Report Code

Jiazhen Tang, Sterre Gesbert, Oliver Hutton, Martha Brenner

2025-10-20

```
# All needed libraries
library(tidyverse)
library(jsonlite)
library(ggplot2)
library(showtext)
library(ggtext)
library(patchwork)
library(lintr)
library(rstudioapi)
```

Loading the data

```
# This likely needs to be adpated to your computer
# Input the file path to the Pressure Cooker data below
file_path <- "~/Documents/Pressure-Cooker-Group-2/sessionInfo"

# Recursively find all JSON files under 'sessionInfo'
json_files <- list.files(path = file_path,
                         pattern = "\\.json$",
                         full.names = TRUE,
                         recursive = TRUE)

results <- list()

# get the values in the data that show whether a students has given a correct
# answer in a session and requested a hint. We also get the data on the question
# type and perhaps the person ID for that arrangement, though we're not sure.
# We did not use the latter two variables in our end results.
for (i in seq_along(json_files)) {
  file_path <- json_files[i]
  try({
    tmp <- fromJSON(txt = file_path)
    items_df <- tmp$elements$items[[1]]
    interaction_row <- items_df[!is.na(items_df$interactionType), ]
    out <- tibble(
      file = basename(file_path),
      correct_answer = tmp$scoring$marksEarned,
      hints_requested = tmp$scoring$penalties$hintsRequested,
      content = interaction_row$interactionType,
      maybe_person_id = interaction_row$id
```

```
    )
    results[[i]] <- out
  }, silent = TRUE)
}

# Combine all extracted rows into a single data frame
data_raw <- bind_rows(results)

#extract all events and timestamps for each session
extract_events_from_file <- function(file_path) {
  tryCatch({
    tmp <- fromJSON(file_path)
    # Extract creation timestamp and convert to POSIX
    creation_ts <- tmp$creationTimestamp
    session_id <- tools::file_path_sans_ext(basename(file_path))
    start_event <- tibble(
      file = session_id,
      event = "START",
      timestamp = as.POSIXct(creation_ts, origin = "1970-01-01", tz = "UTC")
    )
    # Try to get actual events
    events <- tmp$elements$items[[1]]$result$events
    if (!is.null(events) && is.list(events)) {
      real_events <- map_dfr(events, function(e) {
        tibble(
          sessionID = session_id,
          event = e$event,
          timestamp = as.POSIXct(e$timestamp / 1000,
                                 origin = "1970-01-01",
                                 tz = "UTC"),
          progress = e$progress
        )
      })
      # Combine "START" with real events
      bind_rows(start_event, real_events)
    } else {
      # No events, return just the START
      start_event
    }
  }, error = function(e) {
    # On error, return empty tibble
    tibble()
  })
}


# Run the extraction across all files
all_events <- map_dfr(json_files, extract_events_from_file)

# Fill all the sessionID NAs
all_events_filled <- all_events %>%
  arrange(desc(row_number())) %>%
  fill(sessionID, .direction = "down") %>%
```

```r
  arrange(row_number())

# Fill all the file NAs
all_events_filled <- all_events_filled %>%
  arrange(desc(row_number()))  %>%
  fill(file, .direction = "down")

# data_raw had file names ending in .json, while all_events_filled did not
# For joining the two we had to adapt data_raw first
data_raw$file <- sub("\\.json$", "", data_raw$file)

# Create full data table
full_data <- left_join(all_events_filled, data_raw, by = "file")

# The files and sessions didn't have nice names,
# so we created file_ID and session_ID
full_data <- full_data %>%
  mutate(file_id = paste0(dense_rank(file)))

full_data <- full_data %>%
  mutate(session_id = paste0(dense_rank(sessionID)))

