Package: FocusGroup (via r-universe)

October 30, 2026

Type Package

Title Focus Group Simulation Using LLM Agents

Version 0.5.6

Description Moderated focus group discussions simulated with large language models (LLMs). Participants carry personas built from respondent records or labeled survey files. A session runs under a specified turn-taking rule, and one turn can be experimentally manipulated to measure its effect on the next.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Config/testthat/edition 3

Depends R (>= 4.1.0)

Imports R6, dplyr, tidyr, rlang, haven, withr, jsonlite, LLMR (>= 0.8.8)

Suggests testthat (>= 3.0.0), ggplot2 (>= 3.4.0), quanteda, quanteda.textstats, topicmodels, tidytext, knitr, rmarkdown, pkgdown, shiny, bslib, DT, LLMR.shiny (>= 0.1.3)

VignetteBuilder knitr

URL https://github.com/asanaei/FocusGroup, https://asanaei.github.io/FocusGroup/

BugReports https://github.com/asanaei/FocusGroup/issues

NeedsCompilation no

Author Ali Sanaei [aut, cre]

Maintainer Ali Sanaei <sanaei@uchicago.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-30 19:31:07 UTC

RemoteUrl https://github.com/cran/FocusGroup

RemoteRef HEAD

RemoteSha eed540742dc6e78b85e363af4ae490f664fa0eb1

Contents

analyze_focus_group . . . 2
anes_2024_personas . . . 4
ConversationFlow . . . 6
create_agents . . . 8
create_agents_from_data . . . 10
create_agents_from_survey . . . 12
create_conversation_flow . . . 17
fg_after_item . . . 18
fg_default_na_strings . . . 19
fg_injection . . . 19
fg_player . . . 21
fg_probe . . . 23
fg_turns . . . 24
FGAgent . . . 26
focus_group_from_transcript . . . 31
FocusGroup . . . 34
format.focus_group_result . . . 44
get_default_prompt_templates . . . 46
persona_rendering_report . . . 48
run_focus_group . . . 50
run_focus_group_arms . . . 56
run_focus_studio . . . 61
summarize_focus_group . . . 63

Index **68**

`analyze_focus_group` *Analyze Focus Group Results*

Description

Runs descriptive analyses offline. Thematic analysis and a model-generated summary are opt-in and run only when `config` is supplied.

Usage

```
analyze_focus_group(
  focus_group_result,
  num_topics = 5,
  include_plots = TRUE,
  speakers = c("participants", "all"),
  language = "en",
```

```
  config = NULL,
  .runner = NULL
)
```

Arguments

focus_group_result	A FocusGroup object or focus_group_result. A result whose focus_group is NULL, as one saved without its session object, is read back with focus_group_from_transcript().
num_topics	Integer number of topics for topic modeling.
include_plots	Logical. Attempt the four descriptive plots.
speakers	Whose turns the text statistics describe: "participants" (default) excludes the moderator's scripted turns; "all" keeps them. The transcript itself always retains every turn.
language	Language of the transcript (default "en"). The TF-IDF stop-word list and the readability formula are English-specific, so those components are skipped, with a note in issues, for any other value.
config	Optional explicit LLMR::llm_config. When supplied, thematic analysis and a model summary are generated.
.runner	Optional function used instead of live model calls for the two model analyses.

Details

The two model analyses add their usage to the token fields of the FocusGroup object, including calls made before one of them fails. The usage component of a focus_group_result passed in is not changed: it records the session as run. An error from a model analysis is not caught.

Value

A focus_group_analysis containing the returned analysis components, basic_stats, topics, tfidf, readability, themes, model_summary, and plots, and an issues table with columns component and reason for the components that were skipped or failed.

Examples

```
transcript <- data.frame(
  speaker = c("Moderator", "P1", "P2"),
  text = c("What matters most?", "Cost matters.", "Access matters.")
)
fg <- focus_group_from_transcript(transcript)
analysis <- analyze_focus_group(fg, include_plots = FALSE)
analysis$basic_stats$speaker_stats
```

anes_2024_personas *Persona profiles of 100 respondents to the ANES 2024 Time Series Study*

Description

A data frame describing 100 respondents to the American National Election Studies (ANES) 2024 Time Series Study by their demographics and their political attitudes, written as text a language model can read. It serves as ready-made persona material: `create_agents_from_data()` builds participants from any subset of its rows and columns. LLMRpanel ships an identical copy.

Usage

```
anes_2024_personas
```

Format

A data frame with 100 rows and 123 columns. Each row is one respondent; no ANES case identifier is kept. Column names are handles whose prefix names a block, so `startsWith(names(x), "att_iss_")` selects the issue positions:

- `demo_`: 17 demographic fields: age band, sex, education, race/ethnicity, marital status, household income, religion and its importance, religious attendance, census region, rural-urban self-identification, employment status, home ownership, children in the household, union household, military service, and attention to politics.
- `att_id_`: 9 fields of party identification, ideology, and vote choice.
- `att_aff_`: 18 feeling thermometers toward candidates, parties, institutions, and groups.
- `att_eval_`: 14 evaluations: approval of the President, Congress, and the Supreme Court, the economy, personal finances, feelings about the country, and which party would better handle the economy and immigration.
- `att_iss_`: 47 issue positions, among them taxes and spending, immigration, guns, climate, health care, abortion, policing, race, and foreign policy.
- `att_val_`: 17 fields of trust, values, and views of democracy.
- `ideology_score`: A number placing the respondent on one liberal-conservative dimension (see Details); the rows are sorted by it.

Every column but `ideology_score` holds character values:

- An answer with a value label is written as that label without its code ("Strong Democrat", "Favor a great deal"). Parenthetical qualifiers are kept, the interviewer notes "{VOL}" and "{SPECIFY}" are removed, and the four labels of the immigration policy item (V241386) that the Stata file cuts at 80 characters are restored in full from the codebook.
- The six 7-point scales that ANES labels only at their ends are written as sentences, endpoints included: "5 on a scale from 1 (Government should provide many fewer services) to 7 (Government should provide many more services)".
- Feeling thermometers are written as "70 on a 0-100 feeling thermometer".

- "Haven't thought much about this" is kept as an answer where ANES offered it.
- Age is given in six bands and children in the household as "Yes" or "No". Employment status is written from the full ANES labels without their "R " prefix ("Retired (if also working, working < 20 hrs/wk)"). Religious attendance is "Never" for respondents who said they never attend services (V241439 = 2), who were not asked how often (V241440).
- `NA` marks a missing-data code: refused, don't know, inapplicable, no post-election interview, and so on, and a thermometer rating outside 0-100 ("don't know where to rate", "don't recognize"). The 100 rows hold 246 such cells, most of them in the vote items, which do not apply to everyone.

The data frame carries these attributes:

- `dictionary`: A data frame with one row per column except `ideology_score`: `handle`; `question`, the short phrase shown with an answer ("Abortion position"); `anes_variable`, the ANES variable; `domain`, "demographic" or the attitude block; `wording`, the question as printed in the ANES codebook; `wording_variable` and `wording_page`, the variable whose question that is and the codebook page where it begins; and `wording_status`. A status of "verbatim" means the wording is the variable's own question; "from_source_item" means the variable is a summary without a question of its own and the wording is that of the item it summarizes (for party identification, V241221); "stem_only" means the codebook prints only a shared stem ("Where would you place yourself on this scale, or haven't you thought much about this?") and the subject of the scale is in its end labels; "none" means no single question exists (census region, age), and `wording` is `NA`.
- `demographic_fields`: The handles of the demographic columns.
- `n_demographics`, `n_attitudes`: 17 and 105.
- `source`, `citation`: The citation of the ANES release.
- `build`: The R and package versions, the md5 sums of the two ANES files, the seed, and the label overrides used to build the data.
- `build_audit`: For each column, the number of `NA` cells and the number of missing-data codes in the raw file for the same respondents; the two are equal.

Details

The respondents were drawn from the 4,710 of 5,521 who completed both the pre-election and the post-election interview, neither of them on paper, and answered at least 75 percent of the 122 fields. For each code of party identification, race/ethnicity, and liberal-conservative self-placement not yet covered, one respondent with that code was drawn at random (seed 110); the remaining places were filled by greedy maximin selection, each time adding the respondent farthest from those already chosen, under a Gower distance in which the demographic, identity, thermometer, evaluation, issue, and value blocks carry weights 0.25, 0.25, 0.20, 0.15, 0.15, and 0.15. Each respondent's answers are kept together.

The sample spans unusual combinations of answers and does not reproduce population proportions. It carries no survey weights and supports no population inference. Maximin selection overrepresents people with no 2024 presidential vote recorded (38 of the 100, against 18.5 percent of the eligible respondents), and within each party it favors members whose other answers depart from the party's usual positions.

ideology_score comes from a one-dimensional graded-response item response model fitted to 89 attitude items over the 4,710 eligible respondents. Categories enter in the order of their ANES codes (thermometers grouped into five bands, off-scale answers such as "Haven't thought much about this" left out), and items are reversed where they correlate negatively with party identification and ideology. The expected a posteriori scores are standardized over the eligible respondents and oriented so that low is liberal and high is conservative. They correlate 0.89 with party identification and ideology among the eligible respondents and 0.58 among the 100, whose scores have a standard deviation of 0.55.

Items on sexual orientation and gender identity are excluded by design, as are restricted-use variables, geography below the census region, open-ended answers, and items asked only of the spouse/partner sample, only on paper, or only of the GSS re-interview sample, which this release does not include.

Source

American National Election Studies. 2026. ANES 2024 Time Series Study Full Release [dataset and documentation]. May 19, 2026 version. www.electionstudies.org. Work that uses these profiles should cite ANES. ANES bears no responsibility for the analyses or interpretations presented here.

Examples

```
dim(anes_2024_personas)
dictionary <- attr(anes_2024_personas, "dictionary")
dictionary[dictionary$handle == "att_iss_abortion_position",
           c("question", "wording", "wording_status")]
table(anes_2024_personas$att_id_party_identification)
issues <- anes_2024_personas[, startsWith(names(anes_2024_personas), "att_iss_")]
ncol(issues)
# The subset has lost the dictionary; create_agents_from_data() restores
# it, so the answers are shown under their questions.
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
agents <- create_agents_from_data(issues[1:10, 1:3], n_participants = 2,
                                  config = cfg, rows = 1:2)
cat(agents$P1$persona_description)
```

ConversationFlow *ConversationFlow Base Class*

Description

ConversationFlow is an R6 base class defining the interface for turn-taking mechanisms in a focus group simulation. Subclasses implement specific strategies for selecting the next speaker.

Value

An R6 generator meant for subclassing; instances are built by create_conversation_flow(), which returns a ConversationFlow subclass object ready for FocusGroup.

Methods for Subclassing

`initialize(agents, moderator_id)` Sets up the flow. Call `super$initialize(agents, moderator_id)`.

`select_next_speaker(focus_group)` *Required*. Logic to choose the next `FGAgent` to speak. Should return the agent object or `NULL`.

`update_state_post_selection(speaker_id, focus_group)` Optional. Updates internal state after a speaker has spoken. Base implementation updates `self$last_speaker_id`.

`note_moderator_turn()` Optional. Called after every moderator line: a guide item, a call-on line, or an injected moderator message. Base implementation clears `self$last_speaker_id`, so that a flow that passes over the previous speaker bars only back-to-back participant turns.

Public fields

`agents` A named list of `FGAgent` objects participating in the conversation.

`participant_ids` A character vector of agent identifiers, excluding the moderator.

`moderator_id` Character. The ID of the moderator agent.

`last_speaker_id` The ID of the agent who last spoke. Can be `NULL`.

`selection_metadata` Details recorded for the latest selection when it departs from the flow's usual rule, such as a lowered desire threshold or, under `on_desire_failure = "random"`, a desire-scoring failure.

`routing_record` The record of the latest selection, whatever rule made it: a list with `flow`, `rule`, `candidates`, `excluded` (a named character vector giving the reason each excluded participant was passed over), `scores`, `threshold`, `tied`, `tie_break`, and `selected_id`. A running session copies it into its routing record after every pick. A subclass that does not set it is recorded with rule `"custom"`.

Methods

Public methods:

- `ConversationFlow$new()`
- `ConversationFlow$select_next_speaker()`
- `ConversationFlow$update_state_post_selection()`
- `ConversationFlow$note_moderator_turn()`
- `ConversationFlow$clone()`

Method `new()`**:** Initializes the ConversationFlow object.

Usage:

```
ConversationFlow$new(agents, moderator_id)
```

Arguments:

`agents` A named list of `FGAgent` objects.

`moderator_id` Character. The ID of the moderator agent.

Method `select_next_speaker()`**:** Selects the next speaker. Must be implemented by subclasses.

Usage:

```
ConversationFlow$select_next_speaker(focus_group)
```

Arguments:

`focus_group` The `FocusGroup` object managing the simulation, providing context.

Returns: The `FGAgent` object of the selected speaker, or `NULL`.

Method `update_state_post_selection()`**:** Updates internal state after a speaker is selected.

Usage:

```
ConversationFlow$update_state_post_selection(speaker_id, focus_group)
```

Arguments:

`speaker_id` The ID of the agent who was selected and just spoke.

`focus_group` The `FocusGroup` object.

Method `note_moderator_turn()`**:** Notes that the moderator has just spoken. The participant who spoke before the moderator's line is then no longer the previous speaker: whoever closed one question may open the next.

Usage:

```
ConversationFlow$note_moderator_turn()
```

Returns: The flow, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ConversationFlow$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
RandomFlow <- R6::R6Class("RandomFlow", inherit = ConversationFlow,
  public = list(
    select_next_speaker = function(focus_group) {
      ids <- self$participant_ids
      self$agents[[sample(ids, 1)]]
    }))
```

`create_agents` *Create Focus Group Agents*

Description

Creates participant agents from supplied records, direct personas, or the bundled ANES 2024 records, and adds a moderator.

Usage

```
create_agents(
  n_participants,
  config,
  demographics = NULL,
  survey_responses = NULL,
  direct_persona_descriptions = NULL,
  na_strings = fg_default_na_strings(),
  .runner = NULL
)
```

Arguments

n_participants	Integer number of participants to create
config	An explicit LLMR::llm_config for all agents.
demographics	Optional data frame, one row per participant. When it, survey_responses, and direct_persona_descriptions are all NULL, participants are drawn from the bundled ANES 2024 records. Columns with value labels (class haven_labelled) are an error unless the personas are supplied: create_agents_from_data() renders them, where this function would show their codes.
survey_responses	Optional data frame of survey answers, one row per participant, with value labels treated as for demographics.
direct_persona_descriptions	Optional character vector of pre-rendered participant personas. When supplied, these descriptions are used directly and are recycled in order if necessary. Demographics and survey responses remain attached as raw reporting fields.
na_strings	Character vector of regular expressions, matched ignoring case, that mark a text or factor value of demographics or survey_responses as a missing-data label, which the persona leaves out, by the rule described under Rendering answers in create_agents_from_survey(): a value that matches and starts with a negative code or one of 90 and above ("-9. Refused", "[-8] Don't know", "97. Refused"), or one that says nothing but the matched wording ("Refused to answer", "Don't know (VOL)"). Answers that use such words ("I keep missing the bus", "2. Refused to take sides") are kept, and numbers are never read this way. Each value left out is recorded in the rendering report, with a message naming the fields. The default is fg_default_na_strings(); character() turns the screening off.
.runner	Optional function used instead of live model calls and stored on every agent.

Value

A named list of FGAgent objects (participants + 1 moderator), keyed by agent ID. When the participants are drawn from the bundled ANES records, the list carries the source_rows attribute described in create_agents_from_survey(), giving the rows of anes_2024_personas that were drawn. Unless the personas are given in direct_persona_descriptions, the list also carries a rendering_report attribute recording each field supplied for each participant and whether the persona shows it; see persona_rendering_report().

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
agents <- create_agents(6, config = cfg)
names(agents)
agents$P1$persona_description

demo_data <- data.frame(
  age = c(22, 35, 28, 41, 19, 33),
  gender = c("Female", "Male", "Male", "Female", "Male", "Female"),
  education = c("Bachelor's", "Master's", "High School", "PhD",
                "Some College", "Bachelor's")
)
agents <- create_agents(6, config = cfg, demographics = demo_data)
agents$P2$persona_description
```

---

create_agents_from_data

Create agents from an in-memory data frame of respondents

Description

Like `create_agents_from_survey()` but starting from a data frame already in memory (for example anes_2024_personas). Demographic columns are rendered as background; the remaining columns are rendered as survey responses, keyed by their question wording when the frame carries a `dictionary` attribute (see `LLMR::llm_persona_split()`), else by their column names. A frame taken from anes_2024_personas by base subsetting (`[`, `subset()`), which drops the attributes, has its dictionary and demographic fields restored from the package's copy when every column is one of the dataset's, with a message. Columns whose names identify respondents (`id`, `caseid`, `respondent_id`, `respid`, `persona_id`, `uid`, `record_id`, `row_id`, `ResponseId`, `participant_id`, `subject_id`, `user_id`, `PROLIFIC_PID`, `workerId`, `caseno`, or `QKEY`, in any case and with or without spaces or punctuation, as in `ID`, `CaseID`, or `resp_id`) are not rendered: they are kept in the `source_ids` attribute and listed in the rendering report as dropped. The same holds for those columns in a data frame given to `create_agents()`.

