Package 'eam'

May 8, 2026

Type Package

Title Evidence Accumulation Models

Version 1.2.0

LinkingTo Rcpp

Imports Rcpp, dplyr, tidyr, arrow, rlang, distributional, stats, parallel, codetools, grDevices, graphics, ggplot2, gridExtra, data.table, purrr, scales, vctrs, tibble, JuliaConnectoR, NeuralEstimators

Suggests testthat (>= 3.0.0), pbapply, abc

Description Simulation-based evidence accumulation models for analyzing responses and reaction times in single- and multi-response tasks. The package includes simulation engines for five representative models: the Diffusion Decision Model (DDM), Leaky Competing Accumulator (LCA), Linear Ballistic Accumulator (LBA), Racing Diffusion Model (RDM), and Levy Flight Model (LFM), and extends these frameworks to multi-response settings. The package supports user-defined functions for item-level parameterization and the incorporation of covariates, enabling flexible customization and the development of new model variants based on existing architectures. Inference is performed using simulation-based methods, including Approximate Bayesian Computation (ABC) and Amortized Bayesian Inference (ABI), which allow parameter estimation without requiring tractable likelihood functions. In addition to core inference tools, the package provides modules for parameter recovery, posterior predictive checks, and model comparison, facilitating the study of a wide range of cognitive processes in tasks involving perceptual decision making, memory retrieval, and value-based decision making. Key methods implemented in the package are described in Ratcliff (1978) <doi:10.1037/0033-295X.85.2.59>, Usher and McClelland (2001) <doi:10.1037/0033-295X.108.3.550>, Brown and Heathcote (2008) <doi:10.1016/j.cogpsych.2007.12.002>, Tillman, Van Zandt and Logan (2020) <doi:10.3758/s13423-020-01719-6>, Wieschen, Voss and Radev (2020) <doi:10.20982/tqmp.16.2.p120>, Csilléry, François and Blum (2012) <doi:10.1111/j.2041-210X.2011.00179.x>, Beaumont (2019) <doi:10.1146/annurev-statistics-030718-105212>, and Sainsbury-Dale, Zammit-Mangion and Huser (2024) <doi:10.1080/00031305.2023.2249522>.

License MIT + file LICENSE

Encoding UTF-8

Config/testthat/edition 3

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

URL https://y-guang.github.io/eam/, https://github.com/y-guang/eam

NeedsCompilation yes

Author Guangyu Zhu [aut] (ORCID: <https://orcid.org/0009-0005-1571-1782>), Guang Yang [aut, cre] (ORCID: <https://orcid.org/0009-0005-7675-3249>)

Maintainer Guang Yang <guang.spike.yang@gmail.com>

Repository CRAN

Date/Publication 2026-03-29 13:00:02 UTC

Contents

`+.eam_summarise_by_spec`

Add two summarise_by specs together

Description

S3 method for the + operator to combine two ‘eam_summarise_by_spec‘ objects into a single spec that will apply both operations.

Usage

```
## S3 method for class 'eam_summarise_by_spec'
e1 + e2
```

Arguments

`e1`	First eam_summarise_by_spec or eam_summarise_by_tbl object
`e2`	Second eam_summarise_by_spec or eam_summarise_by_tbl object

Value

A combined eam_summarise_by_spec object

`+.eam_summarise_by_tbl`

Join two eam_summarise_by_tbl objects

Description

S3 method for the + operator to join two summary tables created by `summarise_by`. Tables must have identical .wider_by attributes to be joined.

Usage

```
## S3 method for class 'eam_summarise_by_tbl'
e1 + e2
```

Arguments

`e1`	First eam_summarise_by_tbl object
`e2`	Second eam_summarise_by_tbl object

Value

A joined data frame with class "eam_summarise_by_tbl", preserving the .wider_by attribute from the input tables

---

abc_abc *Approximate Bayesian Computation wrapper*

Description

Wrapper around abc to perform ABC inference. This function provides a consistent interface within the eam package and encapsulates the dependency on the abc package.

Usage

```
abc_abc(abc_input, tol, method, transf = "none", ...)
```

Arguments

abc_input	A list with components target, param, and sumstat (typically produced by build_abc_input)
tol	Tolerance level (0 to 1) for ABC acceptance
method	ABC method: "rejection", "loclinear", "neuralnet"
transf	Transformations to apply to parameters: "none" (default), "log", or "logit"
...	Additional arguments passed to abc

Details

This is a thin wrapper around the abc::abc() function. Users should refer to the abc package documentation for detailed parameter descriptions and options.

Value

An object of class abc from abc

Examples

```
# Load example simulation output and observed data
rdm_minimal_example <- system.file("extdata", "rdm_minimal", package = "eam")
sim_output <- load_simulation_output(file.path(rdm_minimal_example, "simulation"))
obs_df <- read.csv(file.path(rdm_minimal_example, "observation", "observation_data.csv"))

# Define a summary-statistics pipeline
summary_pipe <- summarise_by(
  .by = c("condition_idx"),
  rt_mean = mean(rt)
)
```

```
# Summarise simulation output and observed data
sim_summary <- map_by_condition(
  sim_output,
  .progress = FALSE,
  .parallel = FALSE,
  function(cond_df) {
    summary_pipe(cond_df)
  }
)
obs_summary <- summary_pipe(obs_df)

# Build ABC input
abc_input <- build_abc_input(
  simulation_output = sim_output,
  simulation_summary = sim_summary,
  target_summary = obs_summary,
  param = c("V_beta_1", "V_beta_group")
)

# Fit an ABC model
abc_rejection_model <- abc_abc(
  abc_input = abc_input,
  tol = 0.5,
  method = "rejection"
)
```

---

abc_cv *Cross-validation for ABC model*

Description

Wrapper around cv4abc to perform cross-validation of ABC results. This function provides a consistent interface within the eam package and encapsulates the dependency on the abc package.

Usage

```
abc_cv(abc_input, abc_result, nval, tols, ...)
```

Arguments

abc_input	A list with components param and sumstat (typically produced by build_abc_input)
abc_result	Fitted ABC model from abc_abc. Parameters like method, transf, etc. are extracted from this object.
nval	Number of cross-validation folds
tols	Tolerance levels to test during cross-validation
...	Additional arguments passed to cv4abc

Details

This is a thin wrapper around the `abc::cv4abc()` function. When `abc_result` is provided, cv4abc extracts the method, transf, and other settings from the fitted ABC object. Users should refer to the abc package documentation for detailed parameter descriptions and options.