# The code below showed that file and sessionID identified the same things.
mean(full_data$session_id == full_data$file_id)

# We therefore removed some unnecessary variables
full_data <- full_data %>%
  select(-sessionID, -file_id)

# Adding row whether hint event was spammed or not
full_data <- full_data %>%
  arrange(session_id, timestamp) %>%  # ensure chronological order
  group_by(session_id) %>%
  mutate(
    time_diff = timestamp - lag(timestamp),
    prev_hint = lag(event) == "HINT",
    spam_hint = case_when(
      event == "HINT" & prev_hint & time_diff <= 3 ~ 1,
      event == "HINT" ~ 0,
      TRUE ~ NA_real_
    )
  )

#include rows where event 1 and event 2 are both hints
consecutive_hints <- full_data %>%
  arrange(session_id, timestamp) %>%
  group_by(session_id) %>%
  mutate(
    prev_event = lag(event),
    time_between_hints = as.numeric(timestamp - lag(timestamp), units = "secs")
  ) %>%
  filter(event == "HINT" & prev_event == "HINT") %>%
  ungroup()
```

```r
#do the same for spamming evaluate
full_data <- full_data %>%
  arrange(session_id, timestamp) %>%  # ensure chronological order
  group_by(session_id) %>%
  mutate(
    time_diff_eval = timestamp - lag(timestamp),
    prev_eval = lag(event) == "EVALUATE",
    spam_eval = case_when(
      event == "EVALUATE" & prev_eval & time_diff_eval <= 3 ~ 1,
      event == "EVALUATE" ~ 0,
      TRUE ~ NA_real_
    )
  ) %>%
  ungroup()

# include rows where event 1 and event 2 are both evaluate,
# and evaluate 2 is incorrect
consecutive_evals <- full_data %>%
  arrange(session_id, timestamp) %>%
  group_by(session_id) %>%
  mutate(
    prev_event = lag(event),
    prev_progress = lag(progress),
    time_between_evals = as.numeric(timestamp - lag(timestamp),
                                    units = "secs")
  ) %>%
  filter(
    event == "EVALUATE",
    prev_event == "EVALUATE",
    progress <= prev_progress  # progress did not increase
  ) %>%
  ungroup()

# Tidying the environment
rm(all_events, interaction_row, items_df, out, results, tmp, i)
```

Exploring the data

```r
# check number of times where hints or eval were spammed,
# as well as total number of cases with consecutive hints or evals
full_data %>% filter(spam_hint == c(0, 1))
full_data %>% filter(spam_hint == 1)
full_data %>% filter(spam_eval == c(0, 1))
full_data %>% filter(spam_eval == 1)

# proportion of cases where hint was spammed (cases, not people)
25 / 3855

# proportion where eval was spammed (cases, not people)
# - this is also given the second eval was wrong
13424 / 67114
```

```r
# comparing whether people who spam hints
# get the right answer in the end (correct vs incorrect)
# code might be incorrect, and does not count individual sessions,
# rather each event

# out of all sessions, we check how many
# spammed evaluate and got it wrong in the end
full_data %>%
  filter(spam_eval == 1 & correct_answer == 0) %>%
  distinct(file, .keep_all = TRUE)

# comparing whether people who spam evaluate
# get the right answer in the end
full_data %>%
  filter(event == "EVALUATE") %>%
  group_by(spam_eval) %>%
  summarise(correct = mean(correct_answer == 1),
            incorrect = mean(correct_answer == 0))
# doesn't make a difference
# in whether they get the answer correct


