Package 'rollout'

January 13, 2026

Title Tools for Designing, Simulating, and Analyzing Implementation Rollout Trials

Version 0.1.0

Description Provides a unified framework for designing, simulating, and analyzing implementation rollout trials, including stepped wedge, sequential rollout, head-to-head, multi-condition, and rollout implementation optimization designs. The package enables users to flexibly specify rollout schedules, incorporate site-level and nested data structures, generate outcomes under rich hierarchical models, and evaluate analytic strategies through simulation-based power analysis. By separating data generation from model fitting, the tools support assessment of bias, Type I error, and robustness to model misspecification. The workflow integrates with standard mixed-effects modeling approaches and the tidyverse ecosystem, offering transparent and reproducible tools for implementation scientists and applied statisticians.

License MIT + file LICENSE

URL https://github.com/iancero/rollout, https://iancero.github.io/rollout/

BugReports https://github.com/iancero/rollout/issues

Imports broom.mixed, dplyr, glue, lifecycle, parallel, pbapply, purrr, rlang, stats, tidyr

Suggests lme4, lmerTest, testthat (>= 3.0.0), tibble

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.5.0)

NeedsCompilation no

Author Ian Cero [aut, cre] (ORCID: <https://orcid.org/0000-0002-2862-0450>), C. Hendricks Brown [aut] (ORCID: <https://orcid.org/0000-0002-0294-2419>)

Maintainer Ian Cero <ian_cero@urmc.rochester.edu>

Repository CRAN

Date/Publication 2026-01-13 18:20:02 UTC

Contents

`add_binary_outcome` *Create a binary outcome from linear predictors*

Description

Generates a binary outcome by summing effects, computing probabilities via the logistic function, and drawing binary outcomes.

Usage

```
add_binary_outcome(
  data,
  linear_col = "y_linear",
  prob_col = "y_prob",
  binary_col = "y_bin"
)
```

Arguments

`data`	A data frame containing effect columns prefixed with `".."`.
`linear_col`	Name of the column to store the summed linear predictor (default `"y_linear"`).
`prob_col`	Name of the column to store probabilities (default `"y_prob"`).
`binary_col`	Name of the column to store binary outcomes (default `"y_bin"`).

Value

A `tibble` with added linear predictor, probability, and binary outcome columns.

Examples

```
df <- tibble::tibble(.beta = 0.5, .u = rnorm(5), .error = rnorm(5))
add_binary_outcome(df)
```

---

add_error *Add an error term for simulation*

Description

Adds a residual error term (column `.error`) to the data frame, drawn from a normal distribution with specified variance.

Usage

```
add_error(.data, variance = 1)
```

Arguments

`.data`	A data frame to which the error term will be added.
`variance`	Numeric; variance of the residual error (default 1).

Value

A `tibble` with an added `.error` column.

Examples

```
df <- tibble::tibble(x = 1:5)
add_error(df, variance = 2)
```

---

add_fixed_effect *Add a fixed effect column for simulation*

Description

Adds a fixed effect column (prefixed with `"."`) to the design data frame for simulation purposes.

Usage

```
add_fixed_effect(design_df, ...)
```

Arguments

design_df	A data frame containing the rollout design and any parameters.
...	A single named expression specifying the fixed effect to add (e.g., beta = 0.5 * x).

Value

A tibble with the added fixed effect column.

Examples

```
df <- tibble::tibble(x = rnorm(5))
add_fixed_effect(df, beta = 0.5 * x)
```

---

add_linear_outcome *Create a linear outcome by summing effects*

Description

Generates a linear outcome variable by summing all columns that start with "." (representing fixed, random, and error effects).

Usage

```
add_linear_outcome(data, output_col = "y_linear")
```

Arguments

data	A data frame containing effect columns prefixed with ".".
output_col	Name of the column to store the linear outcome (default "y_linear").

Value

A tibble with the added linear outcome column.

Examples

```
df <- tibble::tibble(.beta = 0.5, .u = rnorm(5), .error = rnorm(5))
add_linear_outcome(df)
```

add_parameter *Expand a data frame with parameter combinations for simulation*

Description

Adds combinations of specified parameter values to a data frame for simulation by expanding over all combinations.

Usage

```
add_parameter(df, ...)
```

Arguments

df	A data frame to expand.
...	Named vectors specifying parameter values to expand, provided as param_name = values.

Value

A tibble with added parameter columns for each combination of values.

Examples

```
df <- tibble::tibble(site = "A", condition = "control")
add_parameter(df, beta = c(0, 0.5), sigma = c(1, 2))
```

add_poisson_outcome *Create a Poisson outcome from linear predictors*

Description

Generates a Poisson-distributed count outcome by summing effects, exponentiating to obtain rates, and drawing counts.

