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 ################################
```

```r
# Strip leading "./models/" ---------------------------------------------------

all_callgraphs[, file_clean := sub("^\\./models/", "", file)]

# model_id = first part before "/" --------------------------------------------

all_callgraphs[, model_id := tstrsplit(file_clean, "/", fixed = TRUE, keep = 1L)]

# rest = everything after "model_id/" -----------------------------------------

all_callgraphs[, rest := sub("^[^/]+/", "", file_clean)]

# 2) top_level directory inside each model -------------------------------------

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

## 4) 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(

  # 1) CI and vendored libraries ------------------------------------

  top_level == ".github", "ci_cd",
  top_level == ".lib" | grepl("/\\.lib/", rest),"vendored_lib",

  # 2) 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",

  # 3) Tests (incl. SystemTests, case-insensitive) ------------------

  grepl("SystemTests/", rest, ignore.case = TRUE), "tests",
  grepl("/tests?/|^tests?/", rest, ignore.case = TRUE), "tests",

  # 4) Couplers: NUOPC / LILAC / CESM / cpl -------------------------

  grepl("cpl_|/cesm/|/cpl/|/cpl_",rest), "coupler",

  # 5) Drivers: Fortran code clearly in drivers directories ---------

  grepl("(^|/)drivers(/|$)", rest) & language == "fortran", "driver",
```

```r
  # 6) 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",

  # 7) Everything else: model core ----------------------------------

  default = "model_core"
)]

# Create column to include in risk calculations or not ------------------------

all_callgraphs[, include_in_risk:= (language %chin% c("fortran", "python") &
                                    component %chin% c("model_core", "coupler"))]

# SUmmarize --------------------------------------------------------------------

all_callgraphs[, .N, component]
```

```
##       component     N
##          <char> <int>
## 1:        ci_cd     5
## 2: vendored_lib   100
## 3:  cli_or_tool  1334
## 4:    framework  2091
## 5:   model_core 19946
## 6:        tests  1851
## 7:      coupler   485
```

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

```
##    include_in_risk     N
##             <lgcl> <int>
## 1:           FALSE  5381
## 2:            TRUE 20431
```

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

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

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

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, language)]
```

```r
# 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, lang) {

  comp_sub <- cc_unique[model == m & language == lang]

  # 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(indeg = centrality_degree(mode = "in"),
```

```r
          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, language)]

# 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 --------------------------------------

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

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

all_graphs[!c(model == "VIC" & language == "python"),
           uncertainty_sensitivity:= Map(full_ua_sa_risk_fun, node_df, paths_tbl, N, order)]
```

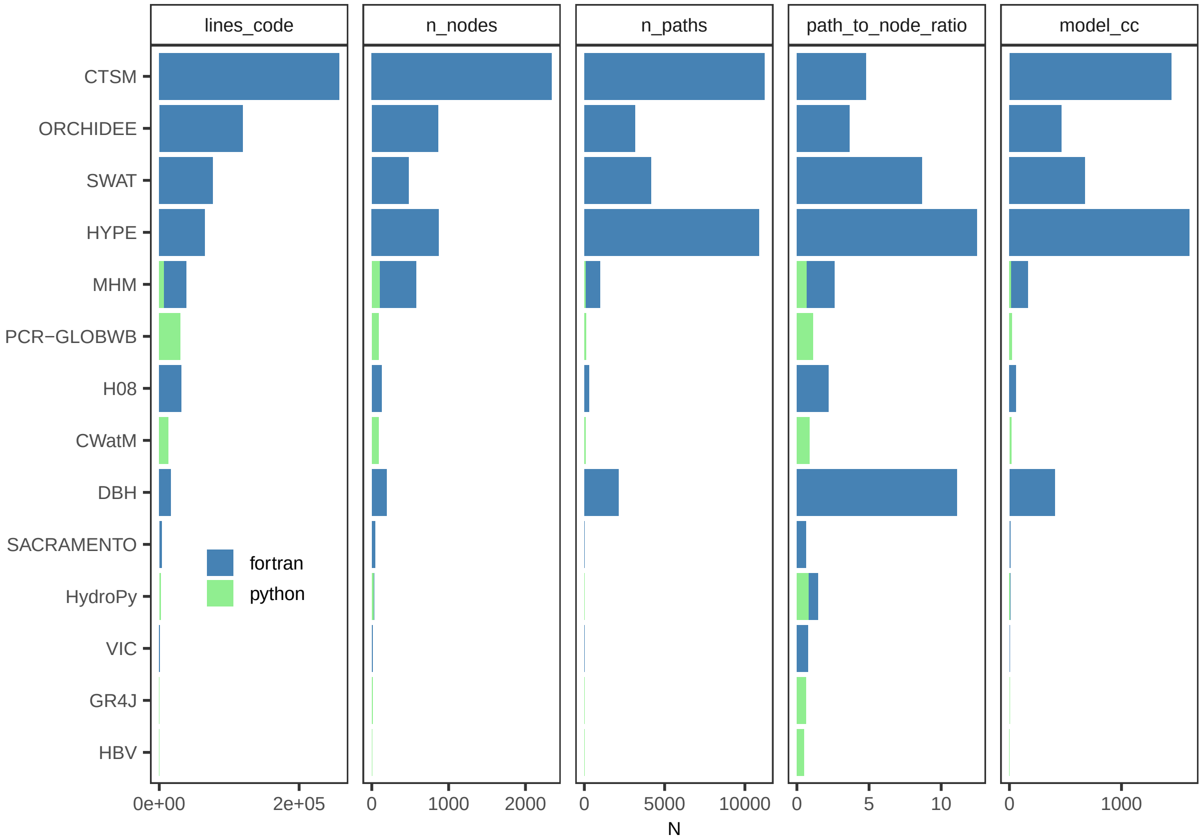
```r
# UNNEST APPROPRIATELY #########################################################

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, language, 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, language, 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, language)] %>%
  .[order(-n_paths)]

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

```r
# Read descriptive_stats_file ---------------------------------------------------

descriptive_stats <- data.table(read_xlsx("./datasets/descriptive_statistics/descriptive_statis
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, fill = language)) +
  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.
```

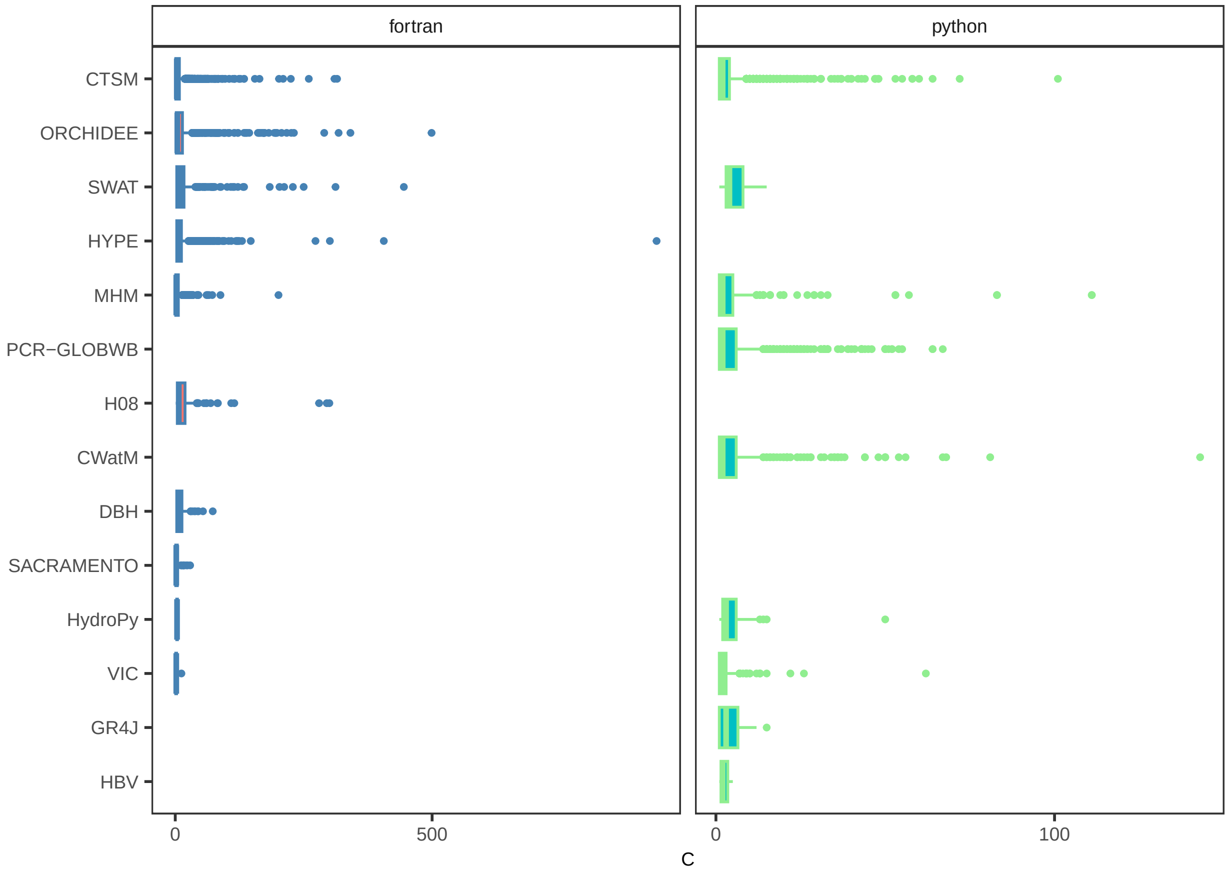
```r
plot_descriptive
```




```r
# 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)))

# Create function to combine files ---------------------------------------------

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

```r
}

# 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 ----------------------------------------------------------------

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 * ")"))
```

```r
# 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])] %>%
  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
```

