpyld	0.16591259	82	4	0
8:	H08	wrte_bints3	0.16557439	18	156	2
9:	H08	calc_resope_F18	0.16539119	83	1	0
10:	H08	read_result	0.15398089	17	114	8
11:	H08	read_i1l2xy	0.14362012	3	47	2
12:	H08	htidw	0.13645485	69	0	7
13:	H08	calc_irgapp_F18	0.12124403	61	1	0
14:	H08	calc_koppen	0.11036789	56	0	4
15:	H08	wrte_binary	0.09522472	1	113	0
16:	H08	prog_flddro	0.08829431	45	0	7
17:	H08	main_fix	0.08428094	43	0	65
18:	H08	htmaskrplc	0.08227425	42	0	4
19:	H08	htmettxt	0.08026756	41	0	2
20:	H08	calc_humact_ftcs	0.07882306	27	1	27
21:	H08	htpointts	0.07625418	39	0	2
22:	H08	htmask	0.07424749	38	0	4
23:	H08	calc_irgapp	0.07191951	36	2	0
24:	H08	calc_humact	0.06319678	25	1	27
25:	H08	calc_humact_F18	0.06262282	26	1	27
26:	H08	prog_K14	0.06220736	32	0	6
27:	H08	htlist2bin	0.06020067	31	0	3
28:	H08	htranktxt	0.06020067	31	0	2
29:	H08	httimetxt	0.06020067	31	0	4
30:	H08	calc_crpcal	0.05819398	30	0	13

```
##      model            name risk_score cyclomatic_complexity indegree outdegree
##     <char>          <char>      <num>                 <num>    <num>     <num>
##     betweenness
##           <num>
##  1:         0.0
##  2:         0.0
##  3:         9.0
##  4:         0.0
##  5:         4.0
##  6:         0.0
##  7:         0.0
##  8:         0.0
##  9:         0.0
## 10:         8.0
## 11:        31.0
## 12:         0.0
## 13:         0.0
## 14:         0.0
## 15:         0.0
## 16:         0.0
## 17:         0.0
## 18:         0.0
## 19:         0.0
## 20:         8.0
## 21:         0.0
## 22:         0.0
## 23:         0.0
## 24:         4.4
## 25:         3.6
## 26:         0.0
## 27:         0.0
## 28:         0.0
## 29:         0.0
## 30:         0.0
##     betweenness
##           <num>
```



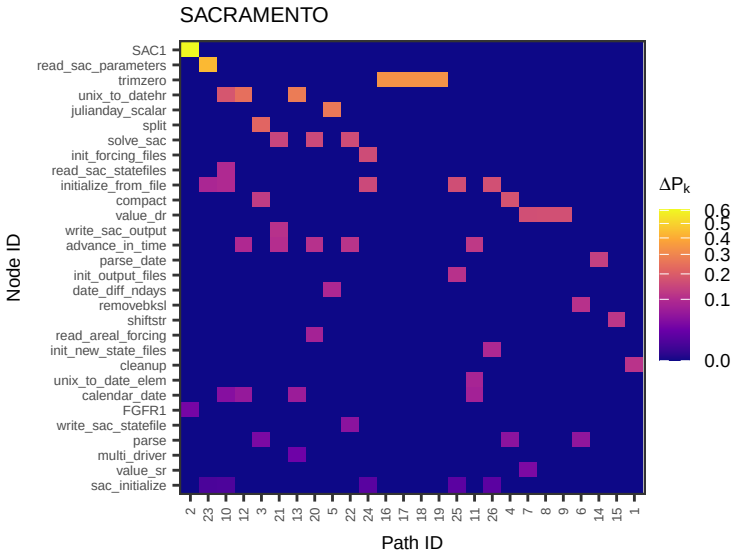
	model	name	risk_score	cyclomatic_complexity
	<char>	<char>	<num>	<int>
1:	HYPE	model	0.70652174	937
2:	HYPE	check_param	0.30571902	9
3:	HYPE	load_coded_info	0.26148852	406
4:	HYPE	add_generic_result	0.20269783	3
5:	HYPE	propagate_external_msg	0.20011706	10
6:	HYPE	load_lakedata	0.19377020	301
7:	HYPE	read_and_calc_basindata	0.17569388	273
8:	HYPE	check_input	0.13000780	4
9:	HYPE	read_next_codestr_on_line	0.12243032	17
10:	HYPE	print_testresult_logical	0.11540134	4
11:	HYPE	calculate_class_outvar_finish	0.11345596	3
12:	HYPE	load_damdata	0.09496386	147
13:	HYPE	initiate_model	0.09495934	125
14:	HYPE	production_pool	0.09260870	1
15:	HYPE	retention_pool	0.09064103	2
16:	HYPE	calculate_flow_for_lakebasin_lake	0.08848927	130
17:	HYPE	soilmodel_0	0.08524710	96
18:	HYPE	set_outvar_xobs	0.08345596	3
19:	HYPE	format_date	0.08334448	13
20:	HYPE	run_model_crit	0.08068341	9
21:	HYPE	reform_inputdata_for_submodel	0.07952274	123
22:	HYPE	load_ascii_qx_observations	0.07860718	121
23:	HYPE	function_to_minim	0.07821127	7
24:	HYPE	load_cropdata	0.07766727	120
25:	HYPE	calculate_class_outvar_add	0.07757525	4
26:	HYPE	HYSS	0.07500000	118
27:	HYPE	assim_initialize	0.07150415	109
28:	HYPE	set_stateinfo	0.06732999	104
29:	HYPE	calculate_criteria	0.06281391	64
30:	HYPE	load_pointsourcedata	0.06114986	94

```