Usage

```
add_poisson_outcome(
  data,
  linear_col = "y_linear",
  rate_col = "y_rate",
  count_col = "y_count"
)
```

Arguments

data	A data frame containing effect columns prefixed with ".".
linear_col	Name of the column to store the summed linear predictor (default "y_linear").
rate_col	Name of the column to store Poisson rates (default "y_rate").
count_col	Name of the column to store Poisson counts (default "y_count").

Value

A tibble with added linear predictor, rate, and count columns.

Examples

```
df <- tibble::tibble(.beta = 0.5, .u = rnorm(5), .error = rnorm(5))
add_poisson_outcome(df)
```

---

add_random_effect *Add a random effect column for simulation*

Description

Adds a random effect column (prefixed with ".") to the design data frame, with optional grouping for nested random effects.

Usage

```
add_random_effect(design_df, ..., .nesting = NULL)
```

Arguments

design_df	A data frame containing the rollout design and any parameters.
...	A single named expression specifying the random effect to add (e.g., u = rnorm(1, 0, 1)).
.nesting	Optional character vector specifying grouping columns for nested random effects (default NULL).

Value

A tibble with the added random effect column.

Examples

```
df <- tibble::tibble(site = rep(1:2, each = 3))
add_random_effect(df, u = rnorm(1, 0, 1), .nesting = "site")
```

`evaluate_model_results` *Summarise simulation results from extracted model estimates*

Description

Computes summary statistics (e.g., power, custom summaries) across a set of extracted model results, typically from `extract_model_results()`, to facilitate simulation evaluation and reporting.

Usage

```
evaluate_model_results(
  results,
  alpha = 0.05,
  ...,
  .summarise_standard_broom = FALSE,
  broom_cols = c("estimate", "std.error", "statistic", "df", "p.value")
)
```

Arguments

`results`	A data frame of extracted model results, typically including columns like `term`, `estimate`, `std.error`, `statistic`, and `p.value`.
`alpha`	Significance level used to compute power. Defaults to `0.05`.
`...`	Additional summary expressions to compute within `dplyr::summarise()`. These may include calls to helper functions like `eval_bias()`, `eval_quantile()`, or direct summaries such as `mean(estimate, na.rm = TRUE)`.
`.summarise_standard_broom`	Logical; if `TRUE`, computes mean and standard deviation for standard `broom` columns present in the data (columns in `broom_cols`). Defaults to `FALSE`.
`broom_cols`	Character vector of standard `broom` columns to summarise if `.summarise_standard_broom = TRUE`. Defaults to `c("estimate", "std.error", "statistic", "df", "p.value")`.

Value

A summarised data frame containing:

- `n_models`: the number of models summarised.
- `power`: the proportion of p-values less than `alpha` (NA if all p-values are NA).
- Additional columns corresponding to custom summaries provided in `...`.
- Mean and SD summaries of `broom` columns if `.summarise_standard_broom = TRUE`.

Examples

```
library(dplyr)
library(purrr)
library(broom.mixed)

# Simulate and fit models
sim_models <- tibble(
  id = 1:50,
  model = map(1:50, ~ lm(mpg ~ wt, data = mtcars))
) |>
  extract_model_results()

# Evaluate power and mean estimate for the slope
sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    alpha = 0.05,
    mean_estimate = mean(estimate, na.rm = TRUE),
    sd_estimate = sd(estimate, na.rm = TRUE)
  )

# Evaluate with .summarise_standard_broom = TRUE
sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    .summarise_standard_broom = TRUE
  )

# Evaluate with eval_bias to compute bias relative to the true value
# Suppose the true slope of wt is -5 (hypothetical)
sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    bias = eval_bias(
      estimate,
      term = c("wt" = -5)
    )
  )
```

`eval_between` *Compute the proportion of values within term-specific intervals within grouped simulation results*

Description

Computes the proportion of `x` values falling within term-specific intervals within each group, typically inside `evaluate_model_results()` for simulation evaluation pipelines.

Usage

```
eval_between(x, term = NULL, na.rm = FALSE)
```

Arguments

x	A numeric vector of estimates or statistics.
term	A named list of numeric vectors of length 2, giving the lower and upper bounds for each term. For example, `list("(Intercept)" = c(-1, 1), x = c(1, 3))`. If `NULL` (default), the interval is assumed to be `[0, 1]`.
na.rm	Logical; whether to remove missing values when computing the proportion. Defaults to `FALSE`.

Details

This function is designed to be used inside `dplyr::summarise()` within a grouped tidyverse pipeline, typically after grouping by `term`.