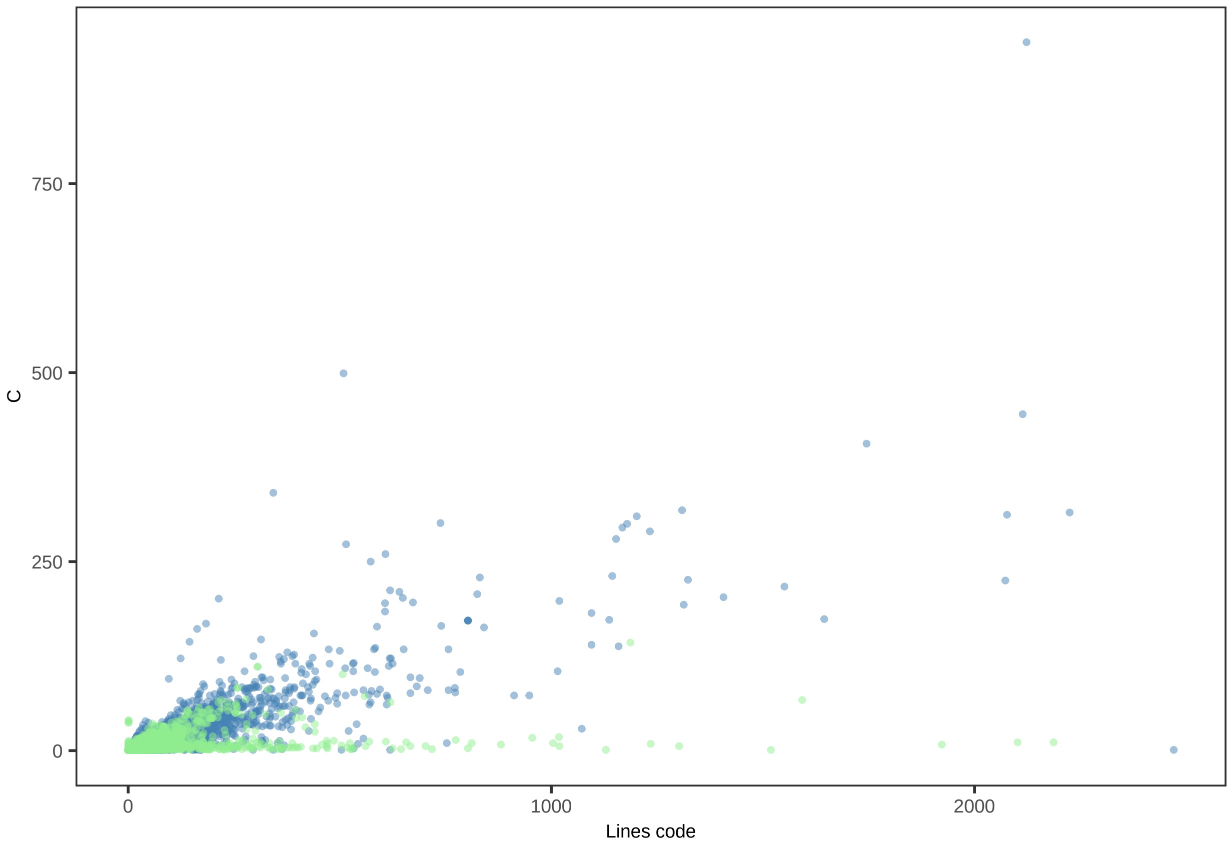



```
## ----plot_scatter_and_bar, dependson="read_metrics_function_data", fig.height=2.5, fig.width=

# 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")
```

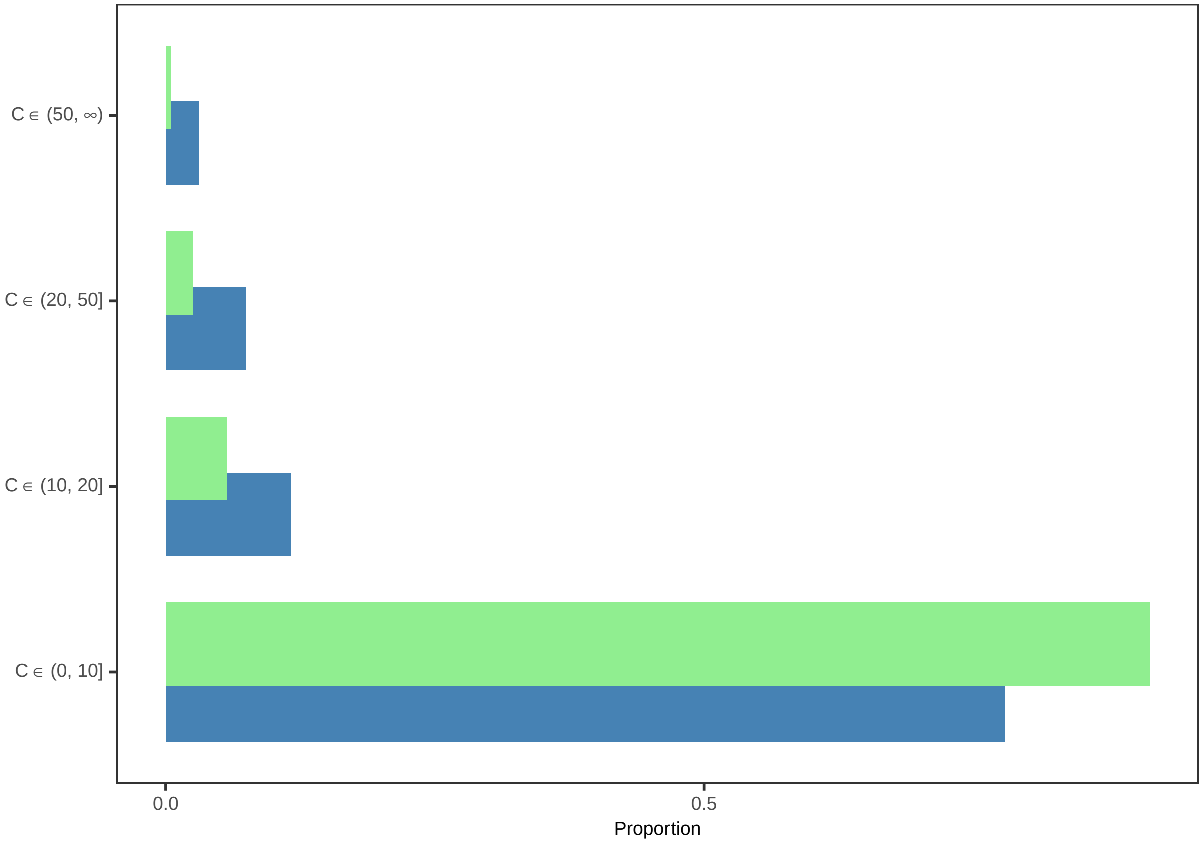
```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```

```
plot_c_vs_loc
```

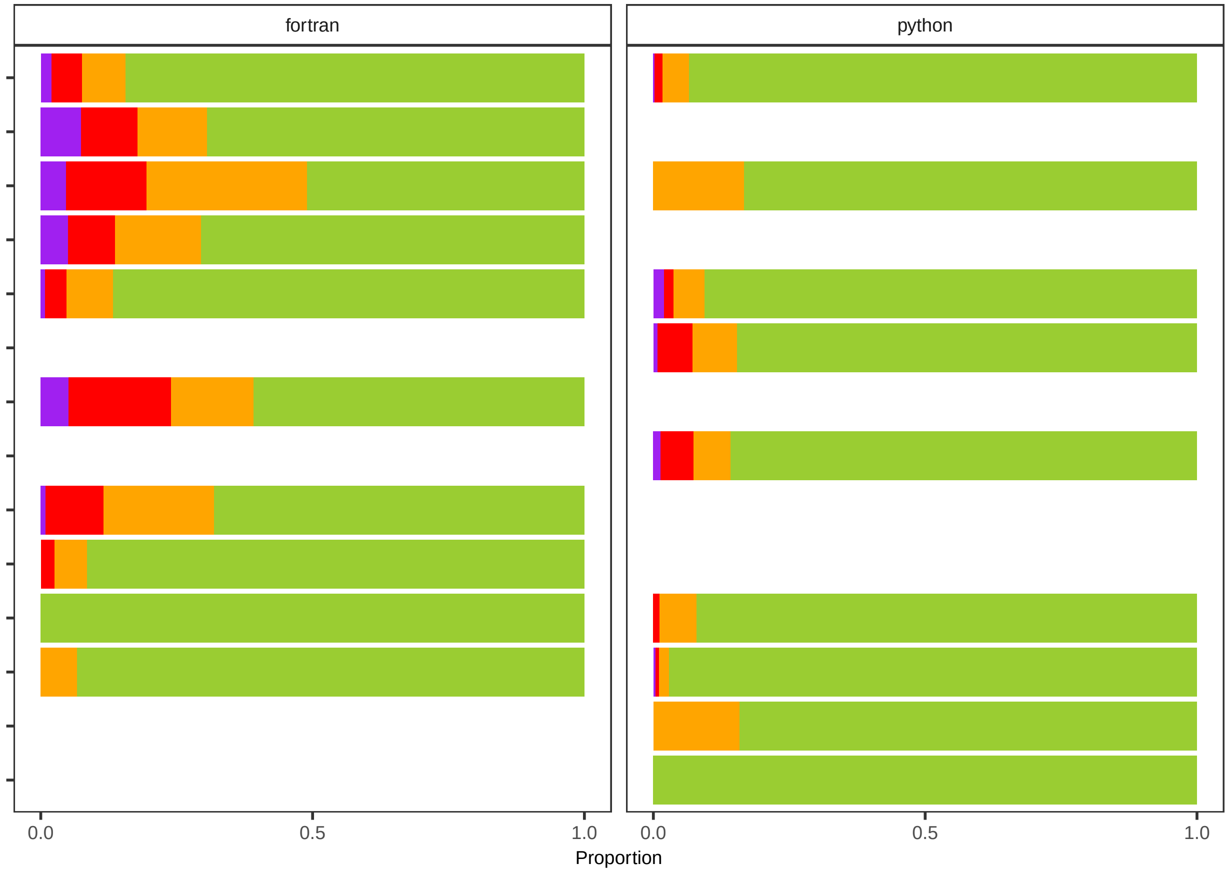
```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```r
# 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

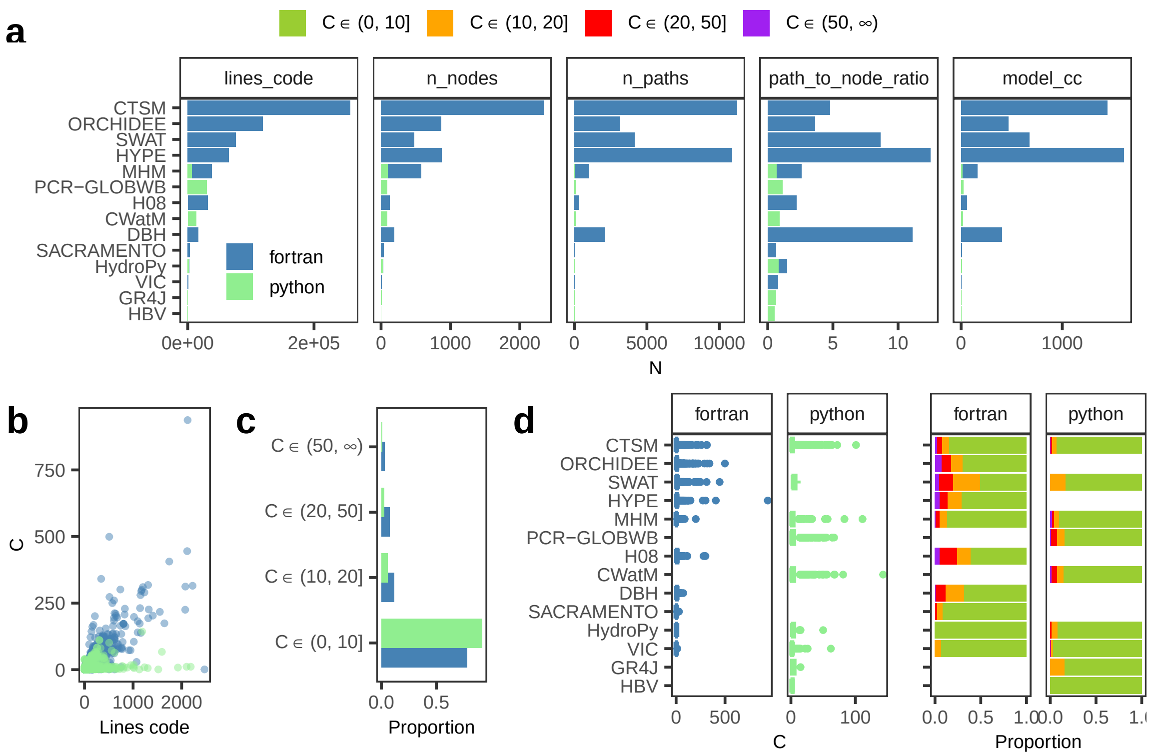
```
# MERGE FIGURES ################################################################

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")
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"))
```