Usage

```
create_agents_from_data(
  data,
  n_participants,
  config,
  demographic_cols = NULL,
  rows = NULL,
  weights = NULL,
  na_strings = fg_default_na_strings(),
  scale_template = .fg_default_scale_template,
  .runner = NULL
)
```

Arguments

data	A data frame, one respondent per row.
n_participants	Integer number of participants (excludes the moderator).
config	An explicit LLMR::llm_config for all agents.
demographic_cols	Character vector of columns to render as demographics. Defaults to the data's "demographic_fields" attribute when present, else a small set of common demographic column names found in data.
rows, weights	See create_agents_from_survey(). Here rows refers to the rows of data, and a predicate receives the demographics frame with its columns relabeled as they appear in the personas and holding the text rendered for each field. Integer positions are kept in the order given.
na_strings	Character vector of regular expressions, matched ignoring case, that mark a text or factor value, or a value label, as missing, as in create_agents_from_survey(). The default is fg_default_na_strings(); character() leaves out only NA and empty values.
scale_template	The sentence that states an answer on a scale whose endpoints alone are labeled, for columns with value labels; see create_agents_from_survey(). Templates are named by column.
.runner	Optional function used instead of live model calls and stored on every agent.

Details

Values are rendered according to their column:

- A column with value labels (class haven_labelled, as haven::read_dta() and haven::read_sav() return them) is rendered by the rules described under Rendering answers in create_agents_from_survey(): a code is shown as its label; a number between the labeled endpoints of a scale is shown in the sentence of scale_template; a missing-data label and a negative code without a label are left out; and a field that holds numbers is shown as numbers. The same warnings are given, and the answers read as they do when the same data are read from a file by create_agents_from_survey(). SPSS user-defined missing values, which haven::read_sav() keeps with user_na = TRUE, are left out.
- Text and factor values are shown as given, except missing-data labels, decided by the rule that decides value labels (see Rendering answers in create_agents_from_survey()). A value is one when it matches na_strings and starts with a negative code or one of 90 and above ("-9. Refused", "[-9] Refused" as haven::as_factor() writes it with levels = "both", "97. Refused"), or when it says nothing but the matched wording ("Refused to answer", "I don't know", "Don't know (VOL)"). Answers that use such words ("I keep missing the bus", "My landlord refused to fix the heat", "2. Refused to take sides") are kept.
- Numbers are shown in full ("100000"); na_strings does not apply to them.

Every value left out is recorded in the rendering report with its reason, and those left out as missing-data labels or codes are announced in a message naming the columns. An NA is left out without a message.

A frame of class `silicon_panel`, or a frame with both `persona` and `persona_id` columns, is treated as a pre-rendered persona panel. Its `persona` text becomes each participant's direct persona description; `persona` and `persona_id` are not rendered as survey answers. This bridge uses the frame's structure and does not require LLMRpanel.

Sampling and the validation of `weights` follow `create_agents_from_survey()`.

Value

A named list of `FGAgent` objects (participants + moderator), keyed by agent ID, with the `source_rows` and `source_ids` attributes described in `create_agents_from_survey()`; here the positions refer to the rows of `data`. Unless the frame is a pre-rendered persona panel, the list also carries a `rendering_report` attribute recording, for each participant and column, the value shown or that it was missing; see `persona_rendering_report()`.

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
agents <- create_agents_from_data(
  anes_2024_personas, n_participants = 6, config = cfg
)
agents$P1$persona_description
```

`create_agents_from_survey` *Create agents from a labeled survey file*

Description

Reads a labeled survey file (Stata `.dta`, SPSS `.sav`, or SAS `.sas7bdat`) and turns each selected respondent into an `FGAgent` whose persona states that respondent's demographics and survey answers. Answers are rendered from the file's own value labels, so the same call works on ANES, GSS, WVS, or any other labeled file; nothing about a particular dataset is hard-coded.

Usage

```
create_agents_from_survey(
  n_participants,
  survey_path,
  config,
  demographic_vars = NULL,
  survey_vars = NULL,
  rows = NULL,
  weights = NULL,
  na_strings = fg_default_na_strings(),
  question_wording = NULL,
  value_labels = NULL,
```

```
  scale_template = .fg_default_scale_template,
  .runner = NULL
)
```

Arguments

n_participants	Integer number of participants (excludes the moderator).
survey_path	Path to the survey file (.dta, .sav, or .sas7bdat).
config	An explicit LLMR::llm_config for all agents.
demographic_vars	Variable names (codes) to render as demographics. May be a named vector, in which case the names are shown as the field labels. If NULL, common demographic variables are auto-detected by their labels.
survey_vars	Variable names (codes) to render as survey responses. May be named; a name is used as the question only for an item with neither a question_wording entry nor a label in the file. If NULL, labeled variables that are not demographics are used, except a case ID, which is left out with a message: a variable without value labels whose values are whole numbers or codes without spaces and differ in every row, in a file of at least 30 rows or under a name or label that reads as an identifier ("id", "case", "respondent", "serial"). A variable whose name identifies respondents (id, caseid, respondent_id, respid, persona_id, uid, record_id, row_id, ResponseId, participant_id, subject_id, user_id, PROLIFIC_PID, workerId, caseno, or QKEY, in any case and with or without spaces or punctuation, as in ID, CaseID, or resp_id) is never rendered.
rows	Optional row selector restricting the eligible respondents before sampling: integer positions of rows in the file, a logical vector with one value per row in the file, or a predicate function(df) -> logical over the demographics frame, whose columns hold the text rendered for each field. The selector refers to the file's own rows and is applied before respondents without any demographic value are left out; a message says how many selected respondents were left out for that reason. Positions must be whole numbers within the file. In a logical selector or a predicate's result, NA counts as not selected, as in subset(), so function(df) df$age != "18-24" leaves out respondents whose age is missing. A position given more than once selects its respondent once.
weights	Optional sampling weights: a column name in the file, or a numeric vector aligned to the file's rows. Used only to weight which respondents are drawn; the weight column is not rendered into the personas, even when demographic_vars or survey_vars name it. Weights must be nonnegative and finite, with at least one positive; numeric values carrying haven value labels are accepted.
na_strings	Character vector of value-label substrings, matched as regular expressions and ignoring case, that mark an answer as missing by the rule described under Rendering answers. The default covers common terms ("inapplicable", "missing", "refused", "don't know", "not asked", "legitimate skip", "no answer", "not applicable", "not available") and the labels the ANES 2024 release gives its negative missing-data codes, such as "-3. RESTRICTED", "-6. No post interview", "-5. Sufficient partial, breakoff", "-4. Error", "-2. DK/RF in V241288 or V241289", and "-2. Open-end responses available in separate file". "Error" and

"RESTRICTED" count only as a whole label. Pass your own vector, or extend `fg_default_na_strings()`, to match another file's missing-data vocabulary.

`question_wording`
: Optional named character vector giving the question as asked for survey items, keyed by variable name, as in `c(V241248 = "Where would you place yourself on abortion?")`. It takes precedence over the variable labels and the names of `survey_vars`; entries for variables not in `survey_vars` are ignored. An item without a wording is shown under its variable label (or, lacking one, its name), with a warning that lists such items: a variable label is often codebook shorthand ("PRE: 7pt scale abortion: self-placement") rather than the question.

`value_labels`
: Optional labels that replace or extend the file's value labels: a data frame with columns `variable`, `code`, and `label`, or a named list of named vectors in the form of haven value labels, as in `list(V241386 = c("Make all unauthorized immigrants felons and send them back to their home country" = 1))`. A supplied label replaces the file's label for its code, or labels a code the file leaves unlabeled; use it to restore labels the file cut short or to label stray codes. Supplied labels are matched against `na_strings` like the file's own. Entries for variables that are not requested are not used, with a warning.

`scale_template`
: The sentence that states an answer placed on a scale whose endpoints alone are labeled (see Rendering answers). The placeholders `{lo}` and `{hi}` are the endpoint codes, `{lo_label}` and `{hi_label}` their labels, and `{value}` the answer, which every template must contain; any other placeholder is an error. The default, "On a scale of {lo} to {hi}, where {lo} is {lo_label} and {hi} is {hi_label}, you place yourself at {value}.", fits self-placement items. Give one template for every item, or templates named by variable with at most one unnamed template as the default for the rest. Items that place someone else or rate a group need their own wording, as in `c(V241240 = "On a scale of {lo} to {hi}, where {lo} is {lo_label} and {hi} is {hi_label}, your placement of the Democratic candidate is {value}.")`. The template used is recorded in the rendering report.

`.runner`
: Optional function used instead of live model calls and stored on every agent.

Details

Each survey answer is shown under its question: the wording supplied in `question_wording`, else the variable's label in the file, else the name given to the variable in `survey_vars`. A label is codebook shorthand rather than the question asked, so items shown without a wording draw a warning. Demographic fields are shown under the names given in `demographic_vars` ("age: 25"), else their labels. The persona draws no inferences from the answers; it states them and lets the model interpret.

Respondents are drawn without replacement, so a participant is never a second copy of another. When the selection leaves more eligible respondents than `n_participants`, that many are sampled, with probability proportional to `weights` when supplied, and the draw needs at least `n_participants` respondents with a positive weight. When the selection leaves exactly `n_participants`, all of them are used, whatever their weights; when it leaves fewer, the call is refused. The option `focusgroup.seed` makes the draw reproducible without changing the caller's random-number state.

Invalid weights are refused rather than repaired. The call stops when the named weight column is not in the file, when the weights are not numeric (a factor, text, or logical values), when their length differs from the number of rows, when any weight is missing, negative, or not finite, when no eligible respondent has a positive weight, or when a draw has fewer respondents with a positive weight than participants to fill. The error names the column or the offending rows.

Value

A named list of `FGAgent` objects (participants + moderator), keyed by agent ID. Two attributes record which respondents were drawn. The `source_rows` attribute is an integer vector of their row positions in the file, named by participant ID. When the file has columns whose names identify respondents (as described for `survey_vars`), the `source_ids` attribute is a data frame of those columns for the drawn respondents, with participant IDs as row names. The moderator appears in neither. A third attribute, `rendering_report`, records for each participant and each requested field and item the text shown or the reason it was left out; read it with `persona_rendering_report()`. Subsetting the list drops the attributes, so read them from the list as returned.

Rendering answers

An answer in a variable with value labels is rendered by the first of the following rules that applies, and the rule used is recorded in the rendering report described under Value.

- A code with a value label is shown as its label, without a leading code of zero or above such as "1. ". A missing-data label, as described below, marks a missing value, and the answer is left out. Two kinds of label from the file are shown but flagged, with a warning naming the variable and code: a label exactly 80 characters long, since some releases, the ANES 2024 Stata file among them, cut longer labels at that length; and the label of a negative code that does not match `na_strings`, shown with its code, since the ANES and WVS releases use negative codes for missing data. Extend `na_strings` to leave the latter out.
- A number without a label that lies between the two labeled endpoints of a scale is shown on that scale, in the sentence given by `scale_template`: "On a scale of 1 to 7, where 1 is Government should provide many fewer services and 7 is Government should provide many more services, you place yourself at 5." The endpoints are the nearest labeled codes on either side of the answer, and they must be the only labels of the scale: no other label may lie below the lower endpoint, and labels above the upper endpoint must lie more than the scale's width beyond it. An off-scale code such as "99. Haven't thought much about this" after a 1 to 7 scale is therefore not an endpoint, while a labeled midpoint, as in a 0 to 10 scale labeled at 0, 5, and 10, rules out the scale form. Negative codes below a scale that starts at zero or above are taken for missing-data codes. Missing-data labels are never endpoints, a scale with a negative endpoint must be centered on zero, and top or bottom codes such as "89 or older" do not make a scale.
- A negative number without a label that is not a point on a scale is taken for a missing-data code, as negative codes are in the ANES and WVS releases, and left out, with a warning. Label it in `value_labels` if it is an answer.
- A number without a label in a variable that holds numbers is shown as the number, as in "age: 25". A variable holds numbers when the file labels at most one of its substantive codes of zero or above, such as the top code "80. Eighty years old or older", or labels only top and bottom codes.

- Any other code without a label is shown as the code, with a warning naming the variable and code. Supply its label in `value_labels`.

A variable without value labels is shown as stored, with numbers written out in full ("100000"). An empty value is left out, and so is a value that is a missing-data label.

One rule decides whether a value label, or a text value of a variable without value labels, is a missing-data label. It is one when it matches `na_strings` and either carries a missing-data code or says nothing but the matched wording. A missing-data code is a negative code or one of 90 and above: the labeled code of a value label, or else a code that leads the text, as in "-9. Refused", "-9 Refused", "[-8] Don't know", "(-8) Don't know", "97: Refused", and "98 - Don't know". A text says nothing but the matched wording when no word remains once its leading code, interviewer markers ("{VOL}", "(VOL)"), parenthetical and bracketed asides, the matched wording, and filler words such as "I", "to", "answer", and "in survey" are taken out: "Refused to answer", "I don't know", "Don't know/Refused (VOL.)", "Not asked in survey", and "9. Missing" are missing-data labels. Answers that use such words are kept, with a substantive code or without one: "2. Refused to take sides", "Missing the old system", "My landlord refused to fix the heat".

Answers left out as missing values are announced in a message. Requested variables that are not in the file are left out of every persona, and a variable requested more than once is rendered once, each with a warning. When two variables would be shown under the same text, each is shown with its variable name appended, with a warning. A variable label exactly 80 characters long shown as a question or field name is also named in a warning.

See Also

`create_agents_from_data()` for an in-memory data frame, `persona_rendering_report()` for the record of each persona, and anes_2024_personas for a ready-made example.

Examples

```
# A small labeled file stands in for a survey release; the call is the same
# against a full ANES, GSS, or WVS file.
toy <- data.frame(
  age = haven::labelled(c(1L, 2L, 3L, 2L),
                        c("18-24" = 1L, "25-34" = 2L, "35-44" = 3L)),
  pid = haven::labelled(c(1L, 2L, 2L, 1L),
                        c("Democrat" = 1L, "Republican" = 2L))
)
path <- file.path(tempdir(), "toy_survey.dta")
haven::write_dta(toy, path)

agents <- create_agents_from_survey(
  n_participants = 2,
  survey_path = path,
  config = LLMR::llm_config("groq", "openai/gpt-oss-20b",
                            api_key = LLMR::llm_api_key_env("GROQ_API_KEY")),
  demographic_vars = c(age = "age"),
  survey_vars = c("Party identification" = "pid"),
  rows = function(df) df$age != "18-24"
)
agents$P1$persona_description
```

create_conversation_flow *Create a Conversation Flow Object*

Description

Factory function to create an instance of a ConversationFlow subclass.

Usage

```
create_conversation_flow(mode, agents, moderator_id, flow_params = list())
```

Arguments

mode	Character. The type of turn-taking flow to create. Supported: "round_robin", "probabilistic", "desire_based".
agents	A named list of FGAgent objects.
moderator_id	Character. The ID of the moderator agent.
flow_params	List. Additional parameters specific to the chosen flow type. For "probabilistic": initial_propensities (named numeric vector), recovery_increment (numeric). For "desire_based": min_desire_threshold (numeric) and on_desire_failure, "error" (the default) or "random". Under "error" a desire score that cannot be obtained, after its retry, stops the session with an error of class focusgroup_desire_failure. Under "random" the speaker is chosen among the participants with valid scores by the usual rule, and uniformly among the eligible participants only when none has a valid score; the failure is recorded in the selected message's metadata$desire_scoring_failure. Earlier versions always chose uniformly at random when any score failed.

Value

An initialized ConversationFlow object (e.g., RoundRobinFlow, ProbabilisticFlow, DesireBasedFlow).

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
agents <- create_agents(
  n_participants = 2, config = cfg,
  direct_persona_descriptions = c("A retired teacher.", "A night-shift nurse."))
flow <- create_conversation_flow("desire_based", agents, "MOD",
                                 flow_params = list(min_desire_threshold = 4))
class(flow)
flow$participant_ids
```

`fg_after_item` *Points in a Session*

Description

A point names a moment in a session at which a message can be injected or a probe asked. `fg_after_item(k)` is the moment just after the moderator's line for item `k` of the guide, before any participant answers it. `fg_before_turn(t)` is the moment just before the `t`-th participant turn of the session, counting from 1 across all questions, before the speaker of that turn is chosen. A probe may also be asked at `"start"`, before the moderator's first line, or at `"end"`, after the last message, the closing line included, and before the final summary. A probe or an injection may be placed at `"before_closing"`, after the last discussion item and the call-ons that follow it and before the closing line; in a guide without a closing item it comes just before `"end"`.

Usage

```
fg_after_item(k)

fg_before_turn(t)
```

Arguments

`k`	One positive whole number, the guide item.
`t`	One positive whole number, the participant turn.

Details

Guide items are numbered in the order the session runs them: with `guide = c(Opening = 1, Icebreaker = 1, Engagement = 2, Exploration = 3, Closing = 1)`, item 4 is the second engagement question.

Participant turns count every reply, those that answer the moderator's call-on lines among them (see `call_on_silent` in `run_focus_group()`). When `fg_before_turn(t)` falls on the reply to a call-on line, it comes after that line. A custom turn-taking flow can end the discussion of an item by choosing no participant (`NULL`) or the moderator; that choice is not a turn and has no routing record. The point has then already come, so when call-ons follow, `fg_before_turn(t)` falls before the call-on line whose reply is turn `t`.

Value

An object of class `fg_point`.

See Also

`fg_injection()`, `fg_probe()`, `run_focus_group_arms()`.

Examples

```
fg_after_item(4)
fg_before_turn(7)
```

fg_default_na_strings *Missing-value patterns used by default for survey files*

Description

Returns the regular expressions that `create_agents_from_survey()` uses by default to recognize value labels marking an answer as missing, such as "Refused", "Don't know", "Inapplicable", and the labels the ANES and WVS give their negative missing-data codes ("-3. RESTRICTED", "-6. No post interview"). Patterns are matched ignoring case. Extend the vector to match another file's vocabulary.