If `term` is provided, the current grouping must include a `term` variable matching the names in `term`. If a term in the group is not found in the provided `term` mapping, the function will return `NA` with a warning.

Value

A numeric scalar representing the proportion of `x` within the term-specific interval within the current group.

Examples

```
library(dplyr)
library(purrr)
library(broom.mixed)

sim_models <- tibble(
  id = 1:50,
  model = map(1:50, ~ lm(mpg ~ wt, data = mtcars))
) |>
  extract_model_results()

sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    prop_between = eval_between(
      estimate,
      term = list("wt" = c(-1, 0))
    )
  )
```

`eval_bias` *Compute bias relative to term-specific true values within grouped simulation results*

Description

Computes the mean bias (difference between estimated values and true values) within each group, typically inside `evaluate_model_results()` for simulation evaluation pipelines.

Usage

```
eval_bias(x, term = NULL, na.rm = FALSE, warnings = TRUE)
```

Arguments

`x`	A numeric vector of estimates (e.g., from a model term).
`term`	A named numeric vector providing the true value for each term. For example, `c("(Intercept)" = 0, x = 2)` to specify the true values for each term. If `NULL` (default), bias is computed relative to zero.
`na.rm`	Logical; whether to remove missing values when computing the mean bias. Defaults to `FALSE`.
`warnings`	Should warnings be returned?

Details

This function is designed to be used inside `dplyr::summarise()` within a grouped tidyverse pipeline, typically after grouping by `term`. It computes the mean of `x` minus the true value for the corresponding term.

If `term` is provided, the current grouping must include a `term` variable matching the names in `term`. If a term in the group is not found in the provided `term` mapping, the function will return `NA` with a warning.

Value

A numeric scalar representing the mean bias within the current group.

Examples

```
library(dplyr)
library(purrr)
library(broom.mixed)

# Simulate and fit models
sim_models <- tibble(
  id = 1:50,
  model = map(1:50, ~ lm(mpg ~ wt, data = mtcars))
) |>
```

```
  extract_model_results()

# Compute bias relative to true value (hypothetical slope = -5)
sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    bias = eval_bias(
      estimate,
      term = c("wt" = -5)
    )
  )

# Compute bias relative to zero for all terms
sim_models |>
  group_by(term) |>
  evaluate_model_results(
    bias = eval_bias(estimate)
  )
```

`eval_greater_than` *Compute the proportion of values above term-specific thresholds within grouped simulation results*

Description

Computes the proportion of `x` values exceeding term-specific thresholds within each group, typically inside `evaluate_model_results()` for simulation evaluation pipelines.

Usage

```
eval_greater_than(x, term = NULL, na.rm = FALSE)
```

Arguments

`x`	A numeric vector of estimates or statistics.
`term`	A named numeric vector providing the threshold for each term. For example, `c("(Intercept)" = 0, x = 2)`. If `NULL` (default), threshold is assumed to be zero.
`na.rm`	Logical; whether to remove missing values when computing the proportion. Defaults to `FALSE`.

Details

This function is designed to be used inside `dplyr::summarise()` within a grouped tidyverse pipeline, typically after grouping by `term`.

If `term` is provided, the current grouping must include a `term` variable matching the names in `term`. If a term in the group is not found in the provided `term` mapping, the function will return `NA` with a warning.

Value

A numeric scalar representing the proportion of x exceeding the term-specific threshold within the current group.

Examples

```
library(dplyr)
library(purrr)
library(broom.mixed)

sim_models <- tibble(
  id = 1:50,
  model = map(1:50, ~ lm(mpg ~ wt, data = mtcars))
) |>
  extract_model_results()

sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    prop_above_0 = eval_greater_than(
      estimate,
      term = c("wt" = 0)
    )
  )
```

---

eval_less_than *Compute the proportion of values below term-specific thresholds within grouped simulation results*

Description

Computes the proportion of x values falling below term-specific thresholds within each group, typically inside `evaluate_model_results()` for simulation evaluation pipelines.

Usage

```
eval_less_than(x, term = NULL, na.rm = FALSE)
```

Arguments

x	A numeric vector of estimates or statistics.
term	A named numeric vector providing the threshold for each term. For example, `c("(Intercept)" = 0, x = 2)`. If NULL (default), threshold is assumed to be zero.
na.rm	Logical; whether to remove missing values when computing the proportion. Defaults to FALSE.

Details

This function is designed to be used inside `dplyr::summarise()` within a grouped tidyverse pipeline, typically after grouping by `term`.