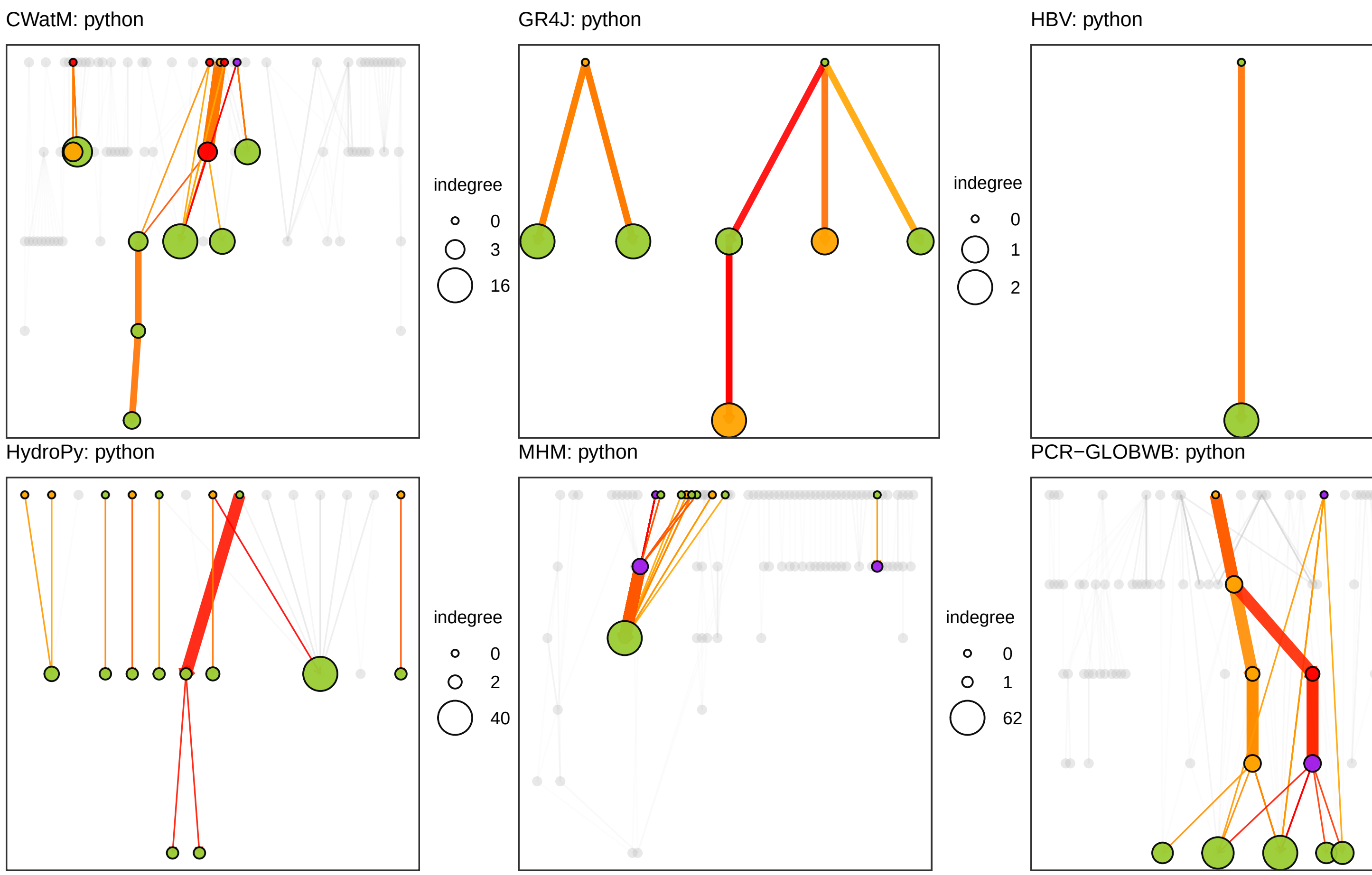
```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

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

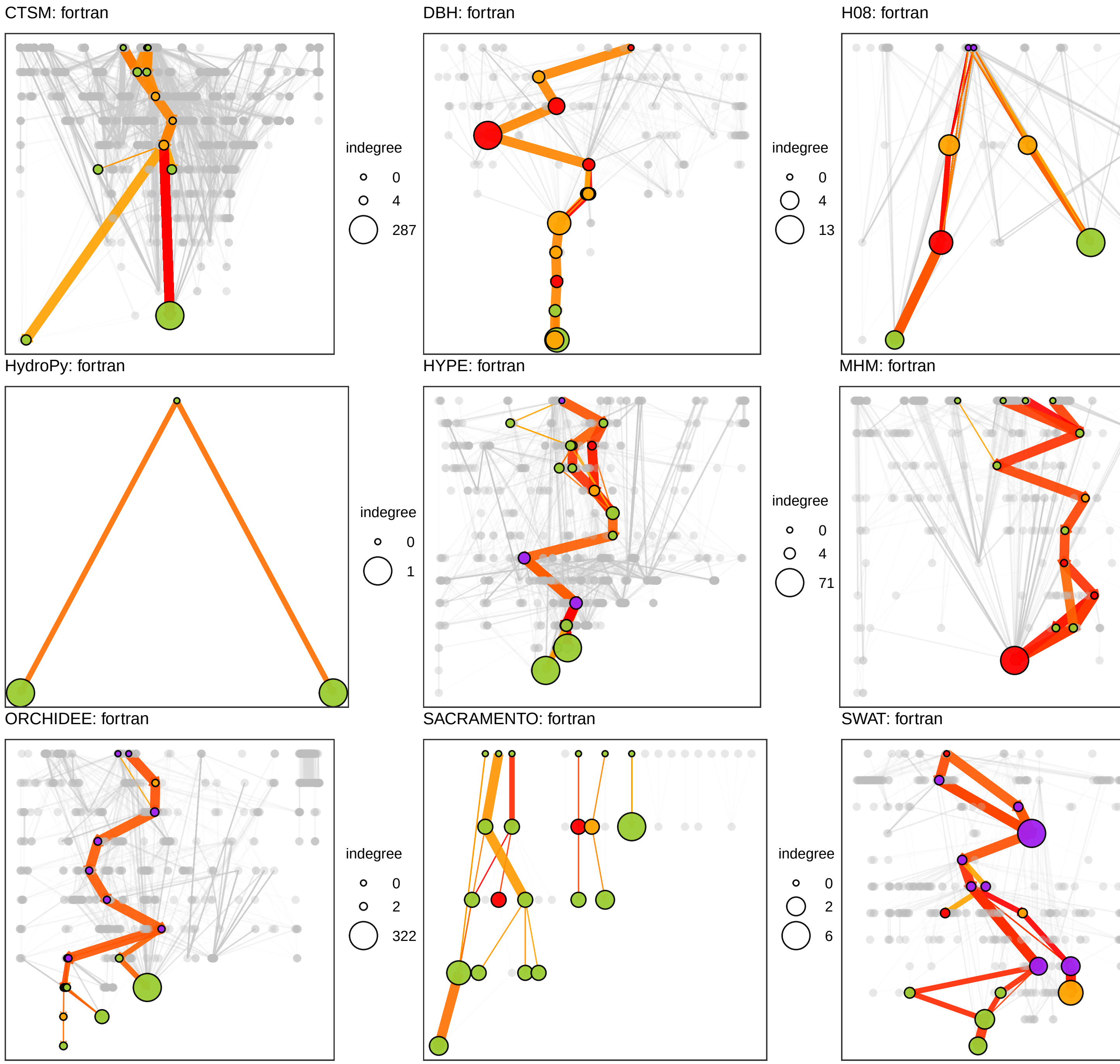


4 Figures

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

# 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)]
```

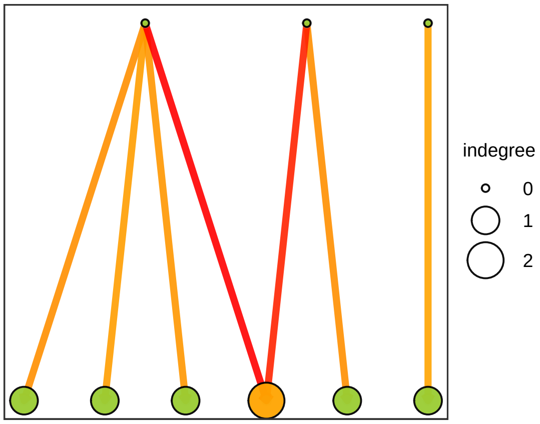



```
## No paths in paths_tbl; skipping plot for: VIC
```



CTSM: fortran
indegree
0
4
287
DBH: fortran
indegree
0
4
13
H08: fortran
HydroPy: fortran
indegree
0
1
HYPE: fortran
indegree
0
4
71
MHM: fortran
ORCHIDEE: fortran
indegree
0
2
322
SACRAMENTO: fortran
indegree
0
2
6
SWAT: fortran