Usage

```
fg_default_na_strings()
```

Value

A character vector of regular expressions.

See Also

`create_agents_from_survey()`

Examples

```
na <- c(fg_default_na_strings(), "prefer not to say")
head(na)
```

fg_injection *A Message Injected into a Session*

Description

Describes one message to be placed into a running session at a stated point, as an experimental manipulation: an objection, a correction, a piece of news. The message is logged like any other, with `metadata$injected = TRUE` and an `injection` record giving its label, arm, point, sender, visibility, and the `message_id` it follows.

Usage

```
fg_injection(
  text,
  at,
  from = "moderator",
  visibility = c("shared", "participants"),
  label = NULL
)
```

Arguments

text	One non-empty string, the message.
at	Where the message goes: fg_after_item(), fg_before_turn(), or "before_closing". When an injection and a probe share a point, the probe is asked first.
from	"moderator", a participant's id or handle, or the name of an outside speaker.
visibility	"shared" or "participants".
label	Optional name for the injection in the records; by default the arm's name in run_focus_group_arms(), and "injection_1", "injection_2", ... in run_focus_group().

Details

A message from = "moderator" is logged under the moderator's id, so the moderator reads it as its own line; so is a message from the moderator's id or handle. A message from a participant's id or handle (the handle in any case) is that participant's turn: at fg_before_turn(t) the participant says text as turn t, with no model call. It is logged under the participant's id with metadata$as_turn = TRUE, recorded in the routing record with the rule "injected", and counted as a participant turn, and it takes the place of the turn the session was about to give: the flow's pick, an addressed turn, or the reply to a call-on line, after which the moderator calls on the next silent participant. A line from someone other than the participant called on takes no one's place: the participant called on still answers next. The record's replaced names which turn was taken, or "none". Such a message is allowed only at fg_before_turn(), with visibility = "shared", and once per point. A from that is one participant's id and another's handle, as under permuted handles, could mean either and stops the run before any call. Any other from names an outside voice, such as "Guest expert", logged under that name and shown to the agents as written; it may not be "System", nor a name of the form "P4", which would read as a participant's handle. A moderator message that puts its last question to one participant by handle ("P2, what do you make of that?") gives that participant the next turn, except at a turn that answers the moderator's call-on line, which stays with the participant called on.

With visibility = "shared" the message enters the shared log and every agent sees it from then on. With visibility = "participants" it is kept out of the moderator's prompts and out of interim summaries, which the moderator also reads; the participants see it as part of the discussion.

Value

An object of class fg_injection.

See Also

run_focus_group(), run_focus_group_arms().

Examples

```
fg_injection(
  "One study found that extended hours cost more than they return.",
  at = fg_after_item(4), from = "Guest expert")
```

fg_player *Replay a finished session in the browser*

Description

Writes one self-contained HTML file that steps through a finished focus group from its saved records: a title page, one page per participant, then every logged message in order, with the record of each speaker selection before the turn it chose and each private answer where it was asked, and an end page. No model is called and nothing is reconstructed: a record the page needs but the session lacks stops the function with an error naming it.

Usage

```
fg_player(
  x,
  file = NULL,
  arm = NULL,
  annotations = NULL,
  mode = c("step", "static"),
  strip_speaker_prefix = FALSE,
  open = interactive()
)
```

Arguments

x	A focus_group_result from run_focus_group(), a focus_group_arms from run_focus_group_arms(), a FocusGroup object after run_simulation(), or the list that FocusGroup$run_arms() returns.
file	Path of the HTML file to write. NULL (the default) writes a temporary file. When x holds several arms and arm is NULL, one file is written per arm, each holding the shared messages and that arm's continuation; file then names a directory (a new temporary one when NULL), and the files are named <arm>.html.
arm	NULL, or the name of the one arm to replay.
annotations	Optional named list, by participant ID, of named character vectors shown on that participant's page and sidebar card, for example list(P1 = c(stake = "high")).
mode	"step" (the default) shows one step at a time; "static" shows every step at once, for printing and archives.
strip_speaker_prefix	FALSE (the default) shows every message as recorded. TRUE drops a speaker label that the model wrote at the start of its own line, such as "**MOD:**" in a moderator line or "P3:" in the reply of the participant whose handle is P3, never another speaker's label; a speaker's own label is its handle, not its ID. The title page says so and counts the lines changed.
open	Open the file with utils::browseURL(). The default opens it in an interactive session.

Details

The page reads these records: `transcript`, with each message's `metadata` (verbatim moderator lines, injected comments, call-ons, and addressed turns); `routing`, for the scores, exclusions, tie draws, and rule behind each participant turn; `probes`, placed after the message named by `after_message_id`; `participants`, for each persona as sent, without the standing rules; `summary`; and `metadata`. Results saved by FocusGroup 0.5.3 and earlier lack `routing` and `probes`, so the function stops for them. For results without the `handle` column of 0.5.5, each speaker's handle is its ID.

The steps, in order:

- a title page: topic and purpose, the number of participants, whether the moderator's lines were read as written or written by the model, the turn-taking rule, responses per question, model, date, package versions, and arm;
- one page per participant: ID and handle, the persona as sent, and any `annotations`, with that participant's sidebar card highlighted;
- one step per logged message: the roster line, moderator lines, injected comments marked with their source, and participant turns;
- before each participant turn, a panel with the selection record: under desire-based selection, the score of every participant asked, those not asked with the reason, the participant chosen, and whether a tie was drawn at random; under probabilistic selection, each participant's chance of being drawn; for a call-on or a participant the moderator addressed, that fact. Round robin gets no panel;
- each private question, as a panel with every answer, placed where it was asked; no agent sees these answers;
- an end page with the turns of every participant, zeros included, and the final summary, which the model wrote after the session.

In `"step"` mode the right arrow, space, Page Down, Enter, or a click shows the next step; the left arrow, Page Up, or Backspace goes back; Home returns to the first page; End shows every step; S shows or hides the selection panels; and F toggles full screen. A presentation clicker sends Page Down and Page Up. Adding `#step=12` to the address opens the page at step 12.

Each participant has one color, used in the sidebar, the turns, and the score bars: the Okabe-Ito palette for up to 7 participants and Paul Tol's muted scheme for 8 or 9, both readable with the common forms of color blindness. Beyond 9 the colors come from `grDevices::hcl.colors(n, "Dark 3")`, with a message, and are harder to tell apart. The moderator is gray.

The file needs no network connection: styles, script, and data are inline. Model text is inserted as text, never as markup; only `**bold**` and `*italics*` are rendered.

Value

The path of the file written, invisibly. For several arms, a character vector of paths named by arm.

See Also

`run_focus_group()` and `run_focus_group_arms()` for the records the page reads.

Examples

```
session <- readRDS(system.file("extdata", "player-session.rds",
                               package = "FocusGroup"))
page <- fg_player(session, file = tempfile(fileext = ".html"),
                  open = FALSE)
file.exists(page)

# Every step at once, with annotations on two participants' pages.
fg_player(session, file = tempfile(fileext = ".html"), mode = "static",
          annotations = list(P1 = c(recruited = "library newsletter"),
                             P2 = c(recruited = "hospital notice board")),
          open = FALSE)
```

---

fg_probe *A Private Question Put to Each Participant*

Description

Describes a question asked of each participant at stated points, whose answers are saved but never enter the shared log: no agent sees a probe or its answers afterwards, so asking it does not change the discussion. Asked before and after an injection, the same question measures a position on both sides of it; asked at `"start"` and `"end"`, it gives ratings before and after a session.

Usage

```
fg_probe(question, at, label = NULL, participants = NULL, scale = NULL)
```

Arguments

`question`	One non-empty string, the question.
`at`	One point, or a list of points, at which to ask it: `fg_after_item()`, `fg_before_turn()`, `"start"`, `"before_closing"`, or `"end"`. `"end"` comes after the closing line; `"before_closing"` comes before it.
`label`	Optional name for the probe in the records; by default `"probe_1"`, `"probe_2"`, ... in the order the probes are given.
`participants`	Optional character vector of participant ids to ask; `NULL` asks every participant.
`scale`	Optional two whole numbers, `c(lo, hi)` with `lo < hi`, the ends of the scale on which the question asks for a number. `NULL`, the default, records the answers as text only.

Details

Each participant answers with its persona, the standing rules, and the discussion so far, as for a turn, and the probe as the closing instruction, worded by the `participant_probe_instruction` template of `get_default_prompt_templates()`. The answer is the reply's final answer, without any reasoning. A reply that is empty or cut off is retried once with a larger budget; one still empty

is recorded as NA, with a warning. The answers are returned in the probes component of the result, with the raw reply beside each.

With scale = c(lo, hi), each answer is also read as a number, recorded in the value column. The reading is strict. Markup ("**4**"), a restatement of the scale ("1-7:", "on a scale of 1 to 7", "from 1 (strongly against) to 7 (strongly in favor)"), and a denominator equal to hi ("5/7", "6 out of 7") are set aside, as are digits joined to letters ("P3", "V241", "2024s") and, on a scale whose ends lie between -999 and 999, numbers of four or more digits, such as years. What remains must be one whole number from lo to hi. An answer with no number, with two different numbers ("3 or 4"), with a fraction of a point ("4.5"), with a number outside the scale ("-2"), or on another scale ("6/10" on a 1 to 7 scale) is not read, and neither are numbers written in words ("seven"); Arabic-Indic and Persian digits are read as digits. Nor is an answer that negates, compares, or reports someone else's number, whose number need not be the answer: one with "minus", "negative", "not", "never", "no", "except", "at least", "at most", "more than", "less than", "fewer than", "higher", "lower", "above", "below", "over", "under", or "between" (outside the scale's own restatement and the description of its ends), a number followed by "+" ("2+"), "others said", "they said", "everyone said", "someone said", or "P3 said", or a fraction written as one character, such as the half sign. Such an answer is followed by one re-ask in the same private exchange, worded by the participant_probe_reask template, asking for one number from lo to hi alone. An empty reply to the re-ask is not retried, and the earlier answer is kept. A value still unread is NA, and the participants concerned are named in one warning for the probe at that point. The column attempts counts the calls made for each answer, and every call, the re-ask included, is in the result's calls.

Value

An object of class fg_probe.

See Also

run_focus_group(), run_focus_group_arms().

Examples

```
fg_probe(paste("Where do you stand on extending library hours, from 1",
               "(strongly against) to 7 (strongly in favor)?",
               "Answer with the number first."),
         at = list("start", fg_after_item(4), "end"), label = "position",
         scale = c(1, 7))
```

---

fg_turns *Turn Table of a Discussion*

Description

One row per logged message, in order, with what the records say about how the speaker came to have the floor. No model is called. The System roster row is left out.

Usage

```
fg_turns(x)
```

Arguments

x	A focus_group_result, a focus_group_arms object, a FocusGroup, or the list that FocusGroup$run_arms() returns, as for summarize_focus_group().

Details

link is "addressed" when the moderator's preceding line named the participant, "called_on" when the moderator called on a participant who had been silent, and otherwise "sequence": the message simply follows the one before it, which says nothing about whom it answers. For the first two, linked_message_id is the moderator's line (the routing record's trigger_message_id); for "sequence", the previous message.

Value

A data frame with segment ("session", or "prefix" and the arm names; an arm's rows start after the branch point), message_id, item, round, phase, speaker_id, handle, is_moderator, injected, visibility ("shared" or "participants"), verbatim, words, rule (the routing rule that chose a participant turn), candidates and excluded (list-columns from the routing record), trigger_message_id, link, and linked_message_id.

Results made before guide items and handles were recorded take item from round (or, for an imported transcript without rounds, start a new item at each moderator message, with item 0 before the first) and handle from speaker_id.

See Also

summarize_focus_group().

Examples

```
transcript <- data.frame(
  speaker = c("Moderator", "Ana", "Ben", "Moderator", "Ana"),
  text = c("What should change?", "Longer evening hours.",
           "More weekend staff.", "Ana, why evenings?",
           "I work days.")
)
fg_turns(focus_group_from_transcript(transcript))
```

FGAgent *FGAgent Class*

Description

FGAgent Class

FGAgent Class

Details

Represents an agent (participant or moderator) in a focus group simulation. Each agent has a unique ID, a persona, an LLM configuration, and methods to generate utterances and express a desire to speak.

Value

An R6 generator. `FGAgent$new(id, agent_details, config, is_moderator = FALSE, .runner = NULL)` returns an `FGAgent`; `create_agents()` and its relatives construct whole rosters.

Customizing Agents

`FGAgent` is designed to be flexible:

- **Per-Agent LLM Configuration**: Each agent is initialized with its own `llm_config` (stored in the `config` field). This allows different agents to use different LLMs, temperatures, or even providers.
- **Persona Definition**: The `agent_details` (containing `demographics`, `survey_responses`, `direct_persona_description`, and/or `communication_style`) are used to construct the `persona_description` and `communication_style_instruction`.
- **Subclassing**: For advanced customization, inherit from `FGAgent` to override methods or add new fields.

Public fields

`id` Character. Unique identifier for the agent.

`persona_description` Character. Textual description of the agent's persona, derived from demographics, survey responses, or direct input.

`communication_style_instruction` Character. A specific instruction about the agent's communication style, to be included in prompts.

`config` An `llm_config` object (from the `LLMR` package) specifying the LLM provider, model, API key, and other parameters for this agent.

`.runner` `NULL` or a function used instead of live model calls. It receives a data frame with `config` and `messages` list-columns and returns those rows with at least `response_text`.

`is_moderator` Logical. `TRUE` if the agent is the moderator, `FALSE` otherwise.

`history` List. A log of utterances made by this agent during the simulation.

`tokens_sent_agent` Numeric. Tokens sent by this agent's calls, as reported; calls that reported no count add nothing.

`tokens_received_agent` Numeric. Tokens received by this agent's calls, as reported.

`tokens_total_agent` Numeric. Total tokens of this agent's calls: each call's reported total, or its sent and received counts when it reported no total.

`usage_unknown_agent` Numeric. Number of this agent's calls whose usage was incomplete, so that the three token fields are lower bounds when it is positive; `NA` when that number cannot be determined.

`role` Character. "moderator" or "participant" for convenience in reports.

`demographics` Named list. Raw demographics used to build persona.

`survey_responses` Named list. Raw survey responses used to build persona.

`handle` Character or `NULL`. The label under which this agent appears in the discussion and by which it recognizes its own turns, set by FocusGroup from its `handles`; `NULL` outside a session, when `id` is used.

Methods

Public methods:

- `FGAgent$new()`
- `FGAgent$generate_utterance()`
- `FGAgent$get_need_to_talk()`
- `FGAgent$answer_probe()`
- `FGAgent$clone()`

Method `new()`**:** Initialize a new FGAgent.

Usage:

```
FGAgent$new(id, agent_details, config, is_moderator = FALSE, .runner = NULL)
```

Arguments:

`id` Character. A unique identifier for the agent.

`agent_details` List. Contains information to build the agent's persona. Can include:

- `demographics`: A named list of demographic attributes (e.g., `list(age = 30, occupation = "teacher")`).
- `survey_responses`: A named list of survey questions and answers.
- `direct_persona_description`: A character string to be used directly as the persona. Overrides demographics/survey if provided.
- `communication_style`: A character string describing the agent's communication style (e.g., "analytical and direct", "empathetic and story-driven").

If `is_moderator` is `TRUE` and no specific details are provided, a default moderator persona is used.

`config` An `llm_config` object from `LLMR::llm_config()`. It may be `NULL` only for an analysis-only agent that will not generate output.

`is_moderator` Logical. `TRUE` if this agent is the moderator, `FALSE` otherwise.

`.runner` `NULL` uses live LLMR calls; a function is used instead of live model calls.

Method `generate_utterance()`**:** Generate an utterance for the agent.

Usage:

```
FGAgent$generate_utterance(
  topic,
  conversation_history_string,
  utterance_prompt_template,
  max_tokens_utterance = 150,
  current_moderator_question = "N/A",
  conversation_summary_so_far = "N/A",
  current_phase = "discussion",
  conversation_log = NULL,
  standing_rules = NULL,
  self_state = TRUE,
  call_params = NULL
)
```

Arguments:

`topic` Character. The current discussion topic.

`conversation_history_string` Character. Formatted string of recent conversation history.

`utterance_prompt_template` Character. The prompt template to use.

`max_tokens_utterance` Integer. Maximum tokens for the generated utterance.

`current_moderator_question` Character. The current question posed by the moderator.

`conversation_summary_so_far` Character. A summary of earlier parts of the conversation.

`current_phase` Character. The current phase of the focus group (e.g., "icebreaker", "exploration").

`conversation_log` List or NULL. The structured conversation log, with speakers under the labels this agent is to see. When supplied (and the template is not a legacy flat template), the message is built role-flipped: this agent's own prior turns, those under its `handle` (or, without one, its `id`), become `assistant` messages and others' become labeled `user` messages. `NULL` keeps the legacy flat single-user-message construction. In a session, the flat message of a participant opens with the sentence that tells it its handle ("In this discussion you are P3.").

`standing_rules` Character or NULL. The system-message standing rules (persona-anchoring, safety, etc.) for the role-flipped path; `NULL` uses a built-in default appropriate to the agent's role.

`self_state` Logical. If `TRUE` (default), a compact "points you have already made" digest of the agent's own prior turns is added to the system message (participants only) to discourage self-repetition.

`call_params` Optional named list of model parameters written over the agent's configuration for this utterance and its retry, such as `list(reasoning_effort = "low")`.