# initial response is incorrect or there is no
# response before pressing hint,
# do they use hint afterwards or not?
session_behaviours <- full_data %>%
  group_by(session_id) %>%
  summarise(
    events = list(event),
    progresses = list(progress)
  ) %>%
  mutate(
    # Condition 1: First two events are START then HINT
    start_then_hint = map_lgl(events, ~ length(.) >= 2 &&
                                .[[1]] == "START" &&
                                .[[2]] == "HINT"),
    # Condition 2: First EVALUATE is incorrect, followed by any HINT
    hint_after_wrong = map2_lgl(events, progresses, function(ev, prog) {
      eval_indices <- which(ev == "EVALUATE")
      hint_indices <- which(ev == "HINT")
      if (length(eval_indices) == 0 || length(hint_indices) == 0) return(FALSE)
      first_eval <- eval_indices[1]
      first_eval_correct <- !is.na(prog[[first_eval]]) &&
        prog[[first_eval]] == 1
    }),
    # Combined flag
    used_hint_after_start_or_wrong = start_then_hint | hint_after_wrong
  )

full_data <- left_join(full_data,
                       session_behaviours %>%
                         select(session_id, used_hint_after_start_or_wrong),
                       by = "session_id")
```

```r
#proportion of people who used a hint after start or wrong answer
full_data %>%
  distinct(session_id, used_hint_after_start_or_wrong) %>%
  summarise(proportion = mean(used_hint_after_start_or_wrong, na.rm = TRUE))


#check whether people improve after pressing "evaluate" - using progress
#code might not be fully correct, check for logic
data_eval <- full_data %>%
  filter(event == "EVALUATE") %>%
  arrange(session_id, timestamp) %>%
  group_by(session_id) %>%
  mutate(
    progress1 = progress,
    progress2 = lead(progress),
    improved = case_when(
      !is.na(progress1) &
        !is.na(progress2) &
        progress2 > progress1 ~ "Improved",
      !is.na(progress1) &
        !is.na(progress2) &
        progress2 <= progress1 ~ "No improvement",
      TRUE ~ NA_character_
    )
  ) %>%
  ungroup()

# remove NAs in improved column, and then do calculations
# proportion of people who improved vs
# didnt improve after pressing "evaluate" once
data_eval %>%
  filter(!is.na(improved)) %>%
  summarise(Improved = mean(improved == "Improved"),
            No_improvement = mean(improved == "No improvement"))
```

Calculating the data we used in our tables

```r
# -------------- Calculating proportions for hints and evaluates ---------------

# 1. Sessions where a hint was used
hint_spam_summary <- full_data %>%
  # only consider sessions with at least one HINT
  filter(event == "HINT") %>%
  group_by(session_id) %>%
  # check if any spam occurred in session
  summarise(spammed_hint = any(spam_hint == 1, na.rm = TRUE))

# Proportion of sessions where hints were spammed
# (among sessions where hint used)
hint_spam_summary %>%
  summarise(
```

```
    prop_spammed = mean(spammed_hint),
    total_sessions = n()
  )

# 2. Sessions where evaluate was spammed (based on progress not increasing)
eval_spam_summary <- full_data %>%
  filter(event == "EVALUATE") %>%
  arrange(session_id, timestamp) %>%
  group_by(session_id) %>%
  mutate(
    time_diff = as.numeric(timestamp - lag(timestamp), units = "secs"),
    progress_diff = progress - lag(progress),
    # define spam as quick repeat without improvement
    is_spam_eval = time_diff <= 3 & progress_diff <= 0
  ) %>%
  summarise(num_evals = n(),
            spammed_eval = any(is_spam_eval, na.rm = TRUE),
            .groups = "drop") %>%
  filter(num_evals >= 2) # exclude sessions with only 1 evaluate

# 3. Get sessions that did not get full marks on first EVALUATE (progress < 1)
sessions_wrong_first_eval <- full_data %>%
  filter(event == "EVALUATE") %>%
  arrange(session_id, timestamp) %>%