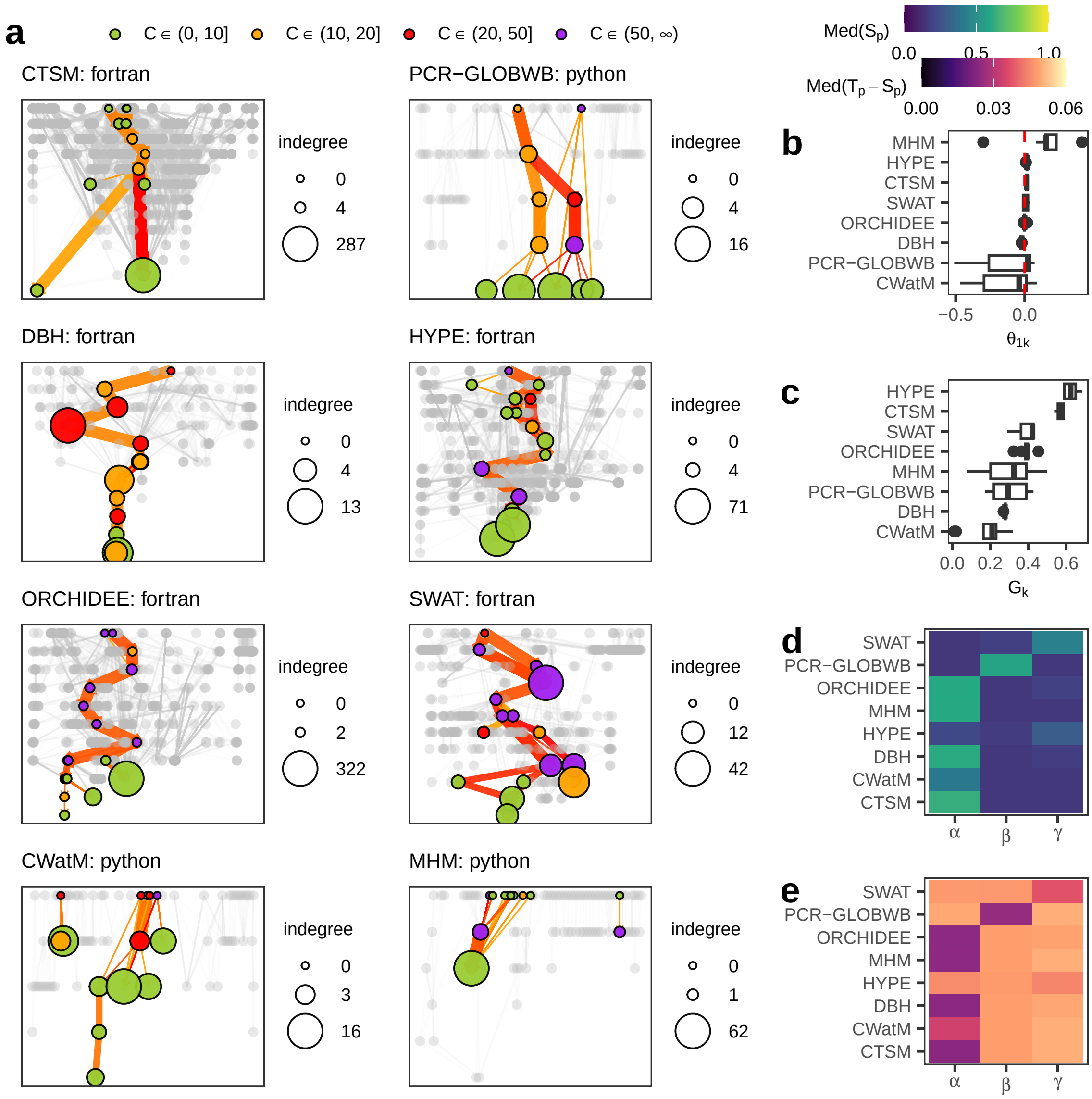
VIC: fortran




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

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

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

tmp <- all_graphs[selected_models, on = .(model, language)]
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, language)] %>%
  .[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, language)] %>%
  .[order(-p_path_fail), .SD[1:10], model] %>%
  ggplot(., aes(reorder(model, gini_node_risk), gini_node_risk)) +
```

```r
  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, language)] %>%
  .[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 <- split(full_sa_df, full_sa_df$language)
tmp2 <- lapply(tmp, data.table) %>%
  lapply(., function(x) {
    dcast(x, name + model + language + parameters ~ sensitivity,
          value.var = "original") %>%
      .[, interaction:= Ti - Si]})

d <- do.call(rbind, tmp2) %>%
  .[selected_models, on = .(model, language)] %>%
  .[, .(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() +
```

```r
  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))
```



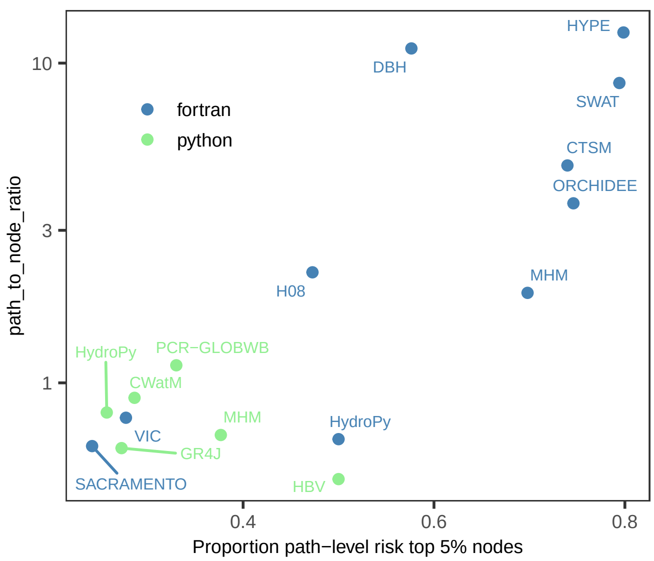

```
# PATH-LEVEL RISK ACCOUNTED FOR THE TOP 5% NODES ###############################

setDT(full_paths_df)

# 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, language, path_id)]
```

```r
# 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, language, node)]

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

node_summary <- merge(node_from_paths, full_node_df, by.x = c("model", "language", "node"),
                      by.y = c("model", "language", "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, language)
  ]
}

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

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

tmp
```

```
##          model language     X n_nodes n_top share_risk_mass_topX
##         <char>   <char> <num>   <int> <num>                <num>
## 1:        HYPE  fortran  0.05     872    44            0.7985366
## 2:        SWAT  fortran  0.05     481    25            0.7942209
## 3:    ORCHIDEE  fortran  0.05     865    44            0.7460796
```

```
## 4:       CTSM  fortran  0.05     2339    117             0.7398243
## 5:        MHM  fortran  0.05      468     24             0.6980374
## 6:        DBH  fortran  0.05      191     10             0.5763146
## 7:        HBV   python  0.05        2      1             0.5000000
## 8:    HydroPy  fortran  0.05        3      1             0.5000000
## 9:        H08  fortran  0.05      129      7             0.4726677
## 10:       MHM   python  0.05       98      5             0.3766507
## 11: PCR-GLOBWB  python  0.05       89      5             0.3300054
## 12:     CWatM   python  0.05       84      5             0.2862374
## 13:       VIC  fortran  0.05        9      1             0.2772493
## 14:      GR4J   python  0.05        8      1             0.2726486
## 15:   HydroPy   python  0.05       26      2             0.2571617
## 16: SACRAMENTO fortran  0.05       41      3             0.2418015
```

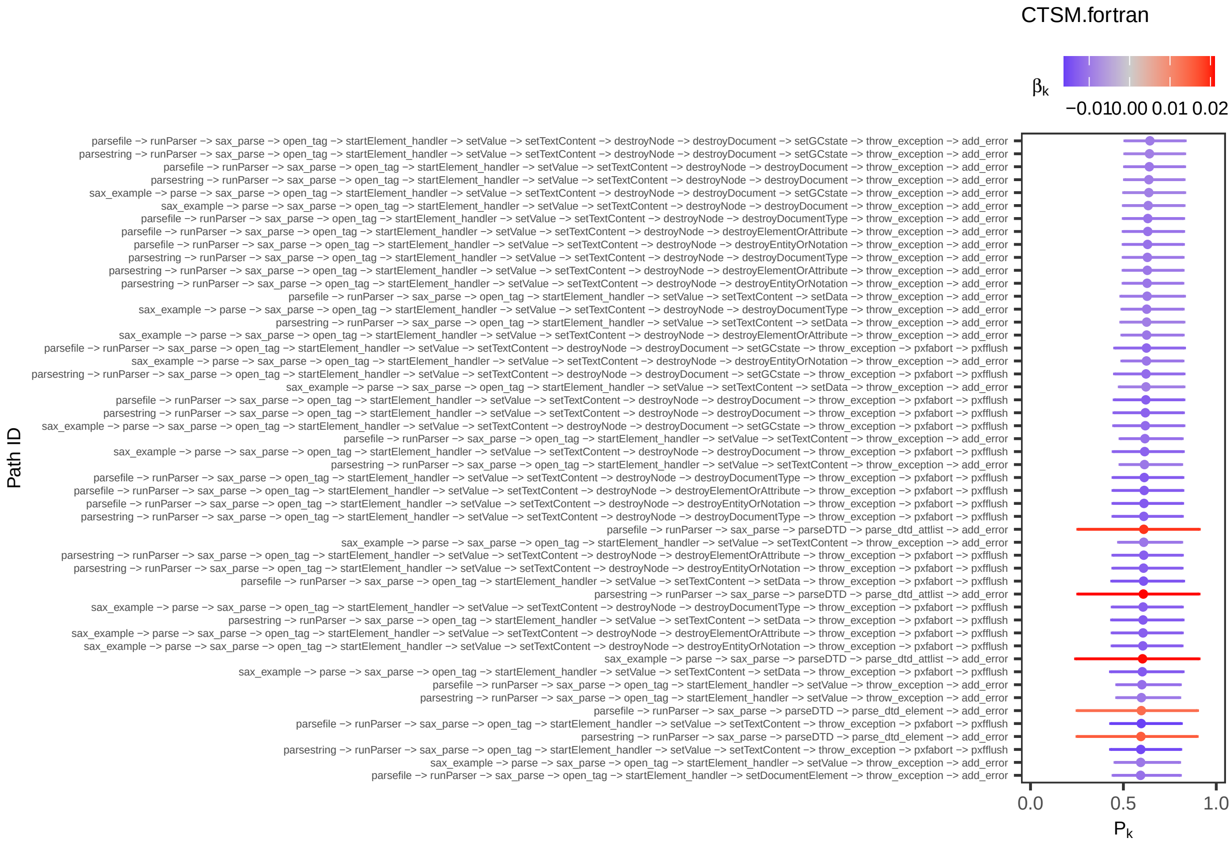
```
# Plot---------------------------------------------------------------------

merge(all_descriptive_df, tmp, by = c("model", "language")) %>%
  ggplot(., aes(share_risk_mass_topX, path_to_node_ratio, color = language)) +
  geom_point() +
  scale_color_manual(values = color_languages, 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))
```



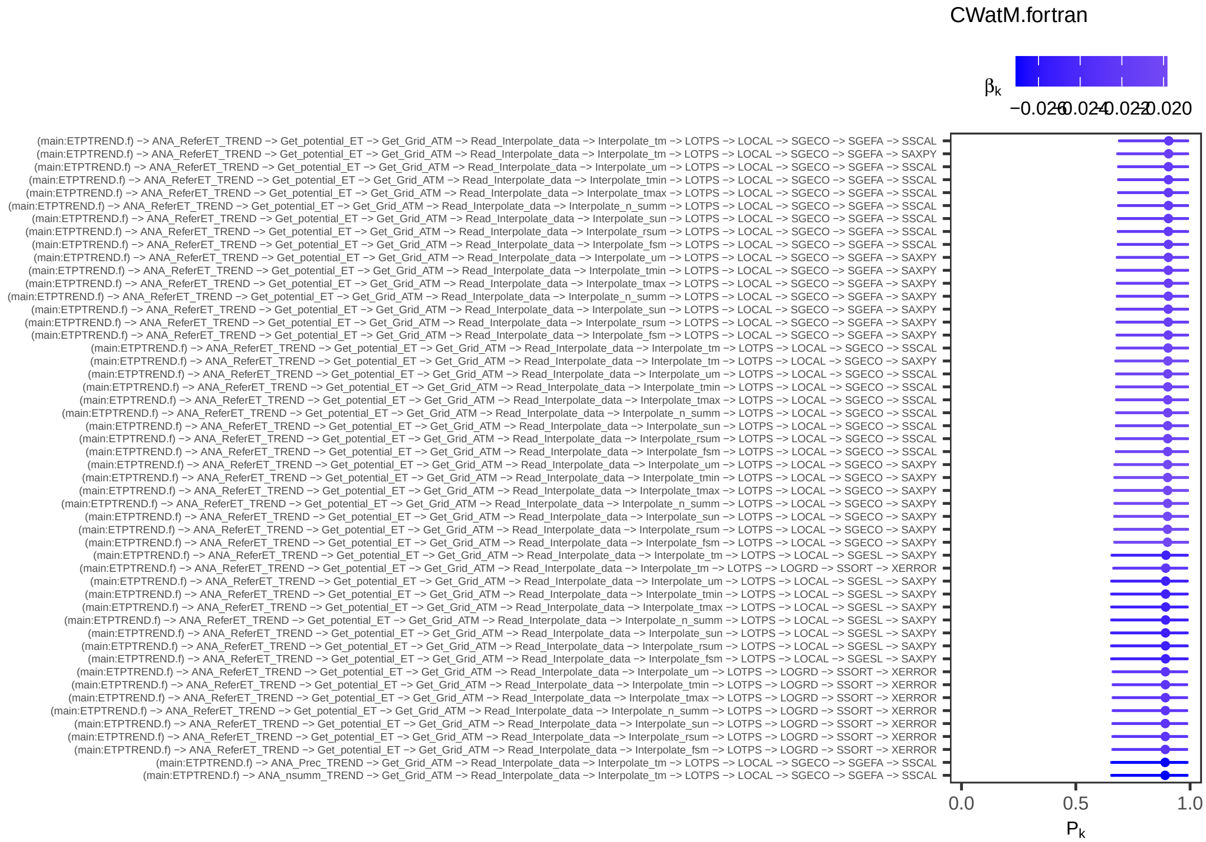