If `term` is provided, the current grouping must include a `term` variable matching the names in `term`. If a term in the group is not found in the provided `term` mapping, the function will return `NA` with a warning.

Value

A numeric scalar representing the proportion of `x` below the term-specific threshold within the current group.

Examples

```
library(dplyr)
library(purrr)
library(broom.mixed)

sim_models <- tibble(
  id = 1:50,
  model = map(1:50, ~ lm(mpg ~ wt, data = mtcars))
) |>
  extract_model_results()

sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    prop_below_0 = eval_less_than(
      estimate,
      term = c("wt" = 0)
    )
  )
```

---

eval_quantile *Compute the observed quantile value for each term within grouped simulation results*

Description

Computes the specified quantile of `x` within each group, typically inside `evaluate_model_results()` for simulation evaluation pipelines.

Usage

```
eval_quantile(x, term = NULL, na.rm = FALSE)
```

Arguments

x	A numeric vector of estimates or statistics.
term	A named numeric vector with quantile probabilities for each term. For example, c("(Intercept)" = 0.05, x = 0.95). If NULL (default), computes the median (0.5).
na.rm	Logical; whether to remove missing values when computing the quantile. Defaults to FALSE.

Details

This function is designed to be used inside dplyr::summarise() within a grouped tidyverse pipeline, typically after grouping by term.

If term is provided, the current grouping must include a term variable matching the names in term. If a term in the group is not found in the provided term mapping, the function will return NA with a warning.

Value

A numeric scalar representing the observed quantile of x within the current group.

Examples

```
library(dplyr)
library(purrr)
library(broom.mixed)

sim_models <- tibble(
  id = 1:50,
  model = map(1:50, ~ lm(mpg ~ wt, data = mtcars))
) |>
  extract_model_results()

sim_models |>
  filter(term == "wt") |>
  group_by(term) |>
  evaluate_model_results(
    lower_quantile = eval_quantile(
      estimate,
      term = c("wt" = 0.05)
    ),
    upper_quantile = eval_quantile(
      estimate,
      term = c("wt" = 0.95)
    )
  )
```

`extract_model_results` *Extract and tidy model results from a column of models*

Description

Applies a tidying function (default `broom.mixed::tidy`) to a column of models, returning a tidy data frame with one row per term per model, suitable for downstream summarisation and evaluation in simulation studies.

Usage

```
extract_model_results(
  models,
  model_col = "model",
  tidy_fun = broom.mixed::tidy,
  .term = NULL
)
```

Arguments

`models`	A data frame containing a column of fitted model objects.
`model_col`	Unquoted column name containing the models. Default is `model`.
`tidy_fun`	A tidying function to apply to each model. Default is `broom.mixed::tidy`. The function must return a data frame with a `term` column.
`.term`	Optional string specifying a term to filter after tidying (e.g., `"(Intercept)"`). If `NULL` (default), all terms are retained.

Value

A tidy data frame with the original columns of `models` joined to the tidied model results, typically including columns such as `term`, `estimate`, `std.error`, `statistic`, and `p.value`.

Examples

```
library(dplyr)
library(purrr)
library(broom.mixed)

# Simulate and fit models
sim_models <- tibble(
  id = 1:5,
  model = map(1:5, ~ lm(mpg ~ wt, data = mtcars))
)

# Extract all terms
extract_model_results(sim_models)
```

```
# Extract only the slope term
extract_model_results(sim_models, .term = "wt")
```

`fit_models` *Fit models in parallel across a list-column of datasets*

Description

Applies a user-specified model-fitting function to each element of a list-column of datasets in `.data`, fitting models in parallel with a progress bar, and returns the original data frame with a new `model` column containing the fitted models.

Usage

```
fit_models(
  .data,
  .x,
  .f,
  packages = NULL,
  n_cores = parallel::detectCores() - 1
)
```

Arguments

`.data`	A data frame containing a list-column of datasets to which the model function will be applied.
`.x`	Unquoted column name of the list-column containing the datasets.
`.f`	A function or formula to apply to each dataset to fit the desired model (e.g., `~ lm(y ~ x, data = .)` or `~ lme4::lmer(y ~ x + (x | group), data = .)`).
`packages`	A character vector of package names to load on each parallel worker, if your model-fitting function requires additional packages. Defaults to `NULL`.
`n_cores`	Number of cores to use for parallel processing. Defaults to `parallel::detectCores() - 1`.

Details

This function is intended for use in simulation pipelines where multiple datasets are generated (e.g., via `simulate_datasets()`), and models need to be fitted to each dataset efficiently in parallel.

It uses `pbapply::pblapply()` to provide a progress bar during model fitting, and `parallel::makeCluster()` for multi-core processing.