##      model                                name risk_score cyclomatic_complexity
##     <char>                              <char>      <num>                 <int>
##     indegree outdegree betweenness
##        <num>     <num>       <num>
##  1:        5       266 3771.000000
##  2:      230         1   22.279642
##  3:        1       159   21.449006
##  4:      154         1   20.597727
##  5:      149         0    0.000000
##  6:        1        29    5.964286
##  7:        1        13    1.152381
##  8:       98         1    9.753023
##  9:       86         0    0.000000
## 10:       87         0    0.000000
## 11:       86         0    0.000000
## 12:        1        15    2.630952
## 13:        6        16  288.333333
## 14:       71         0    0.000000
## 15:       69         0    0.000000
## 16:        1        16  169.416667
## 17:        6        47  623.104094
## 18:       63         0    0.000000
## 19:       58         0    0.000000
## 20:        1        17 2800.000000
## 21:        1         1    0.500000
## 22:        1        33   14.320635
## 23:        7         1 2460.000000
## 24:        1        10    3.049570
## 25:       58         0    0.000000
## 26:        0       136    0.000000
## 27:        1        63   36.542169
## 28:        1         0    0.000000
## 29:        7        83  501.500000
## 30:        1        25    8.678226
##     indegree outdegree betweenness
##        <num>     <num>       <num>
```



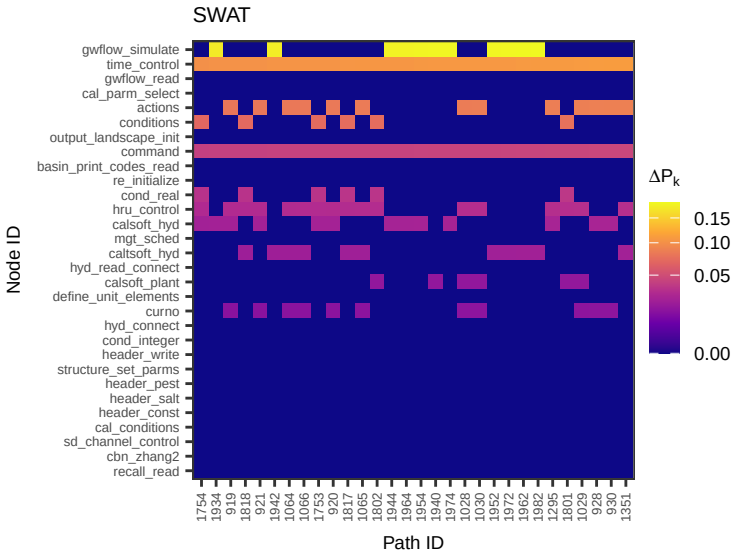

```
##        model                      name risk_score cyclomatic_complexity
##       <char>                    <char>      <num>                  <int>
##  1: ORCHIDEE             stomate_clear  0.6015020                    499
##  2: ORCHIDEE main_grassland_management  0.4181497                    318
##  3: ORCHIDEE      pft_parameters_clear  0.4105702                    341
##  4: ORCHIDEE              hydrol_soil  0.3546497                    290
##  5: ORCHIDEE             stomate_main  0.3177095                    193
##  6: ORCHIDEE                ipslerr_p  0.3024096                      3
##  7: ORCHIDEE           soilcarbon_leak  0.2799343                    231
##  8: ORCHIDEE             routing_flow  0.2726577                    226
##  9: ORCHIDEE      pft_parameters_alloc  0.2611726                    217
## 10: ORCHIDEE                forcesoil  0.2481928                    207
## 11: ORCHIDEE                readstart  0.2382811                    198
## 12: ORCHIDEE              hydrol_init  0.2363009                    196
## 13: ORCHIDEE             sechiba_init  0.2347023                    195
## 14: ORCHIDEE      Animaux_main_dynamic  0.2316875                    173
## 15: ORCHIDEE             Animaux_main  0.2244936                    182
## 16: ORCHIDEE                  nccheck  0.2146302                      2
## 17: ORCHIDEE             stomate_init  0.2104348                    174
## 18: ORCHIDEE ch4_wet_flux_density_wet1  0.2069558                    172
## 19: ORCHIDEE ch4_wet_flux_density_wet2  0.2069558                    172
## 20: ORCHIDEE ch4_wet_flux_density_wet3  0.2069558                    172
## 21: ORCHIDEE ch4_wet_flux_density_wet4  0.2069558                    172
## 22: ORCHIDEE           ioconf_setatt_p  0.2024471                      2
## 23: ORCHIDEE            sechiba_clear  0.2012048                    168
## 24: ORCHIDEE             sechiba_main  0.2001123                     81
## 25: ORCHIDEE    ch4_wet_flux_density_0  0.1985220                    165
## 26: ORCHIDEE     sticslai_io_readstart  0.1961124                    163
## 27: ORCHIDEE          grassmanag_clear  0.1937741                    161
## 28: ORCHIDEE                StomateLpj  0.1918230                    105
## 29: ORCHIDEE            slowproc_main  0.1773529                     78
## 30: ORCHIDEE            slowproc_init  0.1750749                    134
```