Value

A cross-validation object from cv4abc

Examples

```
# Load example simulation output and observed data
rdm_minimal_example <- system.file("extdata", "rdm_minimal", package = "eam")
sim_output <- load_simulation_output(file.path(rdm_minimal_example, "simulation"))
obs_df <- read.csv(file.path(rdm_minimal_example, "observation", "observation_data.csv"))

# Define a summary-statistics pipeline
summary_pipe <- summarise_by(
  .by = c("condition_idx"),
  rt_mean = mean(rt)
)

# Summarise simulation output and observed data
sim_summary <- map_by_condition(
  sim_output,
  .progress = FALSE,
  .parallel = FALSE,
  function(cond_df) {
    summary_pipe(cond_df)
  }
)
obs_summary <- summary_pipe(obs_df)

# Build ABC input and fit an ABC model
abc_input <- build_abc_input(
  simulation_output = sim_output,
  simulation_summary = sim_summary,
  target_summary = obs_summary,
  param = c("V_beta_1", "V_beta_group")
)
abc_model <- abc_abc(
  abc_input = abc_input,
  tol = 0.5,
  method = "rejection"
)

# Run cross-validation for the fitted ABC model
abc_cv_result <- abc_cv(
  abc_input = abc_input,
  abc_result = abc_model,
  nval = 10,
  tols = c(0.1, 0.5)
```

```
)
```

`abc_posterior_bootstrap` *Bootstrap resample ABC posterior samples*

Description

Bootstrap resample ABC posterior samples

Usage

```
abc_posterior_bootstrap(abc_result, n_samples, replace = TRUE)
```

Arguments

`abc_result`	An abc object from abc
`n_samples`	Number of bootstrap samples to draw (default 1000)
`replace`	Logical, whether to sample with replacement (default TRUE)

Value

Data frame of bootstrapped parameter values

Examples

```
# Load an example abc output, you should generate it by applying ABC to your data
# check abc_abc for details on fitting ABC models
rdm_minimal_example <- system.file("extdata", "rdm_minimal", package = "eam")
abc_model <- readRDS(file.path(rdm_minimal_example, "abc", "abc_neuralnet_model.rds"))

# Bootstrap resample posterior parameters
posterior_params <- abc_posterior_bootstrap(
  abc_model,
  n_samples = 100
)

# View the first few rows of the bootstrapped posterior parameters
head(posterior_params)
```

`abc_posterior_predictive_check` *ABC posterior predictive check*

Description

High-level convenience wrapper for posterior predictive checks from `abc_abc()` outputs.

Usage

```
abc_posterior_predictive_check(
  config,
  abc_result,
  observed_df,
  n_conditions = 1,
  n_trials_per_condition = 500,
  n_items = config$n_items,
  n_conditions_per_chunk = NULL,
  output_dir = NULL,
  rt_facet_x = c("item_idx"),
  rt_facet_y = c(),
  accuracy_x = "item_idx",
  accuracy_facet_x = c(),
  accuracy_facet_y = c()
)
```

Arguments

`config`	Simulation configuration object.
`abc_result`	Fitted object from `abc_abc()`.
`observed_df`	Observed trial-level data frame.
`n_conditions`	Number of posterior predictive conditions.
`n_trials_per_condition`	Number of trials per condition.
`n_items`	Number of items per trial.
`n_conditions_per_chunk`	Number of conditions per processing chunk.
`output_dir`	Optional output directory for simulation files.
`rt_facet_x`	Facet columns for `plot_rt()` x facets.
`rt_facet_y`	Facet columns for `plot_rt()` y facets.
`accuracy_x`	Grouping variable for `plot_accuracy()` x-axis.
`accuracy_facet_x`	Facet columns for `plot_accuracy()` x facets.
`accuracy_facet_y`	Facet columns for `plot_accuracy()` y facets.

Details

This function is for teaching and quick demonstrations. It is intentionally specific to one input shape (an abc object). For step checks, follow these functions: abc_posterior_bootstrap(), update_config_from_posterior(), and run_simulation().

This wrapper is mainly a teaching tool. It provides a compact end-to-end posterior predictive workflow, but it intentionally hides several modeling choices by collapsing the posterior to a single summary and then simulating from that reduced representation.

For more serious work, manual posterior predictive simulation is preferred. The recommended workflow is to draw posterior parameter values explicitly with abc_posterior_bootstrap(), inspect or modify those draws as needed, rebuild a simulation configuration explicitly with new_simulation_config so the parameter structure is fully under your control, run the simulation with run_simulation(), and then compare the simulated output with the observed data using plotting or summary functions. update_config_from_posterior() can still be useful for quick checks, but rebuilding the config is the safer option when you need to know exactly how posterior values are mapped back into the model. Following the steps manually makes each assumption visible, including which posterior draw was used, how parameter values entered the simulation config, and how the posterior predictive data were generated.

Value

invisible(NULL). This function is used for plotting side effects only and prints RT and accuracy plots directly.

Examples

```
# Load example simulation config, fitted ABC model, and observed data
base_dir <- system.file("extdata", "rdm_minimal", package = "eam")
sim_output <- load_simulation_output(file.path(base_dir, "simulation"))
abc_model <- readRDS(file.path(base_dir, "abc", "abc_neuralnet_model.rds"))
obs_df <- read.csv(file.path(base_dir, "observation", "observation_data.csv"))

# Run a high-level posterior predictive check
abc_posterior_predictive_check(
  config = sim_output$simulation_config,
  abc_result = abc_model,
  observed_df = obs_df,
  n_conditions = 1,
  n_trials_per_condition = 500,
  rt_facet_x = c("item_idx"),
  rt_facet_y = c(),
  accuracy_x = "item_idx",
  accuracy_facet_x = c("group"),
  accuracy_facet_y = c()
)
```

abc_postpr *ABC model comparison wrapper*

Description

Wrapper function for postpr to facilitate model comparison. This function simplifies the process of comparing multiple models using ABC by automatically stacking summary statistics and creating model indices.

Usage

```
abc_postpr(sumstats = list(), target, ...)
```

Arguments

sumstats	A named list of summary statistics matrices from different models. Each element should be a matrix or data frame with the same columns.
target	Target summary statistics from observed data (vector or matrix)
...	Additional arguments passed to postpr

Value

An object of class "postpr" from postpr

Examples

```
# Load pre-computed ABC input for model comparison
# This example compares the same model to itself for demonstration
rdm_minimal_example <- system.file("extdata", "rdm_minimal", package = "eam")
abc_input <- readRDS(file.path(rdm_minimal_example, "abc", "abc_input.rds"))

# Compare two models using their summary statistics
# In practice, create different abc_input objects for different models:
# abc_input_1 <- build_abc_input(..., simulation_summary = sim_summary_1, ...)
# abc_input_2 <- build_abc_input(..., simulation_summary = sim_summary_2, ...)
postpr_result <- abc_postpr(
  sumstats = list(model1 = abc_input$sumstat, model2 = abc_input$sumstat),
  target = abc_input$target,
  tol = 0.5,
  method = "rejection"
)

# View model comparison results
summary(postpr_result)
```

abc_resample *ABC with resampling*

Description

Performs ABC inference with resampling to assess stability and uncertainty. Each iteration draws a random sample from the simulation pool and runs ABC, producing multiple posterior estimates for comparison.

Usage

```
abc_resample(
  target,
  param,
  sumstat,
  n_iterations,
  n_samples,
  replace = FALSE,
  ...
)
```

Arguments

target	Target summary statistics from observed data
param	Parameter values matrix or data frame
sumstat	Summary statistics matrix or data frame
n_iterations	Number of resample iterations
n_samples	Number of samples to draw in each iteration
replace	Logical, whether to sample with replacement (default FALSE)
...	Additional arguments passed to abc_abc

Value

A list of length n_iterations, where each element is an object of class abc returned by abc. Each list element contains the ABC posterior for one bootstrap iteration, allowing assessment of stability and uncertainty in parameter estimates.