  group_by(session_id) %>%
  summarise(first_progress = first(progress)) %>%
  filter(first_progress < 1) %>%
  pull(session_id)

# 4. Get list of session_ids for spammed hint and evaluate
spam_hint_sessions <- hint_spam_summary %>%
  filter(spammed_hint) %>%
  pull(session_id)

spam_eval_sessions <- full_data %>%
  filter(event == "EVALUATE", session_id %in% sessions_wrong_first_eval) %>%
  arrange(session_id, timestamp) %>%
  group_by(session_id) %>%
  mutate(
    time_diff = as.numeric(timestamp - lag(timestamp), units = "secs"),
    progress_diff = progress - lag(progress),
    is_spam_eval = time_diff <= 3 & progress_diff <= 0
  ) %>%
  summarise(spammed_eval =
              any(is_spam_eval, na.rm = TRUE), .groups = "drop") %>%
  filter(spammed_eval) %>%
  pull(session_id)

# Proportion of sessions where evaluate was spammed
# (among sessions where the first evaluate wasn't correct)
tibble(
  total_sessions = length(sessions_wrong_first_eval),
  spammy_sessions = length(spam_eval_sessions),
```

```
  proportion_spam = spammy_sessions / total_sessions
)

# 5. Sessions that spammed either hint OR evaluate
spammy_sessions <- union(spam_hint_sessions, spam_eval_sessions)

# 6. Restrict to sessions where first evaluate was wrong
spammy_in_wrong_first <- intersect(spammy_sessions, sessions_wrong_first_eval)

# 7. Final result tibble
total_sessions <- length(sessions_wrong_first_eval)
spammy_sessions <- length(spammy_in_wrong_first)

result <- tibble(
  total_sessions = total_sessions,
  spammy_sessions = spammy_sessions,
  proportion_spam = spammy_sessions / total_sessions
)

print(result)



# ---------- Proportion of users who improved with/without hints --------------


# Step 1: Get sessions where the first event is not EVALUATE and correct
excluded_sessions <- full_data %>%
  arrange(session_id, timestamp) %>%
  group_by(session_id) %>%
  slice(1) %>%
  filter(event == "EVALUATE", progress == 1) %>%
  pull(session_id)

# Step 2: Filter out excluded sessions
filtered_data <- full_data %>%
  filter(!session_id %in% excluded_sessions)

# Step 3: Arrange data
filtered_data <- filtered_data %>%
  arrange(session_id, timestamp)

# Step 4: For each session, calculate progress improvement after HINT
hint_improvement <- filtered_data %>%
  group_by(session_id) %>%
  mutate(
    prev_eval_progress = lag(progress, default = 0),
    next_eval_progress = lead(progress),
    next_event = lead(event),
    prev_event = lag(event)
  ) %>%
  filter(event == "HINT" & next_event == "EVALUATE") %>%
  mutate(progress_change = next_eval_progress -
```

```
            if_else(prev_event == "EVALUATE",
                    prev_eval_progress,
                    0)) %>%
  ungroup()

# Step 5: For each session, calculate progress improvement after EVALUATE
eval_improvement <- filtered_data %>%
  group_by(session_id) %>%
  mutate(
    curr_progress = progress,
    next_eval_progress = lead(progress),
    next_event = lead(event)
  ) %>%
  filter(event == "EVALUATE" & next_event == "EVALUATE") %>%
  mutate(progress_change = next_eval_progress - curr_progress) %>%
  ungroup()

# Step 6: Summary - average progress change
summary_table <- tibble(
  source = c("After Hint", "After Evaluate"),
  mean_progress_increase = c(
    mean(hint_improvement$progress_change, na.rm = TRUE),
    mean(eval_improvement$progress_change, na.rm = TRUE)
  ),
  proportion_improved = c(
    mean(hint_improvement$progress_change > 0, na.rm = TRUE),
    mean(eval_improvement$progress_change > 0, na.rm = TRUE)
  ),
  n_cases = c(nrow(hint_improvement), nrow(eval_improvement))
)

summary_table

full_data %>%
  filter(progress >= 0.9 & progress < 1 & spam_eval == 1)
```

Our code for creating our plots

The density plots

```
#plot the density on y axis and x axis is time between consecutive hints
