Triage Against the Machine: Can AI Reason Deliberatively?

Francesco Veri, Gustavo Umbelino

2025-04-04

Preparing data for analysis

Define helper functions

Maybe move this to it's own package…

```
create_file_path <- function(provider, model, survey, file_type) {
  file.path("llm_data", provider, model, survey, paste0(file_type, ".csv"))
}

output.file.exists <- function(file_name) {
  file.exists(paste(OUTPUT_DIR, file_name, sep = "/"))
}

read_output_csv <- function(file_name) {
  read_csv(paste(OUTPUT_DIR, file_name, sep = "/"), show_col_types = FALSE)
}

write_output_csv <- function(data, file_name) {
  write_csv(data, paste(OUTPUT_DIR, file_name, sep = "/"))
}

now_utc <- function() {
   now <- Sys.time()
   attr(now, "tzone") <- "UTC"
   now
}
```

Get available LLMs

```
# get models info
models <- read_csv(LLMS_FILE, show_col_types = FALSE)

# initialize a vector to store the 'has_data' values
has_data_flags <- logical(nrow(models))

# iterate over each row in the models data frame
for (i in 1:nrow(models)) {
  provider <- models$provider[i]
  model <- models$model[i]

  # create the data path
```

```r
  path <- paste0("llm_data/", provider, "/", model)

  # check if the path exists and set the 'has_data' flag accordingly
  has_data_flags[i] <- file.exists(path)
}

# add the 'has_data' column to the models data frame
models <- models %>%
  mutate(has_data = has_data_flags) %>%
  arrange(provider, model)

# print rows where has_data is TRUE
if (!any(models$has_data)) {
  warn("No data available!")
}

# print ordered survey names
for (i in 1:nrow(models)){
  cat(paste0(i, ". ", models[i,]$provider, "/", models[i,]$model, if(models[i,]$has_data) " | has data"
}
```

```
## 1. anthropic/claude-3-5-haiku-20241022 | has data
## 2. anthropic/claude-3-5-sonnet-20241022 | has data
## 3. anthropic/claude-3-7-sonnet-20250219 | has data
## 4. anthropic/claude-3-haiku-20240307 | has data
## 5. anthropic/claude-3-opus-20240229 | has data
## 6. anthropic/claude-3-sonnet-20240229 | has data
## 7. cohere/command | has data
## 8. cohere/command-r-08-2024 | has data
## 9. cohere/command-r-plus-08-2024 | has data
## 10. cohere/command-r7b-12-2024 | has data
## 11. deepseek/deepseek-chat | has data
## 12. deepseek/deepseek-reasoner | has data
## 13. deepseek/deepseek-v2 | has data
## 14. deepseek/deepseek-v2.5
## 15. google/gemini-1.5-flash | has data
## 16. google/gemini-1.5-flash-8b | has data
## 17. google/gemini-1.5-pro | has data
## 18. google/gemini-2.0-flash | has data
## 19. google/gemma | has data
## 20. google/gemma2:27b | has data
## 21. google/gemma3:12b | has data
## 22. meta/llama2:13b | has data
## 23. meta/llama2:70b | has data
## 24. meta/llama3.1:405B-turbo | has data
## 25. meta/llama3.2 | has data
## 26. meta/llama3.3:70b | has data
## 27. meta/llama3:70b | has data
## 28. microsoft/phi | has data
## 29. microsoft/phi2
## 30. microsoft/phi3 | has data
## 31. microsoft/phi3.5 | has data
## 32. microsoft/phi4 | has data
## 33. mistralai/ministral-3b-latest | has data
## 34. mistralai/ministral-8b-latest | has data
## 35. mistralai/mistral-large-latest | has data
## 36. mistralai/mistral-small-latest | has data
## 37. mistralai/open-mistral-7b | has data
## 38. mistralai/open-mistral-nemo | has data
## 39. mistralai/open-mixtral-8x22b | has data
## 40. mistralai/open-mixtral-8x7b | has data
## 41. openai/gpt-3.5-turbo | has data
## 42. openai/gpt-4 | has data
## 43. openai/gpt-4-turbo | has data
## 44. openai/gpt-4o | has data
## 45. openai/gpt-4o-mini | has data
## 46. openai/o1 | has data
## 47. openai/o1-mini | has data
## 48. openai/o3-mini | has data
## 49. qwen/qwen-max | has data
## 50. qwen/qwen-plus | has data
## 51. qwen/qwen-turbo | has data
## 52. qwen/qwen1.5-110b-chat | has data
## 53. qwen/qwen1.5-72b-chat | has data
## 54. qwen/qwen2-72b-instruct | has data
## 55. qwen/qwen2.5-72b-instruct | has data
## 56. qwen/qwq-plus | has data
```

There are 54 models with data out of 56 (96%).

Surveys

```r
# read the sheet names of the Excel file
survey_names <- excel_sheets(SURVEY_FILE)

# remove invalid and "template"
survey_names <- sort(survey_names[!grepl("^~", survey_names) & survey_names != "template"])

get_surveys <- function(survey_names) {

  surveys <- list()

  # Iterate over each sheet in the workbook
  for (survey_name in survey_names) {

    # Read the current sheet into a data frame
    df <- read_excel(SURVEY_FILE, sheet = survey_name)

    # Check if required columns exist
    required_columns <- c("considerations", "policies", "scale_max", "q-method")
    missing_cols <- setdiff(required_columns, colnames(df))
    if (length(missing_cols) > 0) {
      cat(
        "Sheet",
        survey_name,
        "is missing the following columns:",
        paste(missing_cols, collapse = ", "),
        "\n\n"
      )
      next
    }

    # Calculate the number of non-NA rows in "considerations" column
    n_c <- sum(!is.na(df$considerations))

    # Calculate the number of non-NA rows in "policies" column
    n_p <- sum(!is.na(df$policies))

    # Extract integer values from "scale_max" column, assuming they are already integers
    scale_max <- as.integer(na.omit(df$scale_max))

    # Extract logical (boolean) values from "q-method" column
    q_method <- as.logical(na.omit(df$`q-method`))

    surveys[[length(surveys) + 1]] <- tibble(
      survey = survey_name,
      considerations = n_c,
      policies = n_p,
      scale_max,
      q_method,
    )

  }

  surveys <- bind_rows(surveys)
  surveys

}


surveys <- get_surveys(survey_names)


# define file types
file_types <- c("considerations", "policies", "reasons")

kable(surveys %>% arrange(survey), caption = "Surveys")
```