```
##        model                     name risk_score cyclomatic_complexity
##       <char>                   <char>      <num>                 <int>
##     indegree outdegree betweenness
##        <num>     <num>       <num>
##  1:        1         4     8.00000
##  2:        1        14   495.00000
##  3:        1         0     0.00000
##  4:        1        24    77.50000
##  5:        1        28  1198.60000
##  6:      322         0     0.00000
##  7:        2         3    13.50000
##  8:        1         6     9.00000
##  9:        1         0     0.00000
## 10:        0       107     0.00000
## 11:        1         3     0.00000
## 12:        1        58     6.02381
## 13:        1         1     0.50000
## 14:        1        49   330.00000
## 15:        1        12    77.00000
## 16:      229         1     1.00000
## 17:        1         2    15.00000
## 18:        1         0     0.00000
## 19:        1         0     0.00000
## 20:        1         0     0.00000
## 21:        1         0     0.00000
## 22:      216         0     0.00000
## 23:        0        10     0.00000
## 24:        4        18  1402.65000
## 25:        1         0     0.00000
## 26:        1         0     0.00000
## 27:        1         1     1.00000
## 28:        1        36   920.00000
## 29:        2        12  1160.25000
## 30:        1        79   195.00000
##     indegree outdegree betweenness
##        <num>     <num>       <num>
```



	model	name	risk_score	cyclomatic_complexity	indegree
	<char>	<char>	<num>	<num>	<num>
1:	SACRAMENTO	SAC1	0.66428571	29.000000	1
2:	SACRAMENTO	read_sac_parameters	0.54285714	24.000000	1
3:	SACRAMENTO	trimzero	0.32857143	7.000000	4
4:	SACRAMENTO	unix_to_datehr	0.31428571	3.000000	4
5:	SACRAMENTO	julianday_scalar	0.30000000	1.000000	6
6:	SACRAMENTO	split	0.28571429	12.000000	1
7:	SACRAMENTO	solve_sac	0.20000000	4.000000	1
8:	SACRAMENTO	init_forcing_files	0.20000000	8.000000	1
9:	SACRAMENTO	read_sac_statefiles	0.19285714	5.000000	1
10:	SACRAMENTO	initialize_from_file	0.19285714	3.000000	1
11:	SACRAMENTO	compact	0.18571429	5.000000	2
12:	SACRAMENTO	value_dr	0.17142857	2.000000	3
13:	SACRAMENTO	write_sac_output	0.15714286	6.000000	1
14:	SACRAMENTO	advance_in_time	0.15000000	1.000000	1
15:	SACRAMENTO	parse_date	0.13571429	5.000000	1
16:	SACRAMENTO	init_output_files	0.13571429	5.000000	1
17:	SACRAMENTO	date_diff_ndays	0.12857143	7.000000	0
18:	SACRAMENTO	removebksl	0.11428571	4.000000	1
19:	SACRAMENTO	shiftstr	0.11428571	4.000000	1
20:	SACRAMENTO	read_areal_forcing	0.11428571	4.000000	1
21:	SACRAMENTO	init_new_state_files	0.11428571	4.000000	1
22:	<NA>	cleanup	0.11055901	3.826087	1
23:	SACRAMENTO	unix_to_date_elem	0.10714286	3.000000	1
24:	SACRAMENTO	calendar_date	0.10000000	1.000000	2
25:	SACRAMENTO	FGFR1	0.09285714	3.000000	1
26:	SACRAMENTO	write_sac_statefile	0.07142857	2.000000	1
27:	<NA>	parse	0.06055901	3.826087	0
28:	SACRAMENTO	multi_driver	0.04285714	3.000000	0
29:	SACRAMENTO	value_sr	0.04285714	3.000000	0
30:	SACRAMENTO	sac_initialize	0.02142857	2.000000	0

```