font_add_google("Tenor Sans", "tenor_sans")
showtext_auto()

# Time between events defined as spam
threshold <- 3
epsilon <- 0.1

### DENSITIES

# Compute densities manually for each event
```

```r
# Separate standardized tables for hints and evaluations
hints <- consecutive_hints %>%
  mutate(event_type = "HINT") %>%
  rename(time_between = time_between_hints)

evals <- consecutive_evals %>%
  mutate(event_type = "EVALUATE") %>%
  rename(time_between = time_between_evals)

# Combine
combined <- bind_rows(hints, evals)

densities <- combined %>%
  group_by(event_type) %>%
  do({
    dens <- density(.$time_between, from = 0, to = 30)
    data.frame(x = dens$x, y = dens$y, event_type = unique(.$event_type))
  })

# Split into two segments based on threshold
densities <- densities %>%
  mutate(segment = case_when(
    x <= threshold + epsilon ~ "before",
    TRUE ~ "after"
  ))

## Hint density
dens_hint <- density(consecutive_hints$time_between_hints, from = 0, to = 20)
dens_df_hint <- data.frame(x = dens_hint$x, y = dens_hint$y)
# Split
dens_df_hint$segment <- ifelse(dens_df_hint$x <= threshold, "before", "after")

## Eval density
dens_eval <- density(consecutive_evals$time_between_evals, from = 0, to = 60)
dens_df_eval <- data.frame(x = dens_eval$x, y = dens_eval$y)
# Split
dens_df_eval$segment <- ifelse(dens_df_eval$x <= threshold, "before", "after")

### THEME
base_theme <- theme_minimal(base_size = 16)

title_left_theme <- theme(
  plot.title = element_text(
    hjust = 0.5,          # left align
    vjust = 1,          # slightly higher up
    face = "bold",
    size = 16,
    color = "#383838"
  )
)

### TITLES
# Optional: the thoughts was to add the titles as a textblock to the left
```

```r
# of the plots.

# Title block for hint
title_h <- ggplot() +
  annotate("text",
           x = 0,
           y = 0.5,
           label = "Density of Time Between\nConsecutive Hints",
           hjust = 0, vjust = 0.5, size = 2, fontface = "bold") +
  theme_void() +
  theme(plot.margin = margin(0, 0, 0, 0))

# Title block for evaluation
title_e <- ggplot() +
  annotate("text",
           x = 0,
           y = 0.5,
           label = "Density of Time Between\nConsecutive Evaluations",
           hjust = 0, vjust = 0.5, size = 2, fontface = "bold") +
  theme_void() +
  theme(plot.margin = margin(0, 0, 0, 0))

### PLOTS

# Plot the density on y axis and x axis as time between consecutive hints
# Hint plot
hint_dist <- ggplot(dens_df_hint, aes(x = x, y = y, fill = segment)) +
  geom_area(data = subset(dens_df_hint, segment == "before"),
            fill = "#01317a", alpha = 0.7) +
  geom_line(data = subset(dens_df_hint, segment == "before"),
            color = "#01317a", alpha = 0.7) +
  geom_area(data = subset(dens_df_hint, segment == "after"),
            fill = "#01317a", alpha = 0.3) +
  geom_line(data = subset(dens_df_hint, segment == "after"),
            color = "#01317a", alpha = 0.3) +
  geom_vline(xintercept = threshold,
             linetype = "solid",
             colour = "#383838",
             linewidth = 0.6) +
  labs(
    title = "Density of Time Between Consecutive Hints",
    x = "Time Between Hints (seconds)",
    y = "Density"
  ) +
  xlim(c(0, 20)) +
  base_theme +
  title_left_theme

# Hint block
hint_block <- title_h | hint_dist

# Plot the density on y axis and x axis as time between consecutive evaluates
# Eval plot
```

```
eval_dist <- ggplot(dens_df_eval, aes(x = x, y = y, fill = segment)) +
  # Area before threshold
  geom_area(data = subset(dens_df_eval, segment == "before"),
            fill = "#d55e00", alpha = 0.7) +
  geom_line(data = subset(dens_df_eval, segment == "before"),
            color = "#d55e00", alpha = 0.7) +
  # Area after threshold
  geom_area(data = subset(dens_df_eval, segment == "after"),