Table 1: Surveys

survey	considerations	policies	scale_max	q_method
acp	48	5	11	FALSE
auscj	45	8	7	FALSE
bep	43	7	7	FALSE
biobanking_mayo_ubc	38	7	11	FALSE
biobanking_wa	49	7	11	FALSE
ccps	33	7	11	FALSE
ds_aargau	33	7	7	FALSE
ds_bellinzona	32	7	7	FALSE
energy_futures	45	9	11	FALSE
fnqcj	42	5	12	FALSE

survey	considerations	policies	scale_max	q_method
forestera	45	7	11	FALSE
fremantle	36	6	11	TRUE
gbr	35	7	7	FALSE
swiss_health	24	6	7	FALSE
uppsala_speaks	42	7	7	FALSE
valsamoggia	36	4	11	TRUE
zh_thalwil	31	7	7	FALSE
zh_uster	31	7	7	FALSE
zh_winterthur	30	6	7	FALSE
zukunft	20	7	7	FALSE

Read and format LLM data

```r
get_llm_data <- function() {

  # initialize an empty list to store the data frames
  data_list <- list()

  # iterate over each survey
  for (survey_name in survey_names) {

    # iterate over each row in the models data frame where has_data is TRUE
    for (i in 1:nrow(models)) {
      if (!models$has_data[i]) {
        next
      }

      provider <- models$provider[i]
      model <- models$model[i]
      min_iterations <- models$min_iterations[i]

      # check if any file for the survey exists
      survey_path <- paste0("llm_data/", provider, "/", model, "/", survey_name, "/")
      if (!any(file.exists(paste0(survey_path, file_types, ".csv")))) {
        next
      }

      # iterate over each file type
      for (file_type in file_types) {
        # create the file path
        file_path <- create_file_path(provider, model, survey_name, file_type)

        # check if the file exists
        if (!file.exists(file_path)) {
          break
        }

        # read the CSV file
        temp_data <- read_csv(file_path, show_col_types = FALSE)

        # skip file if file exists but has no data
        if (nrow(temp_data) == 0) {
          break
        }

        # select the relevant columns based on file type
        if (file_type == "considerations") {
          # initialize survey_data
          survey_data <- temp_data %>%
            rename_with(~ paste0("C", seq_along(.)),
                        starts_with("C", ignore.case = FALSE))

          # add column "survey" to meta data
          survey_data <- survey_data %>%
            mutate(survey = survey_name) %>%
            relocate(survey, .after = model)

          # ensure survey_data has columns up to C50
          # skip 8 rows of meta data
          for (j in (ncol(survey_data) - 7):50) {
            survey_data[[paste0("C", j)]] <- as.numeric(NA)
          }

          # go to next file type
          next

        } else if (file_type == "policies") {
          temp_data <- temp_data %>%
            select(cuid, starts_with("P", ignore.case = FALSE)) %>%
            rename_with(~ paste0("P", seq_along(.)),
                        starts_with("P", ignore.case = FALSE))

          # ensure temp_data has columns up to P10
          for (j in (ncol(temp_data)):10) {
            temp_data[[paste0("P", j)]] <- as.numeric(NA)
          }

        } else if (file_type == "reasons") {
          temp_data <- temp_data %>%
            select(cuid, reason) %>%
            rename(R = reason)
        }

        # merge the data frames by 'cuid' and keep all rows
        survey_data <- full_join(survey_data, temp_data, by = c("cuid"))

      }

      # add the survey_data to the list
      if (exists("survey_data")) {
        data_list[[length(data_list) + 1]] <- survey_data

        # remove the survey_data data frame to free up memory
        rm(survey_data)
      }
    }
  }

  # Combine all data frames in the list into a single data frame
  llm_data <- bind_rows(data_list)

  return(llm_data)

}

# get llm data
llm_data <- get_llm_data()

# aggregate llm_data by provider, model, and survey and N the number of rows
llm_surveys <- llm_data %>%
  group_by(provider, model, survey) %>%
  summarise(
    N = n(),
    mean_input_tokens = as.integer(mean(input_tokens)),
    mean_output_tokens = as.integer(mean(output_tokens)),
    .groups = 'drop'
  )

models <- llm_surveys %>%
  group_by(provider, model) %>%
  summarise(
    surveys_with_data = n(),
    min_iterations_completed = min(N, na.rm = TRUE),
    max_iterations_completed = max(N, na.rm = TRUE),
    .groups = 'drop'
  ) %>%
  full_join(models, by = c("provider", "model")) %>%
  mutate(done = (min_iterations <= min_iterations_completed) &
           (surveys_with_data == length(survey_names)))

models["done"][is.na(models["done"])] <- FALSE

# write resutls to file
write_csv(llm_data, paste(OUTPUT_DIR, "llm_data.csv", sep = "/"))
write_csv(llm_surveys, paste(OUTPUT_DIR, "llm_surveys.csv", sep = "/"))
```

Cost analysis

```
# calculate costs in tokens
cost_tokens <- llm_data %>%
  group_by(provider, model) %>%
  summarise(
    total_iterations_completed = n(),
    total_input_tokens = as.integer(sum(input_tokens)),
    total_output_tokens = as.integer(sum(output_tokens)),
    input_output_ratio = total_input_tokens/total_output_tokens,
    .groups = 'drop'
```

```r
  )

models <- full_join(models, cost_tokens, by = c("provider", "model")) %>%
  mutate(
    cost_input = (total_input_tokens / 1000000) * price_1M_input,
    cost_output = (total_output_tokens / 1000000) * price_1M_output,
    total_cost = cost_input + cost_output
  )


api_costs <- read_excel(EXPENSES_FILE)

api_costs <- api_costs %>%
  group_by(api) %>%
  summarise(
    credits_paid = sum(as.numeric(credits), na.rm = TRUE),
    total_cost = sum(as.numeric(paid), na.rm = TRUE),
    .groups = "drop"
  )

api_costs <- models %>%
  group_by(api) %>%
  summarise(
    credits_used = sum(total_cost, na.rm = TRUE),
    estimate = sum(total_estimate, na.rm = TRUE)) %>%
  full_join(api_costs, by="api") %>%
  mutate(
    credits_left = credits_paid - credits_used,
  ) %>%
  select(
    api,
    credits_paid,
    credits_used,
    credits_left,
    total_cost,
    estimate) %>%
  arrange(desc(credits_paid))

write_csv(api_costs, paste(OUTPUT_DIR, "api_costs.csv", sep = "/"))

cat("Total spent: USD", sum(api_costs$credits_paid, na.rm = TRUE))
```

```
## Total spent: USD 225.19
```