Examples

```
# Load ABC input data from example simulation
abc_input <- readRDS(
  system.file("extdata", "rdm_minimal", "abc", "abc_input.rds", package = "eam")
)

# Perform ABC resampling
results <- abc_resample(
```

```
  target = abc_input$target,
  param = abc_input$param,
  sumstat = abc_input$sumstat,
  n_iterations = 2,
  n_samples = 2,
  tol = 0.5,
  method = "rejection"
)

# check the abc results
str(results)
```

`abi_assess` *Assess neural estimator using trained estimator*

Description

A wrapper around `NeuralEstimators::assess()` that automatically unpacks the trained estimator and ABI input from a trained estimator object created by `abi_train`.

Usage

```
abi_assess(
  trained_estimator,
  estimator_name = NULL,
  use_gpu = TRUE,
  verbose = TRUE
)
```

Arguments

`trained_estimator`	A trained estimator object returned by `abi_train`. Must be of class `eam_abi_trained_estimator` and contain `trained_estimator` and `abi_input` elements.
`estimator_name`	Character string; optional name for the estimator (default: NULL).
`use_gpu`	Logical; whether to use GPU for assessment (default: TRUE).
`verbose`	Logical; whether to print progress information (default: TRUE).

Details

This function extracts the trained estimator and ABI input from the trained estimator object, then extracts test parameters and summary statistics from the ABI input, along with parameter names (`theta`), and passes them to `NeuralEstimators::assess()`. The test set (`theta_test` and `Z_test`) is used for assessment.

The returned object has class `eam_abi_assess`, which enables the use of S3 methods like `plot_cv_recovery` for visualization.

Value

A list with class `eam_abi_assess` containing:

estimates Data frame with columns: m, k, j, estimator, parameter, estimate, truth

runtimes Data frame with runtime information

Note

This function initializes the global Julia environment on first call.

Examples

```
## Not run:
# Train an estimator first
trained_estimator <- abi_train(
  estimator = estimator,
  abi_input = abi_input,
  epochs = 100
)

# Assess the trained estimator
assessment <- abi_assess(
  trained_estimator = trained_estimator,
  estimator_name = "MyEstimator",
  use_gpu = TRUE,
  verbose = TRUE
)

# View the assessment results
str(assessment)

# Plot parameter recovery
plot_cv_recovery(assessment)

## End(Not run)
```

`abi_estimate` *Estimate parameters using trained neural estimator*

Description

A wrapper around `NeuralEstimators::estimate()` that automatically extracts the trained estimator from a trained estimator object created by `abi_train`.

Usage

```
abi_estimate(trained_estimator, Z, X = NULL, batchsize = 32, use_gpu = TRUE)
```

Arguments

trained_estimator	A trained estimator object returned by abi_train. Must be of class eam_abi_trained_estimator and contain a trained_estimator element.
Z	Data in a format amenable to the neural-network architecture of estimator. Can be a single data set or a list of data sets.
X	Additional inputs to the neural network (default: NULL). If provided, the call will be of the form estimator((Z, X)).
batchsize	Integer; the batch size for applying estimator to Z (default: 32). Batching occurs only if Z is a list, indicating multiple data sets.
use_gpu	Logical; whether to use the GPU if available (default: TRUE).

Details

This function extracts the trained neural estimator from the trained estimator object and applies it to the provided data Z. The data Z should be in the same format as the summary statistics used during training (e.g., Z_train, Z_val, or Z_test from the ABI input).

Value

A matrix of outputs resulting from applying the trained estimator to Z (and possibly X).

Note

This function initializes the global Julia environment on first call.

Examples

```
## Not run:
# Train an estimator first
trained_estimator <- abi_train(
  estimator = estimator,
  abi_input = abi_input,
  epochs = 100
)

# Estimate parameters for test data
point_est <- abi_estimate(
  trained_estimator = trained_estimator,
  Z = abi_input$Z_test[[1]]
)

# Estimate for multiple data sets
estimates <- abi_estimate(
  trained_estimator = trained_estimator,
  Z = abi_input$Z_test,
  batchsize = 16
)

## End(Not run)
```

abi_posterior_predictive_check *ABI posterior predictive check*

Description

High-level convenience wrapper for posterior predictive checks from ABI-trained estimators.

Usage

```
abi_posterior_predictive_check(
  config,
  trained_estimator,
  estimator_type = c("point", "posterior"),
  observed_df,
  Z = NULL,
  posterior_dataset_id = 1,
  posterior_n_samples = 1000,
  n_conditions = 1,
  n_trials_per_condition = 500,
  n_items = config$n_items,
  n_conditions_per_chunk = NULL,
  output_dir = NULL,
  rt_facet_x = c("item_idx"),
  rt_facet_y = c(),
  accuracy_x = "item_idx",
  accuracy_facet_x = c(),
  accuracy_facet_y = c()
)
```

Arguments

config	Simulation configuration object.
trained_estimator	Trained ABI estimator from abi_train().
estimator_type	Character string: "point" or "posterior".
observed_df	Observed trial-level data frame.
Z	Input data for ABI estimation/sampling. If NULL, uses Z_test.
posterior_dataset_id	Dataset id used when estimator_type = "posterior".
posterior_n_samples	Number of samples when estimator_type = "posterior".
n_conditions	Number of posterior predictive conditions.

`n_trials_per_condition`
: Number of trials per condition.

`n_items`
: Number of items per trial.

`n_conditions_per_chunk`
: Number of conditions per processing chunk.

`output_dir`
: Optional output directory for simulation files.

`rt_facet_x`
: Facet columns for `plot_rt()` x facets.

`rt_facet_y`
: Facet columns for `plot_rt()` y facets.

`accuracy_x`
: Grouping variable for `plot_accuracy()` x-axis.

`accuracy_facet_x`
: Facet columns for `plot_accuracy()` x facets.

`accuracy_facet_y`
: Facet columns for `plot_accuracy()` y facets.

Details

This function is for teaching and quick demonstrations. It is intentionally specific to one estimator workflow at a time, selected by `estimator_type`. For step checks, follow these functions: `abi_estimate()` or `abi_sample_posterior()`, then `update_config_from_posterior()`, and `run_simulation()`.

This wrapper is mainly a teaching tool. It provides a compact end-to-end posterior predictive workflow for ABI, but it intentionally hides several modeling choices behind a single helper call.

For more serious work, manual posterior predictive simulation is preferred. The recommended workflow is to obtain parameter values explicitly with `abi_estimate()` for point estimators or `abi_sample_posterior()` for posterior estimators, inspect or summarise those values, rebuild a simulation configuration explicitly with `new_simulation_config` so the parameter structure is fully under your control, run the simulation with `run_simulation()`, and then compare simulated and observed data using plotting or summary functions. `update_config_from_posterior()` can still be useful for quick checks, but rebuilding the config is the safer option when you need to know exactly how inferred values are mapped back into the model. Following the steps manually makes each assumption visible, including which ABI output was used, how posterior summaries were constructed, and how the posterior predictive data were generated.

Value

`invisible(NULL)`. This function is used for plotting side effects only and prints RT and accuracy plots directly.