Returns: A list with `text` (the generated utterance) and `meta` (a list of call metadata: `response_id`, `finish_reason`, `sent_tokens`, `rec_tokens`, `total_tokens`, `usage_unknown`, `duration_s`, `provider`, `model`, and `metadata`). The token counts cover the first call and, when the reply was incomplete, its retry; each is `NA` unless every one of those calls reported it, and `usage_unknown` counts the calls whose usage was incomplete. The other fields describe the final reply. A label the agent wrote for itself at the start of the reply, such as `P3:` or `**MOD:**`, is removed from `text`, and `metadata` then holds `self_label_removed = TRUE`; another speaker's label is kept.

A short opening self-introduction ("As a retired nurse, ") is removed too, and `metadata` then holds `preamble_removed = TRUE`; a longer opening clause, or one that is part of the statement, is kept. The agent's token fields are updated even when the retry fails.

Method `get_need_to_talk()`**:** Get the agent's "desire to talk" score. This method queries the LLM to rate how strongly the agent feels the need to contribute to the discussion at the current moment.

Usage:

```
FGAgent$get_need_to_talk(
  topic,
  conversation_history_string,
  desire_prompt_template,
  max_tokens_desire = 20,
  current_moderator_question = "N/A",
  last_speaker_id = "N/A",
  last_utterance_text = "N/A",
  conversation_log = NULL,
  max_tokens_retry = NULL,
  call_params = NULL
)
```

Arguments:

`topic` Character. The current discussion topic.

`conversation_history_string` Character. Formatted string of recent conversation history.

`desire_prompt_template` Character. The prompt template to use for this query.

`max_tokens_desire` Integer. Maximum tokens for the LLM's response to the desire query.

`current_moderator_question` Character. The current question posed by the moderator.

`last_speaker_id` Character. The label of the agent who spoke last, as the agents see it.

`last_utterance_text` Character. The text of the last utterance.

`conversation_log` List or NULL. The structured conversation log, with speakers under the labels this agent is to see. When supplied (and the template is not a legacy flat template), desire scoring is role-flipped so the agent reads its own prior turns, those under its handle, as its own voice; `NULL` keeps the legacy flat construction.

`max_tokens_retry` Integer or NULL. The budget of the one retry made when the first reply is empty or cut off; `NULL` uses 256. No retry is made when `max_tokens_desire` is already at least this large.

`call_params` Optional named list of model parameters written over the agent's configuration for these calls, such as `list(reasoning_effort = "low")`.

Returns: Numeric. A score from 0 (no desire) to 10 (very strong desire), read from the reply's final answer only: reasoning the provider returns separately, or that the model writes inside `<think>` tags or the gpt-oss channel markup, is never read. A reply that cannot be read as a score is an error; the usage of the calls made, including a retry, is recorded on the agent first. The configuration's temperature, when it sets one, is used; none is added when it does not.

Method `answer_probe()`**:** Answer a private question put by the researcher. The agent reads its persona, the standing rules, and the discussion so far, as for a turn, and the question as the closing instruction. The answer is returned, not logged, and is not added to `history`.

Usage:

```
FGAgent$answer_probe(
  question,
  conversation_log,
  topic = NULL,
  template = "{{probe_question}}",
  max_tokens = NULL,
  conversation_summary = NULL,
  call_params = NULL,
  scale = NULL,
  reask_template = NULL
)
```

Arguments:

`question` Character. The question.

`conversation_log` List. The discussion so far, with speakers under the labels this agent is to see.

`topic` Character or NULL. The session topic, available to the template as `{{topic}}`.

`template` Character. The instruction template; `{{probe_question}}` is replaced by the question.

`max_tokens` Integer or NULL. The budget; `NULL` uses the configuration's limit, or 300 when it sets none.

`conversation_summary` Character or NULL. A summary of earlier discussion to add to the system message.

`call_params` Optional named list of model parameters written over the configuration for this call.

`scale` Optional two whole numbers, `c(lo, hi)`. When given, the answer is read as one number on that scale, and an answer from which none can be read is followed by one re-ask (see `fg_probe()`).

`reask_template` Character or NULL. The re-ask, with placeholders `{{lo}}`, `{{hi}}`, and `{{probe_question}}`; `NULL` uses `participant_probe_reask` of `get_default_prompt_templates()`.

Returns: A list with `answer` (the final answer, trimmed, or `NA` when the reply is empty after one retry with a larger budget), `raw_text` (the reply as returned), `finish_reason`, `call_id` (the call's row in the session's call record, or `NA` outside a session), `value` (the number read from the answer on `scale`, or `NA`), and `attempts` (the number of calls made). After a re-ask that returned text, `answer`, `raw_text`, `finish_reason`, and `call_id` describe the re-ask.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FGAgent$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
```

```
a <- FGAgent$new(
  id = "P1",
  agent_details = list(direct_persona_description = "A retired teacher."),
  config = cfg)
a$persona_description
```

`focus_group_from_transcript` *Build a FocusGroup object from a transcript or a session file*

Description

Turns a transcript that the package did not generate – a human focus group, a transcript exported from other software, or a saved CSV – or a session file the package wrote into a FocusGroup object whose `conversation_log` holds the turns. The returned object supports the analysis methods (`analyze()`, `analyze_participation_balance()`, `analyze_topics()`, `analyze_tfidf()`, `analyze_readability()`, `analyze_themes()`), the plotting methods, `analyze_focus_group()`, `fg_turns()`, and `summarize_focus_group()`, and it can be continued with `run_focus_group_arms()`. No model call is made: importing and the descriptive analyses run offline, and only the model-dependent methods (`analyze_themes()`, `summarize()`) contact a provider when you call them.

Usage

```
focus_group_from_transcript(
  data,
  speaker_col = "speaker",
  text_col = "text",
  topic = NULL,
  moderator_id = NULL,
  personas = NULL
)
```

Arguments

`data`	A data frame with one row per turn, in order; the path of a session file (ending in `.jsonl`); a character vector of its lines; a `focus_group_result`; or a FocusGroup. A result or a session object is read as the file `format.focus_group_result()` writes for it, so the object returned is always a new one, without model configurations or runners.
`speaker_col`	Character. Name of the column holding the speaker id. Default "speaker". For a data frame only.
`text_col`	Character. Name of the column holding the utterance text. Default "text". Missing text becomes an empty utterance, with a warning giving the number of such turns. For a data frame only.
`topic`	Character or `NULL`. The discussion topic, used by the model-dependent analyses and stored on the object. `NULL` keeps the topic of a session file and otherwise falls back to "Imported transcript".

`moderator_id` Character or `NULL`. Exact speaker id to treat as the moderator. When `NULL`, an `is_moderator` column or the exact aliases described above are used.

`personas` `NULL`, or a character vector of personas named by speaker id, which replace the placeholder or recorded persona of those speakers; a speaker a continued session lets speak needs one.

Details

One lightweight agent is created per speaker so that speaker-aware methods have a roster to report on. Unless `personas` or the file gives one, an agent carries a placeholder persona, and a session cannot be continued while a participant has it. Set `moderator_id` when the transcript identifies the moderator. When it is `NULL` and `data` has no `is_moderator` column, the moderator is the speaker whose id, ignoring case and surrounding spaces, is exactly one of "mod", "moderator", "facilitator", or "interviewer"; an id such as "Moderator 1" or "Modesto" does not match, and two matching speakers are an error. When no speaker is the moderator, a non-speaking "MOD" agent is added so the object is complete, a message says so, and every speaker is analyzed as a participant. Rows whose speaker is missing or blank are dropped with a warning; rows with speaker "System" are kept in the log but excluded from analyses, matching how the simulator treats its own roster message.

Columns that a transcript exported by this package carries are kept when `data` has them, so a saved transcript can be read back without losing its record: `message_id`, `round`, `item`, `is_moderator`, `phase`, `response_id`, `finish_reason`, `sent_tokens`, `rec_tokens`, `total_tokens`, `duration_s`, `provider`, `model`, `timestamp`, and the list-column `metadata`. A field that is absent takes these values: messages are numbered in order, `is_moderator` follows the moderator found as described above, `round` counts the moderator's turns (0 throughout when there is no moderator), `item` starts a new guide item at each moderator turn (0 before the first and for "System" rows), `phase` is `"imported"`, `metadata` is an empty list, and the other fields are `NA`. In particular, token counts that the transcript does not supply are unknown (`NA`), not zero, and importing adds no model calls to the object's usage counters.

The speakers keep their labels: the object's `handles` are the speaker ids, as with `handles = "ids"` in FocusGroup, and every message's `handle` is its speaker id. A `handle` column in `data` is not read.

Supplied fields are checked rather than coerced. `message_id` must hold distinct positive whole numbers increasing down the rows; `round`, `item`, and the token counts nonnegative whole numbers; `duration_s` nonnegative numbers; and `metadata` lists or `NULL`. Numbers stored as text are accepted, and `NA` is allowed except in `message_id`. A value that fails these checks is an error naming the column and its rows.

An `is_moderator` column must hold `TRUE` or `FALSE` in every row, as logical values or as the text "TRUE" and "FALSE"; it must give each speaker a single role and mark at most one speaker; it must agree with `moderator_id` when both are given. A column that marks no speaker means the transcript has no moderator, and the aliases are not consulted. "System" cannot be the moderator. If a participant is already named "MOD", the placeholder moderator is named "MOD_2", or the next free suffix.

Timestamps may be date-times, dates, numbers of seconds since 1970-01-01 UTC, or ISO 8601 text such as "2026-03-01", "2026-03-01 14:30:00", or "2026-03-01T14:30:00.5-05:00". Text without an offset is read as UTC, and all timestamps are stored in UTC. A timestamp that cannot be read becomes `NA`, with one warning giving the number of such values.

Value

A FocusGroup object with a populated `conversation_log`.

Session files

A session file has one JSON object per line, each with a `type`; the file `system.file("fg-session-spec.md", package = "FocusGroup")` describes the format for a person or a language model converting a transcript. `format.focus_group_result()` writes the file of a result, which this function reads back with every setting and record: the roster with personas, styles, and handles; the guide; the turn-taking rule and its parameters; the settings, such as the token limits, the call-on points, and the templates; the summary; and the routing, probe, injection, and call records. A result written and read back gives the same file when written again.

Only `message` lines with `speaker_id` and `text` are required. Every problem in a file (a line that is not JSON or not an object, an unknown line type or field, a value of the wrong type, a missing `text`, message ids that do not increase, two moderators, a speaker not declared when participants are, an item outside the guide, a negative token count, a file of one JSON value written over several lines) is reported at once, by an error of class `focusgroup_format_error` whose `problems` field is a data frame of `row` (the line), `field`, and `problem`. A field whose name starts with `x_` is one the format does not define: on a `session` or `item` line it is kept, without the prefix, in the metadata or the guide item, as `format.focus_group_result()` writes such fields; on other lines it is ignored. A byte-order mark and Windows line endings are accepted. A path that is not a `.jsonl` file, such as a CSV or RDS file, is refused: read it into R first.

A file without `participant` lines takes its speakers from the messages, with the moderator found as for a data frame. A message injected from an outside voice (`metadata$injected`) never makes its speaker a participant. A file that records no run, such as a converted human transcript, is read with these defaults: each moderator line becomes a guide item delivered as written, a `generic_discussion` item when a participant replies to it, a `transition` when none does, and the `closing` when it is the last line and unanswered; call-ons are off; the handles are the speaker ids; `max_participant_responses` is the longest run of participant turns between moderator lines; and the turn-taking rule is round robin, continuing from the last speaker. A moderator line that gives its `phase` gives its item that phase.

The record of a run made by FocusGroup 0.5.5 or earlier, saved without its session object, has no `item` lines and no `flow_params`. Its guide is rebuilt from the moderator's recorded lines, one item per guide line with the phase that line records, and every item is marked verbatim: a line that was delivered verbatim gives the item's text, while for a line the moderator's model generated the guide text is unknown, so a warning (of class `focusgroup_guide_rebuilt`) names those items and says that a continuation delivers their lines as recorded. The failure rule of `"desire_based"` is read from the session line's `on_desire_failure`.

See Also

`analyze_focus_group()` to run the descriptive analyses in one call; `run_focus_group()` to simulate a discussion instead.

Examples

```
transcript <- data.frame(
  speaker = c("Chair", "Ana", "Ben", "Ana", "Chair", "Ben"),
```

```
  text = c(
    "Welcome, everyone. What do you make of the new library hours?",
    "They help working parents like me; evenings matter most.",
    "I see it differently. Mornings are now far too crowded.",
    "Crowded mornings still beat being locked out after work.",
    "Ben, say more about what changed in the mornings.",
    "Staff are stretched thin before noon, so lines are longer."
  )
)
# "Chair" is not one of the moderator aliases, so name the moderator.
fg <- focus_group_from_transcript(transcript, topic = "library hours",
                                  moderator_id = "Chair")
fg$analyze_participation_balance()$participation_stats
```

`FocusGroup` *FocusGroup R6 Class*

Description

Main R6 class for managing and running a focus group simulation. It orchestrates the simulation through distinct phases (Opening, Icebreaker, Engagement, Exploration, Closing) guided by a question script, manages agent interactions via a turn-taking flow, and provides methods for analysis and visualization of the conversation.

Details

The group's token fields count every model call made on its behalf: the moderator's and participants' turns with their retries, desire scores, interim and final summaries, and model analyses run later, such as `analyze_themes()`. `run_focus_group()` describes how unreported usage is handled. A transcript imported with `focus_group_from_transcript()` adds no calls. Objects saved by FocusGroup 0.5.2 and earlier lack `total_tokens` and `calls_usage_unknown`.

A session run by `run_focus_group()` carries a `session_status` attribute, a list with `complete` and `failure` as in that function's `metadata`.

Every prompt shows the agents under their handles, never under their IDs: the participants' turns, the moderator's lines, desire scores, probes, and the interim summary that stands in for messages the prompt window no longer holds. The researcher's records, the log and the routing record among them, keep the IDs. Each participant's system message names its handle ("In this discussion you are P3.").

The moderator calls on silent participants at the guide items given by `call_on_silent`. When the discussion of such an item ends, each participant who has not spoken since the previous call-on point (or the start of the session) is asked, in roster order, by the line in `call_on_text`, delivered as written with no model call; the participant then answers, without a desire score, and the routing record gives the turn the rule `"called_on"`. With the default guide of `run_focus_group()` the points are items 2, 4, and 7. Krueger (2002) describes the opening question of a focus group as a round robin; the first point gives every participant a turn early in the session.

A moderator line whose last question opens or closes with one participant's handle ("P3, do you agree?", "P3: what do you think?", "Could you say more, P3?"), or is a short follow-up naming

one ("And P3?", "What about P3?", "How about P3?"), and does not also ask the group, gives that participant the next turn, with the rule `"addressed"`.

Value

An R6 generator. `FocusGroup$new()` returns a session object; most analyses run through `run_focus_group()`, which builds, runs, and returns one. Construction alone makes no model call.

Public fields

`topic` Character. The main topic of the focus group discussion.

`purpose` Character. The primary purpose or research objective of conducting the focus group.

`agents` Named list. A list of `FGAgent` objects participating in the simulation, indexed by their IDs.

`moderator_id` Character. The ID of the agent designated as the moderator.

`conversation_log` List. A chronological log of all messages. Each message is a list that includes at least `message_id`, `round`, `item`, `speaker_id`, `handle`, `is_moderator`, `text`, `timestamp`, `phase`, and `metadata`, along with call metadata: `response_id`, `finish_reason`, `sent_tokens`, `rec_tokens`, `total_tokens`, `duration_s`, `provider`, and `model`. `message_id` is unique message order; moderator and participant messages from one moderator cycle share `round`, and `item` is the guide item the message belongs to (0 for the roster row). `speaker_id` is the agent ID and `handle` the label the agents saw.

`handles` Named character vector. The label under which each agent appears to the other agents, named by agent ID; see the `handles` argument of `new()`.

`call_on_silent` List. The guide items at which the moderator calls on the participants who have not spoken, as `points` (the name of each point: `"opening"`, `"midpoint"`, `"end"`, or `"item:<k>"`) and `items` (the item numbers). Both are empty when call-ons are off.

`call_on_text` Character. The moderator's call-on line; `{participant}` is replaced by the participant's handle.

`turn_taking_flow` A `ConversationFlow` object dictating participant turn-taking.

`prompt_templates` List. Holds prompt templates for agent/moderator actions.

`question_script` List. A structured list defining phases and specific questions/actions for the moderator. Each element is a list with `phase` (e.g., "opening", "icebreaker_question") and optionally `text` (for specific questions).

`current_question_text` Character. Text of the current question being discussed.

`current_conversation_summary` Character. An LLM-generated summary of earlier parts of the conversation, used for managing context length in prompts.

`final_summary` Character. Final LLM-generated summary from the most recent simulation run.

`message_mode` Character. The message construction used by the most recent `run_simulation()` ("roleflip" or "flat"), recorded so a saved object can be replayed (e.g. in the GUI continuation experiment) with the same construction.

`admin_config` An `llm_config` object for group-level model tasks.

`max_tokens_utterance` Integer. Default max tokens for participant utterances.

`max_tokens_moderator` Integer. Default max tokens for moderator utterances.

`max_tokens_desire` Integer. Default max tokens for desire-to-talk queries.

`max_tokens_desire_retry` Integer. Max tokens for the one retry of a desire query whose reply was empty or cut off.

`max_tokens_summary` Integer. Max tokens for the final summary.

`max_tokens_interim_summary` Integer. Max tokens for an interim summary.

`max_tokens_probe` Integer or `NULL`. Max tokens for a probe answer; `NULL` uses the agent configuration's limit, or 300 when it sets none.

`reasoning_effort` `NULL`, one reasoning-effort setting for every call type, or a named list of settings by call type (`"moderator"`, `"participant"`, `"desire"`, `"summary"`, `"probe"`). A setting is written over the configuration's `reasoning_effort` for those calls; `NULL` leaves the configurations as they are.

`max_participant_responses` Integer. Maximum number of participant exchanges per round before the moderator can intervene. Can also be set globally via `options(focusgroup.max_participant_responses = N)`.

`total_tokens_sent` Numeric. Tokens sent across all LLM calls in the group, as reported; calls that reported no count add nothing.

`total_tokens_received` Numeric. Tokens received across all LLM calls, as reported.

`total_tokens` Numeric. Total tokens across all LLM calls: each call's reported total, or its sent and received counts when it reported no total.

`calls_usage_unknown` Numeric. Number of calls whose usage was incomplete, so that the three token fields are lower bounds when it is positive; `NA` when that number cannot be determined, as after a call that raised an error.

`records` List. The record of the most recent session: `calls`, one entry per model call the session made; `routing`, one entry per speaker selection; `probes` and `injections`, described in `run_focus_group()`; and `info`, the versions, standing rules, and settings in force when the session started. `run_focus_group()` returns the first four as data frames. Objects saved by FocusGroup 0.5.3 and earlier lack it.

Methods

Public methods:

- `FocusGroup$new()`
- `FocusGroup$run_simulation()`
- `FocusGroup$run_arms()`
- `FocusGroup$summarize()`
- `FocusGroup$analyze()`
- `FocusGroup$analyze_topics()`
- `FocusGroup$analyze_tfidf()`
- `FocusGroup$analyze_readability()`
- `FocusGroup$analyze_themes()`
- `FocusGroup$analyze_statistics()`
- `FocusGroup$analyze_participation_balance()`
- `FocusGroup$analyze_response_patterns()`

- FocusGroup$analyze_question_patterns()
- FocusGroup$analyze_key_phrases()
- FocusGroup$plot_participation_timeline()
- FocusGroup$plot_word_count_distribution()
- FocusGroup$plot_participation_by_agent()
- FocusGroup$plot_message_length_timeline()
- FocusGroup$clone()

Method `new()`**:** Initialize a new FocusGroup simulation.

Usage:

```
FocusGroup$new(
  topic,
  purpose,
  agents,
  moderator_id,
  turn_taking_flow,
  question_script = list(),
  prompt_templates = list(),
  admin_config = NULL,
  max_tokens_config = list(),
  max_participant_responses = NULL,
  reasoning_effort = NULL,
  handles = NULL,
  call_on_silent = c("opening", "midpoint", "end"),
  call_on_text = NULL
)
```

Arguments:

`topic` Character. The main discussion topic.

`purpose` Character. The primary purpose of the focus group.

`agents` Named list of initialized `FGAgent` objects, each named by its `id`.

`moderator_id` Character. The ID of the agent acting as moderator.

`turn_taking_flow` An initialized `ConversationFlow` object.

`question_script` List. Moderator's script defining phases and questions. If empty, a minimal default script (opening, generic discussion, closing) is used. An item with `verbatim = TRUE` is delivered as its `text` reads, with no model call; it must have a text.

`prompt_templates` List. Custom prompt templates. Defaults are used if not provided.

`admin_config` An `llm_config` object for administrative model tasks. If `NULL`, the moderator's explicitly supplied `config` is used.

`max_tokens_config` List. Optional. Named list with `utterance`, `moderator`, `desire`, `desire_retry`, `summary`, `interim_summary`, and `probe` to override the default max token limits (160, 400, 16, 256, 400, 250, and the configuration's own limit or 300) for these call types. A model that reasons before answering needs `desire` and `desire_retry` large enough to hold its reasoning and the score.

`max_participant_responses` Integer. Optional. Maximum participant exchanges per round before moderator intervention. Defaults to `getOption("focusgroup.max_participant_responses", 2L)`. It must be one positive whole number.

`reasoning_effort` Optional reasoning-effort setting, as described for the `reasoning_effort` field.

`handles` The labels under which the agents see each other. `NULL` (the default) shows the participants as `"P1"`, `"P2"`, ... in roster order and the moderator as `"Moderator"`, so the IDs given to the agents never reach a prompt; a participant whose ID is already `"P1"` to `"P99"` keeps it, and the others take the numbers that are no agent's ID, the moderator's included, so that a neutral handle never names a different agent's ID. `"ids"` shows the agent IDs, with a message listing them. A character vector named by agent ID sets handles of your own, such as invented first names, and the agents it does not name keep their neutral handles. A handle must be non-empty and distinct, cannot be `"System"` or the name of an injected voice, and cannot contain a standalone number from 0 to 10, which a desire score could be mistaken for. No participant's ID may be the moderator's handle (in any case), which would make the two indistinguishable.

`call_on_silent` Where the moderator calls on the participants who have not spoken since the previous such point: any of `"opening"` (the first discussion item), `"midpoint"` (discussion item `ceiling(D / 2)` of `D`), and `"end"` (the last discussion item), or guide item numbers, which count every item, as `fg_after_item()` does, and must be items participants answer. `TRUE` gives the three named points; `"none"`, `FALSE`, or `NULL` turns call-ons off. Points falling on the same item count once.

`call_on_text` The call-on line, delivered as written with no model call. It must contain `{participant}`, which is replaced by the participant's handle. `NULL` uses "{participant}, I'd like to hear from you as well. What are your thoughts?".

Method `run_simulation()`**:** Run the full focus group simulation. Iterates through the `question_script` phases or a specified number of rounds.

Usage:

```
FocusGroup$run_simulation(
  num_rounds = NULL,
  verbose = FALSE,
  injections = NULL,
  probes = NULL,
  final_summary = TRUE
)
```

Arguments:

`num_rounds` Integer. Optional. Maximum number of moderator cycles to run. If `NULL` (default), the simulation runs until the `question_script` is exhausted or the moderator decides to end. If both `num_rounds` and `question_script` are provided, the simulation stops at whichever condition is met first.

`verbose` Logical. If `TRUE`, prints progress and utterances to the console.

`injections` Optional `fg_injection()` or list of injections to place into the session at their points.

`probes` Optional `fg_probe()` or list of probes to ask at their points; the answers go to `records$probes`, never to the log.

`final_summary` Logical. `TRUE` (the default) ends the session with a model-written summary in `final_summary`; `FALSE` makes no summary call and leaves `final_summary` `NULL`.

Returns: Invisibly returns the `conversation_log`. When a call fails, the error propagates and the messages completed so far remain in `conversation_log`. A failed final summary is not an

error: it warns and leaves `final_summary` as `NA`. Every model call the session makes, and every speaker selection, is recorded in `records`. An injection or probe whose point the session never reaches is reported with a warning.

Method `run_arms()`**:** Run the session up to a branch point once, then continue a copy of it once per arm. `run_focus_group_arms()` describes the design and builds results from the objects returned here.

Usage:

```
FocusGroup$run_arms(
  arms,
  probes = NULL,
  num_rounds = NULL,
  verbose = FALSE,
  final_summary = TRUE
)
```

Arguments:

`arms` A named list of arms, each `NULL`, an `fg_injection()`, or a list of injections; the first injection of every arm that has one must name the same point.

`probes` Optional `fg_probe()` or list of probes.

`num_rounds` Integer. Optional maximum number of moderator cycles.

`verbose` Logical. Print progress.

`final_summary` Logical. Whether each arm ends with a final summary, as in `run_simulation()`.

Returns: A list with `prefix`, this object, stopped at the branch point, and `arms`, a named list of `FocusGroup` objects, each a copy of the prefix continued to the end of the session under one arm. A failure in an arm raises an error of class `focusgroup_branch_failure` whose fields `arm`, `branch` (the failed arm's object), `completed`, `phase`, and `parent` describe it. A session that ends before the branch point raises an error of class `focusgroup_injection_unreached`.

Method `summarize()`**:** Generate a summary of the conversation using an LLM.

Usage:

```
FocusGroup$summarize(
  config,
  summary_level = 1,
  max_tokens = NULL,
  internal_call = FALSE,
  transcript_override = NULL,
  .runner = NULL
)
```

Arguments:

`config` An explicit `llm_config` object for the summarization model.

`summary_level` Integer (1-3). 1: Prose overview, 2: Detailed bulleted, 3: Short bulleted takeaways.

`max_tokens` Integer. Optional. Max tokens for the summary.

`internal_call` Logical. Marks a summary made for context management during the session. It does not change the call: the call's usage is added to the group's token fields either way, and to no agent's. A session's call record lists such a summary as `"interim_summary"`.

`transcript_override` Character. Optional. If provided, this transcript is summarized instead of `self$conversation_log`.

`.runner` Optional function used instead of live model calls. It receives a data frame with `config` and `messages` list-columns and returns the rows with at least `response_text`. `NULL` makes a live call through `config`, also on a session that ran with a runner.

Returns: Character string containing the generated summary. The transcript made from the log shows each speaker under its handle, with the agent ID beside it when the two differ, and leaves out the roster row; such a summary is for the researcher. The interim summary the agents read is made from a `transcript_override` that shows handles only. A log or transcript with nothing to summarize is an error, as is an empty reply; its usage is recorded first.

Method `analyze()`**:** Basic analysis of the conversation log.

Usage:

```
FocusGroup$analyze(message_ids = NULL, speaker_ids = NULL)
```

Arguments:

`message_ids` Integer vector. Optional `message_id` values to analyze. If `NULL`, all messages are analyzed.

`speaker_ids` Character vector. Optional. Specific speakers to analyze. If `NULL`, analyzes all speakers.

Returns: A list with `speaker_stats` (a tibble: speaker_id, utterance_count, total_words, avg_words_per_utterance) and `full_transcript` (character string).

Method `analyze_topics()`**:** Perform LDA topic modeling on the conversation.

Usage:

```
FocusGroup$analyze_topics(
  num_topics = 5,
  min_doc_length = 20,
  top_n_terms = 10,
  message_ids = NULL,
  speaker_ids = NULL,
  seed = 110,
  unit = c("message", "speaker"),
  ...
)
```

Arguments:

`num_topics` Integer. Number of topics to identify.

`min_doc_length` Integer. Minimum words for a text to count as a document.

`top_n_terms` Integer. Number of top terms per topic to return.

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

`seed` Integer or NULL. Seed for the LDA control (default 110); `NULL` leaves it unset.

`unit` Character. What counts as a document: `"message"` (default) treats each utterance as a document; `"speaker"` pools every turn by the same speaker, which leaves as many documents as there are speakers and tends to recover who spoke rather than what was discussed.

`...` Additional arguments to `topicmodels::LDA()`.

Returns: A list with LDA model, topic terms, and document-topic proportions. NULL on failure.

Method `analyze_tfidf()`**:** Calculate TF-IDF scores for terms per participant.

Usage:

```
FocusGroup$analyze_tfidf(
  top_n_terms = 10,
  message_ids = NULL,
  speaker_ids = NULL,
  ...
)
```

Arguments:

`top_n_terms` Integer. Number of top TF-IDF terms per participant.

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

`...` Additional arguments to `tidytext::unnest_tokens`.

Returns: A tibble with TF-IDF scores.

Method `analyze_readability()`**:** Calculate readability scores for each participant's aggregated text.

Usage:

```
FocusGroup$analyze_readability(
  measures = "Flesch",
  message_ids = NULL,
  speaker_ids = NULL
)
```

Arguments:

`measures` Character vector. Readability measure(s) from `quanteda.textstats::textstat_readability`.

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

Returns: A tibble with readability scores.

Method `analyze_themes()`**:** Perform LLM-assisted thematic analysis on the transcript.

Usage:

```
FocusGroup$analyze_themes(
  config,
  message_ids = NULL,
  speaker_ids = NULL,
  .runner = NULL
)
```

Arguments:

`config` An explicit `llm_config` object for thematic analysis.

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

`.runner` Optional function used instead of live model calls. It receives a data frame with `config` and `messages` list-columns and returns the rows with at least `response_text`. `NULL` makes a live call through `config`, also on a session that ran with a runner.

Returns: The thematic summary as a character string. An empty transcript returns `character(0)`. Provider failures are propagated.

Method `analyze_statistics()`**:** Perform statistical analysis on conversation patterns.

Usage:

```
FocusGroup$analyze_statistics(message_ids = NULL, speaker_ids = NULL)
```

Arguments:

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

Returns: A list with ANOVA results, phase participation stats, and correlations.

Method `analyze_participation_balance()`**:** Analyze participation balance and dominance patterns.

Usage:

```
FocusGroup$analyze_participation_balance(
  message_ids = NULL,
  speaker_ids = NULL
)
```

Arguments:

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

Returns: A list with participation statistics and balance metrics. Every agent in the roster has a row, an agent who did not speak in the analyzed messages with zero messages and words and an `NA` average. The `called_on` column counts the turns that answered the moderator's call-on line.

Method `analyze_response_patterns()`**:** Analyze response patterns and interaction behaviors.

Usage:

```
FocusGroup$analyze_response_patterns(message_ids = NULL, speaker_ids = NULL)
```

Arguments:

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

Returns: A list with response and interaction pattern metrics.

Method `analyze_question_patterns()`**:** Analyze question asking patterns during the conversation.

Usage:

```
FocusGroup$analyze_question_patterns(message_ids = NULL, speaker_ids = NULL)
```

Arguments:

`message_ids` Integer vector. Optional `message_id` values to analyze.

speaker_ids Character vector. Optional. Specific speakers to analyze.

Returns: A list with question pattern analysis.

Method `analyze_key_phrases()`**:** Extract and analyze key phrases using n-grams.

Usage:

```
FocusGroup$analyze_key_phrases(
  min_freq = 2,
  message_ids = NULL,
  speaker_ids = NULL
)
```

Arguments:

`min_freq` Integer. Minimum frequency for phrases to be considered key.

`message_ids` Integer vector. Optional `message_id` values to analyze.

`speaker_ids` Character vector. Optional. Specific speakers to analyze.

Returns: A list with bigram and trigram analysis.

Method `plot_participation_timeline()`**:** Create a participation timeline of cumulative messages by participant across phases.

Usage:

```
FocusGroup$plot_participation_timeline()
```

Returns: ggplot object

Method `plot_word_count_distribution()`**:** Create word count distribution plot showing message length patterns.

Usage:

```
FocusGroup$plot_word_count_distribution()
```

Returns: ggplot object

Method `plot_participation_by_agent()`**:** Create a participation by agent plot showing total messages per participant.

Usage:

```
FocusGroup$plot_participation_by_agent()
```

Returns: ggplot object

Method `plot_message_length_timeline()`**:** Create a timeline showing message length over message order.

Usage:

```
FocusGroup$plot_message_length_timeline()
```

Returns: ggplot object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FocusGroup$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Krueger, R. A. (2002). *Designing and Conducting Focus Group Interviews*. University of Minnesota.

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
agents <- create_agents(
  n_participants = 2, config = cfg,
  direct_persona_descriptions = c("A retired teacher.",
                                  "A night-shift nurse."))
flow <- create_conversation_flow("round_robin", agents, "MOD")
fg <- FocusGroup$new(topic = "Neighborhood safety",
                     purpose = "Pilot the moderator guide.",
                     agents = agents,
                     moderator_id = "MOD",
                     turn_taking_flow = flow)
fg$topic
```

`format.focus_group_result` *Write a Session as JSON Lines*

Description

Returns the lines of a session file: one JSON object per line, each one row of a table of the result, so that `writeLines(format(result), "session.fg.jsonl")` saves the session and `focus_group_from_transcript()` reads it back.

Usage

```
## S3 method for class 'focus_group_result'
format(x, ...)

## S3 method for class 'FocusGroup'
format(x, ...)

## S3 method for class 'focus_group_arms'
format(x, ...)
```

Arguments

`x`	A `focus_group_result`, or a FocusGroup.
`...`	Unused.

Details

Each line has a `type`, and its other fields are the columns of the table it comes from:

`session` One line, first: the fields of `metadata` (topic, purpose, the turn-taking rule and its parameters, the settings, the status, the versions), with `summary` and `usage`, and `format` and `version`, which name the file format. `metadata$handles` and `metadata$guide` are written in the lines below. A field of `metadata` the format does not define, such as a researcher's `metadata$condition`, is written with the prefix `x_` (`"x_condition"`), and the reader restores it under its name.

`participant` One line per row of `participants`, the moderator's included.

`item` One line per guide item, in order: `item` (its number) and the item's `phase`, `text`, and `verbatim`, where set; any other field of the item, such as a `label`, with the prefix `x_`.

`message` One line per row of `transcript`.

`routing`, `probe`, `injection`, `call` One line per row of `routing`, `probes`, `injections`, and `calls`. A call line leaves out `messages`, the messages the call sent, which grow with the square of the session's length and are not needed to read or continue it.

The writing is canonical, so that a file read back and written again is the same file. A missing value is `null`. A whole number stored as a double is written with a decimal point (`160.0`) and an integer without one, and every double is written with as many digits as it needs to be read back exactly. A named vector, such as the desire `scores` of a pick, is an object; `candidates` and `tied` are always arrays. An empty list is `{}`. Times are written in UTC to the microsecond (`"2026-10-24T14:30:00.25Z"`). Every character outside ASCII is written as a `\uXXXX` escape, so the file is plain ASCII.