Execution analysis

NOTE: Execution data is NOT completely accurate. A few (3-5) executions failed and, as a result, we have no record of it.

```r
# read exec log
# NOTE: not all executions were logged due to technical issues
# so the number of completions in this log will not add up to those in llm_data
exec_log <- read_csv(EXEC_LOG_FILE, show_col_types = FALSE)
```

```r
# fill missing columns
exec_log <- exec_log %>%
  arrange(model) %>%
  mutate(
    fixed_num_errors = (`num surveys` * `num iterations`) - (`num fail completions` + `num success compl
  ) %>%
  relocate(fixed_num_errors, .before = `num errors`) %>%

  # remove trial columns
  filter(is.na(`template success rate (%)`)) %>%
  select(-`template success rate (%)`) %>%

  # remove models with no completion data
  filter(`num completions` > 0)


exec_models <- exec_log %>%
  group_by(provider, model) %>%
  summarise(
    num_exec = n(),
    total_cost_USD = sum(`total cost ($)`, na.rm = TRUE),
    total_time_min = sum(`total elapsed time (min)`, na.rm = TRUE),
    num_completions = sum(`num completions`, na.rm = TRUE),
    num_success = sum(`num success completions`, na.rm = TRUE),
    num_error = sum(fixed_num_errors, na.rm = TRUE),
    num_fail = sum(`num fail completions`, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  mutate(
    fail_rate = num_fail / num_completions,
    success_rate = num_success / num_completions,
    cost_per_completion = total_cost_USD / num_completions,
    cost_per_success = total_cost_USD / num_success,
    time_per_completion_s = total_time_min / num_completions * 60
  )

models <- full_join(models, exec_models, by=c("provider", "model"))

exec_models_surveys <- exec_log %>%
  pivot_longer(19:38, names_to = "survey", values_to = "success_rate", values_drop_na = TRUE) %>%
  mutate(
    success_iterations = round(success_rate / 100 * `num iterations`),
    survey = str_extract(survey, "([^ ]+)\\s*") %>%
           trimws()

  ) %>%
  group_by(provider, model, survey) %>%
  summarise(
    num_execs = n(),
    num_success = sum(success_iterations, na.rm = TRUE),
    num_iterations = sum(`num iterations`, na.rm = TRUE),
    success_rate = num_success / num_iterations,
    .groups = "drop"
  )

exec_surveys <- exec_models_surveys %>%
  group_by(survey) %>%
  summarise(
    num_execs = sum(num_execs),
    num_iterations = sum(num_iterations),
    num_success = sum(num_success),
    success_rate = num_success/num_iterations,
    .groups = "drop"
  )

exec_total <- models %>%
  summarise(
    hours = sum(total_time_min, na.rm = TRUE) / 60,
    cost_USD = sum(total_cost, na.rm = TRUE),
    .groups = "drop"
  )

exec_total <- as.data.frame(t(exec_total))
colnames(exec_total) <- c("measure")
exec_total <- rownames_to_column(exec_total, "metric")

# write summary to file
write_csv(exec_models, paste(OUTPUT_DIR, "exec_models.csv", sep = "/"))
write_csv(exec_surveys, paste(OUTPUT_DIR, "exec_surveys.csv", sep = "/"))
write_csv(exec_models_surveys, paste(OUTPUT_DIR, "exec_models_surveys.csv", sep = "/"))
write_csv(exec_total, paste(OUTPUT_DIR, "exec_total.csv", sep = "/"))
```

Time & costs to complete data collection

```r
data_left <- models %>%
  filter(!done, included) %>%
  mutate(
    completions_total = min_iterations * length(survey_names),
    completions_left = completions_total - total_iterations_completed,
    time_left_min = (time_per_completion_s * completions_left) / 60,
    cost_left = cost_per_completion * completions_left
  ) %>%
  select(
    provider, model,
    completions_total,
    completions_left,
    time_left_min,
    cost_left,
  )

write_csv(data_left, paste(OUTPUT_DIR, "data_left.csv", sep = "/"))
```

Compile model info

```r
included_models_summary <- models %>%
  filter(included) %>%
  select(
    provider,
    model,
    type,
    api,
    done,
    cost_per_completion,
    total_cost,
    fail_rate,
    time_per_completion_s,
    total_time_min,
    comment
  )

write_output_csv(included_models_summary, "models_summary.csv")

excluded_models <- models %>% filter(!included) %>% select(provider, model, reason)

cat("\n", nrow(excluded_models), "out of", nrow(models), "were excluded from the analysis for the follo
```

14 out of 58 were excluded from the analysis for the following reasons.

```r
kable(excluded_models %>% arrange(provider, model), caption = "Excluded models and reasons")
```

Table 2: Excluded models and reasons

provider	model	reason
anthropic	claude-3-sonnet-20240229	not available in Anthropic API anymore
deepseek	deepseek-v2	high fail rate (85%)
deepseek	deepseek-v2.5	too big to run locally; not available through APIs
meta	llama2:13b	does not respond to prompts correctly
meta	llama2:70b	does not respond to prompts correctly
meta	llama3.2	3% success rate on auscj
microsoft	phi	does not respond to prompts correctly
microsoft	phi2	same model as phi
microsoft	phi3	does not respond to prompts correctly
microsoft	phi3.5	10% success rate for biobanking_wa
mistralai	open-mistral-7b	11% success rate for auscj, uppsala_speaks, and biobanking_wa
mistralai	open-mixtral-8x7b	6% success rate on fremantle only
openai	o1-mini	0% success rate on uppsala_speaks only; responds with "I'm sorry, but I can't help with that."
qwen	qwen1.5-110b-chat	has API limit of 10 RPM; too slow