Packages specified in `packages` will be loaded on each worker to ensure model-fitting functions that depend on those packages work correctly in parallel.

Value

The original `.data` data frame with an additional `model` column containing the fitted model objects returned by `.f`.

Examples

```
library(dplyr)
library(purrr)
library(lme4)

# Create example grouped datasets for mixed models
datasets <- tibble(
  id = 1:5,
  data = map(1:5, ~ {
    df <- sleepstudy[sample(nrow(sleepstudy), 50, replace = TRUE), ]
    df$Subject <- factor(df$Subject)
    df
  })
)

# Fit linear mixed models in parallel
fitted_models <- fit_models(
  datasets,
  .x = data,
  .f = ~ lme4::lmer(Reaction ~ Days + (Days | Subject), data = .),
  packages = c("lme4"),
  n_cores = 1
)

# Inspect the first fitted mixed model
summary(fitted_models$model[[1]])

# Tidy the fitted models using extract_model_results() for further evaluation
extracted <- extract_model_results(fitted_models)
head(extracted)

# Summarise estimates for 'Days' across simulated fits
extracted |>
  filter(term == "Days") |>
  evaluate_model_results(
    mean_estimate = mean(estimate, na.rm = TRUE),
    sd_estimate = sd(estimate, na.rm = TRUE)
  )
```

---

initialize_replicates *Add replicate identifiers for simulation replicates*

Description

Expands a long-format schedule to include a replicate identifier for running multiple simulation replicates efficiently.

Usage

```
initialize_replicates(long_schedule, n)
```

Arguments

long_schedule	A long-format rollout schedule.
n	Integer specifying the number of replicates to generate.

Value

A tibble with an added sample_id column for replicate indexing.

Examples

```
schedule <- tibble::tibble(site = "A", cohort = 1, chron_time = 0, condition = "control")
initialize_replicates(schedule, n = 3)
```

---

join_info *Join unit-level information to a long-format rollout schedule*

Description

Merges unit-level characteristics or parameters into a long-format rollout schedule and optionally expands rows based on count variables to create multiple units per site.

Usage

```
join_info(
  long_schedule,
  unit_info,
  by = NULL,
  uncount_vars = NULL,
  .ids = NULL
)
```

Arguments

long_schedule	A long-format schedule (output from pivot_schedule_longer).
unit_info	A data frame with unit-level information to join.
by	Columns used to join long_schedule and unit_info (default NULL uses shared columns).

uncount_vars	Optional character vector or list of quosures indicating count variables to expand rows.
.ids	Optional character vector specifying names of id columns when uncounting, one per uncount_var.

Value

A tibble with joined and optionally expanded rows to reflect unit counts.

Examples

```
schedule <- tibble::tibble(site = "A", cohort = 1, chron_time = 0, condition = "control")
unit_info <- tibble::tibble(site = "A", n_units = 3)
join_info(schedule, unit_info, by = "site", uncount_vars = "n_units")
```

---

pivot_schedule_longer *Pivot a rollout schedule from wide to long format with local time calculation*

Description

Transforms a wide-format rollout schedule into a long-format schedule, extracting chronological time from column names, converting condition columns to factors, and adding local time within each cohort if desired.

Usage

```
pivot_schedule_longer(
  schedule,
  time_cols,
  names_to = "chron_time",
  names_pattern = ".*(\\d+)",
  names_transform = as.numeric,
  values_to = "condition",
  values_transform = as.factor,
  cohort_name = "cohort",
  local_time = TRUE
)
```

Arguments

schedule	A data frame containing the rollout schedule in wide format.
time_cols	Columns containing time-specific condition assignments (tidyselect syntax).
names_to	Name of the new column to store extracted chronological time (default "chron_time").
names_pattern	Regular expression to extract the numeric time from column names (default ".*(\\d+)").

names_transform
: Function to transform extracted time values (default `as.numeric`).

values_to
: Name of the new column to store condition values (default `"condition"`).

values_transform
: Function to transform condition values (default `as.factor`).

cohort_name
: The column indicating cohort membership for local time calculation (default `cohort`).

local_time
: Logical; if `TRUE`, adds a `local_time` column indicating time since rollout start for each cohort and condition (default `TRUE`).

Value

A long-format `tibble` with columns for cohort, condition, chronological time, and optionally local time.

Examples

```
library(dplyr)
library(tidyr)
schedule <- tibble::tibble(
  site = c("A", "B"),
  cohort = c(1, 2),
  t1 = c("control", "intervention"),
  t2 = c("intervention", "intervention")
)
pivot_schedule_longer(schedule, time_cols = starts_with("t"))
```

Index