##          model                name risk_score cyclomatic_complexity indegree
##         <char>              <char>      <num>                 <num>    <num>
##     outdegree betweenness
##         <num>       <num>
##  1:         1           1
##  2:         0           0
##  3:         0           0
##  4:         1           5
##  5:         0           0
##  6:         1           0
##  7:         3           6
##  8:         0           0
##  9:         1           4
## 10:         5           7
## 11:         0           0
## 12:         0           0
## 13:         0           0
## 14:         3           7
## 15:         0           0
## 16:         0           0
## 17:         6           0
## 18:         0           0
## 19:         0           0
## 20:         0           0
## 21:         0           0
## 22:         0           0
## 23:         1           1
## 24:         0           0
## 25:         0           0
## 26:         0           0
## 27:         3           0
## 28:         2           0
## 29:         1           0
## 30:         1           0
##     outdegree betweenness
##         <num>       <num>
```



	model	name	risk_score	cyclomatic_complexity	indegree
	<char>	<char>	<num>	<int>	<num>
1:	SWAT	gwflow_simulate	0.60955084	445	1
2:	SWAT	time_control	0.47567568	57	42
3:	SWAT	gwflow_read	0.42741313	312	1
4:	SWAT	cal_parm_select	0.42247597	250	12
5:	SWAT	actions	0.41668101	229	12
6:	SWAT	conditions	0.38678632	212	14
7:	SWAT	output_landscape_init	0.28011583	203	1
8:	SWAT	command	0.27139271	134	1
9:	SWAT	basin_print_codes_read	0.25444015	184	1
10:	SWAT	re_initialize	0.24961390	6	34
11:	SWAT	cond_real	0.23320463	15	30
12:	SWAT	hru_control	0.20981355	115	1
13:	SWAT	calsoft_hyd	0.17357035	122	1
14:	SWAT	mgt_sched	0.16674556	108	3
15:	SWAT	caltsoft_hyd	0.15977874	112	1
16:	SWAT	hyd_read_connect	0.13169463	35	12
17:	SWAT	calsoft_plant	0.12818783	89	1
18:	SWAT	define_unit_elements	0.12606178	15	15
19:	SWAT	curno	0.11663466	2	16
20:	SWAT	hyd_connect	0.11621622	87	0
21:	SWAT	cond_integer	0.11177606	15	13
22:	SWAT	header_write	0.10984556	77	1
23:	SWAT	structure_set_parms	0.10607235	26	10
24:	SWAT	header_pest	0.10444015	73	1
25:	SWAT	header_salt	0.10444015	73	1
26:	SWAT	header_const	0.10444015	73	1
27:	SWAT	cal_conditions	0.10190945	71	1
28:	SWAT	sd_channel_control	0.09324324	70	0
29:	SWAT	cbn_zhang2	0.09279399	64	1
30:	SWAT	recall_read	0.08687259	60	1

```
##      model                  name risk_score cyclomatic_complexity indegree
##     <char>                <char>      <num>                 <int>    <num>
##     outdegree betweenness
##         <num>       <num>
##  1:         5    70.00000
##  2:        13  2907.00000
##  3:         0     0.00000
##  4:        13     8.00000
##  5:        40   664.50000
##  6:        43    48.00000
##  7:         0     0.00000
##  8:        46  2457.00000
##  9:         0     0.00000
## 10:         0     0.00000
## 11:         0     0.00000
## 12:        67  1413.28571
## 13:        27    84.70924
## 14:        33    21.00000
## 15:        14    76.62500
## 16:         1     1.00000
## 17:        23    61.80435
## 18:         0     0.00000
## 19:         1    29.00000
## 20:        22     0.00000
## 21:         0     0.00000
## 22:         0     0.00000
## 23:         1    25.00000
## 24:         0     0.00000
## 25:         0     0.00000
## 26:         0     0.00000
## 27:        12     5.00000
## 28:        16     0.00000
## 29:         8    15.00000
## 30:         0     0.00000
##     outdegree betweenness
##         <num>       <num>
```