```
# PLOT THE TOP 50 PATHS PER MODEL ##############################################

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

```r
# Remove empty slots --------------------------------------------------------

tmp2 <- tmp[sapply(tmp, function(x) nrow(x) > 0)]

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

out <- list()

for ( i in 1:length(tmp2)) {

  out[[i]] <- ggplot(tmp2[[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`.
```

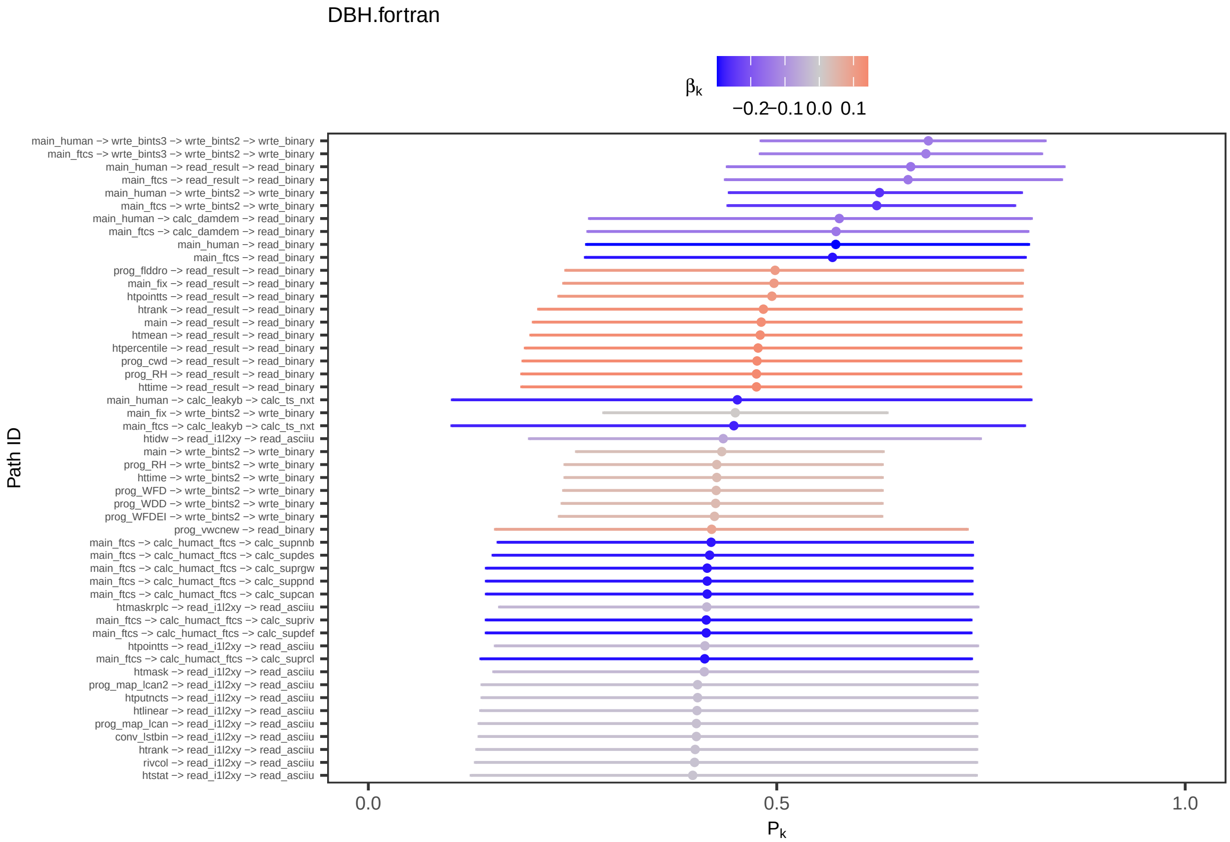


```
##
## [[2]]

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



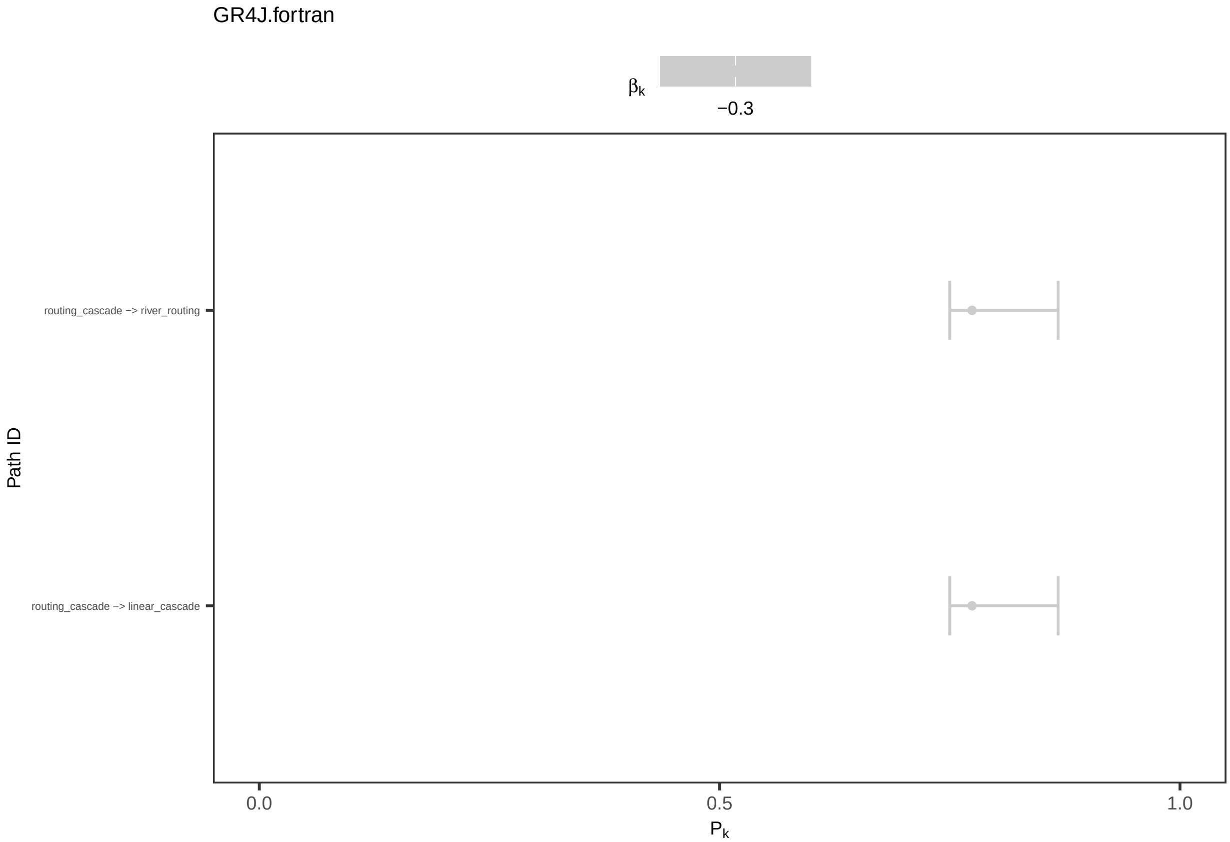
```
##
## [[3]]
```

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




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

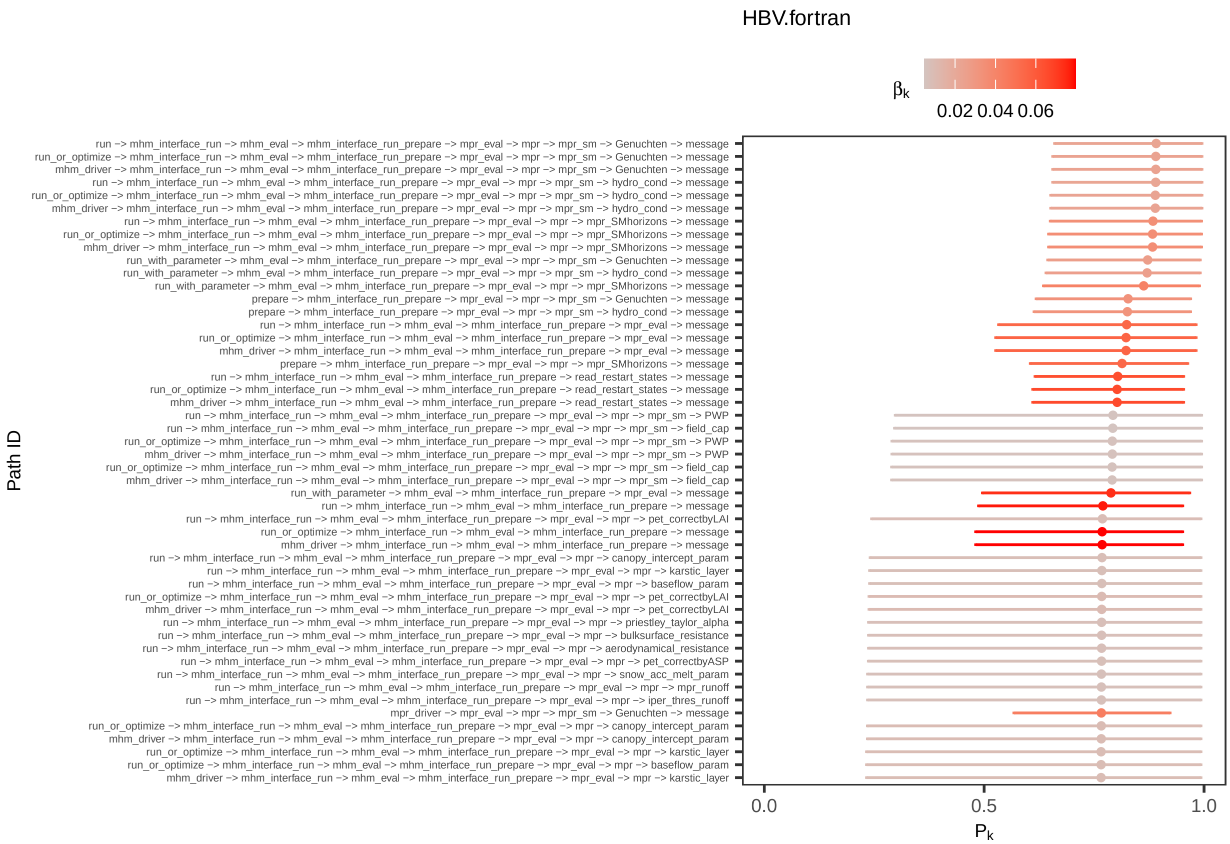
```
## `height` was translated to `width`.
```