Calculate Cronbach's Alpha

We calculate Cronbach's Alpha from the top 30

```r
# Initialize an empty list to store the alpha results
alpha_results <- list()

models_with_data <- llm_data %>%
  distinct(provider, model)

if (output.file.exists(ALPHA_RESULTS_FILE)) {
  last_alpha_results <- read_output_csv(ALPHA_RESULTS_FILE)
}

# Iterate over each unique provider/model combination
for (row in 1:nrow(models_with_data)) {
  provider <- models_with_data[row, ]$provider
  model <- models_with_data[row, ]$model

  # filter the data for the current provider/model
  provider_model_data <- llm_data %>%
    filter(model == !!model)

  # iterate over each survey
  for (survey_name in unique(provider_model_data$survey)) {

    # filter the data for the current survey
    survey_data <- provider_model_data %>%
      filter(survey == !!survey_name) %>%

      # get only first x iterations, where x = MAX_ITERATIONS
      arrange(created_at) %>%
      head(MAX_ITERATIONS)

    # SKIP THIS ITERATION IF PREVIOUS RECORD EXISTS
    # get date of the most data generation
    last_updated <- max(survey_data$created_at, na.rm = TRUE)

    # if there is a previous record
    if (exists("last_alpha_results")) {
      last_alpha <- last_alpha_results %>% filter(model == !!model, survey == !!survey_name)

      ## and last record is still valid, save it and skip
      ## valid records are more recent than the latest data generated
      if (nrow(last_alpha) == 1 && (last_alpha$created_at > last_updated)) {
        alpha_results[[length(alpha_results) + 1]] <- last_alpha
        next
      } else {
        cat("updating:", model, "/", survey_name, "\n")
      }
    }

    # Calculate Cronbach's Alpha for considerations (C1..C50)
    considerations_data <- survey_data %>% select(C1:C50)

    if (nrow(considerations_data) > 1) {

      # Check if policies are all equal (no variance)
      # this can happen when there are few iterations
      c_all_equal <- all(apply(considerations_data, 1, function(row)
        all(row == considerations_data[1, ], na.rm = TRUE)), na.rm = TRUE)
      # TODO: FIXME!
      # NOTE: assign alpha = 1, which should NOT exist!
      # if (c_all_equal) {
      #   alpha_considerations <- 1
      # } else {
        alpha_considerations <- alpha(
          considerations_data,
          check.keys = TRUE,
          warnings = FALSE,
        )$total$raw_alpha
      # }
    } else {
      alpha_considerations <- NA
    }

    # Calculate Cronbach's Alpha for policies (P1..P10)
    policies_data <- survey_data %>% select(P1:P10)

    if (nrow(policies_data) > 1) {

      # Check if policies are all equal (no variance)
      # this can happen when there are few iterations
      p_all_equal <- all(apply(policies_data, 1, function(row)
        all(row == policies_data[1, ], na.rm = TRUE)), na.rm = TRUE)

      # NOTE: assign alpha = 1, which should NOT exist!
      if (p_all_equal) {
        alpha_policies <- 1
      }

      # normal case, calculate alpha
      else {
        alpha_policies <- alpha(
          policies_data,
          check.keys = TRUE,
          warnings = FALSE,
        )$total$raw_alpha
      }
    } else {
      alpha_policies <- NA
    }

    if (nrow(policies_data) > 1 && nrow(considerations_data) > 1) {
      all_data <- cbind(considerations_data, policies_data)
      alpha_all <- alpha(
        all_data,
        check.keys = TRUE,
        warnings = FALSE,
      )$total$raw_alpha
    } else {
      alpha_all <- NA
    }
```

```r
    # Store the results in the list
    alpha_results[[length(alpha_results) + 1]] <- tibble(
      provider = provider,
      model = model,
      survey = survey_name,
      N = nrow(considerations_data),
      created_at = now_utc(),
      alpha_considerations = alpha_considerations,
      alpha_policies = alpha_policies,
      alpha_all = alpha_all
    )
  }
}

# Combine all results into a single data frame
alpha_results <- bind_rows(alpha_results)

rm(models_with_data)
rm(considerations_data)
rm(survey_data)
rm(policies_data)
rm(provider_model_data)

# write summary to file
write_csv(alpha_results, paste(OUTPUT_DIR, "alpha_results.csv", sep = "/"))
```

Check alpha results per model

```r
# Aggregate alpha_results by model and calculate summary statistics
alpha_summary <- alpha_results %>%
  group_by(provider, model) %>%
  summarise(
    N = sum(N),
    mean_alpha_all = mean(alpha_all, na.rm = TRUE),
    min_alpha_considerations = min(alpha_considerations, na.rm = TRUE),
    max_alpha_considerations = max(alpha_considerations, na.rm = TRUE),
    mean_alpha_considerations = mean(alpha_considerations, na.rm = TRUE),
    std_alpha_considerations = sd(alpha_considerations, na.rm = TRUE),
    min_alpha_policies = min(alpha_policies, na.rm = TRUE),
    max_alpha_policies = max(alpha_policies, na.rm = TRUE),
    mean_alpha_policies = mean(alpha_policies, na.rm = TRUE),
    std_alpha_policies = sd(alpha_policies, na.rm = TRUE),

  )
```

```
## Warning: There were 4 warnings in `summarise()`.
## The first warning was:
## i In argument: `min_alpha_considerations = min(alpha_considerations, na.rm =
##   TRUE)`.
## i In group 45: `provider = "qwen"` `model = "qwen1.5-110b-chat"`.
## Caused by warning in `min()`:
## ! no non-missing arguments to min; returning Inf
## i Run `dplyr::last_dplyr_warnings()` to see the 3 remaining warnings.
## `summarise()` has grouped output by 'provider'. You can override using the
## `.groups` argument.
```