            fill = "#d55e00", alpha = 0.3) +
  geom_line(data = subset(dens_df_eval, segment == "after"),
            color = "#d55e00", alpha = 0.3) +
  geom_vline(xintercept = threshold,
             linetype = "solid",
             colour = "#383838",
             linewidth = 0.6) +
  labs(
    title = "Density of Time Between Consecutive Evaluations",
    x = "Time Between Evaluations (seconds)",
    y = "Density"
  ) +
  xlim(c(0, 20)) +
  base_theme +
  title_left_theme

# Eval block
eval_block <- title_e | eval_dist

### FINAL DENSITY PLOT
dense_plot <- hint_dist / eval_dist +
  plot_layout(widths = c(0.5, 2))

dense_plot
```

The plots comparing how much progress is made after spamming hint or evaluate

```
# ------------------- Code for the Data Needed for the Plot -------------------
second_plot_data <- full_data %>%
  arrange(desc(row_number())) %>%
  fill(progress, .direction = "down") %>%
  arrange(row_number()) %>%
  filter(spam_hint == 1 | spam_eval == 1)

# Bin width
bin_width <- 0.1

second_plot_data <- second_plot_data %>%
  filter(event %in% c("HINT", "EVALUATE")) %>%
  mutate(
    progress_bin = cut(
      progress,
      breaks = seq(0, 1, by = bin_width),
      include.lowest = TRUE,
```

```
      right = FALSE,
      labels = paste0(seq(0, 0.95, by = bin_width) * 100,
                      "%-",
                      seq(bin_width, 1, by = bin_width) * 100, "%")
    )
  ) %>%
  group_by(event, progress_bin) %>%
  summarise(count = n(), .groups = "drop") %>%
  group_by(event) %>%
  mutate(percentage = count / sum(count) * 100)

# ------------------------------ Code for the Plot ------------------------------
# THEME
plot2_theme <- theme(
  plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
  axis.text.x = element_text(angle = 45, hjust = 1)
)

# Set colors and labels for legend
fill_scale <- scale_fill_manual(
  name = NULL,
  values = c("HINT" = "#355d9b", "EVALUATE" = "#dc823d"),
  labels = c(
    "HINT" = "Spammed\nHint",
    "EVALUATE" = "Spammed\nEvaluation"
  ),
  drop = FALSE
)

# HINT
spam_hint <- ggplot(
  subset(second_plot_data, event == "HINT"),
  aes(x = progress_bin, y = percentage, fill = event)
) +
  geom_col(linewidth = 0.4) +
  fill_scale +
  labs(x = NULL, y = "Percentage of Events") +
  scale_x_discrete(
    labels = c("0-10", "50-60", "90-100")
  ) +
  plot2_theme +
  base_theme +
  scale_y_continuous(labels = scales::label_percent(scale = 1)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1, size = 9)
  )

# EVALUATE
spam_eval <- ggplot(
  subset(second_plot_data, event == "EVALUATE"),
  aes(x = progress_bin, y = percentage, fill = event)
) +
  geom_col(linewidth = 0.4) +
  fill_scale +
```

```
  labs(x = "Progress (%)", y = NULL) +
  scale_x_discrete(
    labels = c("0-10", "10-20", "20-30", "30-40", "40-50",
               "50-60", "60-70", "70-80", "80-90", "90-100")
  ) +
  plot2_theme +
  base_theme +
  ylim(0, 60) +
  theme(axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y  = element_blank(),
    axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1, size = 9),
    axis.title.x = element_text(hjust = 0)
  )

# Combine side by side and collect the single legend
spam_combined_plot <- (spam_hint | spam_eval) +
  plot_layout(widths = c(0.33, 1), guides = "collect") &
  theme(
    legend.position = "right"
  )


### FINAL PROGRESS PLOT
spam_prog_plot <- spam_combined_plot +
  plot_annotation(
    title = "Distribution of Spammed Hints and Evaluations Across Progress",
    subtitle = "",
    theme = theme(plot.title = element_text(hjust = 0,
                                            vjust = -2,
                                            face = "bold",
                                            color = "#383838")
    )
  )