The file holds no model configuration and no runner, and so no key or endpoint from a configuration: `metadata` holds none, and the recorded call settings are the allowlisted model parameters. The message of a failed call is written as recorded, except that key-like values in it (the values of query parameters such as `key=`, bearer tokens, the values of `x-api-key:` and `api-key:` headers and of JSON fields such as `"api_key"`, `"apiKey"`, `"token"`, and `"secret"`, and strings of the forms of OpenAI, Groq, xAI, Hugging Face, and Google API keys) are replaced by `<redacted>`. `saveRDS()` of the result, by contrast, saves `focus_group`, which holds the configuration (and so any literal key or private endpoint in it) and the runner; set `result$focus_group <- NULL` first to save the record alone. `analyze_focus_group()` and the Studio rebuild the session object of such a result from its tables.

For a result saved by FocusGroup 0.5.5 or earlier, the guide, the flow parameters, and the communication styles, which it lacks, are read from the fields of its `focus_group`, when it has one; without one, `focus_group_from_transcript()` rebuilds them from the record as its help page describes.

`format()` of a FocusGroup, such as one read by `focus_group_from_transcript()`, gives the lines of its session file in the same way, from its log and records. A `focus_group_arms` object holds several sessions and has no one file: format an arm, `format(x$arms[["name"]])`, or the shared prefix, `format(x$prefix)`.

Value

A character vector of JSON lines, one per row, in the order session, participants, items, messages, routing, probes, injections, calls.

See Also

focus_group_from_transcript() to read the lines back, and vignette("architecture", package = "FocusGroup").

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
scripted <- function(experiments, ...) {
  experiments$response_text <- "Evening hours matter most to me."
  experiments
}
result <- run_focus_group("Library hours", cfg, n_participants = 2,
                          guide = list(Engagement = "Which hours matter?",
                                       Closing = "Close the session."),
                          flow = "round_robin", verbose = FALSE,
                          .runner = scripted)
path <- tempfile(fileext = ".fg.jsonl")
writeLines(format(result), path)
substr(readLines(path)[1:3], 1, 70)
fg <- focus_group_from_transcript(path)
fg$conversation_log[[3]]$text
```

---

get_default_prompt_templates *Get Default Prompt Templates*

Description

Retrieves a list containing the default prompt templates used by the FocusGroup package. These templates guide the behavior of LLM agents (participants and moderator) during different phases and actions within the simulation.

Usage

```
get_default_prompt_templates()
```

Details

Users can retrieve this list, modify specific templates, and then pass the modified list to FocusGroup$new() or higher-level wrapper functions to customize the simulation's prompts.

Value

A named list where each element is a character string representing a prompt template. Placeholders like \{\{topic\}\}, \{\{persona_description\}\}, etc., are used within the templates and will be filled dynamically during the simulation.

Participant templates and message construction

Two participant templates ship here, and which one is used depends on the message construction (see the message_mode argument of run_focus_group(), or options(focusgroup.message_mode=)).

- participant_turn_instruction is the canonical template for the default role-flipped construction. It carries only the current question and the turn cue, because the persona, standing rules, and the transcript are supplied structurally (a system message plus role-separated turns).
- participant_utterance_subtle_persona is the flat template. It inlines the persona and the whole transcript into one user message, which for a participant in a session opens with the sentence that tells it its handle ("In this discussion you are P3."). A custom template that contains \{\{conversation_history\}\} or \{\{persona_description\}\} is treated as a flat template and routed through the flat path.

Probe templates

participant_probe_instruction closes the messages of a private probe (fg_probe()); \{\{probe_question\}\} is replaced by the question. participant_probe_reask follows an answer to a probe with a scale from which no number could be read, in the same private exchange; its placeholders are \{\{lo\}\} and \{\{hi\}\}, the ends of the scale, and \{\{probe_question\}\}.

Summary templates

summary_participant, summary_item, summary_disagreements, summary_trajectory, summary_turn, and summary_session are the prompts of summarize_focus_group(), one per unit, and summary_reask is the message that returns a failing reply with its failures. Their placeholders are \{\{topic\}\}, \{\{unit\}\}, \{\{roster\}\} (speakers and message counts, computed in code), \{\{transcript\}\}, \{\{probes\}\} (empty unless probe answers are included), \{\{stages\}\} (trajectory only), \{\{format\}\} (the JSON shape the reply must take), and, in summary_reask, \{\{failures\}\}. They are read by summarize_focus_group() alone; changing them in a session's prompt_templates has no effect, and the templates argument of summarize_focus_group() replaces them.

Examples

```
default_prompts <- get_default_prompt_templates()
names(default_prompts)
default_prompts$participant_utterance_subtle_persona

my_prompts <- get_default_prompt_templates()
my_prompts$moderator_opening <- "Welcome all! Let us talk about {{topic}} today."

cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
agents <- create_agents(2, config = cfg,
                        direct_persona_descriptions = c("A retired teacher.",
                                                        "A night-shift nurse."))
fg <- FocusGroup$new(
  topic = "Library hours",
  purpose = "Pilot the revised opening.",
  agents = agents,
  moderator_id = "MOD",
```

```
  turn_taking_flow = create_conversation_flow("round_robin", agents, "MOD"),
  prompt_templates = my_prompts)
fg$prompt_templates$moderator_opening
```

`persona_rendering_report` *Report how each persona was rendered*

Description

Returns the record, kept with a roster, of how every requested demographic field and survey item was rendered into each participant's persona: the text shown, or the reason it was left out. Rosters built from respondent records by `create_agents_from_survey()`, `create_agents_from_data()`, and `create_agents()` carry the report; check it before a session runs.

Usage

```
persona_rendering_report(x)
```

Arguments

`x`	A roster returned by one of the constructors, a FocusGroup built from such a roster, or the result of `run_focus_group()`.

Details

A field is "rendered" when the persona states it as the source gives it. It is a "fallback" when the persona shows it in a form that needs checking: a code that has no value label, shown as the code; a label exactly 80 characters long, which some survey releases (the ANES 2024 Stata file among them) produce by cutting longer labels; or the label of a negative code that does not match `na_strings`, where the release may use negative codes for missing data; or a field of a supplied record that holds several values, of which the persona shows the first. It is "dropped" when the persona leaves it out, which happens when there is no answer to show: no value, a value whose label is a missing-data label or is empty, a text value that is a missing-data label (see Rendering answers in `create_agents_from_survey()`), a negative code without a label, an SPSS user-defined missing value, a record field that does not hold a single value, or a variable that is not in the file. An identifier is dropped too, with the reason "identifier column": a column whose name identifies respondents (`id`, `caseid`, `respondent_id`, `respid`, `persona_id`, `uid`, `record_id`, `row_id`, `ResponseId`, `participant_id`, `subject_id`, `user_id`, `PROLIFIC_PID`, `workerId`, `caseno`, or `QKEY`, in any case and with or without separators, as in `ID` or `CaseID`), or, among the items `create_agents_from_survey()` chooses when `survey_vars` is `NULL`, a variable that looks like a case ID, as described there. The agents would otherwise read the respondent's ID; the roster keeps it in `source_ids`.

Value

A data frame with one row per participant and rendered field or item, in the order requested. A variable requested more than once has one row, and a sampling-weight column, which is never rendered, has none. The columns are:

agent_id The participant ID.

kind `"demographic"` or `"item"`.

variable The variable in the file, or the column of the data.

question The text shown for it: the field label of a demographic, the question of an item.

question_source Where that text came from: `"wording"` (`question_wording`), `"name"` (a name given to the variable), `"dictionary"` (the data's dictionary), `"label"` (the variable label), or `"variable"` (the variable name itself).

code The stored value, as text; `NA` when there is none.

text The answer as it appears in the persona; `NA` when dropped.

rendered_as `"label"`, `"supplied_label"` (from `value_labels`), `"scale"`, `"value"`, or `"code"`; `NA` when dropped.

status `"rendered"`, `"fallback"`, or `"dropped"`.

reason Why a field is a fallback or was dropped; `NA` otherwise.

template For an answer shown on a scale, the `scale_template` used; `NA` otherwise.

The call stops when `x` carries no report, as for a roster built from supplied persona text or a list subset from a roster.

Examples

```
toy <- data.frame(
  age = c(25, 41, 80),
  abortion = haven::labelled(c(2, 7, -9), c(
    "1. Always permitted" = 1, "7. Never permitted" = 7, "-9. Refused" = -9))
)
path <- file.path(tempdir(), "toy_scale.dta")
haven::write_dta(toy, path)
agents <- create_agents_from_survey(
  n_participants = 3, survey_path = path,
  config = LLMR::llm_config("groq", "openai/gpt-oss-20b",
                            api_key = LLMR::llm_api_key_env("GROQ_API_KEY")),
  demographic_vars = c(age = "age"), survey_vars = "abortion",
  question_wording = c(abortion = "Where would you place yourself on abortion?")
)
report <- persona_rendering_report(agents)
report[, c("agent_id", "variable", "text", "status")]
```

run_focus_group *Run a Focus Group Simulation*

Description

Constructs the agents, moderator guide, and turn-taking flow, then runs one focus group session. `config` has no default; every agent and group-level call uses the one supplied. Before a live run, the function estimates the number of generated model outputs and applies the `max_calls` limit unless `confirm` is `TRUE`.

Usage

```
run_focus_group(
  topic,
  config,
  n_participants = 6,
 guide = c(Opening = 1, Icebreaker = 1, Engagement = 2, Exploration = 3, Closing = 1),
  demographics = NULL,
  survey_responses = NULL,
  flow = "desire_based",
  seed = NULL,
  message_mode = c("roleflip", "flat"),
  verbose = TRUE,
  max_participant_responses = NULL,
  max_calls = 120L,
  confirm = FALSE,
  max_tokens_config = NULL,
  reasoning_effort = NULL,
  on_desire_failure = c("error", "random"),
  injections = NULL,
  probes = NULL,
  moderator_verbatim = FALSE,
  call_on_silent = c("opening", "midpoint", "end"),
  call_on_text = NULL,
  handles = NULL,
  final_summary = TRUE,
  .runner = NULL
)
```

Arguments

`topic`	Character. Focus group topic.
`config`	An explicit `LLMR::llm_config` used by all agents and group-level model tasks.
`n_participants`	Integer. Number of participants, excluding the moderator.
`guide`	A named numeric vector or named list. The names are phases, among `"Opening"`, `"Icebreaker"`, `"Engagement"`, `"Exploration"`, and `"Closing"`, each at most

once and in that order. Numeric values select that many moderator instructions from the phase banks. Character vectors supply the ordered instructions directly. The default is a compact session (one opening, one icebreaker, two engagement and three exploration questions, one closing); with the default flow, participants, response limit, and call-ons it plans at most 111 model outputs, inside the default `max_calls` of 120. Larger guides multiply quickly: each question round costs one output per response plus, under `"desire_based"`, one scoring output per participant. `NULL` runs a minimal script: an opening, one open discussion of `topic`, and a closing.

`demographics` Optional participant demographics, one row per participant. With this, `survey_responses`, and personas all absent, the participants are drawn from the bundled ANES 2024 records.

`survey_responses` Optional participant survey answers, one row per participant.

`flow` Character. One of `"round_robin"`, `"probabilistic"`, or `"desire_based"`.

`seed` Optional whole number (a fraction such as 1.7 is an error) governing in-package sampling: the respondents drawn from the bundled records and the random choices of the turn-taking flow, such as the speaker draws of `"probabilistic"` and the tie-breaks of `"desire_based"`. The random waits of a provider's retries do not draw from this stream. The caller's random-number state is restored when the function returns. The option `focusgroup.seed`, when set, fixes the respondent draw instead.

`message_mode` Character. `"roleflip"` or `"flat"`.

`verbose` Logical. Print session progress.

`max_participant_responses` Optional integer maximum number of participant responses per moderator question. `NULL`, the default, uses the option `focusgroup.max_participant_responses`, which is 2 unless you set it.

`max_calls` Integer. Maximum estimated live model outputs allowed without confirmation; a finite whole number.

`confirm` Logical. Permit a live run whose estimate exceeds `max_calls`.

`max_tokens_config` Optional named list of token limits by call type, as for `FocusGroup$new()`: `utterance`, `moderator`, `desire`, `desire_retry`, `summary`, and `interim_summary`. A model that reasons before it answers needs `desire` and `desire_retry` large enough to hold its reasoning and the score.

`reasoning_effort` Optional reasoning-effort setting passed to the provider: one string, such as `"low"`, for every call, or a named vector by call type (`"moderator"`, `"participant"`, `"desire"`, `"summary"`, `"probe"`). It is written over the `reasoning_effort` of `config` for those calls; `NULL` leaves `config` as it is. LLMR forwards it on OpenAI-compatible providers.

`on_desire_failure` Character. Under `"desire_based"`, what happens when a participant's desire score cannot be obtained after its retry. `"error"` (the default) stops the session with an error of class `focusgroup_desire_failure`, returned as a partial

session as described in Details. "random" chooses the speaker among the participants with valid scores by the usual rule, and uniformly among the eligible participants only when none has a valid score; the failure is recorded in the selected message's metadata$desire_scoring_failure.

injections — Optional fg_injection() or list of injections, each placed into the session at its point. To compare sessions that share everything before an injection, use run_focus_group_arms().

probes — Optional fg_probe() or list of probes, private questions asked of each participant at stated points. The answers are returned in probes and never enter the discussion.

moderator_verbatim — FALSE (the default) has the moderator's model render each guide item as a line of its own. TRUE delivers every guide item exactly as written, with no model call, and a vector of item numbers delivers those items so. A verbatim message carries metadata$verbatim = TRUE. The opening and closing instructions of the numeric phase banks are instructions to the moderator, so verbatim delivery suits a guide written as the lines to be said.

call_on_silent — Where the moderator calls on the participants who have not spoken since the previous such point, as described in Details: any of "opening", "midpoint", and "end" (the default is all three), guide item numbers, or "none" (or FALSE) for no call-ons.

call_on_text — The call-on line. It must contain {participant}, which is replaced by the participant's handle; NULL uses "{participant}, I'd like to hear from you as well. What are your thoughts?".

handles — The labels under which the agents see each other: NULL (the default) for "P1", "P2", ... and "Moderator", "ids" for the agent IDs, or a character vector of handles named by agent ID, as for FocusGroup$new().

final_summary — Logical. TRUE (the default) ends the session with a model-written summary; FALSE makes no summary call and leaves summary empty.

.runner — Optional function used instead of live model calls. It receives a data frame with config and messages list-columns and returns those rows with at least response_text. Token columns such as sent_tokens, rec_tokens, and total_tokens are optional; when they are absent, usage is unknown.

Details

At the end of each call-on item's discussion the moderator calls on every participant who has not spoken since the previous call-on point, or since the start for the first, in roster order. "opening" is the first guide item participants answer, "midpoint" the middle one, and "end" the last; points that fall on the same item count once, and item numbers count every guide item, as fg_after_item() does. Each call-on line is delivered as written, with no model call, and carries metadata$call_on; the participant then answers, with a cue in the turn instruction and no desire score, and the reply carries metadata$called_on = TRUE. With the default guide the points are items 2, 4, and 7, so every participant speaks during the icebreaker. Krueger (2002) describes the opening question of a focus group as a round robin.

The participants and the moderator see each other under handles, never under the IDs of the records: the IDs of create_agents() are already "P1", "P2", ..., and the moderator appears as

"Moderator". When the moderator's last question opens or closes with one participant's handle ("P3, do you agree?", "P3: what do you think?", "Could you say more, P3?"), or is a short follow-up naming one ("And P3?", "What about P3?"), and does not also put it to the group, that participant takes the next turn, without a desire score and with a sentence in its turn instruction saying that the question was put to it, and the reply carries metadata$addressed = TRUE. A label a speaker writes for itself at the start of its reply (**MOD:**, P3:) is removed, and the message carries metadata$self_label_removed = TRUE. A short opening self-introduction ("As a teacher, ", "As a retired nurse, ") is removed too, and the message carries metadata$preamble_removed = TRUE; a longer opening clause, or one that belongs to the statement ("As a teacher I grade papers at night, ...", "As a result, ...", "As an alternative, ..."), is kept. In both cases the reply as returned stays in calls.

The estimate counts one output for each moderator turn the model writes, one for each participant response a question round allows, one for each participant at each call-on point, one for each probe answer (two when the probe has a scale, for a possible re-ask), and one for the final summary. Under "desire_based" it adds one desire score per participant before each response the flow chooses. With the default guide, six participants, and the default flow, two responses per question give an estimate of 111 outputs (93 without call-ons) and three give 153. A session can make fewer calls, since the desire-based flow does not score the previous speaker, an addressed turn needs no scores, and a call-on point calls on only the participants who have been silent, or more, since incomplete replies are retried and a long discussion triggers interim summaries. max_calls applies to the estimate, and only to live runs: with .runner the session runs whatever the estimate.

When a model call fails partway through the session (a provider error that persists after retries, an error from .runner, a reply still incomplete after its retry, or a failed interim summary), the turns already completed are not lost. The function raises an error of class focusgroup_incomplete_session whose partial field holds the focus_group_result built from those turns, with metadata$complete set to FALSE. The error's phase field names the script phase that was running, such as "engagement_question", or "interim_summary", and its parent field holds the original error. Errors raised before the session starts, such as an invalid argument or the max_calls check, carry no partial result. A failed final summary is not an error of this kind: the function warns and returns the complete transcript with an NA summary.

Token usage that a provider did not report is unknown, not zero. A call's total is its reported total, or the sum of its sent and received counts when both are known. The usage component adds the reported counts of every call FocusGroup makes, including retries of incomplete replies, desire scores, and interim summaries, and its unknown element counts the calls whose usage was incomplete. When unknown is positive the subtotals are lower bounds, and when it is NA, as after a call that raised an error, the number of such calls cannot be determined. Retries made inside LLMR::call_llm_robust() are not visible here and are not counted. A reported count that is not a nonnegative whole number is treated as unknown, with a warning.

In the transcript, a message's token count is NA unless every call behind the message reported it. The message's metadata element usage_unknown gives the number of those calls whose usage was incomplete.

Value

A focus_group_result, a list with these components:

focus_group The FocusGroup object that ran the session. It keeps the model configuration, so model analyses can be run on it later.

transcript A data frame with one row per logged message, starting with the "System" message that lists the roster. Its columns are message_id, round, item (the guide item the message belongs to, 0 for the roster row), speaker_id (the agent ID), handle (the label the agents saw), is_moderator, text, phase, response_id, finish_reason, sent_tokens, rec_tokens, total_tokens, duration_s, provider, model, timestamp (in UTC), and the list-column metadata, which marks, among others, delivered lines (verbatim), injected messages (injected, injection), call-on lines (call_on, a list of point, participant, and handle), the replies to them (called_on), replies to a moderator line that addressed the speaker (addressed), replies whose self-label was removed (self_label_removed), replies whose opening self-introduction was removed (preamble_removed), and injected participant turns (as_turn; see fg_injection()).

summary The final summary. It is NA when the final summary call failed, and character(0) in a partial result that stopped before it or in a session run with final_summary = FALSE.

participants A data frame with one row per agent: id, handle, role, is_moderator, persona, communication_style (the agent's style instruction), demographics, survey_responses, provider, and model. When the participants were drawn from the bundled ANES records, the rows drawn are recorded in attr(result$focus_group$agents, "source_rows").

usage A list of the token subtotals sent, rec, and total, and the count unknown described in Details. It records the session as run; model analyses run later add to the counters of focus_group but not to this list.

metadata A list with topic, purpose, flow, message_mode, n_participants, estimated_calls, provider, model, complete (TRUE for a finished session, FALSE for a partial one), and failure (NULL, or a list with the phase and message of the error that stopped the session); then the conditions of the run: max_participant_responses, seed (NA when none was given), on_desire_failure, reasoning_effort, max_tokens (the token limit of each call type), call_settings (a data frame giving, for each call type and agent, the provider, model, and model parameters the calls start from), prompt_templates, standing_rules (the system-message rules for participants and the moderator), disclosure, versions (of FocusGroup, LLMR, and R), started, and branch (NULL, or for a session run by run_focus_group_arms() its arm and branch point), handles (a data frame of agent_id and handle), call_on_silent (a list of the points and the guide items they fall on, both empty when call-ons are off), call_on_text, final_summary, guide (the guide items as the session ran them, each a list with phase and, when set, text and verbatim), and flow_params (the parameters of the turn-taking rule: initial_propensities and recovery_increment for "probabilistic", min_desire_threshold and on_desire_failure for "desire_based", none otherwise). It holds no configuration object.

calls A data frame with one row per model call the session made, in order, including retries and failed calls: call_id, call_type ("moderator", "participant", "desire", "summary", "interim_summary", or "probe"), agent_id, attempt (2 for a retry), round, phase, message_id (the message the call produced, if any), provider, model, the list-columns settings (the model parameters sent) and messages (the messages sent), raw_text (the reply before any cleaning), thinking (reasoning the provider returned separately), cleaned_text, score (for desire calls), finish_reason, response_id, sent_tokens, rec_tokens, total_tokens, reasoning_tokens, duration_s, error (the error a failed call raised), timestamp, and arm (the arm of a session continued by run_focus_group_arms(), otherwise NA). Only scalar sampling and length parameters enter settings; keys and endpoint addresses never do.

routing A data frame with one row per speaker selection: `pick`, `round`, `phase`, `turn` (the participant turn it chose for), `flow`, `rule` (such as `"rotation"`, `"propensity_draw"`, `"highest_score"`, `"threshold_lowered"`, `"highest_below_threshold"`, `"uniform_after_failure"`, or, for turns the flow did not choose, `"called_on"`, `"addressed"`, and `"injected"`), `selected_id`, `message_id`, `trigger_message_id` (for `"called_on"` and `"addressed"`, the moderator line that gave the turn; otherwise `NA`), `threshold`, `tie_break` (`"none"`, `"random"`, or `"draw"`), `scoring_mode`, and the list-columns `candidates`, `excluded` (named by participant, giving the reason, such as `"previous_speaker"` or `"score_failed"`), `scores` (desire scores, or selection probabilities for `"probabilistic"`), and `tied`, followed by `arm`.

probes A data frame with one row per probe answer: `probe` (its label), `point`, `agent_id`, `question`, `answer` (the final answer, or `NA` when none came), `value` (the number read from the answer for a probe with a `scale`, otherwise `NA`; see `fg_probe()`), `raw_text`, `finish_reason`, `attempts` (the calls made for the answer, retries and the re-ask included), `call_id` (the row in `calls` of the call that gave the answer), `round`, `phase`, `turn` (the next participant turn), `after_message_id` (the last message before it), and `arm`.

injections A data frame with one row per injection: `label`, `arm`, `point`, `delivered` (`FALSE` for one whose point the session never reached, which is also reported with a warning), `message_id`, `speaker_id`, `from`, `visibility`, `text`, `cut_after_message_id` (the message it follows), `round`, `phase`, `turn`, `as_turn` (`TRUE` for a participant's injected turn), and `replaced` (for such a turn, the turn it took the place of: `"flow"`, `"addressed"`, or `"called_on"`; `"none"` for a line from someone other than the participant called on, who still answers next).

Results saved by FocusGroup 0.5.2 and earlier lack `usage$unknown`, `metadata$complete`, and `metadata$failure`; results saved by 0.5.3 and earlier lack `calls`, `routing`, `probes`, `injections`, and the metadata that follows `failure`; results saved by 0.5.4 and earlier lack the `item` and `handle` columns, `trigger_message_id`, and the metadata that follows `branch`; results saved by 0.5.5 and earlier lack the `value` and `attempts` columns of `probes`, `communication_style` in `participants`, `as_turn` and `replaced` in `injections`, and `metadata$guide` and `metadata$flow_params`.

A result is saved whole with `saveRDS()`, and its `focus_group` then holds the model configuration, which may contain a literal API key or a private endpoint, and the runner. `format()` writes the record without them, one JSON object per line, for a file that `focus_group_from_transcript()` reads back; see `format.focus_group_result()`.

References

Krueger, R. A. (2002). *Designing and Conducting Focus Group Interviews*. University of Minnesota.

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
guide <- list(
  Opening = "Welcome the participants and state the ground rules.",
  Exploration = "Which funding priority deserves attention first?",
  Closing = "Thank the participants and close the session."
)

scripted <- function(experiments, ...) {
```

```
  experiments$response_text <- "Evening opening hours matter most to me."
  experiments
}
result <- run_focus_group(
  topic = "Library funding priorities",
  config = cfg,
  n_participants = 2,
  guide = guide,
  flow = "round_robin",
  verbose = FALSE,
  .runner = scripted
)
result$transcript[, c("speaker_id", "phase", "text")]

# The scripted runner reports no token counts, so usage is unknown.
result$usage

# A session that fails partway returns what it completed.
failing <- local({
  calls <- 0
  function(experiments, ...) {
    calls <<- calls + 1
    if (calls == 3) stop("The provider is unavailable.")
    scripted(experiments)
  }
})
partial <- tryCatch(
  run_focus_group("Library funding priorities", cfg, n_participants = 2,
                  guide = guide, flow = "round_robin", verbose = FALSE,
                  .runner = failing),
  focusgroup_incomplete_session = function(e) e$partial
)
partial$metadata[c("complete", "failure")]
```