Examples

```
## Not run:
# Load an observed dataset and a trained ABI estimator prepared in your environment
observed_data <- sim_output$open_dataset() |>
  dplyr::filter(chunk_idx == 1, condition_idx == 1) |>
  dplyr::collect()

# Point-estimate workflow
```

```
abi_posterior_predictive_check(
  config = sim_config,
  trained_estimator = trained_point_estimator,
  estimator_type = "point",
  observed_df = observed_data,
  Z = abi_input$Z_test[[1]],
  rt_facet_x = c("item_idx"),
  rt_facet_y = c(),
  accuracy_x = "item_idx",
  accuracy_facet_x = c("ndt_beta_0"),
  accuracy_facet_y = c()
)

# Posterior-sampling workflow
abi_posterior_predictive_check(
  config = sim_config,
  trained_estimator = trained_posterior_estimator,
  estimator_type = "posterior",
  observed_df = observed_data,
  Z = abi_input$Z_test,
  posterior_dataset_id = 1,
  posterior_n_samples = 1000,
  rt_facet_x = c("item_idx"),
  rt_facet_y = c(),
  accuracy_x = "item_idx",
  accuracy_facet_x = c("ndt_beta_0"),
  accuracy_facet_y = c()
)

## End(Not run)
```

`abi_sample_posterior` *Sample from posterior distribution using trained neural estimator*

Description

A wrapper around `NeuralEstimators::sampleposterior()` that automatically extracts the trained estimator from a trained estimator object created by `abi_train` and returns posterior samples in a 3D array format.

Usage

```
abi_sample_posterior(trained_estimator, Z = NULL, N = 1000, ...)
```

Arguments

`trained_estimator`
: A trained estimator object returned by `abi_train`. Must be of class `eam_abi_trained_estimator` and contain a `trained_estimator` element.

Z	Data in a format amenable to the neural-network architecture of estimator. Can be a single data set or a list of data sets. If NULL (default), uses Z_test from the ABI input object.
N	Integer; number of approximate posterior samples to draw (default: 1000).
...	Additional keyword arguments passed to the Julia version of sampleposterior(), applicable when estimator is a likelihood-to-evidence-ratio estimator.

Details

This function extracts the trained neural posterior estimator from the trained estimator object and uses it to sample from the approximate posterior distribution given data Z. The samples are stacked using Julia's stack() function, then converted to a tibble in R for easy manipulation and summarization.

Value

A tibble of class eam_abi_posterior_samples containing posterior samples. Each row represents one posterior sample for one dataset. Columns include dataset_id (integer dataset identifier) and one column for each parameter.

Note

This function initializes the global Julia environment on first call.

Examples

```
## Not run:
# Train an estimator first
trained_estimator <- abi_train(
  estimator = estimator,
  abi_input = abi_input,
  epochs = 100
)

# Sample from posterior using test data (default)
posterior_samples <- abi_sample_posterior(
  trained_estimator = trained_estimator,
  N = 1000
)

# Sample from posterior for specific data
posterior_samples <- abi_sample_posterior(
  trained_estimator = trained_estimator,
  Z = abi_input$Z_test,
  N = 2000
)

## End(Not run)
```

abi_train *Train neural estimator using ABI input*

Description

A wrapper around `NeuralEstimators::train()` that automatically unpacks parameters and summary statistics from an ABI input object created by `build_abi_input`.

Usage

```
abi_train(
  estimator,
  abi_input,
  train_subset = "train",
  val_subset = "val",
  loss = "absolute-error",
  learning_rate = 1e-04,
  epochs = 100,
  batchsize = 32,
  savepath = NULL,
  stopping_epochs = 5,
  use_gpu = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`estimator`	A neural estimator to train, or a character string of Julia code that evaluates to an estimator. See `NeuralEstimators::train` for details.
`abi_input`	An ABI input object created by `build_abi_input`. Must contain `theta_train`, `Z_train`, `theta_val`, and `Z_val` elements.
`train_subset`	Character string specifying which subset to use for training: "train", "val", or "test" (default: "train").
`val_subset`	Character string specifying which subset to use for validation: "train", "val", or "test" (default: "val").
`loss`	Character string specifying the loss function: 'absolute-error' for mean-absolute-error loss or 'squared-error' for mean-squared-error loss (default: 'absolute-error'). Can also be a string of Julia code defining a custom loss function.
`learning_rate`	Numeric; learning rate for the ADAM optimizer (default: 1e-4).
`epochs`	Integer; number of training epochs (default: 100).
`batchsize`	Integer; batch size for stochastic gradient descent (default: 32).
`savepath`	Character string; path to save the trained estimator and training information. If NULL (default), nothing is saved.

stopping_epochs
: Integer; stop training if validation risk doesn't improve for this many epochs (default: 5).

use_gpu
: Logical; whether to use GPU if available (default: TRUE).

verbose
: Logical; whether to print training information (default: TRUE).

...
: Additional arguments passed to `NeuralEstimators::train()`.

Details

This function extracts training and validation parameters and summary statistics from the ABI input object and passes them to `NeuralEstimators::train()`. The training data (`theta_train` and `Z_train`) are used for updating the estimator via stochastic gradient descent, while the validation data (`theta_val` and `Z_val`) are used for monitoring performance and early stopping.

If `savepath` is provided, the neural network parameters will be saved as BSON files during training, along with loss values in `loss_per_epoch.csv` and the best parameters in `best_network.bson`.

Value

A list with class `eam_abi_trained_estimator` containing:

original_estimator The initial estimator before training

trained_estimator The trained neural estimator

abi_input The ABI input object used for training

Note

This function initializes the global Julia environment on first call.

Examples

```
## Not run:
# Train a neural estimator with ABI input
trained_estimator <- abi_train(
  estimator = estimator,
  abi_input = abi_input,
  epochs = 100,
  learning_rate = 1e-4,
  batchsize = 32,
  use_gpu = TRUE
)

# Train with custom save path
trained_estimator <- abi_train(
  estimator = estimator,
  abi_input = abi_input,
  epochs = 200,
  savepath = "path/to/save"
)

## End(Not run)
```

build_abc_input *Build input for Approximate Bayesian Computation (ABC)*

Description

Prepares simulation output, summary statistics, and target data for ABC analysis using the abc package. Extracts parameters and summary statistics from simulation results and formats them into matrices suitable for ABC parameter estimation.

Usage

```
build_abc_input(simulation_output, simulation_summary, target_summary, param)
```

Arguments

simulation_output	A eam_simulation_output object containing that is from run_simulation or load_simulation_output.
simulation_summary	A data frame containing summary statistics for each simulated condition. Should have a 'condition_idx' column and be created by summarise_by.
target_summary	A data frame containing target summary statistics to match against simulation results. Should have the same summary statistic columns as simulation_summary (excluding 'wider_by' columns).
param	Character vector of parameter names to extract from simulation_output. These parameters will be used as the parameter space for ABC estimation.

Details

This function provides a streamlined workflow for preparing ABC inputs, but it requires that all components be constructed using this package's functions. Specifically, simulation_output must be created by run_simulation or load_simulation_output, and both simulation_summary and target_summary must be generated using summarise_by. If your data originates from external sources or custom pipelines, you should manually construct the ABC input list instead, ensuring proper matrix formatting and column alignment as expected by abc_abc.

Value

A list with components suitable for abc_abc

Required format for summary statistics

Both simulation_summary and target_summary must be created using summarise_by. This ensures consistent column naming and data structure required for ABC analysis. See summarise_by for details on generating properly formatted summaries, and map_by_condition for typical workflow examples. If you want more flexibility in summary statistic calculation, you can manually construct the ABC input list. It is not necessary to use this function if you are familiar with the abc package.

Examples

```
# Load the example dataset
rdm_minimal_example <- system.file("extdata", "rdm_minimal", package = "eam")
sim_output <- load_simulation_output(file.path(rdm_minimal_example, "simulation"))
obs_df <- read.csv(file.path(rdm_minimal_example, "observation", "observation_data.csv"))