```
##
## [[5]]
## `height` was translated to `width`.
```

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

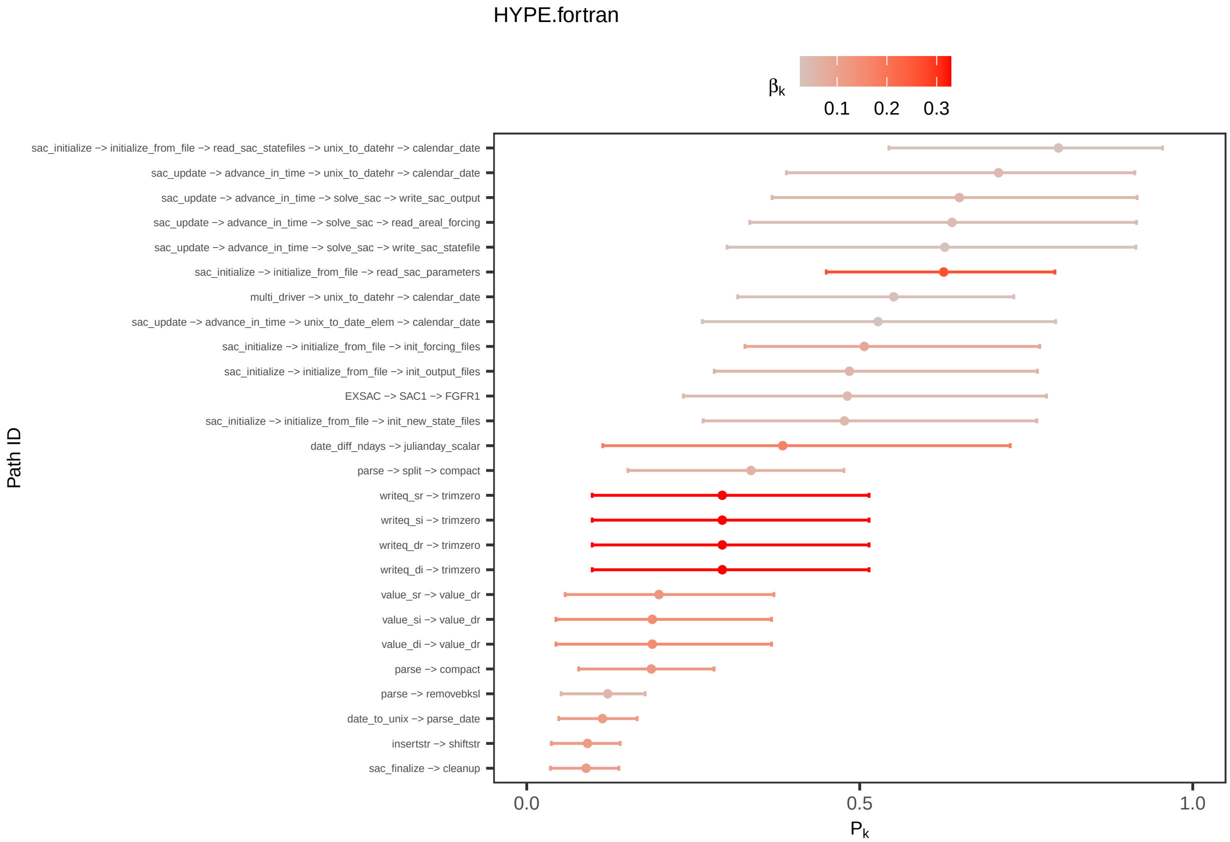
```
## `height` was translated to `width`.
```




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

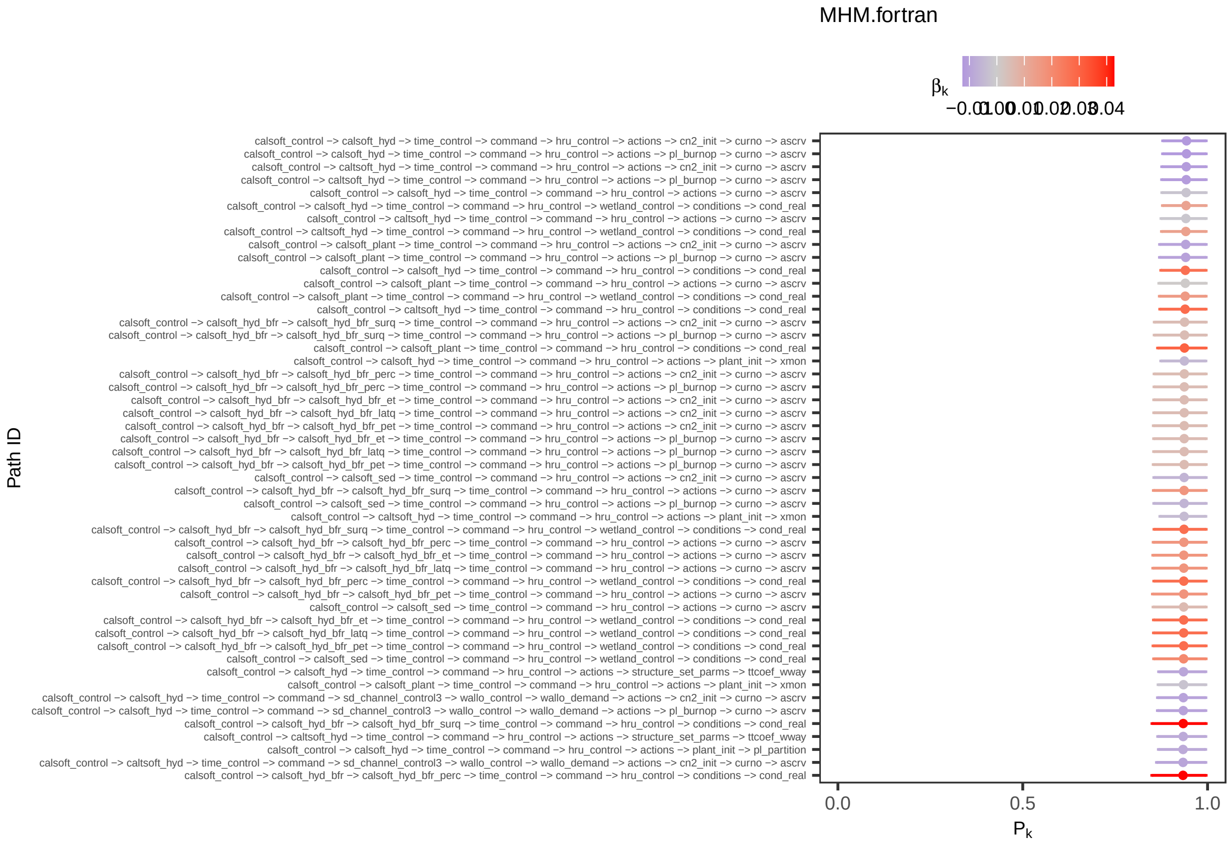
```
## `height` was translated to `width`.
```


```
##
## [[8]]
## `height` was translated to `width`.
```



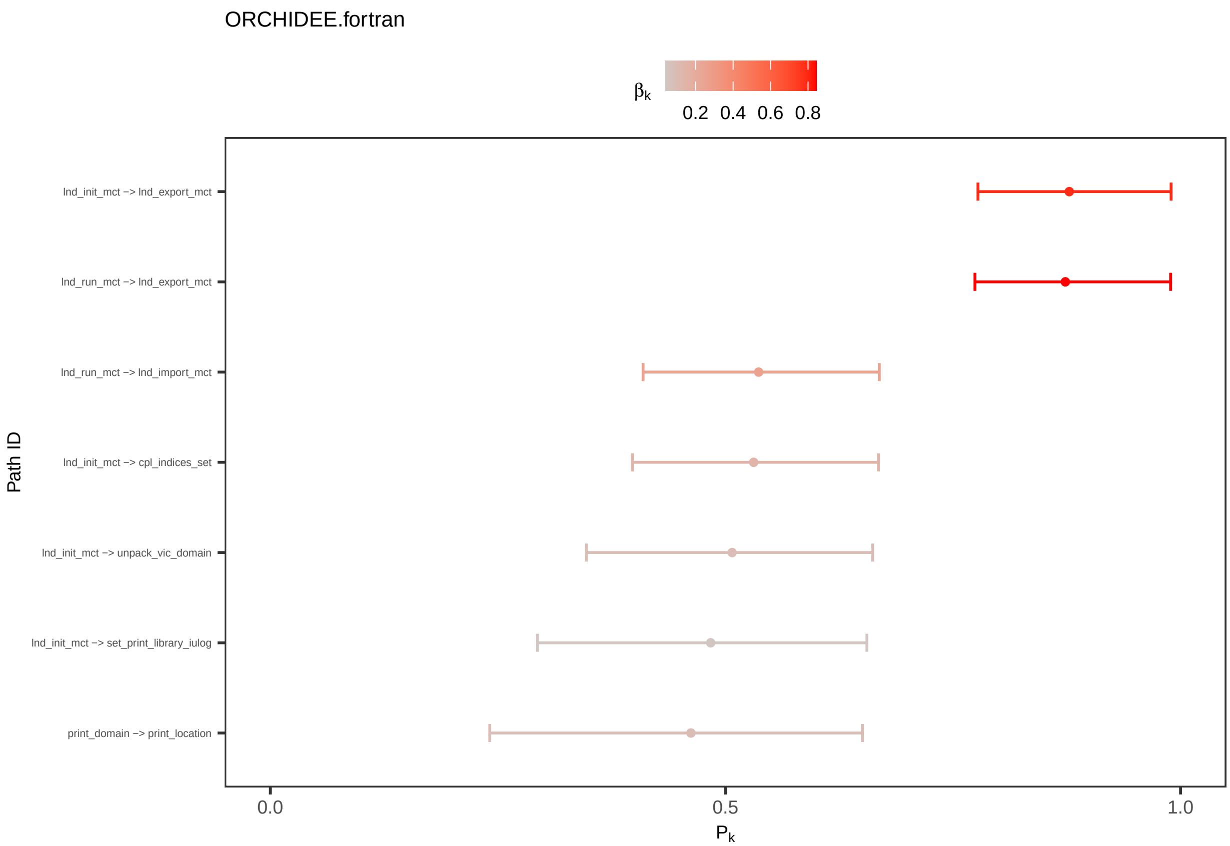

