

Triage Against the Machine: Can AI Reason Deliberatively?

Francesco Veri, Gustavo Umbelino

2025-03-24

Define 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"))  
}
```

Get available LLMs

```
# read the CSV file into a data frame and remove duplicates  
models <- read_csv(LLMS_FILE, show_col_types = FALSE) %>%  
  distinct(provider, model)  
  
# 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  
  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
2. anthropic/claude-3-5-sonnet-20241022
3. anthropic/claude-3-haiku-20240307
4. anthropic/claude-3-opus-20240229
5. anthropic/claude-3-sonnet-20240229
6. cohere/command
7. cohere/command-r
8. cohere/command-r-plus
9. cohere/command-r7b-12-2024
10. deepseek/deepseek-chat | has data
11. deepseek/deepseek-reasoner | has data
12. deepseek/deepseek-v2
13. deepseek/deepseek-v2.5
14. google/gemini-1.5-flash | has data
15. google/gemini-1.5-flash-8b | has data
16. google/gemini-1.5-pro | has data
17. google/gemini-2.0-flash | has data
18. google/gemma
19. google/gemma2 | has data
20. google/gemma3
21. meta/llama2
22. meta/llama3
23. meta/llama3.1
24. meta/llama3.2 | has data
25. meta/llama3.3
26. microsoft/phi
27. microsoft/phi2
28. microsoft/phi3
29. microsoft/phi3.5
30. microsoft/phi4 | has data
31. mistralai/ministral-3b-latest
32. mistralai/ministral-8b-latest | has data
33. mistralai/mistral-large-latest | has data
34. mistralai/mistral-small-latest | has data
35. mistralai/open-mistral-7b
36. mistralai/open-mistral-nemo | has data
37. mistralai/open-mixtral-8x22b | has data
38. mistralai/open-mixtral-8x7b
39. openai/gpt-3.5-turbo | has data
40. openai/gpt-4
41. openai/gpt-4-turbo
42. openai/gpt-4o | has data
43. openai/gpt-4o-mini | has data
44. openai/o1
45. openai/o1-mini | has data
46. openai/o3-mini
47. qwen/qwen-max
48. qwen/qwen-plus
49. qwen/qwen-turbo
50. qwen/qwen1.5
51. qwen/qwen2
52. qwen/qwen2.5
53. qwen/qwq-plus

There are 18 models with data out of 53 (34%).

Get available surveys

```
# 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"])

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

# print ordered survey names
for (i in 1:length(survey_names)){
  cat(paste0(i, ". ", survey_names[i], "\n"))
}
```

```
## 1. acp
## 2. auscj
## 3. bep
## 4. biobanking_mayo_ubc
## 5. biobanking_wa
## 6. ccps
## 7. ds_aargau
## 8. ds_bellinzona
## 9. energy_futures
## 10. fnqcj
## 11. forestera
## 12. fremantle
## 13. gbr
## 14. swiss_health
## 15. uppsala_speaks
## 16. valsamoggia
## 17. zh_thalwil
## 18. zh_uster
## 19. zh_winterthur
## 20. zukunft
```

Read and format LLM data

```
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]

# 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'
  )

# calculate costs in tokens
cost_tokens <- llm_data %>%
  group_by(provider, model) %>%
  summarise(
    N = n(),
    input_tokens = as.integer(sum(input_tokens)),
    output_tokens = as.integer(sum(output_tokens)),

```

```

    .groups = 'drop'
  )

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

```

Calculate Cronbach's Alpha

```

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

# Iterate over each unique provider/model combination
for (provider_model in unique(paste(llm_data$provider, llm_data$model, sep = "/"))) {

  # filter the data for the current provider/model
  provider_model_data <- llm_data %>%
    filter(paste(provider, model, sep = "/") == provider_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)

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

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

    # Store the results in the list
    alpha_results[[length(alpha_results) + 1]] <- tibble(
      provider_model = provider_model,
      survey = survey_name,

```

```

    N = nrow(considerations_data),
    alpha_considerations = alpha_considerations,
    alpha_policies = alpha_policies
  )
}
}

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

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

```

# Aggregate alpha_results by model and calculate summary statistics
alpha_summary <- alpha_results %>%
  group_by(provider_model) %>%
  summarise(
    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)
  )

```

Define aggregation functions

```

# 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 columns to aggregate
  if (ncol(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 columns 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$selected)
  ranked_policies <- c(results$selected, 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

```

aggregate_llm_data <- function(data) {

  # initialize an empty list to store the alpha results
  aggregation_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(provider == !!provider, model == !!model, survey == !!survey)

    # 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
    aggregation_result <- tibble(
      provider = provider,
      model = model,
      survey = survey,
      N = N,
      aggregated_considerations,
      aggregated_policies,
    )

    aggregation_results[[length(aggregation_results) + 1]] <- aggregation_result
  }
}