# Define summary statistics pipeline
summary_pipe <- summarise_by(
  .by = c("condition_idx"),
  rt_mean = mean(rt)
)

# Calculate summary statistics for simulation and observation
sim_summary <- map_by_condition(
  sim_output,
  .progress = FALSE,
  .parallel = FALSE,
  function(cond_df) {
    summary_pipe(cond_df)
  }
)
obs_summary <- summary_pipe(obs_df)

# Build ABC input
abc_input <- build_abc_input(
  simulation_output = sim_output,
  simulation_summary = sim_summary,
  target_summary = obs_summary,
  param = c("V_beta_1", "V_beta_group")
)

# Perform ABC parameter estimation using rejection method
abc_rejection_model <- abc_abc(
  abc_input = abc_input,
  tol = 0.5,
  method = "rejection"
)
```

`build_abi_input` *Build input for Amortized Bayesian Inference (ABI)*

Description

Prepares simulation output for Amortized Bayesian Inference (ABI) analysis using the `NeuralEstimators` package. Extracts parameters and summary statistics from simulation results, splits data into training and validation sets, and formats them into matrices suitable for neural network training.

Usage

```
build_abi_input(
  simulation_output,
  theta,
  Z,
  train_ratio = 0.8,
  n_test = 100,
  rank_levels = NULL
)
```

Arguments

simulation_output	A eam_simulation_output object from run_simulation or load_simulation_output.
theta	Character vector of parameter names to extract from simulation_output. These parameters will be used as the target variables for inference.
Z	Character vector of summary statistic column names to extract from the simulation output dataset (e.g., "rt", "item_idx", "choice").
train_ratio	Numeric value between 0 and 1 specifying the proportion of non-test conditions to use for training, with the remainder used for validation (default: 0.8).
n_test	Integer specifying the number of conditions to use for testing (default: 10).
rank_levels	Numeric vector specifying which rank indices to include. If NULL (default), uses all ranks from 1 to n_items from simulation config.

Details

This function provides a streamlined workflow for preparing ABI inputs. It requires that simulation_output be created by run_simulation or load_simulation_output. The function automatically handles missing trials and ranks by filling with zeros to ensure complete data matrices.

The output format is optimized for the abi package's training functions, with parameters formatted as matrices (each column is a condition) and summary statistics formatted as lists of matrices (one per condition, with trials as columns).

Value

A list with components suitable for abi package training:

theta_train Matrix of training parameters (parameters × conditions)

theta_val Matrix of validation parameters (parameters × conditions)

theta_test Matrix of test parameters (parameters × conditions)

Z_train List of matrices, one per training condition (ranks*Z × trials)

Z_val List of matrices, one per validation condition (ranks*Z × trials)

Z_test List of matrices, one per test condition (ranks*Z × trials)

train_idx Vector of condition indices used for training

val_idx Vector of condition indices used for validation

test_idx Vector of condition indices used for testing

train_ratio The training ratio used (train / non-test conditions)

n_test The number of test conditions used

rank_levels The rank levels included in Z matrices

theta Character vector of parameter names used

Z Character vector of summary statistic names used

Examples

```
# Load the example dataset
rdm_minimal_example <- system.file("extdata", "rdm_minimal", package = "eam")
sim_output <- load_simulation_output(file.path(rdm_minimal_example, "simulation"))

# build the ABI input
abi_input <- build_abi_input(
  sim_output,
  c(
    "V_beta_1",
    "V_beta_group"
  ),
  c(
    "item_idx",
    "rt",
    "choice"
  )
)

# view the structure of the ABI input
str(abi_input)

## Not run:
# Example of using the ABI input for training
# (requires NeuralEstimators package and build your estimator first, see our tutorials)
train(
  estimator,
  theta_train = abi_input$theta_train,
  theta_val = abi_input$theta_val,
  Z_train = abi_input$Z_train,
  Z_val = abi_input$Z_val,
  epochs = 500,
  stopping_epochs = 200
)

## End(Not run)
```

`load_simulation_output` *Rebuild eam_simulation_output from an existing output directory*

Description

This function reconstructs a eam_simulation_output object from a previously saved simulation output directory.

Usage

```
load_simulation_output(output_dir)
```

Arguments

output_dir	The directory containing the simulation results and config

Value

A eam_simulation_output object

Examples

```
# Load simulation output from package data
sim_output_path <- system.file(
  "extdata", "rdm_minimal", "simulation",
  package = "eam"
)
sim_output <- load_simulation_output(sim_output_path)

# Access the configuration
sim_output$simulation_config

# Access the dataset (check arrow documentation for working with the dataset)
dataset <- sim_output$open_dataset()
```

---

map_by_condition *Map a function by condition across simulation output chunks*

Description

This function processes simulation output by gathering all chunks, iterating through them one by one, filtering and collecting data by chunk, then applying a user-defined function by condition within each chunk.

Usage

```
map_by_condition(
  simulation_output,
  .f,
  ...,
  .combine = dplyr::bind_rows,
```

```
  .parallel = NULL,
  .n_cores = NULL,
  .progress = FALSE
)
```

Arguments

simulation_output	A eam_simulation_output object containing the dataset and configuration
.f	A function to apply to each condition's data. The function should accept a data frame representing one condition's results
...	Additional arguments passed to the function .f
.combine	Function to combine results (default: dplyr::bind_rows)
.parallel	Logical or NULL.
.n_cores	Integer. Number of CPU cores to use for parallel processing. If NULL, uses `detectCores() - 1`. Only used when `.parallel = TRUE`.
.progress	Logical, whether to show a progress bar (default: FALSE)

Details

This function handles out-of-core computation automatically using Apache Arrow, so you don't need to understand Arrow internals. It loads data chunk by chunk to avoid memory issues with large simulations.

If you prefer to manually work with the raw Arrow dataset, you can access it via `simulation_output$open_dataset()`, which returns an Arrow Dataset object. You can then use `dplyr` verbs to filter and query before calling `dplyr::collect()` to load data into memory.

Value

A list containing the results of applying .f to each condition, with names corresponding to condition indices

Examples

```
# Load simulation output
sim_output_path <- system.file(
  "extdata", "rdm_minimal", "simulation",
  package = "eam"
)
sim_output <- load_simulation_output(sim_output_path)

# Define a summary pipeline
summary_pipe <- summarise_by(
  .by = c("condition_idx"),
  rt_mean = mean(rt),
  rt_quantiles = quantile(rt, probs = c(0.1, 0.5, 0.9))
)

# Apply function to each condition
```

```
sim_sumstat <- map_by_condition(
  sim_output,
  .progress = FALSE,
  .parallel = FALSE,
  function(cond_df) {
    summary_pipe(cond_df)
  }
)
```

`new_simulation_config` *Create a new simulation configuration*

Description

This function creates a new eam simulation configuration object that contains all parameters needed to run a simulation.

Usage