`run_focus_group_arms` *Run Arms of a Focus Group from One Shared Prefix*

Description

Runs a session up to an injection point once, then continues the full remainder of the session (every later question and turn, and the final summary) once per arm, so that every arm shares each message before the injection. Comparing the arms then compares sessions that differ only in what was injected and in what followed from it.

Usage

```
run_focus_group_arms(
  topic,
  config,
  arms,
```

```
  n_participants = 6,
 guide = c(Opening = 1, Icebreaker = 1, Engagement = 2, Exploration = 3, Closing = 1),
  demographics = NULL,
  survey_responses = NULL,
  flow = "desire_based",
  seed = NULL,
  message_mode = c("roleflip", "flat"),
  verbose = TRUE,
  max_participant_responses = NULL,
  max_calls = 120L,
  confirm = FALSE,
  max_tokens_config = NULL,
  reasoning_effort = NULL,
  on_desire_failure = c("error", "random"),
  probes = NULL,
  moderator_verbatim = FALSE,
  call_on_silent = c("opening", "midpoint", "end"),
  call_on_text = NULL,
  handles = NULL,
  final_summary = TRUE,
  .runner = NULL
)
```

Arguments

topic	Character. Focus group topic. Or a recorded session to continue, a FocusGroup or a focus_group_result; see "Continuing a recorded session".
config	An explicit LLMR::llm_config used by all agents and group-level model tasks.
arms	A named list of at least two arms, as described in Details.
n_participants	Integer. Number of participants, excluding the moderator.
guide	A named numeric vector or named list. The names are phases, among "Opening", "Icebreaker", "Engagement", "Exploration", and "Closing", each at most once and in that order. Numeric values select that many moderator instructions from the phase banks. Character vectors supply the ordered instructions directly. The default is a compact session (one opening, one icebreaker, two engagement and three exploration questions, one closing); with the default flow, participants, response limit, and call-ons it plans at most 111 model outputs, inside the default max_calls of 120. Larger guides multiply quickly: each question round costs one output per response plus, under "desire_based", one scoring output per participant. NULL runs a minimal script: an opening, one open discussion of topic, and a closing.
demographics	Optional participant demographics, one row per participant. With this, survey_responses, and personas all absent, the participants are drawn from the bundled ANES 2024 records.
survey_responses	Optional participant survey answers, one row per participant.
flow	Character. One of "round_robin", "probabilistic", or "desire_based".

`seed` Optional whole number (a fraction such as 1.7 is an error) governing in-package sampling: the respondents drawn from the bundled records and the random choices of the turn-taking flow, such as the speaker draws of `"probabilistic"` and the tie-breaks of `"desire_based"`. The random waits of a provider's retries do not draw from this stream. The caller's random-number state is restored when the function returns. The option `focusgroup.seed`, when set, fixes the respondent draw instead.

`message_mode` Character. `"roleflip"` or `"flat"`.

`verbose` Logical. Print session progress.

`max_participant_responses`
Optional integer maximum number of participant responses per moderator question. `NULL`, the default, uses the option `focusgroup.max_participant_responses`, which is 2 unless you set it.

`max_calls` Integer. Maximum estimated live model outputs allowed without confirmation; a finite whole number.

`confirm` Logical. Permit a live run whose estimate exceeds `max_calls`.

`max_tokens_config`
Optional named list of token limits by call type, as for `FocusGroup$new()`: `utterance`, `moderator`, `desire`, `desire_retry`, `summary`, and `interim_summary`. A model that reasons before it answers needs `desire` and `desire_retry` large enough to hold its reasoning and the score.

`reasoning_effort`
Optional reasoning-effort setting passed to the provider: one string, such as `"low"`, for every call, or a named vector by call type (`"moderator"`, `"participant"`, `"desire"`, `"summary"`, `"probe"`). It is written over the `reasoning_effort` of `config` for those calls; `NULL` leaves `config` as it is. LLMR forwards it on OpenAI-compatible providers.

`on_desire_failure`
Character. Under `"desire_based"`, what happens when a participant's desire score cannot be obtained after its retry. `"error"` (the default) stops the session with an error of class `focusgroup_desire_failure`, returned as a partial session as described in Details. `"random"` chooses the speaker among the participants with valid scores by the usual rule, and uniformly among the eligible participants only when none has a valid score; the failure is recorded in the selected message's `metadata$desire_scoring_failure`.

`probes` Optional `fg_probe()` or list of probes. A probe at or before the branch point is asked once, in the prefix; a later one is asked in every arm.

`moderator_verbatim`
`FALSE` (the default) has the moderator's model render each guide item as a line of its own. `TRUE` delivers every guide item exactly as written, with no model call, and a vector of item numbers delivers those items so. A verbatim message carries `metadata$verbatim = TRUE`. The opening and closing instructions of the numeric phase banks are instructions to the moderator, so verbatim delivery suits a guide written as the lines to be said.

`call_on_silent` Where the moderator calls on the participants who have not spoken since the previous such point, as described in Details: any of `"opening"`, `"midpoint"`,

and "end" (the default is all three), guide item numbers, or "none" (or FALSE) for no call-ons.

call_on_text — The call-on line. It must contain {participant}, which is replaced by the participant's handle; NULL uses "{participant}, I'd like to hear from you as well. What are your thoughts?".

handles — The labels under which the agents see each other: NULL (the default) for "P1", "P2", ... and "Moderator", "ids" for the agent IDs, or a character vector of handles named by agent ID, as for FocusGroup$new().

final_summary — Logical. TRUE (the default) ends the session with a model-written summary; FALSE makes no summary call and leaves summary empty.

.runner — Optional function used instead of live model calls. It receives a data frame with config and messages list-columns and returns those rows with at least response_text. Token columns such as sent_tokens, rec_tokens, and total_tokens are optional; when they are absent, usage is unknown.

Details

Each arm is NULL, which continues the prefix with nothing injected, one fg_injection(), or a list of injections. The first injection of every arm that has one must name the same point; that point is where the session branches. An arm's later injections may name later points. A neutral injection at the branch point, rather than NULL, gives a control arm that also receives a message there.

The prefix is run once, with its probes, and a copy of the whole session (agents, turn-taking state, records, and token counters) is taken at the branch point; each arm continues its own copy. Each arm also starts from the random-number state at the branch point, so the random draws that follow, such as tie-breaks and probabilistic speaker choices, use the same stream in every arm; a provider's retries, which wait a random time, do not draw from it. With seed, the whole run is reproducible; without it, the arms still share the stream from the branch point on.

A branch point must lie within the guide: fg_after_item(k) needs k no larger than the number of guide items, and fg_before_turn(t) needs t no larger than the number of participant turns the guide allows. A point the session still does not reach, as when a custom flow chooses no speaker, stops the run with an error of class focusgroup_injection_unreached.

The estimate checked against max_calls is that of one session times the number of arms, an upper bound, since the prefix is paid once.

Value

A focus_group_arms object, a list with:

arms A named list of focus_group_result objects, one per arm, each a complete session as returned by run_focus_group(). Its transcript, calls, and records begin with the shared prefix; its usage counts the whole session, prefix included; and its metadata$branch records the arm, the branch point, the number of prefix messages and calls, and prefix_hash, a hash of the prefix transcript that is the same in every arm.

prefix The shared prefix as a focus_group_result with metadata$complete set to FALSE and no summary.

branch_point The branch point, as written, such as "fg_after_item(4)".

injections A data frame of every arm's injections, as in `run_focus_group()`.

probes A data frame of every probe answer: those asked in the prefix once, with `arm` set to `NA`, and those asked in each arm.

usage The token usage of all the calls made: the prefix once and each arm's remainder. Adding up the arms' own `usage` would count the prefix once per arm.

metadata A list with `topic`, `branch_point`, `arms` (the arm names), `estimated_calls`, `seed`, `prefix_messages`, `prefix_hash`, `complete`, and `failure`.

When a call fails, the run stops with an error of class `focusgroup_incomplete_arms` whose `partial` field holds the `focus_group_arms` object built from what was completed, with the arm that failed, if any, as a partial session; its `arm` field names that arm and its `parent` field holds the original error.

Continuing a recorded session

When `topic` is a recorded session, a `FocusGroup` or a `focus_group_result` (for example one read from a session file by `focus_group_from_transcript()`), no prefix is run: each arm continues the record from a participant turn. The branch point must be `fg_before_turn(t)`, where turn `t` is the `t`-th message by a participant. The messages before it are the shared prefix, taken from the record as they are, and the record's turn `t` and everything after it are replaced by each arm's continuation. An arm `fg_injection(text, at = fg_before_turn(t), from = "P2")` has P2 say `text` as turn `t` (see `fg_injection()`); a `NULL` arm regenerates turn `t` and the rest; an outside voice or a moderator line can also be injected there.

The session is rebuilt as it stood just before turn `t`: the guide item and its stage, the turns already given on it, the call-on queue, a participant the moderator's last line addressed, the question in force, the interim summary in force, and the turn-taking rule's state, with the record's personas, handles, guide, settings, templates, standing rules, and message construction. `config` and `.runner` are given to every agent; `reasoning_effort` and `max_tokens_config`, when given, replace the record's. The arguments that the record fixes (`n_participants`, `guide`, `demographics`, `survey_responses`, `flow`, `message_mode`, `max_participant_responses`, `on_desire_failure`, `moderator_verbatim`, `call_on_silent`, `call_on_text`, and `handles`) must be left out. A probe at the branch point itself is asked once, on the rebuilt session, before the arms, so that every arm shares its answers, as a probe there is asked in the prefix of a new session. A probe or an injection at a point the record has already passed is an error, and so is a transcript without moderator lines, which has no guide item to continue, as are a custom turn-taking flow, whose state cannot be rebuilt, a record without a topic, and a participant who has only the placeholder persona of an imported transcript; give personas with `focus_group_from_transcript(x, personas = ...)`. Everything is checked before any call. Under round robin with a model that replies the same way to the same messages, a `NULL` arm repeats the record. Each arm's `calls` and `usage` hold the record's calls before the cut and the arm's own after it. With `seed`, the caller's random-number state is restored afterwards.

A continuation of a human transcript is a model's continuation of a human discussion: the later moderator lines are the recorded ones, delivered as written, and the participants' turns are generated from the personas given. Arms are comparable with one another, not with the people who took part.

See Also

`fg_injection()`, `fg_probe()`, `run_focus_group()`.