```
# FUNCTIONS TO SELECT THE TOP TEN RISKY PATHS PER MODEL AND
# PRINT THEM OUT FOR LATEX ##########################################################

tmp <- full_paths_df[order(-p_path_fail), .SD[1:10], model] %>%
  .[, .(model, path_str)] %>%
  na.omit() %>%
  split(., .$model)


to_tex_list_fun(tmp)
```

5 Session information

```
# SESSION INFORMATION ############################################################

sessionInfo()
```

```
## R version 4.5.2 (2025-10-31)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.6.1
##
## Matrix products: default
## BLAS:   /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framew
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/London
## tzcode source: internal
##
## attached base packages:
## [1] tools     parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] benchmarkme_1.0.8  sensobol_1.1.6     ggraph_2.2.2       foreach_1.5.2
##  [5] igraph_2.2.1       tidygraph_1.3.1    here_1.0.2         tidytext_0.4.3
##  [9] ggrepel_0.9.6      readxl_1.4.5       cowplot_1.2.0.9000 scales_1.4.0
## [13] openxlsx_4.2.8     lubridate_1.9.4    forcats_1.0.1      stringr_1.6.0
## [17] dplyr_1.1.4        purrr_1.2.0        readr_2.1.5        tidyr_1.3.2
## [21] tibble_3.3.0       ggplot2_4.0.1.9000 tidyverse_2.0.0    data.table_1.18.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.1      viridisLite_0.4.2     farver_2.1.2
##  [4] viridis_0.6.5         S7_0.2.1              fastmap_1.2.0
##  [7] tweenr_2.0.3          janeaustenr_1.0.0     digest_0.6.39
## [10] timechange_0.3.0      lifecycle_1.0.4       tokenizers_0.3.0
## [13] magrittr_2.0.4        compiler_4.5.2        rlang_1.1.6
## [16] yaml_2.3.12           knitr_1.51            labeling_0.4.3
## [19] graphlayouts_1.2.2    RColorBrewer_1.1-3    withr_3.0.2
## [22] grid_4.5.2            polyclip_1.10-7       iterators_1.0.14
## [25] MASS_7.3-65           tinytex_0.58          cli_3.6.5
## [28] crayon_1.5.3          rmarkdown_2.30        generics_0.1.4
## [31] RcppParallel_5.1.11-1 rstudioapi_0.17.1     httr_1.4.7
## [34] tzdb_0.5.0            cachem_1.1.0          ggforce_0.5.0
## [37] cellranger_1.1.0      vctrs_0.6.5           Matrix_1.7-4
## [40] hms_1.1.3             ineq_0.2-13           glue_1.8.0
## [43] benchmarkmeData_1.0.4 codetools_0.2-20      rngWELL_0.10-10
```

```
## [46] stringi_1.8.7         gtable_0.3.6          randtoolbox_2.0.5
## [49] pillar_1.11.1         htmltools_0.5.9       R6_2.6.1
## [52] zigg_0.0.2            Rdpack_2.6.4          doParallel_1.0.17
## [55] rprojroot_2.1.1       evaluate_1.0.5        lattice_0.22-7
## [58] rbibutils_2.4         SnowballC_0.7.1       Rfast_2.1.5.2
## [61] memoise_2.0.1         Rcpp_1.1.0            zip_2.3.3
## [64] gridExtra_2.3         xfun_0.55             pkgconfig_2.0.3
```

```
## Return the machine CPU ------------------------------------------------------

cat("Machine:     "); print(get_cpu()$model_name)
```

```
## Machine:
## [1] "Apple M1 Max"
```

```
## Return number of true cores -------------------------------------------------

cat("Num cores:   "); print(detectCores(logical = FALSE))
```

```
## Num cores:
## [1] 10
```

```
## Return number of threads ----------------------------------------------------

cat("Num threads: "); print(detectCores(logical = FALSE))
```

```
## Num threads:
## [1] 10
```