```
new_simulation_config(
  prior_params = list(),
  prior_formulas = list(),
  between_trial_formulas = list(),
  item_formulas = list(),
  n_conditions_per_chunk = NULL,
  n_conditions,
  n_trials_per_condition,
  n_items,
  max_reached = n_items,
  max_t,
  dt = 0.001,
  noise_mechanism = "add",
  noise_factory = NULL,
  model = "ddm",
  parallel = FALSE,
  n_cores = NULL,
  rand_seed = NULL
)
```

Arguments

`prior_params`	A list or data frame of initial values for prior
`prior_formulas`	A list of formulas defining prior distributions for condition-level parameters
`between_trial_formulas`	A list of formulas defining between-trial parameters
`item_formulas`	A list of formulas defining item-level parameters

`n_conditions_per_chunk`
: Number of conditions to process per chunk (optional, typically does not need to be set. It determine the storage and in-memory size of each chunk, if you find memory issues, try reducing this number)

`n_conditions`
: Total number of conditions to simulate

`n_trials_per_condition`
: Number of trials per condition

`n_items`
: Number of items per trial

`max_reached`
: Maximum number of items that can be recalled (default: n_items)

`max_t`
: Maximum simulation time

`dt`
: Time step size (default: 0.001)

`noise_mechanism`
: Noise mechanism ("add", "mult_evidence", or "mult_t", default: "add")

`noise_factory`
: Function that creates noise functions.

`model`
: Model name or backend names (e.g., "ddm", "rdm", "lca")

`parallel`
: Whether to run in parallel (default: FALSE)

`n_cores`
: Number of cores for parallel processing (default: NULL, auto-detect)

`rand_seed`
: Random seed for parallel processing (default: NULL)

Details

This function only creates the configuration object and does not run the simulation. To actually execute the simulation, you must pass the returned configuration object to `run_simulation`.

Supported Models:

This package supports three evidence accumulation models. The appropriate backend is automatically selected based on the `model` parameter and the parameters defined in your formulas.

DDM (Drift Diffusion Model) Models evidence accumulation towards a single upper threshold. Items either reach the threshold and are recalled, or time out.

Required parameters (must appear in `prior_formulas`, `between_trial_formulas`, or `item_formulas`):

- `A` - Upper decision boundary/threshold
- `V` - Drift rate (evidence accumulation rate)
- `Z` - Starting point of evidence
- `ndt` - Non-decision time

Set `model = "ddm"`

RDM (Racing Diffusion Model) Models multiple racing evidence accumulators, each with upper and lower thresholds for binary decisions (correct/incorrect).

Required parameters:

- `A_upper` - Upper decision boundary (correct response)
- `A_lower` - Lower decision boundary (incorrect response)
- `V` - Drift rate
- `Z` - Starting point of evidence

- ndt - Non-decision time

Set model = "rdm". Note: If you set model = "ddm" but define A_upper instead of A, the model will automatically switch to RDM.

LCA (Leaky Competing Accumulator) Models competitive evidence accumulation with leakage and mutual inhibition between accumulators.

Required parameters:

- A - Decision threshold
- V - Input strength/drift rate
- Z - Starting point of evidence
- ndt - Non-decision time
- beta - Self-excitation/leak parameter
- k - Lateral inhibition strength

Set model = "lca"

LFM (Lévy Flight Model) Uses the same parameters as DDM. See DDM above.

Set model = "lfm"

LBA (Linear Ballistic Accumulator) Uses the same parameters as RDM. See RDM above.

Set model = "lba"

Note: All required parameters must be defined at least once across prior_params, prior_formulas, between_trial_formulas, and item_formulas.

Parameter Hierarchy and Formula Evaluation:

The simulation uses a hierarchical parameter system with sequential formula evaluation, allowing later formulas to reference earlier ones:

1. **prior_params** - Initial constant values available to all formulas
2. **prior_formulas** - Evaluated once per condition, can reference prior_params. Use for condition-level parameters that vary across conditions.
3. **between_trial_formulas** - Evaluated once per trial within each condition. Can reference both prior_params and variables from prior_formulas. Use for trial-level variability.
4. **item_formulas** - Evaluated once per item within each trial. Can reference all previous parameters. Use for item-specific parameters.

Using Distributions:

You can use the distributional package to define random parameters. For example:

- A ~ distributional::dist_uniform(0.5, 2.0) - Uniform distribution
- V_condition ~ distributional::dist_normal(1.0, 0.2) - Normal distribution
- sigma ~ 0.5 - Constant value
- V ~ distributional::dist_normal(V_condition, sigma) - Reference earlier parameters

Each formula is evaluated sequentially, so you can build complex parameter dependencies. For instance, you might define a base drift rate V in prior_formulas, then add trial-level noise in between_trial_formulas, and finally scale by item position in item_formulas.

Value

An S3 object of class `eam_simulation_config` containing validated simulation parameters. This object should be passed to `run_simulation` to execute the simulation.

Examples

```
# Define formulas for the simulation
prior_formulas <- list(
  V ~ distributional::dist_uniform(0.1, 1.0),
  ndt ~ 0.3,
  noise_coef ~ 1
)

between_trial_formulas <- list()

item_formulas <- list(
  A_upper ~ 1,
  A_lower ~ -1,
  V ~ V
)

# Define noise factory
noise_factory <- function(context) {
  noise_coef <- context$noise_coef
  function(n, dt) {
    noise_coef * rnorm(n, mean = 0, sd = sqrt(dt))
  }
}

# Create configuration
config <- new_simulation_config(
  prior_formulas = prior_formulas,
  between_trial_formulas = between_trial_formulas,
  item_formulas = item_formulas,
  n_conditions = 10,
  n_trials_per_condition = 10,
  n_items = 5,
  max_reached = 5,
  max_t = 10,
  dt = 0.01,
  noise_mechanism = "add",
  noise_factory = noise_factory,
  model = "ddm",
  parallel = FALSE
)

# print the config
config

# Run simulation
sim_output <- run_simulation(config)
sim_output
```

plot_accuracy *Plot accuracy comparison between posterior and observed data*

Description

Visualizes accuracy metrics comparing posterior simulation results with observed data. Creates side-by-side bar plots for easy comparison across conditions.

Usage

```
plot_accuracy(
  simulated_output,
  observed_df,
  x = "item_idx",
  facet_x = c(),
  facet_y = c()
)
```

Arguments

simulated_output	Posterior simulation output from run_simulation()
observed_df	Observed data frame
x	Variable for x-axis (default: "item_idx")
facet_x	Variables for faceting columns
facet_y	Variables for faceting rows

Value

A ggplot2 object

Examples

```
# Load posterior simulation output and observed data
base_dir <- system.file("extdata", "rdm_minimal", package = "eam")
post_output <- load_simulation_output(file.path(base_dir, "abc", "posterior", "neuralnet"))
obs_df <- read.csv(file.path(base_dir, "observation", "observation_data.csv"))

# Plot accuracy comparison between posterior and observed data
# The plot shows side-by-side bars comparing hit rates or accuracy
plot_accuracy(
  post_output,
  obs_df,
  facet_x = c("group")
)
```

plot_cv_pair_correlation *Plot CV parameter pair correlations*

Description

Create a matrix of pairwise plots for cross-validation parameter estimates, including scatter plots with fitted trends, rank correlations, and marginal distributions.

Usage

```
plot_cv_pair_correlation(data, ...)

## S3 method for class 'cv4abc'
plot_cv_pair_correlation(data, ...)
```

Arguments

data	A cv4abc object containing true parameters and cross-validated estimates.
...	Additional arguments:
	interactive Logical; whether to pause between tolerance levels and wait for input

Value

Invisibly returns 'NULL'. Called for its side effect of producing plots.

See Also

plot_cv_pair_correlation.cv4abc

Examples

```
# Load CV output from saved file
cv_file <- system.file(
  "extdata", "rdm_minimal", "abc", "cv", "neuralnet.rds",
  package = "eam"
)
abc_neuralnet_cv <- readRDS(cv_file)

# Plot parameter pair correlations
plot_cv_pair_correlation(abc_neuralnet_cv)
```

plot_cv_recovery *Plot CV parameter recovery*

Description

Visualize parameter recovery from cross-validation results, showing estimated vs. true parameter values and residual distributions for each parameter.

Usage

```
plot_cv_recovery(data, ...)