Define aggregation functions

```r
# function to calculate mode of data, same as stat_function
calc_mode <- function(data) {
  as.numeric(names(sort(table(data), decreasing = TRUE)[1]))
}

# function to bootstrap mode
bootstrap_mode <- function(data, n_bootstrap = 1000) {

  # return NA if data contains any NA
  if (any(is.na(data))) {
    return(NA)
  }

  # define the statistic function for bootstrapping to find mode
  stat_function <- function(data, indices) {
    as.numeric(names(sort(table(data[indices]), decreasing = TRUE)[1]))
  }

  # perform bootstrap
  results <- boot(data = data,
                  statistic = stat_function,
                  R = n_bootstrap)

  # calculate bootstrapped mode
  b_mode <- calc_mode(results$t)

  # return the bootstrapped modes
  return(b_mode)
}


aggregate_llm_considerations <- function(considerations) {

  # ensure there are at least 2 rows to aggregate
  if (nrow(considerations) < 2) {
    return(considerations)
  }

  # Calculate the mode for each column
  mode_considerations <- considerations %>%
    summarise(across(everything(), bootstrap_mode))

  return(mode_considerations)

}

aggregate_llm_policies <- function(policies) {

  # ensure there are at least 2 rows to aggregate
  if (nrow(policies) < 2) {
    return(policies)
  }

  # Remove columns with NAs
  valid_policies <- policies[, colSums(is.na(policies)) != nrow(policies)]

  # Convert the policies to a ranked matrix
  ranked_matrix <- as.matrix(valid_policies)

  # Define the number of winners to all - 1 policies
  # stv complains if winners == all policies
  num_winners <- ncol(valid_policies) - 1

  # Run the Single Transferable Vote algorithm
  results <- stv(ranked_matrix, num_winners, quiet = TRUE)

  # add last policy to ranked result
  last_policy <- setdiff(colnames(valid_policies), results$elected)
  ranked_policies <- c(results$elected, last_policy)

  policy_order <- colnames(valid_policies)

  order <- match(policy_order, ranked_policies)

  # calculate the number of missing values needed to reach length 10
  missing_columns <- ncol(policies) - length(order)

  # fill in the missing values with NA
  order <- c(order, rep(NA, missing_columns))

  # create a new data frame with aggregated results
  policy_ranks <- data.frame(t(order))
  colnames(policy_ranks) <- colnames(policies)

  return(policy_ranks)
}
```

Aggregate considerations and preferences

```r
aggregate_llm_data <- function(data) {

  # get last aggregation results, if it exists
  if (output.file.exists(AGGREGATION_RESULTS_FILE)) {
    last_aggr_results <- read_output_csv(AGGREGATION_RESULTS_FILE)
  }

  # initialize an empty list to store the alpha results
  aggr_results <- list()

  # iterate over each unique provider/model/survey combination
  for (row in 1:nrow(llm_surveys)) {
    provider <- llm_surveys[row, ]$provider
    model <- llm_surveys[row, ]$model
    survey <- llm_surveys[row, ]$survey
    N <- llm_surveys[row, ]$N

    # filter the data for the current survey
    survey_data <- data %>%
      filter(model == !!model, survey == !!survey) %>%

      # get only first x iterations, where x = MAX_ITERATIONS
      arrange(created_at) %>%
      head(MAX_ITERATIONS)

    # SKIP THIS ITERATION IF PREVIOUS RECORD EXISTS
    # get date of the most data generation
    last_updated <- max(survey_data$created_at, na.rm = TRUE)

    # if there is a previous record
    if (exists("last_aggr_results")) {
      last_aggr <- last_aggr_results %>% filter(model == !!model, survey == !!survey)

      ## and last record is still valid, save it and skip
      ## valid records are more recent than the latest data generated
      if (nrow(last_aggr) == 1 &&
          (last_aggr$created_at > last_updated)) {
        aggr_results[[length(aggr_results) + 1]] <- last_aggr
        next
      } else {
        cat("updating:", model, "/", survey, "\n")
      }
    }

    # aggregate considerations C1:C50
    considerations_data <- survey_data %>% select(C1:C50)
    aggregated_considerations <- aggregate_llm_considerations(considerations_data)

    # aggregate policies P1:P10
    policies_data <- survey_data %>% select(P1:P10)
    aggregated_policies <- aggregate_llm_policies(policies_data)

    # store the results in the list
    aggr_result <- tibble(
      provider = provider,
      model = model,
      survey = survey,
      N = N,
      created_at = now_utc(),
      aggregated_considerations,
      aggregated_policies,
    )

    aggr_results[[length(aggr_results) + 1]] <- aggr_result

  }
```

```
  # Combine all results into a single data frame
  aggr_results <- bind_rows(aggr_results)

  return(aggr_results)

}

time_start <- Sys.time()
llm_data_aggregated <- aggregate_llm_data(llm_data)
time_end <- Sys.time()
elapsed_time <- difftime(time_end, time_start, units = "auto")

print(
  paste(
    "Aggregation of",
    nrow(llm_data),
    "LLM responses across",
    length(unique(llm_data$survey)) ,
    "surveys completed in",
    round(as.numeric(elapsed_time), 2),
    units(elapsed_time)
  )
)
```

```
## [1] "Aggregation of 29431 LLM responses across 20 surveys completed in 2.06 secs"
```

```
# write summary to file
write_output_csv(llm_data_aggregated, AGGREGATION_RESULTS_FILE)
```

It takes 2.06 secs to run the aggregation script.

Read and format human data

```
# Import the CSV file into a data frame
human_data <- read_csv(HUMAN_DATA_FILE, show_col_types = FALSE)

# Rename columns to be consistent with LLM data
human_data <- human_data %>%
  rename_with( ~ sub("^U0|^U", "C", .), starts_with("U", ignore.case = FALSE)) %>%
  rename_with( ~ sub("^Pref", "P", .), starts_with("Pref", ignore.case = FALSE)) %>%
  filter(Study != "Sydney CC Adaptation")

# Read the mapping file
study_survey_map <- read_csv("data/study_survey_map.csv", show_col_types = FALSE)

# Add a new column 'Survey' to human_data by matching 'Study' with 'survey'
human_data <- human_data %>%
  left_join(study_survey_map, by = c("Study" = "study")) %>%
  relocate(survey, .after = "Study")
```