```

```

}

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

return(aggregation_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)) ,

## [1] "Aggregation of 4385 LLM responses across 20 surveys completed in 11.46 mins"
print(head(llm_data_aggregated))

## # A tibble: 6 x 64
##   provider model    survey      N    C1    C2    C3    C4    C5    C6    C7    C8
##   <chr>      <chr>    <chr>  <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 deepseek deepsee~ acp      10     6     7     7     7     7     7     8     9
## 2 deepseek deepsee~ auscj    10     4     5     4     3     5     3     6     2
## 3 deepseek deepsee~ bep      10     2     3     6     5     2     2     3     2
## 4 deepseek deepsee~ bioba~   10     8     8     5     6    10     6     5     8
## 5 deepseek deepsee~ bioba~   10     8     8     5     7    10     5     4     8
## 6 deepseek deepsee~ ccps     10     2     5     3     9     5     3     5     9
## # i 52 more variables: C9 <dbl>, C10 <dbl>, C11 <dbl>, C12 <dbl>, C13 <dbl>,
## #   C14 <dbl>, C15 <dbl>, C16 <dbl>, C17 <dbl>, C18 <dbl>, C19 <dbl>,
## #   C20 <dbl>, C21 <dbl>, C22 <dbl>, C23 <dbl>, C24 <dbl>, C25 <dbl>,
## #   C26 <dbl>, C27 <dbl>, C28 <dbl>, C29 <dbl>, C30 <dbl>, C31 <dbl>,
## #   C32 <dbl>, C33 <dbl>, C34 <dbl>, C35 <dbl>, C36 <dbl>, C37 <dbl>,
## #   C38 <dbl>, C39 <dbl>, C40 <dbl>, C41 <dbl>, C42 <dbl>, C43 <dbl>,
## #   C44 <dbl>, C45 <dbl>, C46 <dbl>, C47 <dbl>, C48 <dbl>, C49 <dbl>, ...
# write summary to file
write_csv(llm_data_aggregated, paste(OUTPUT_DIR, "llm_data_aggregated.csv", sep = "/"))

```

It takes 11.46 mins to run the aggregation script.

Read and format human data

```

# Import the CSV file into a data frame
human_data <- read_csv("data/total_dataset_clean.csv", 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")

# rename column names for consistency
# colnames(human_data) <- lapply(colnames(human_data), tolower)

human_data

## # A tibble: 1,032 x 70
##   Datacheck StudyID Study  survey CaseID Case  data_type StageID Stage_Analysis
##   <dbl>    <dbl> <chr>  <chr>   <dbl> <chr> <chr>         <dbl> <chr>
## 1         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 2         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 3         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 4         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 5         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 6         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 7         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 8         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 9         1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## 10        1         1 Uppsa~ uppsa~     1 Acti~ Likert         1 Pre-Delib
## # i 1,022 more rows
## # i 61 more variables: PNum <dbl>, C1 <dbl>, C2 <dbl>, C3 <dbl>, C4 <dbl>,
## #   C5 <dbl>, C6 <dbl>, C7 <dbl>, C8 <dbl>, C9 <dbl>, C10 <dbl>, C11 <dbl>,
## #   C12 <dbl>, C13 <dbl>, C14 <dbl>, C15 <dbl>, C16 <dbl>, C17 <dbl>,
## #   C18 <dbl>, C19 <dbl>, C20 <dbl>, C21 <dbl>, C22 <dbl>, C23 <dbl>,
## #   C24 <dbl>, C25 <dbl>, C26 <dbl>, C27 <dbl>, C28 <dbl>, C29 <dbl>,
## #   C30 <dbl>, C31 <dbl>, C32 <dbl>, C33 <dbl>, C34 <dbl>, C35 <dbl>, ...

```

Original DRI analysis

```

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)
}

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 ##
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'
    )
  }

  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')

```

```

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

#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")
#cumdist_pre_post <- cum_test(IC$IC_PRE, IC$IC_POST, nboots = 1000)not necessary

#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)
  )

  # 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
}

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

  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))

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

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

```



```

} # end for each case study

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 <- DRIInd.LLMs %>%
  group_by(provider, model) %>%
  summarise(N = n(), .groups = "drop",
            agg_bm_index = mean(bm_index)) %>%
  arrange(desc(agg_bm_index))

llm_sd <- DRIInd.LLMs %>%
  group_by(survey, case) %>%
  summarise(N = n(), .groups = "drop",
            llm_sd = sd(bm_index)) %>%
  arrange(desc(llm_sd))

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

DRI_benchmark %>%
  mutate(label = paste(provider, model, sep="/")) %>%
  ggplot(aes(x = reorder(label, desc(agg_bm_index)), y = agg_bm_index)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(agg_bm_index, 3)), vjust = -0.3) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        plot.background = element_rect(fill = "white")) + scale_y_continuous(limits = c(0, 1)) +
  labs(x = "", y = "DRI benchmark") -> plot

DRIInd.LLMs %>%
  mutate(label = paste(provider, model, sep="/")) %>%
  ggplot(aes(x = reorder(label, bm_index, FUN = median), y = bm_index)) +
  geom_boxplot() +
  coord_flip() +
  theme_minimal() +
  theme(
    plot.background = element_rect(fill = "white")) + scale_y_continuous(limits = c(-0.1, 1)) +
  labs(x = "", y = "DRI benchmark") -> plot

```

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

## Warning: Removed 5 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
# Read the sheet names of the Excel file
exec_log <- read_csv(EXEC_LOG_FILE, show_col_types = FALSE)

## Warning: One or more parsing issues, call `problems()` on your data frame for details,
## e.g.:
##   dat <- vroom(...)
##   problems(dat)
# TODO
```