```
##
## [[9]]
```

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



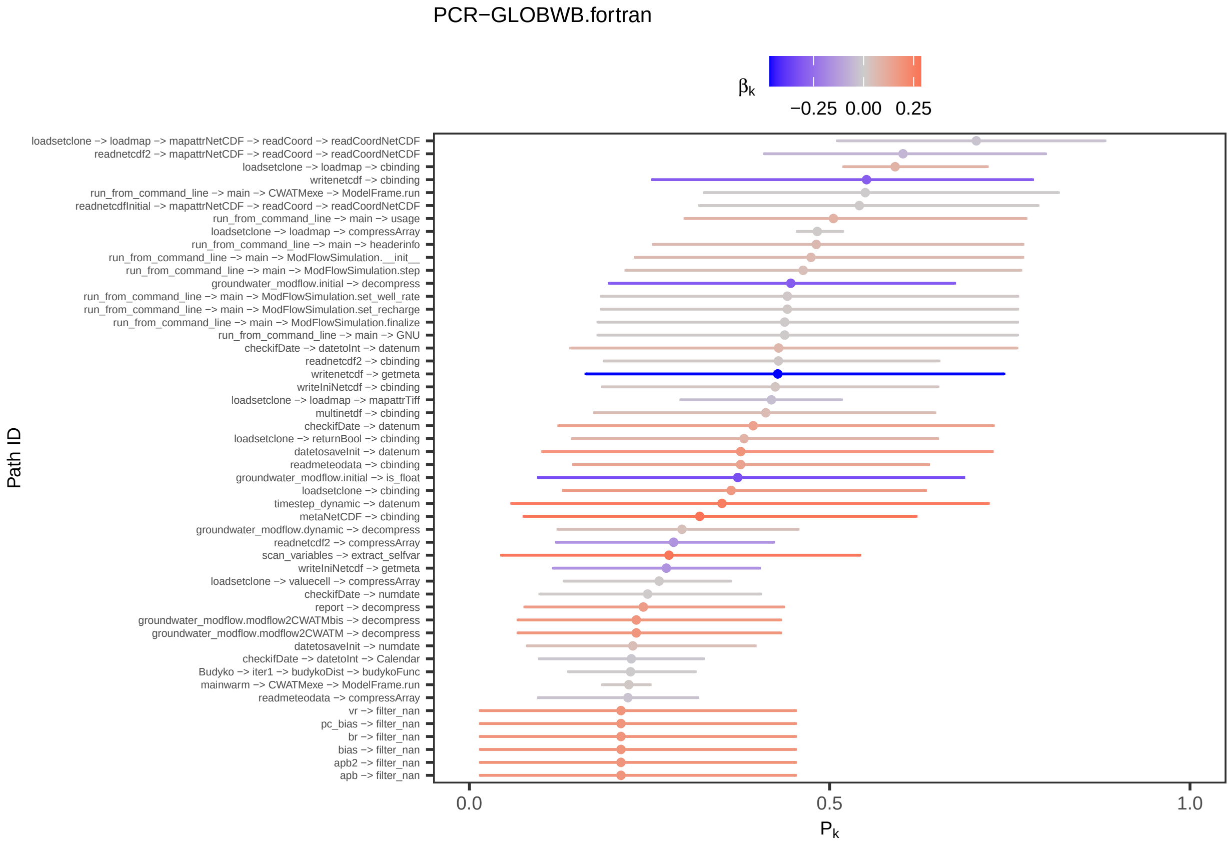

```
##
## [[10]]
```

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

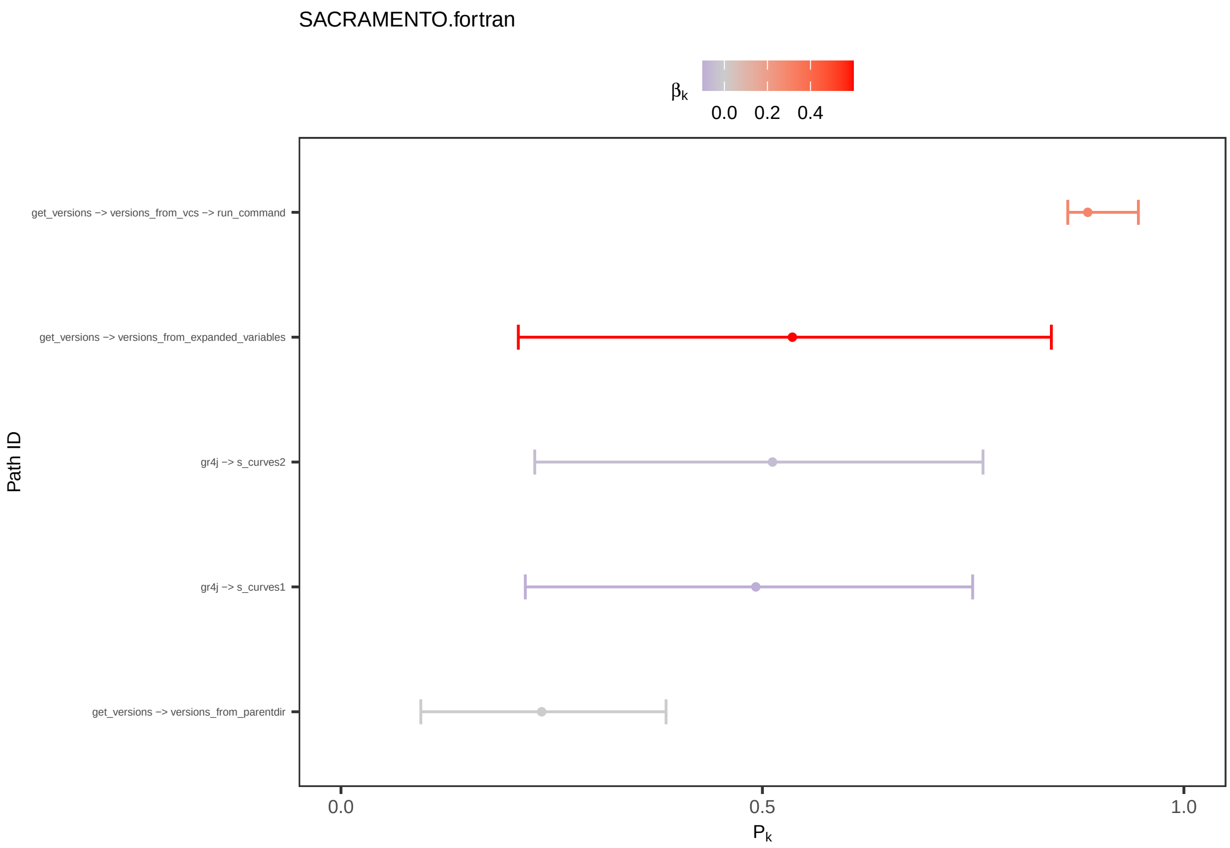



```
##
## [[11]]
```

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



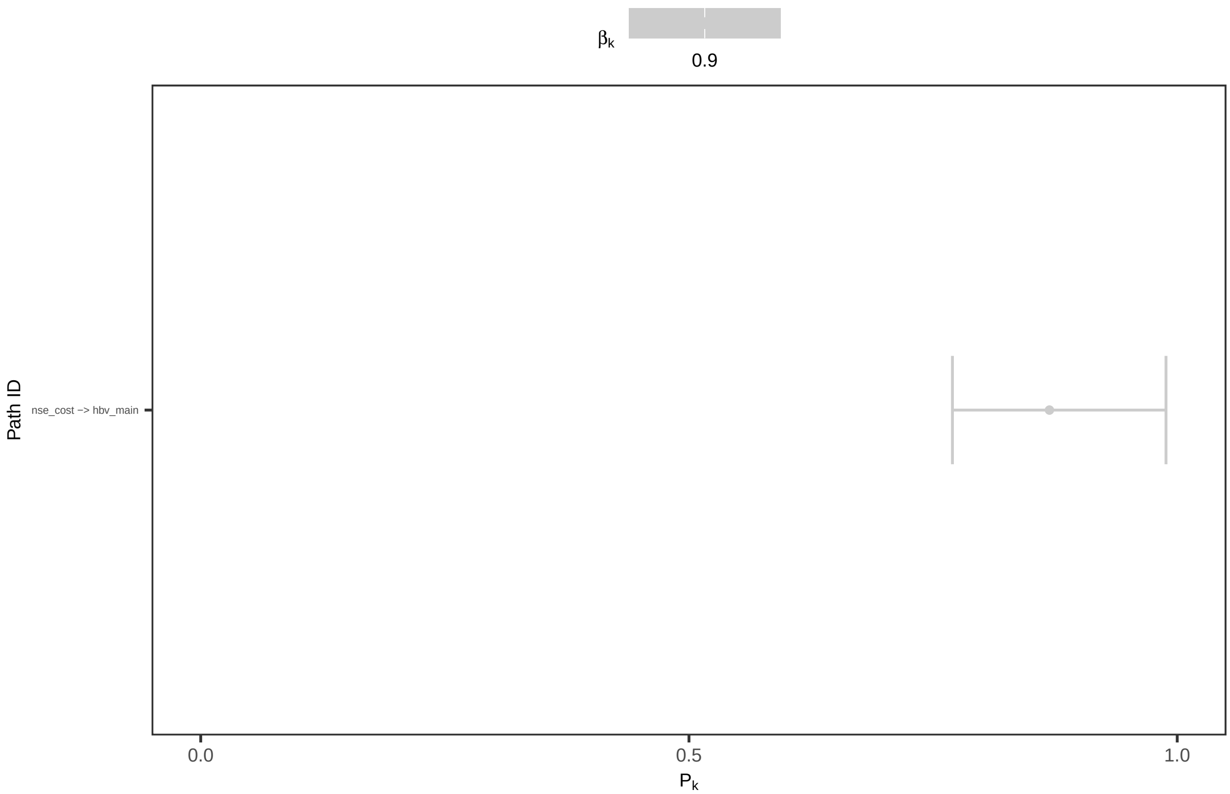