Generate random participants

```r
get_random_data <- function(sheet_names, file_path=SURVEY_FILE) {

  rand <- list()

  # Iterate over each sheet in the workbook
  for (sheet_name in sheet_names) {
    #cat("Processing sheet:", sheet_name, "\n")

    # Read the current sheet into a data frame
    df <- read_excel(file_path, sheet = sheet_name)

    # Check if required columns exist
    required_columns <- c("considerations", "policies", "scale_max", "q-method")
    missing_cols <- setdiff(required_columns, colnames(df))
    if (length(missing_cols) > 0) {
      cat(
        "Sheet",
        sheet_name,
        "is missing the following columns:",
        paste(missing_cols, collapse = ", "),
        "\n\n"
      )
      next
    }

    # Calculate the number of non-NA rows in "considerations" column
    n_c <- sum(!is.na(df$considerations))

    # Calculate the number of non-NA rows in "policies" column
    n_p <- sum(!is.na(df$policies))

    # Extract integer values from "scale_max" column, assuming they are already integers
    scale_max <- as.integer(na.omit(df$scale_max))

    # Extract logical (boolean) values from "q-method" column
    q_method <- as.logical(na.omit(df$`q-method`))

    # Print the results for each sheet
    #cat("Sheet:", sheet_name, "\n")
    #cat("Number of considerations:", n_c, "\n")
    #cat("Number of policies:", n_p, "\n")
    #cat("Integer value from 'scale_max' column:", scale_max, "\n")
    #cat("Logical value from 'q-method' column:", q_method, "\n")


    ### Make random survey
    # Generate data for C1:C50 with Likert scale values [1, scale_max] randomly assigned
    if (q_method) {

      # get normally distributed data
      c_df <- data.frame(t(round(rnorm(n = n_c, mean = scale_max / 2, sd = scale_max / 4))))

      # replace values below lower and above upper bound
      c_df[c_df < 1] <- 1
      c_df[c_df > scale_max] <- scale_max

    } else {
      c_df <- data.frame(t(replicate(n_c, sample(1:scale_max, 1, replace = TRUE))))
    }

    # fill and rename columns
    c_df[1, (n_c + 1):50] <- NA
    colnames(c_df) <- paste0("C", 1:50)

    # Generate data for P1:P10 with random unique ranks 1 to n_p for each row
    p_df <- data.frame(t(apply(matrix(0, nrow = 1, ncol = n_p), 1, function(x)
      sample(1:n_p))))
    p_df[1, (n_p + 1):10] <- NA
    colnames(p_df) <- paste0("P", 1:10)

    # Validate data
    c_valid <- !(any(c_df < 1, na.rm = TRUE) || any(c_df > scale_max, na.rm = TRUE))
    p_valid <- !(any(duplicated(p_df[!is.na(p_df)])) ||
      any(p_df < 1, na.rm = TRUE) ||
      any(p_df > n_p, na.rm = TRUE))

    #cat("Valid considerations:", c_valid, "\n")
    #cat("Valid policies:", p_valid, "\n")

    if (!c_valid || !p_valid) {
      warn(paste("Random generation produced invalid data for", sheet_name))
    }

    # Combine the two datasets into one data frame
    dataset <- as.data.frame(cbind(c_df, p_df))
    dataset <- dataset %>%
      mutate(survey = sheet_name) %>%
      relocate(survey, .before = 1)

    rand[[length(rand) + 1]] <- dataset

    #cat(rep("-", 40), "\n")


  }

  rand <- bind_rows(rand)
  return(rand)

}

# Example usage
rand_data <- get_random_data(survey_names)
```