## S3 method for class 'cv4abc'
plot_cv_recovery(data, ...)

## S3 method for class 'eam_abi_assess'
plot_cv_recovery(data, ...)

## S3 method for class 'eam_abi_posterior_samples'
plot_cv_recovery(data, trained_estimator = NULL, theta = NULL, ...)
```

Arguments

data	An eam_abi_posterior_samples object from abi_sample_posterior containing posterior samples with columns dataset_id and parameter columns. The median of each parameter for each dataset is used as the point estimate for recovery assessment.
...	Additional arguments: **n_rows** Integer; number of rows in the plot grid (default: 3) **n_cols** Integer; number of columns in the plot grid, multiplied by 2 for paired plots (default: 1) **method** Character; smoothing method for geom_smooth (default: "lm") **formula** Formula; used in geom_smooth (default: y ~ x) **resid_tol** Numeric; quantile threshold for filtering residuals by absolute value. If specified, only observations with residuals below this quantile are plotted (default: NULL, no filtering) **interactive** Logical; whether to pause between pages and wait for user input (default: FALSE)
trained_estimator	Optional. A trained estimator object returned by abi_train. If provided, the true parameter values are extracted from trained_estimatorabi_inputtheta_test. Either trained_estimator or theta must be provided, but not both.
theta	Optional. A matrix of true parameter values with parameters as rows and datasets as columns. Column count must match the number of unique dataset_id values in data. Either trained_estimator or theta must be provided, but not both.

Value

Invisibly returns ‘NULL‘. Called for its side effect of producing plots.

See Also

plot_cv_recovery.cv4abc, plot_cv_recovery.eam_abi_assess, plot_cv_recovery.eam_abi_posterior_samples

Examples

```
# Load CV output from saved file
cv_file <- system.file(
  "extdata", "rdm_minimal", "abc", "cv", "neuralnet.rds",
  package = "eam"
)
abc_neuralnet_cv <- readRDS(cv_file)

# Plot parameter recovery
plot_cv_recovery(
  abc_neuralnet_cv,
  n_rows = 2,
  n_cols = 1,
  resid_tol = 0.99
)

## Not run:
# Train a posterior estimator
trained_estimator <- abi_train(
  estimator = posterior_estimator,
  abi_input = abi_input,
  epochs = 50
)

# Sample from posterior using test data (default)
posterior_samples <- abi_sample_posterior(
  trained_estimator = trained_estimator,
  N = 1000
)

# Plot recovery using trained_estimator to get true values
plot_cv_recovery(
  posterior_samples,
  trained_estimator = trained_estimator
)

# Alternatively, provide true parameter values directly
plot_cv_recovery(
  posterior_samples,
  theta = abi_input$theta_test
)

## End(Not run)
```

plot_posterior_parameters *Plot parameter posterior distributions*

Description

Plotting posterior distributions (and optionally prior distributions) from ABC results.

Usage

```
plot_posterior_parameters(data, ...)

## S3 method for class 'abc'
plot_posterior_parameters(data, abc_input = NULL, ...)
```

Arguments

data	An abc object containing posterior samples in adj.values or unadj.values.
...	Additional arguments: **n_rows** Integer; number of rows in the plot grid (default: 2) **n_cols** Integer; number of columns in the plot grid (default: 2) **interactive** Logical; whether to pause between pages and wait for input
abc_input	Optional abc_input object containing prior samples for comparison.

Value

Invisibly returns 'NULL'. Called for its side effect of producing plots.

See Also

plot_posterior_parameters.abc

Examples

```
# Load ABC output from saved file
abc_file <- system.file(
  "extdata", "rdm_minimal", "abc", "abc_rejection_model.rds",
  package = "eam"
)
abc_rejection_model <- readRDS(abc_file)

# Load ABC input for prior comparison
abc_input_file <- system.file(
  "extdata", "rdm_minimal", "abc", "abc_input.rds",
  package = "eam"
)
abc_input <- readRDS(abc_input_file)
```

```
# Plot posterior distributions with prior comparison
plot_posterior_parameters(abc_rejection_model, abc_input)
```

`plot_resample_forest` *Plot resample forest plots*

Description

Create forest plots showing parameter ranges across resample iterations. Each iteration is displayed as a horizontal line with quantile intervals.

Usage

```
plot_resample_forest(
  data,
  n_rows = 2,
  n_cols = 2,
  interactive = FALSE,
  ci_level = 0.95
)
```

Arguments

`data`	List of abc results from abc_resample
`n_rows`	Number of rows in plot grid (default 2)
`n_cols`	Number of columns in plot grid (default 2)
`interactive`	Whether to pause between pages (default FALSE)
`ci_level`	quantile intervals (default 0.95 for 95% interval)

Value

No return value, called for side effects (plotting). Creates forest plots displayed in the graphics device.

Examples

```
# Load ABC input data from example simulation
abc_input <- readRDS(
  system.file("extdata", "rdm_minimal", "abc", "abc_input.rds", package = "eam")
)

# Perform ABC resampling
results <- abc_resample(
  target = abc_input$target,
  param = abc_input$param,
```

```
  sumstat = abc_input$sumstat,
  n_iterations = 100,
  n_samples = 100,
  tol = 0.5,
  method = "rejection"
)

# plot forest plots showing parameter ranges
plot_resample_forest(results, ci_level = 0.95)
```

`plot_resample_medians` *Plot resample median distributions*

Description

Plot density distributions of parameter medians across resample iterations.

Usage

```
plot_resample_medians(data, n_rows = 2, n_cols = 2, interactive = FALSE)
```

Arguments

`data`	List of abc results from abc_resample
`n_rows`	Number of rows in plot grid (default 2)
`n_cols`	Number of columns in plot grid (default 2)
`interactive`	Whether to pause between pages (default FALSE)

Value

No return value, called for side effects (plotting). Creates density plots displayed in the graphics device.

Examples

```
# Load ABC input data from example simulation
abc_input <- readRDS(
  system.file("extdata", "rdm_minimal", "abc", "abc_input.rds", package = "eam")
)

# Perform ABC resampling
results <- abc_resample(
  target = abc_input$target,
  param = abc_input$param,
  sumstat = abc_input$sumstat,
  n_iterations = 100,
  n_samples = 100,
  tol = 0.5,
```

```
  method = "rejection"
)

# plot the resample medians for each parameter
plot_resample_medians(results)
```

`plot_rt` *Plot reaction time distributions*

Description

Visualize reaction time distributions from your model predictions. Overlay observed experimental data for reference.

Usage

```
plot_rt(simulated_output, observed_df, facet_x = c("item_idx"), facet_y = c())
```

Arguments

`simulated_output`	Output from `run_simulation` containing posterior predictions
`observed_df`	Your observed data as a data frame
`facet_x`	Variables to split plots horizontally. Default is `"item_idx"` to show separate plots for each item
`facet_y`	Variables to split plots vertically. Default is none (`c()`)

Details

Posterior predictions are plotted directly at the trial level. This pools all simulated trials for the requested facets without condition-level aggregation.

Value

A plot showing predicted RT distributions (blue), with observed data (red) if provided

Examples