spam_prog_plot
# Possible conclusion:
# Spamming patterns might suggest initial
# lack of direction and end-of-task disengagement.
```

The plots comparing how much progress is made after using a hint or evaluation

```
# ------------------- Code for the Data Needed for the Plot -------------------
first_plot_data <- full_data %>%
  arrange(desc(row_number())) %>%
  fill(progress, .direction = "down") %>%
  arrange(row_number())

bin_width <- 0.1

first_plot_data <- first_plot_data %>%
  filter(event %in% c("HINT", "EVALUATE")) %>%
```

```r
  mutate(
    progress_bin = cut(
      progress,
      breaks = seq(0, 1, by = bin_width),
      include.lowest = TRUE,
      right = FALSE,
      labels = paste0(seq(0, 0.95, by = bin_width) * 100,
                      "%-",
                      seq(bin_width, 1, by = bin_width) * 100, "%")
    )
  ) %>%
  group_by(event, progress_bin) %>%
  summarise(count = n(), .groups = "drop") %>%
  complete(event = c("HINT", "EVALUATE"),
           progress_bin = levels(progress_bin),
           fill = list(count = 0)) %>%
  group_by(event) %>%
  mutate(percentage = count / sum(count) * 100)

# ---------------------------- Code for the Plot ----------------------------

# HINT
p_hint <- ggplot(
  subset(first_plot_data, event == "HINT"),
  aes(x = progress_bin, y = percentage, fill = event)
) +
  geom_col(linewidth = 0.4) +
  fill_scale +
  labs(x = "Progress (%)", y = "Percentage of Events") +
  scale_x_discrete(
    labels = c("0-10", "10-20", "20-30", "30-40", "40-50",
               "50-60", "60-70", "70-80", "80-90", "90-100")
  ) +
  plot2_theme +
  base_theme +
  scale_y_continuous(limits = c(0, 60),
                     labels = scales::label_percent(scale = 1)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1, size = 9),
        axis.title.x = element_text(hjust = 0))

# EVALUATE
p_eval <- ggplot(
  subset(first_plot_data, event == "EVALUATE"),
  aes(x = progress_bin, y = percentage, fill = event)
) +
  geom_col(linewidth = 0.4) +
  fill_scale +
  labs(x = NULL, y = NULL) +
  scale_x_discrete(
    labels = c("0-10", "10-20", "20-30", "30-40", "40-50",
               "50-60", "60-70", "70-80", "80-90", "90-100")
  ) +
  plot2_theme +
```

```
  base_theme +
  scale_y_continuous(limits = c(0, 60)) +
  theme(axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y  = element_blank(),
    axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1, size = 9),
  )


# Combine side by side and collect the single legend
combined_plot <- (p_hint | p_eval) +
  plot_layout(widths = c(1, 1), guides = "collect") &
  theme(
    legend.position = "right"
  )

### FINAL PROGRESS PLOT

prog_plot <- combined_plot +
  plot_annotation(
    title = "Distribution of All Hints and Evaluations Across Progress",
    subtitle = "",
    theme = theme(plot.title = element_text(hjust = 0,
                                            vjust = -2,
                                            face = "bold",
                                            color = "#383838")
    )
  )

prog_plot
```

```
lint(rstudioapi::getSourceEditorContext()$path)
```

```
## i No lints found.
```