DRI Analysis

```r
dri_calc <- function(data, v1, v2) {
  lambda <- 1 - (sqrt(2) / 2)
  dri <- 2 * (((1 - mean(abs((data[[v1]] - data[[v2]]) / sqrt(2)
  ))) - (lambda)) / (1 - (lambda))) - 1

  return(dri)
}

dri_calc_v2 <- function(data, v1, v2) {
  # Calculate orthogonal distance for each pair
  d <- abs((data[[v1]] - data[[v2]]) / sqrt(2))

  # Define lambda as in the original
  lambda <- 1 - (sqrt(2) / 2)  # ??? 0.293

  # Calculate penalty: 0.5 if both correlations are in [0, 0.2], 1 otherwise
  penalty <- ifelse(data[[v1]] >= 0 & data[[v1]] <= 0.2 & #0.3
                      data[[v2]] >= 0 & data[[v2]] <= 0.2, # 0.3
                    0, 1)#0.4

  # Adjusted consistency per pair
  consistency <- (1 - d) * penalty

  # Average consistency across all pairs
  avg_consistency <- mean(consistency)

  # Scale to [-1, 1] as in the original
  dri <- 2 * ((avg_consistency - lambda) / (1 - lambda)) - 1

  return(dri)
}

get_IC <- function(data, survey, case) {

  # loop through analysis stages (pre/post)
  for (stage in 1:max(data$StageID)) {

    # select specific data to analyse
    data_stage <- data %>% filter(StageID == stage)

    # make sure there's data to analyze
    if (nrow(data_stage) > 0) {
      # get participant numbers/ids
      PNums <- data_stage$PNum

      # variables for reading COLUMN data
      # Q is a list considerations (Likert scale)
      # - there are up to 50 questions
      # R is a list ratings (rankings)
      Q <- data_stage %>% select(C1:C50)
      R <- data_stage %>% select(P1:P10)
      # remove all NA columns (in case there are less than 50
      # consideration questions
      Q <- Q[, colSums(is.na(Q)) != nrow(Q)]
      R <- R[, colSums(is.na(R)) != nrow(R)]

      # transpose data
      Q <- t(Q)
      R <- t(R)

      # format data as data frame
      Q <- as.data.frame(Q)
      R <- as.data.frame(R)

      # name columns with participant numbers
      colnames(Q) <- PNums
      colnames(R) <- PNums

      # obtain a list of correlations without duplicates
      # cor() returns a correlation matrix between Var1 and Var2
      # Var1 and Var2 are the variables being correlated
      # Freq is the correlation
      QWrite <- subset(as.data.frame(as.table(cor(Q, method = "spearman"))),
                       match(Var1, names(Q)) > match(Var2, names(Q)))

      RWrite <- subset(as.data.frame(as.table(cor(R, method = "spearman"))),
                       match(Var1, names(R)) > match(Var2, names(R)))

      # initialize the output in the first iteration
      if (stage == 1) {
        IC <- data.frame("P_P" = paste0(QWrite$Var1, '-', QWrite$Var2))
        IC$P1 <- as.numeric(as.character(QWrite$Var1))
        IC$P2 <- as.numeric(as.character(QWrite$Var2))
      }

      # prepare QWrite
      QWrite <- as.data.frame(QWrite$Freq)
      names(QWrite) <- paste0("Q", stage)

      # prepare RWrite for merge
      RWrite <- as.data.frame(RWrite$Freq)
      names(RWrite) <- paste0('R', stage)

      # merge
      IC <- cbind(IC, QWrite, RWrite)
    }

  }

  # append case & study info
  IC$survey <- survey
  IC$case <- case

  ## IC Points calculations ##
```

```r
  IC$IC_PRE <- 1 - abs((IC$R1 - IC$Q1) / sqrt(2))
  IC$IC_POST <- 1 - abs((IC$R2 - IC$Q2) / sqrt(2))

  return(IC)
}

get_ind_DRI <- function(IC) {

  Plist <- unique(c(IC$P1, IC$P2))

  Plist <- Plist[order(Plist)]

  DRIInd <- data.frame('participant' = Plist)
  DRIInd$survey <- survey
  DRIInd$case <- data_case_study$Case[1]

  DRIInd <- DRIInd[c("survey", "case", "participant")]

  #Add individual-level metrics
  for (i in 1:length(Plist)) {
    DRIInd$DRIPre[i] <- dri_calc(
      data = IC  %>% filter(P1 == Plist[i] | P2 == Plist[i]),
      v1 = 'R1',
      v2 = 'Q1'
    )
    DRIInd$DRIPost[i] <- dri_calc(
      data = IC  %>% filter(P1 == Plist[i] | P2 == Plist[i]),
      v1 = 'R2',
      v2 = 'Q2'
    )

    # calculate updated DRI
    DRIInd$DRIPreV2[i] <- dri_calc_v2(
      data = IC  %>% filter(P1 == Plist[i] | P2 == Plist[i]),
      v1 = 'R1',
      v2 = 'Q1'
    )
    DRIInd$DRIPostV2[i] <- dri_calc_v2(
      data = IC  %>% filter(P1 == Plist[i] | P2 == Plist[i]),
      v1 = 'R2',
      v2 = 'Q2'
    )

  }

  return(DRIInd)

}

get_case_DRI <- function(IC, type="human_only") {

  ## Group DRI level ##
  DRI_PRE <- dri_calc(data = IC, v1 = 'R1', v2 = 'Q1')
  DRI_POST <- dri_calc(data = IC, v1 = 'R2', v2 = 'Q2')

  ## Group DRI level V2 ##
  DRI_PRE_V2 <- dri_calc_v2(data = IC, v1 = 'R1', v2 = 'Q1')
  DRI_POST_V2 <- dri_calc_v2(data = IC, v1 = 'R2', v2 = 'Q2')

  #CaseDRI Dataframe
  DRI.Case <- data.frame(
    survey = survey,
    case = case,
    type = type,
    DRI_PRE,
    DRI_PRE_V2,
    DRI_POST,
    DRI_POST_V2
  )

  #Tests for groups
  DRIOverallSig <- wilcox.test(IC$IC_POST,
                               IC$IC_PRE,
                               paired = TRUE,
                               alternative = "greater")
  DRIOverallSig_twoside <- wilcox.test(IC$IC_POST,
                                       IC$IC_PRE,
                                       paired = TRUE,
                                       alternative = "two.side")

  #Adding the results to case data
  DRI.Case$DRI_one_tailed_p <- DRIOverallSig$p.value
  DRI.Case$DRI_twoside_p <- DRIOverallSig_twoside$p.value

  return(DRI.Case)

}

mini_publics <- human_data %>%
  group_by(survey, Case) %>%
  summarise(.groups = "drop")

# get_llm_data <- function(provider, model, survey) {
#   llm_participant <- llm_data_aggregated %>%
#     filter(provider == !!provider, model == !!model, survey == !!survey)
#   return(llm_participant)
# }

get_ind_LLM_DRI <- function(data, provider, model) {

  llm_DRI <- data %>%
    filter(participant == 0) %>%
    select(-participant) %>%
    mutate(provider = !!provider, model = !!model) %>%
    relocate(provider, model, .before = 1)
  return(llm_DRI)
}

add_llm_participant <- function(data, provider, model, survey) {

  # print(paste("adding", paste(provider, model, survey, sep = "/"), "to human data."))

  # get llm data
  llm_participant <- llm_data_aggregated %>%
    filter(provider == !!provider, model == !!model, survey == !!survey)

  # check if it exists
  if (nrow(llm_participant) == 0) {
    warn(paste("No human participant found for", paste(provider, model, survey, sep = "/")))
  }

  # create 2 participants, PRE and POST
  llm_participants <- bind_rows(llm_participant, llm_participant)
  llm_participants$PNum <- 0 # PNum = 0 is LLM
    llm_participants$StageID <- c(1,2)

  data_with_llm <- bind_rows(data, llm_participants)

  return(data_with_llm)

}


DRIInd.LLMs <- list()

# for each study [1:N], N = 26
for (case_study in 1:nrow(mini_publics)) {

  # select study data
  survey <- mini_publics[case_study, ]$survey
  case <- mini_publics[case_study, ]$Case

  # get human data for this case study
  data_case_study <- human_data %>% filter(survey == !!survey &
                                             Case == !!case)

  # intersubject correlations (IC)
  IC <- get_IC(data_case_study, survey, case)

  ## GROUP DRI ##
  DRI.Case <- get_case_DRI(IC)

  ## INDIVIDUAL DRI ##
  DRIInd <- get_ind_DRI(IC)

  # get human average
  # NOTE: this should be the same as human_only group DRI
  human_ind_DRI_mean <- tibble(
    DRIPre = mean(DRIInd$DRIPre),
    DRIPost = mean(DRIInd$DRIPost)
  )

  human_ind_DRI_meanV2 <- tibble(
    DRIPreV2 = mean(DRIInd$DRIPreV2),
    DRIPostV2 = mean(DRIInd$DRIPostV2)
  )

  # Global dataframes for depositing results
  # initialize *.Global
  if (case_study == 1) {
    IC.Global <- IC
    DRIInd.Global <- DRIInd
    DRI.Global <- DRI.Case
  }

  # append to *.Global
  else {
    IC.Global <- rbind(IC.Global, IC)
    DRIInd.Global <- rbind(DRIInd.Global, DRIInd)
    DRI.Global <- rbind(DRI.Global, DRI.Case)
  }

  # check if there are LLM data for this survey
  llms <- llm_surveys %>% filter(survey == !!survey)
  if (nrow(llms) == 0) {
    next
  }

  # iterate through each llm
  for (llm in 1:nrow(llms)) {

    provider <- llms[llm,]$provider
    model <- llms[llm,]$model
    type <- paste0("human+",paste(provider, model, sep = "/"))

    data_with_llm <- add_llm_participant(data_case_study, provider, model, survey)

    IC.LLM <- get_IC(data_with_llm, survey, case)
    DRI.Case.LLM <- get_case_DRI(IC.LLM, type)
    DRIInd.LLM <- get_ind_DRI(IC.LLM)
    DRIInd.LLM.Model <- get_ind_LLM_DRI(DRIInd.LLM, provider, model)

    DRIInd.LLM.Model$human_only_DRIPre_mean <- human_ind_DRI_mean$DRIPre
    DRIInd.LLM.Model$human_only_DRIPost_mean <- human_ind_DRI_mean$DRIPost

    ## DRI V2
    DRIInd.LLM.Model$human_only_DRIPre_meanV2 <- human_ind_DRI_meanV2$DRIPreV2
    DRIInd.LLM.Model$human_only_DRIPost_meanV2 <- human_ind_DRI_meanV2$DRIPostV2

    get_bm_index <- function(diff) {
      bm_index <- (diff + 2) / 4
```

```
      return(bm_index)
    }

    DRIInd.LLM.Model <- DRIInd.LLM.Model %>%
      mutate(DRIPre_diff = DRIPre - human_only_DRIPre_mean,
             DRIPost_diff = DRIPost - human_only_DRIPost_mean) %>%

      # benchmark index = use DRIPost & normalize it to be >= 0
      mutate(bm_index = get_bm_index(DRIPost_diff)) %>%

          mutate(DRIPre_diffV2 = DRIPreV2 - human_only_DRIPre_meanV2,
             DRIPost_diffV2 = DRIPostV2 - human_only_DRIPost_meanV2) %>%

      # benchmark index = use DRIPost & normalize it to be >= 0
      mutate(bm_indexV2 = get_bm_index(DRIPost_diffV2))


    DRIInd.LLMs[[length(DRIInd.LLMs) + 1]] <- DRIInd.LLM.Model

    DRI.Global <- rbind(DRI.Global, DRI.Case.LLM)

  }

} # end for each case study
```

```
## Warning in cor(Q, method = "spearman"): the standard deviation is zero
## Warning in cor(Q, method = "spearman"): the standard deviation is zero
## Warning in cor(Q, method = "spearman"): the standard deviation is zero
## Warning in cor(Q, method = "spearman"): the standard deviation is zero
```

```
DRIInd.LLMs <- bind_rows(DRIInd.LLMs)

missing <-setdiff(unique(llm_data$survey), unique(DRIInd.LLMs$survey))

if (length(missing) > 0) {
  warn(paste("Missing", missing, "from DRIInd.LLMs!"))
}
```

```
## Warning: Missing swiss_health from DRIInd.LLMs!
```

```
# add delta column
DRI.Global <- DRI.Global %>%
  mutate(DRI_DELTA = DRI_POST - DRI_PRE)

# write summary to file
write_csv(DRIInd.LLMs, paste(OUTPUT_DIR, "DRIInd_LLMs.csv", sep = "/"))
write_csv(DRI.Global, paste(OUTPUT_DIR, "DRI_global.csv", sep = "/"))
```