```
##
## [[12]]
## `height` was translated to `width`.
```




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

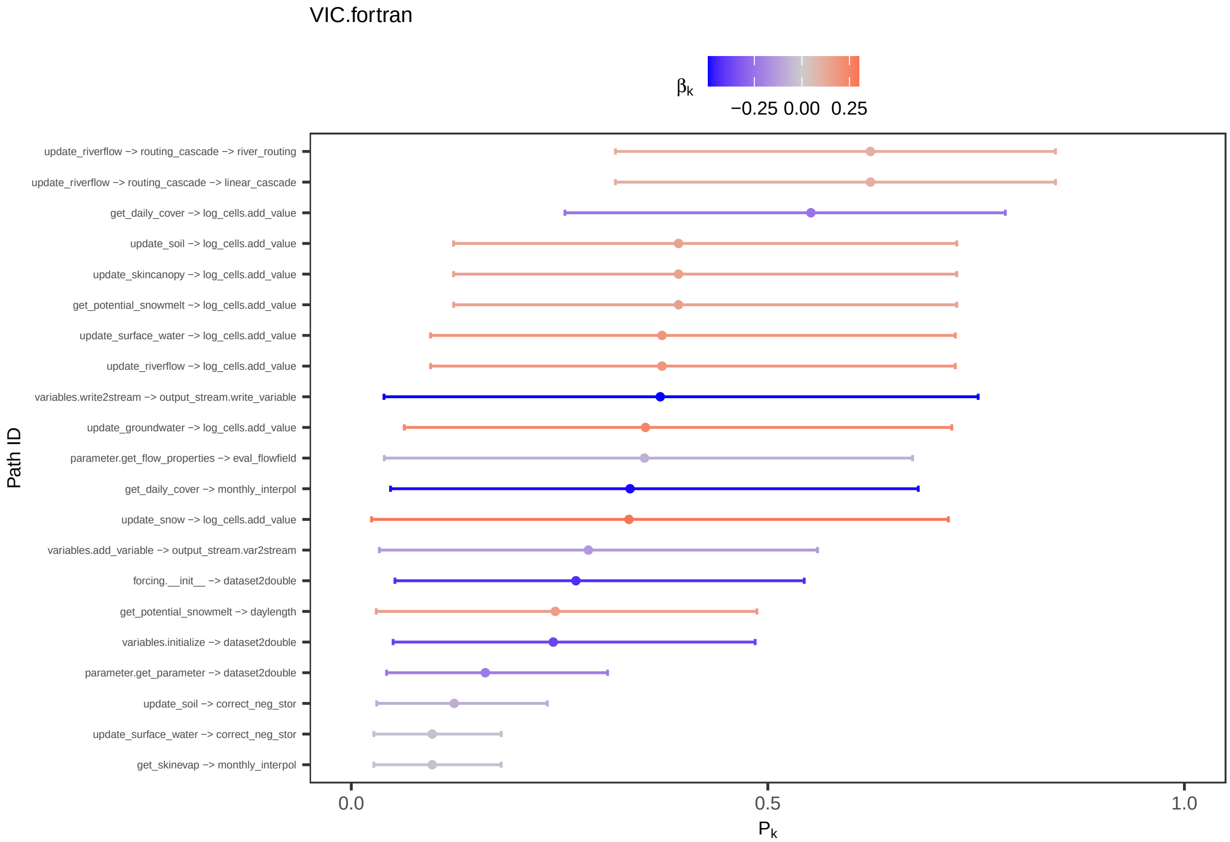
```
## `height` was translated to `width`.
```

SWAT.fortran



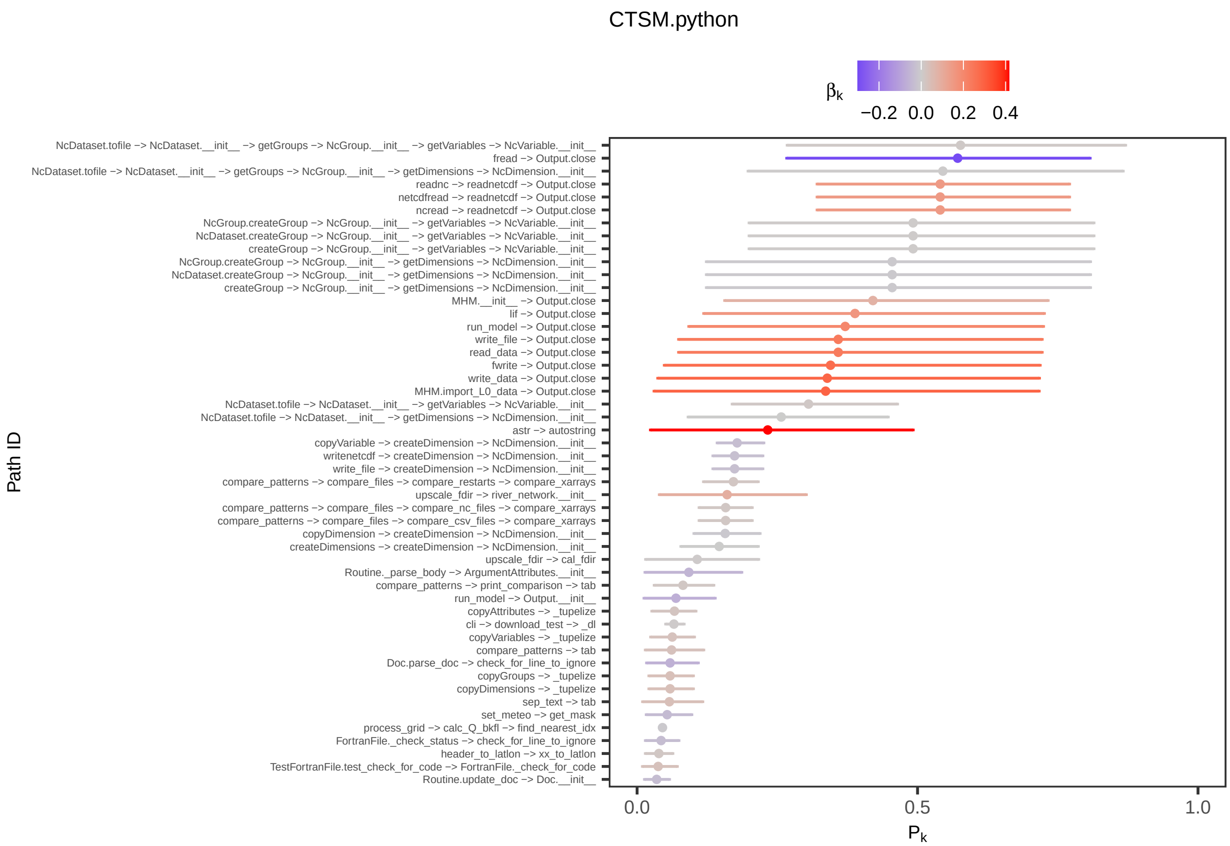

```
##
## [[14]]
```

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



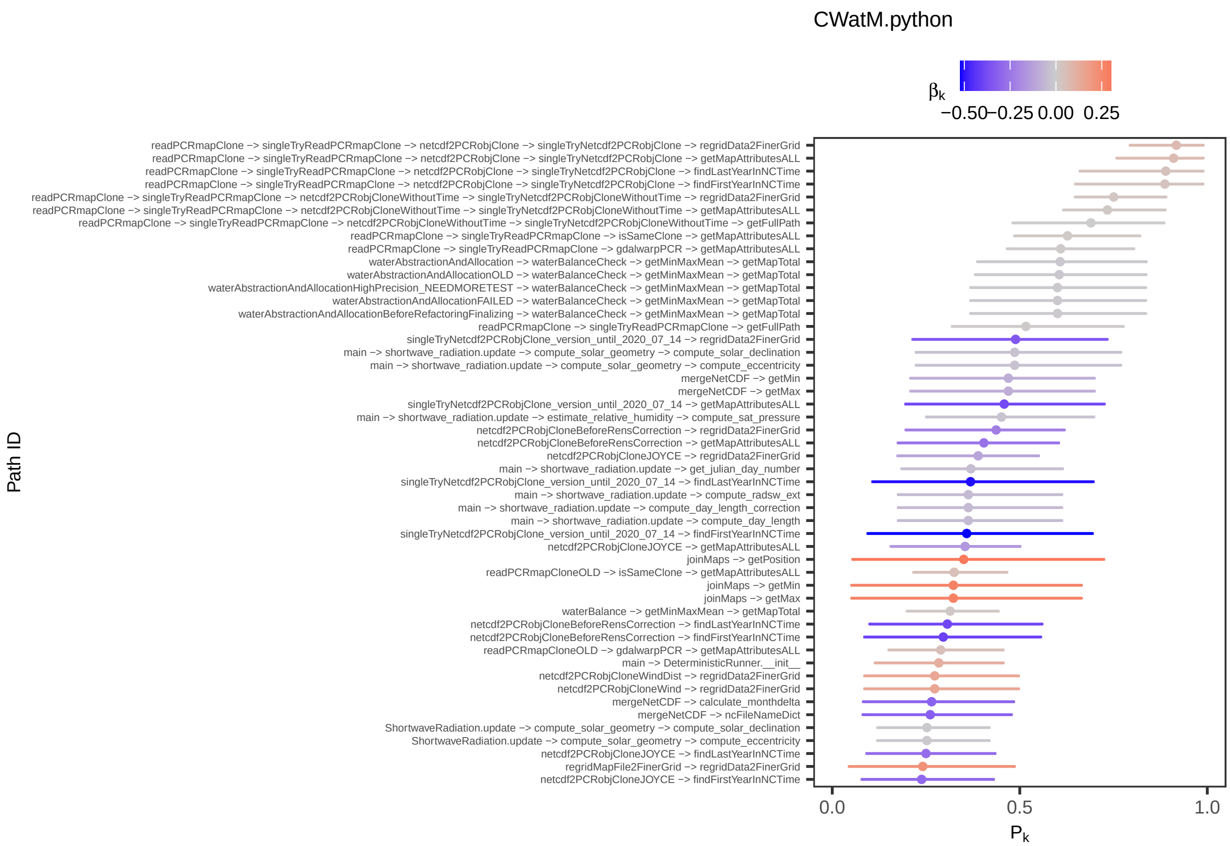

```
##
## [[15]]
```

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



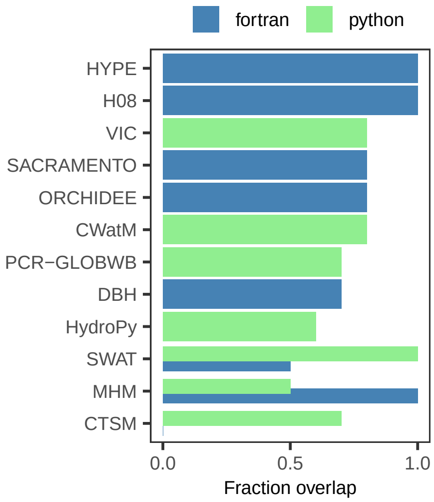

```
##
## [[16]]
```

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



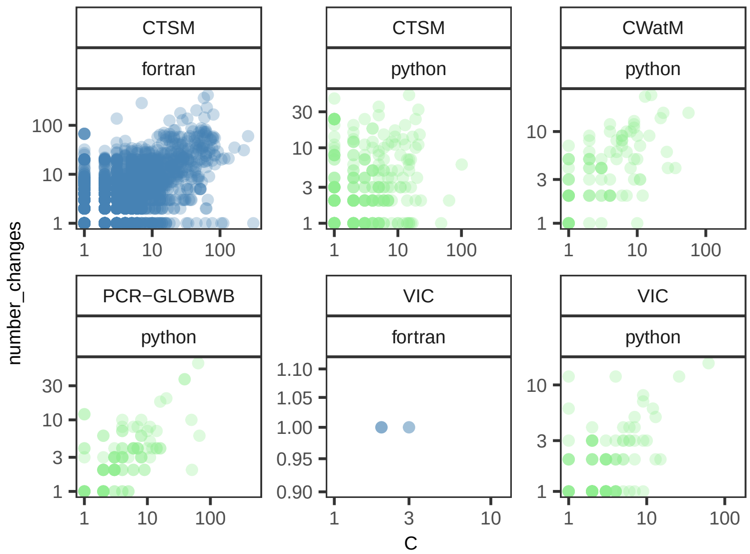

```r
# READ RANKING OF THE UA / SA DATASET ##########################################

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)

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

plot_logs <- lapply(csv_files, fread) %>%
  lapply(., function(x) x[, .(model, language, cyclomatic_complexity, number_changes,
                              change_per_days, age_days)]) %>%
  rbindlist() %>%
  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~language, scales = "free") +
  theme_AP() +
  theme(legend.position = "none")

plot_logs
```

```
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.

## Warning: Removed 6629 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```r
# 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, language)] %>%
  .[, .(model, language, path_str)]

tmp2 <- split(tmp, list(tmp$model, tmp$language))

tmp3 <- tmp2[sapply(tmp2, nrow) > 0] %>%
  lapply(., function(x) na.omit(x) %>%
           .[, .(path_str)])

to_tex_list_fun(tmp3)
```

5 Session information

```r
# 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.framewo
## 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_2.1.4       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_1.5.2
## [17] dplyr_1.1.4        purrr_1.1.0        readr_2.1.5        tidyr_1.3.1
## [21] tibble_3.3.0       ggplot2_4.0.0.9000 tidyverse_2.0.0    data.table_1.17.8
##
## 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_0.2.0             fastmap_1.2.0
##  [7] tweenr_2.0.3         janeaustenr_1.0.0    digest_0.6.37
## [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_2.3.10          knitr_1.50           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_0.57         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_0.5.8.1     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_2.3          SnowballC_0.7.1       Rfast_2.1.5.1
## [61] memoise_2.0.1          Rcpp_1.1.0            zip_2.3.3
## [64] gridExtra_2.3          xfun_0.53             pkgconfig_2.0.3
```

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

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

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

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

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

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

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

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

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