Examples

```
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
scripted <- function(experiments, ...) {
  experiments$response_text <- "Evening hours matter most to people who work."
  experiments
}
guide <- list(Opening = "Welcome the participants.",
              Engagement = c("Which hours matter most?",
                             "Who is left out by the current hours?"),
              Exploration = "What would you change first?",
              Closing = "Thank the participants and close.")
arms <- list(
  objection = fg_injection(
    "Extended hours would cost the branch two weekday programs.",
    at = fg_after_item(3), from = "Guest expert"),
  control = NULL)
out <- run_focus_group_arms(
  "Library hours", cfg, arms = arms, n_participants = 2, guide = guide,
  flow = "round_robin", verbose = FALSE,
  probes = fg_probe("From 1 to 7, how strongly do you favor longer hours?",
                    at = list(fg_after_item(3), "end")),
  .runner = scripted)
out
out$probes[, c("arm", "point", "agent_id", "answer")]

# Continue a recorded session from its third participant turn: as recorded,
# and with P2 saying something else there.
session <- run_focus_group("Library hours", cfg, n_participants = 2,
                           guide = guide, flow = "round_robin",
                           verbose = FALSE, .runner = scripted)
path <- tempfile(fileext = ".fg.jsonl")
writeLines(format(session), path)
again <- run_focus_group_arms(
  focus_group_from_transcript(path), cfg,
  arms = list(regenerated = NULL,
              changed = fg_injection("Mornings matter more to me.",
                                     at = fg_before_turn(3), from = "P2")),
  verbose = FALSE, .runner = scripted)
again$arms$changed$transcript[, c("speaker_id", "text")]
```

`run_focus_studio` *Launch the FocusGroup Shiny GUI*

Description

A point-and-click front end with three tabs. Run a focus group starts a fresh moderated session live from a topic and a handful of participants, shows the transcript, and offers a complete `.rds` session and transcript CSV. Analyze accepts a finished run, saved session, CSV, or bundled example and

runs the package's descriptive analyses offline; thematic analysis and a model summary are optional live actions. The continuation experiment takes a saved focus group up to a message and compares repeated next-message responses under one control history and several labeled treatment histories. Running and the continuation experiment generate text and need an API key; analysis works offline.

Usage

```
run_focus_studio(...)
```

Arguments

`...`	Passed to `shiny::runApp()`.

Details

The GUI is optional. It needs the suggested packages shiny, bslib, DT, and LLMR.shiny; install them first. Keys are read from environment variables only, never pasted into the app.

The Studio needs LLMR.shiny 0.1.3 or later and stops with an error when an older version is installed.

The usage tile in the sidebar counts the model calls each action made and the tokens they reported, including the calls an action made before it failed. When some calls reported no usage, the token counts are shown as lower bounds. If usage cannot be recorded, a notification says so, and the action's results remain available.

A run that fails partway keeps the turns it completed: the partial transcript can be analyzed and downloaded, and a notice names the phase in which the run stopped. A downloaded session keeps its completion status and the rows its participants were drawn from, but no model configuration or runner function, since a configuration can hold a literal API key.

A continuation from a session saved by FocusGroup 0.5.2 or earlier rebuilds the saved `FGAgent` agents with the current class and their saved persona, style, and records, so that their prompts are unchanged and their usage is recorded as described in `run_focus_group()`; agents of other classes are used as saved.

Value

Invisibly, the value of `shiny::runApp()`; called for the side effect of starting the app.

Examples

```
if (interactive() &&
    all(vapply(c("shiny", "bslib", "DT", "LLMR.shiny"),
               requireNamespace, logical(1), quietly = TRUE))) {
  run_focus_studio()
}
```

summarize_focus_group *Summarize a Discussion by Participant, Item, Disagreement, Trajectory, Turn, or Session*

Description

Summarizes a finished or imported discussion with a model, in one of six units and at one or more lengths. The model is asked for claims as JSON: each claim states one point, its status, the messages that support it, and any words quoted from them. Every claim is checked against the transcript before it is used, and the prose of the summary is rendered from the claims that pass. No message is added to the session and no agent sees the summary.

Usage

```
summarize_focus_group(
  x,
  config,
  by = "participant",
  compression = "brief",
  include_probes = FALSE,
  on_fail = c("error", "drop"),
  max_calls = 30L,
  confirm = FALSE,
  max_tokens = NULL,
  reasoning_effort = NULL,
  templates = NULL,
  verbose = FALSE,
  .runner = NULL
)
```

Arguments

x	A focus_group_result from run_focus_group(), a focus_group_arms object from run_focus_group_arms(), a FocusGroup, including one made by focus_group_from_transcript(), or the list that FocusGroup$run_arms() returns (or its named list of arms), summarized as a focus_group_arms object is.
config	An explicit llm_config object for the summarizing model. The session's own models and runner are never used.
by	The unit: one of "participant", "item", "disagreements", "trajectory", "turn", or "session".
compression	One or more of "headline", "brief", and "detailed".
include_probes	Logical. Send private probe answers to the model as a separate, labeled block.
on_fail	"error" or "drop"; see Checks.
max_calls	Maximum estimated model calls allowed without confirm.

`confirm` Logical. Permit a run whose estimate exceeds `max_calls`.

`max_tokens` Optional token limit for each reply. By default, the configuration's own limit, or else 1000, 2000, or 3000 tokens for the longest length asked for, with 120 more per message under `by = "turn"` and 300 more per stage under `by = "trajectory"`.

`reasoning_effort` Optional reasoning effort for every summary call, one string, written over the configuration's setting.

`templates` Optional named list of templates replacing the defaults of the same name: `summary_participant`, `summary_item`, `summary_disagreements`, `summary_trajectory`, `summary_turn`, `summary_session`, and `summary_reask` (see `get_default_prompt_templates()`). A unit template must contain `\{\{transcript\}\}` and `\{\{format\}\}`, and `summary_reask` must contain `\{\{failures\}\}`; a placeholder without a value is an error.

`verbose` Logical. Print one line per call.

`.runner` Optional function used instead of live model calls. It receives a data frame with `config` and `messages` list-columns and returns the rows with at least `response_text`.

Value

A `focus_group_summary`, a list with:

summaries One row per unit and length: `segment` (`"session"`, or `"prefix"` and the arm names), `unit`, `level`, `text` (rendered from the claims kept; `NA` when there is none), `n_claims`, `words`, `budget`, `n_turns`, `speaker_id` (the user ID, for participant and turn units), and `note`.

claims The claims kept, one row each: `segment`, `unit`, `claim`, `kind`, `status`, `text`, `speakers` (handles as the model saw them), `speaker_ids`, `evidence`, `context`, `probe_evidence`, `act`, `ending`, `ending_message_id`, `levels`, and `attempt`.

quotes The quotes of the claims kept, each cut from its source: `text` as the model gave it, `verbatim` as it appears in the message, and `start` and `end`, character positions in the message text, with `spans` giving each part of a quote joined by "...".

checks Every check run, one row per check, claim, and attempt, with `passed` and, on failure, `detail`.

rejected Claims dropped under `on_fail = "drop"`, with the reasons.

calls One row per model call, in the format of the `calls` component of `run_focus_group()`, with `segment` and `unit`.

turns The table from `fg_turns()`, with `label`, the handle the model saw.

metadata `by`, `compression`, `include_probes`, `on_fail`, `provider`, `model`, `settings`, `max_tokens`, `reasoning_effort`, `structured_output`, `templates`, `handles` (user ID, handle, and the label sent to the model), `segments`, `source_hash`, `units`, `estimated_calls`, `max_calls`, `budgets`, `usage`, `versions`, `created`, `complete`, and `failure`.

`as.data.frame()` returns `summaries`, or another table named by its `component` argument.

Units

`"participant"` One call per participant: positions and reasons, changes of position, reactions to others, and guide items not addressed. A participant with no turns gets a row with `n_turns = 0` and no call.

`"item"` One call per guide item with participant turns, sending only that item's messages: positions and their holders, agreement and disagreement. Who spoke, and how often, is counted in code and given to the model.

`"disagreements"` One call per session: one entry per disagreement or position held by one participant alone, with its holders, its messages, and how it ended (a concession, the moderator moving on, or unresolved).

`"trajectory"` One call per session. The stages are fixed by the package: thirds of the participant turns at `"headline"`, one stage per guide item at `"brief"` and `"detailed"`.

`"turn"` One call per guide item (an item without participant turns joins the next one): a gist of every message, with an act code at `"detailed"`.

`"session"` One call per session for the whole group.

For a `focus_group_arms` object the shared prefix is summarized once and each arm from the branch point on; an arm's call sees only the messages after the branch.

Lengths

`compression` names the lengths: `"headline"` (one sentence per unit, 20 words per turn), `"brief"` (a paragraph of about 100 to 120 words; 40 words per turn), and `"detailed"` (longer, with quotes). All the lengths asked for come back in one call per unit, as lists of claims drawn from one set, so they agree with each other. The word budgets are recorded beside each summary; they are not enforced.

What the model sees

The transcript, with each message's id and its speaker's handle, the label the agents saw. No user ID labels a speaker: a speaker whose handle is its user ID (an imported transcript, `handles = "ids"`, or a session made before handles existed) is shown as `P1`, `P2`, ... in roster order, unless that ID already has this form or another voice uses it, and the moderator as `Moderator`. The messages themselves are sent as recorded: the speakers wrote them under the handles they saw, and a name in them ("Ana, why evenings?", "Will the library stay open in May?") is left as written. Personas are never sent, nor is the System roster row. Injected messages are marked as injected, and those shown to participants only are marked as such. Private probe answers are sent only with `include_probes = TRUE`, as a separate block with ids `Q1`, `Q2`, ...

Checks

Each reply is checked in code, with no model involved. The JSON must parse (the reply is the first complete JSON object with a `claims` field, in a code block or in the text; only the final answer counts, so reasoning a model writes into its reply is ignored) and the reply must not have been cut off at the token limit. Every message a claim cites must exist, lie within the unit, and not be the System row. The messages given as evidence must be by the speakers the claim names, and a claim other than `"not_stated"` must cite at least one message by each speaker it names.

A quote must appear in the text of the cited message, ignoring only differences of case, whitespace, and quotation marks (and, for Persian, the zero-width non-joiner and the Arabic forms of yeh and kaf). It must have at least 3 words and start and end on word boundaries, an apostrophe or hyphen between letters counting as part of a word. The parts of a quote joined by `"..."` must appear in order in the same message, each with at least 2 words, and the words left out between them may not include a negation such as "not" or "n't". Words a claim puts in double quotation marks in its text must likewise appear in a message it cites as evidence.

A claim of change must cite two messages of one speaker, earlier first. Every handle of the form `P` and a number in a claim must be on the roster; a participant the text names must be among the claim's speakers or the speaker of a message it cites as context; any speaker the text names, by the label shown to the model or, for an imported transcript, by the speaker's name (matched in its case, as a whole word), must be among the claim's speakers or the speakers of the messages it cites, unless the name with a word beside it repeats a cited message ("open in May" from a message asking about May), so that "Ben said ..." citing only Ana's message fails; a claim that is not `"not_stated"` may not name a participant with no message in the unit; a claim that a participant said nothing fails when that participant spoke; and a user ID or name the model was not shown, as a speaker's label or in the text of a message, may not appear. A claim that speaks for the whole group ("everyone", "all of them", "no one", "consensus", "unanimous", "the whole group", and similar wording) must cite a message from every participant who spoke on each item it cites. Unless `include_probes = TRUE`, no claim may share a run of 6 words with a probe answer that does not also occur in the transcript.

Coverage is counted over the claims that pass: every participant, every participant who spoke on the item, every message, or every stage, as the unit requires, and every length asked for. A participant counts as covered only by a claim that reports what they said, with a message of theirs as evidence. The prose shows the status of a claim that is not `"stated"` at every length.

A reply that fails is sent back once, with each failure stated (for instance, that a quote is not in message 14, followed by the text of message 14); after a reply cut off at the token limit, the limit is doubled for the re-ask. If the second reply fails too, `on_fail = "error"` raises an error of class `focusgroup_summary_check_failure` with fields `checks` (the checks of both replies), `replies` (both raw replies), `segment`, `unit`, and `partial` (the summary of the units completed before it). `on_fail = "drop"` instead moves the failing claims to `rejected` and renders the summary from the rest, but still raises the error when a dropped claim reports a disagreement, a position held by one participant alone, or a change, or when the reply as a whole fails (invalid JSON, a cut-off reply, missing lengths, or coverage that the claims kept do not give).

Calls and structured output

Calls are made one at a time, with call types `"summary_<by>"`, and are recorded in the `calls` component of the result, never in the session's records; the random-number state is the same after the call as before. The estimate checked against `max_calls` counts one re-ask per unit, an upper bound. A unit whose prompt is longer than the prompt window (64,000 estimated tokens, counting bytes for text outside ASCII) stops the run before any call, with an error of class `focusgroup_summary_input_too_long`; `by = "item"` sends one guide item at a time. The token limit is written into the field the configuration uses (`max_completion_tokens`, `max_output_tokens`, or `max_tokens`). When the installed LLMR provides `LLMR::enable_structured_output()`, the request also carries the JSON schema of the reply, unless the configuration already sets a response format or is an Anthropic configuration with thinking enabled, which cannot take a forced tool call; set `options(focusgroup.summary_structured_output = FALSE)` for a model that rejects a

schema. Otherwise the format is stated in the prompt alone. Either way every reply is validated locally. The schema and the mode are recorded in `metadata$structured_output`. No temperature or seed is set.

See Also

`fg_turns()`; the `summarize()` method of FocusGroup for the interim summary that agents read during a session.

Examples

```
transcript <- data.frame(
  speaker = c("Moderator", "Ana", "Ben"),
  text = c("What should change about the library hours?",
           "Evening hours would help working parents like me.",
           "I disagree, mornings are what retirees need most.")
)
fg <- focus_group_from_transcript(transcript, topic = "Library hours")
# A scripted model reply for each participant, as a live model would give.
scripted <- function(experiments, ...) {
  prompt <- experiments$messages[[1]][[1]]$content
  unit <- if (grepl("participant, P1", prompt, fixed = TRUE)) "P1" else "P2"
  claim <- if (unit == "P1") {
    list(id = 1, unit = "P1", kind = "position", status = "stated",
         text = "P1 wants evening hours for working parents.",
         speakers = list("P1"), evidence = list(2), context = list(1),
         quotes = list(list(message_id = 2, speaker = "P1",
                            text = "Evening hours would help")))
  } else {
    list(id = 1, unit = "P2", kind = "position", status = "stated",
         text = "P2 disagrees and puts retirees' mornings first.",
         speakers = list("P2"), evidence = list(3), context = list(2),
         quotes = list(list(message_id = 3, speaker = "P2",
                            text = "mornings are what retirees need")))
  }
  experiments$response_text <- as.character(jsonlite::toJSON(
    list(claims = list(claim), levels = list(headline = list(1))),
    auto_unbox = TRUE))
  experiments
}
cfg <- LLMR::llm_config("openai", "gpt-4o-mini")
out <- summarize_focus_group(fg, cfg, compression = "headline",
                             .runner = scripted)
out
out$quotes[, c("unit", "message_id", "verbatim")]
```

Index

∗ **datasets**
anes_2024_personas, 4

analyze_focus_group, 2
analyze_focus_group(), *31*, *33*, *45*
anes_2024_personas, 4, *9, 10*, *16*

ConversationFlow, 6
create_agents, 8
create_agents(), *10*, *26*, *48*, *52*
create_agents_from_data, 10
create_agents_from_data(), *4*, *9*, *16*, *48*
create_agents_from_survey, 12
create_agents_from_survey(), *9–12*, *19*, *48*
create_conversation_flow, 17
create_conversation_flow(), *6*

fg_after_item, 18
fg_after_item(), *20*, *23*, *38*, *52*
fg_before_turn *(*fg_after_item*)*, 18
fg_before_turn(), *20*, *23*
fg_default_na_strings, 19
fg_default_na_strings(), *9*, *11*, *14*
fg_injection, 19
fg_injection(), *18*, *38, 39*, *52*, *54*, *59, 60*
fg_player, 21
fg_probe, 23
fg_probe(), *18*, *30*, *38, 39*, *47*, *52*, *55*, *58*, *60*
fg_turns, 24
fg_turns(), *31*, *64*, *67*
FGAgent, 26
focus_group_from_transcript, 31
focus_group_from_transcript(), *3*, *34*, *44–46*, *55*, *60*, *63*
FocusGroup, *6*, *21*, *25*, *27*, *31–33*, 34, *44, 45*, *48*, *63*, *67*
format.focus_group_arms *(*format.focus_group_result*)*, 44
format.focus_group_result, 44
format.focus_group_result(), *31*, *33*, *55*
format.FocusGroup *(*format.focus_group_result*)*, 44

get_default_prompt_templates, 46
get_default_prompt_templates(), *23*, *30*, *64*

haven::as_factor(), *11*
haven::read_dta(), *11*
haven::read_sav(), *11*

LLMR::enable_structured_output(), *66*
LLMR::llm_persona_split(), *10*

persona_rendering_report, 48
persona_rendering_report(), *9*, *12*, *15, 16*

run_focus_group, 50
run_focus_group(), *18*, *20–22*, *24*, *33–36*, *47, 48*, *59, 60*, *62–64*
run_focus_group_arms, 56
run_focus_group_arms(), *18*, *20–22*, *24*, *31*, *39*, *52*, *54*, *63*
run_focus_studio, 61

saveRDS(), *45*, *55*
subset(), *10*, *13*
summarize_focus_group, 63
summarize_focus_group(), *25*, *31*, *47*

utils::browseURL(), *21*