DRI Benchmark

```
DRI_benchmark <- DRIInd.LLMs %>%
  group_by(provider, model) %>%
  summarise(
    N = n(),
    .groups = "drop",
    agg_bm_index = mean(bm_indexV2, na.rm = TRUE)
  ) %>%
  filter(N == max(N)) %>% # only include models with all surveys
  arrange(desc(agg_bm_index))


DRI_benchmark %>%
  mutate(label = paste(provider, model, sep = "/")) %>%
  ggplot(aes(
    x = reorder(label, agg_bm_index),
    y = agg_bm_index,
    fill = provider
  )) +
  geom_bar(stat = "identity") +
  coord_flip() +
  geom_text(aes(label = round(agg_bm_index, 3)), hjust = -0.3, size = 3) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.background = element_rect(fill = "white"),
    legend.position = "none"
  ) +
  scale_y_continuous(limits = c(0, 1)) +
  labs(x = "", y = "DRI benchmark") -> plot

ggsave(
  paste(OUTPUT_DIR, "benchmarkV1.png", sep = "/"),
  plot,
  width = 10,
  height = 6
)

DRIInd.LLMs %>%
  filter(model %in% DRI_benchmark$model) %>% # only include models with all surveys
  mutate(label = paste(provider, model, sep = "/")) %>%
  ggplot(aes(
    x = reorder(label, bm_indexV2, FUN = median, na.rm = TRUE),
    y = bm_indexV2,
    color = provider
  )) +
  geom_boxplot() +
  coord_flip() +
  theme_minimal() +
  theme(plot.background = element_rect(fill = "white"),
        legend.position = "none") + scale_y_continuous(limits = c(-0.1, 1)) +
  labs(x = "", y = "DRI benchmark") -> plot

ggsave(
  paste(OUTPUT_DIR, "benchmarkV2.png", sep = "/"),
  plot,
  width = 10,
  height = 6
```

```
)
```

```
## Warning: Removed 17 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Trying a 2-dimension benchmark
DRI_benchmark <- full_join(alpha_results, DRIInd.LLMs, by = c("provider", "model", "survey")) %>%
  filter(model %in% DRI_benchmark$model) # only include models with all surveys


DRI_benchmark %>%
  #  filter(provider == "google") %>%
  #mutate(label = paste(provider, model, sep = "/")) %>%
  group_by(provider, model, .groups = "drop") %>%
  summarise(
    mean_alpha = mean(alpha_all, na.rm = TRUE),
    mean_bm_index = mean(bm_indexV2, na.rm = TRUE),
    se_alpha = sd(alpha_all, na.rm = TRUE),
    se_bm_index = sd(bm_indexV2, na.rm = TRUE)

  ) %>%
  ggplot(aes(x = mean_alpha, y = mean_bm_index, color = provider)) +
  geom_point(size = 5) + # Adjust size as needed
  #geom_errorbar(aes(ymin = mean_bm_index - se_bm_index, ymax = mean_bm_index + se_bm_index), width = 0
  #geom_errorbarh(aes(xmin = mean_alpha - se_alpha, xmax = mean_alpha + se_alpha), height = 0.02) + # E
  geom_text(aes(label = model), vjust = -2, size = 2) + # Add labels above each dot
  #scale_x_continuous(limits = c(0, 1)) +
  #scale_y_continuous(limits = c(0, 1)) +
  labs(x = "Cronbach's alpha (mean)", y = "DRI Benchmark Index (mean)", color = "Provider") +
  theme_minimal() +
  theme(plot.background = element_rect(fill = "white"),
        legend.position = "none") -> plot # Remove text from color legend
```

```
## `summarise()` has grouped output by 'provider', 'model'. You can override using
## the `.groups` argument.
```

```
# Set the theme to minimal


ggsave(
  paste(OUTPUT_DIR, "benchmarkV3.png", sep = "/"),
  plot,
  width = 10,
  height = 6
)
```