```
# Load example posterior simulation output
post_output_path <- system.file(
  "extdata", "rdm_minimal", "abc", "posterior", "neuralnet",
  package = "eam"
)
post_output <- load_simulation_output(post_output_path)

# Load example observed data
obs_file <- system.file(
  "extdata", "rdm_minimal", "observation", "observation_data.csv",
```

```
  package = "eam"
)
obs_df <- read.csv(obs_file)

# Plot RT distributions by item
plot_rt(post_output, obs_df, facet_x = c("item_idx"))

# Plot RT distributions by item and group
plot_rt(
  post_output,
  obs_df,
  facet_x = c("item_idx"),
  facet_y = c("group")
)
```

`print.eam_simulation_config` *Print method for eam simulation configuration*

Description

Print method for eam simulation configuration

Usage

```
## S3 method for class 'eam_simulation_config'
print(x, ...)
```

Arguments

`x`	A eam_simulation_config object
`...`	Additional arguments (ignored)

Value

Invisibly returns the input object

run_simulation *Run a simulation with specified configuration*

Description

This function runs a complete simulation based on the provided eam_simulation_config object, which is generated by the new_simulation_config function.

Usage

```
run_simulation(config, output_dir = NULL)
```

Arguments

config	A eam_simulation_config object containing all simulation parameters, you should use new_simulation_config to create one.
output_dir	The directory to save out-of-core results (optional, will use temp directory if not provided)

Details

This function uses an out-of-core approach to handle potentially large simulation results. Instead of returning a data frame directly, it persists the data to disk and returns an eam_simulation_output object that contains metadata and file system paths.

To access the simulation data, use the following methods on the returned object:

- open_dataset() - Returns an Arrow Dataset containing the simulation results, e.g. sim_output$open_dataset()
- open_evaluated_conditions() - Returns an Arrow Dataset containing the evaluated condition parameters, e.g. sim_output$open_evaluated_conditions()

Both methods return Arrow Dataset objects rather than data frames, allowing for efficient querying and filtering before loading data into memory. To convert to a data frame, use dplyr::collect() or as.data.frame().

Throughout this package, the eam_simulation_output object is used as the standard parameter for downstream analysis functions, rather than passing Arrow objects or data frames directly.

For multi-item backends, at each discrete time point, only one item can reach the threshold. The precision of this detection depends on the dt parameter. This design choice was made for performance considerations. For almost all experimental scenarios, it is negligible. But users should be aware of this limitation, if it is critical, try to increase the temporal resolution by reducing dt. For implementation details, refer to the backend source code (accumulate_evidence_* functions).

Value

A S3 object of class eam_simulation_output containing the output information

Examples

```
# Define formulas for the simulation
prior_formulas <- list(
  V ~ distributional::dist_uniform(0.1, 1.0),
  ndt ~ 0.3,
  noise_coef ~ 1
)

between_trial_formulas <- list()

item_formulas <- list(
  A_upper ~ 1,
  A_lower ~ -1,
  V ~ V
)

# Define noise factory
noise_factory <- function(context) {
  noise_coef <- context$noise_coef
  function(n, dt) {
    noise_coef * rnorm(n, mean = 0, sd = sqrt(dt))
  }
}

# Create configuration
config <- new_simulation_config(
  prior_formulas = prior_formulas,
  between_trial_formulas = between_trial_formulas,
  item_formulas = item_formulas,
  n_conditions = 10,
  n_trials_per_condition = 10,
  n_items = 5,
  max_reached = 5,
  max_t = 10,
  dt = 0.01,
  noise_mechanism = "add",
  noise_factory = noise_factory,
  model = "ddm",
  parallel = FALSE
)

# Run simulation
sim_output <- run_simulation(config)

# Access results
dataset <- sim_output$open_dataset()
dataset # an arrow dataset object

# if you want to load it into memory, you can use:
df <- as.data.frame(dataset)
head(df)
```

```
# Access evaluated condition parameters
cond_dataset <- sim_output$open_evaluated_conditions()
df_cond <- as.data.frame(cond_dataset)
head(df_cond)
```

---

summarise_by *Summarise data by groups with optional pivoting*

Description

This function provides a flexible way to group data, compute summary statistics, and reshape results. It works similar to ‘dplyr::summarise()‘ but with added capabilities for pivoting results wider.

Usage

```
summarise_by(
  .data = NULL,
  ...,
  .by = c("condition_idx"),
  .wider_by = c("condition_idx")
)
```

Arguments

.data	A data frame to summarise, or NULL to create a reusable summary function
...	Summary expressions using dplyr-style syntax. Named arguments become column names in the output (e.g., ‘mean_rt = mean(rt)‘).
.by	Character vector of grouping column names. Default is "condition_idx".
.wider_by	Character vector of columns to keep as identifiers when pivoting. Default is "condition_idx". Must be a subset of ‘.by‘. When ‘.wider_by‘ differs from ‘.by‘, the extra columns in ‘.by‘ will be spread across as column suffixes.

Details

You can use ‘summarise_by()‘ in two ways: 1. **Direct use**: Pass your data directly and get results immediately 2. **Build-then-apply**: Create reusable summary functions, combine them with ‘+‘, then apply to your data later

The build-then-apply approach is useful when you want to compute different types of summaries (e.g., RT statistics and accuracy statistics) and automatically join them together.

Value

- If ‘.data‘ is provided: A data frame with summarised results - If ‘.data‘ is NULL: A function that can be applied to data later

Usage with ABC workflows

If you plan to use build_abc_input for ABC analysis, you must use summarise_by() to generate summary statistics (or manually handle the arrow output format). This function typically works together with map_by_condition to process simulation results. See map_by_condition for workflow examples.

Examples

```
# Example 1: Direct use - pass data and get results immediately
trial_data <- data.frame(
  condition_idx = rep(1:2, each = 4),
  item_idx = rep(1:2, 4),
  rt = c(0.5, 0.6, 0.7, 0.8, 0.55, 0.65, 0.75, 0.85),
  accuracy = c(1, 1, 0, 1, 1, 0, 1, 1)
)

# Compute mean RT and accuracy by condition and item
result <- summarise_by(
  trial_data,
  mean_rt = mean(rt),
  mean_acc = mean(accuracy),
  .by = c("condition_idx", "item_idx"),
  .wider_by = "condition_idx"
)
# Result has columns: condition_idx, mean_rt_item_idx_1, mean_rt_item_idx_2, etc.
result

# Example 2: Build-then-apply - create reusable summary functions
# Build separate summary functions for different statistics
rt_summary_pipe <- summarise_by(
  mean_rt = mean(rt),
  sd_rt = stats::sd(rt),
  .by = c("condition_idx", "item_idx"),
  .wider_by = "condition_idx"
)

acc_summary_pipe <- summarise_by(
  mean_acc = mean(accuracy),
  n_trials = length(accuracy),
  .by = c("condition_idx", "item_idx"),
  .wider_by = "condition_idx"
)

# Combine with + and apply to data
combined_summary_pipe <- rt_summary_pipe + acc_summary_pipe
result <- combined_summary_pipe(trial_data)
# Result has all summaries joined by condition_idx
result
```

summarise_posterior_parameters
Summarise posterior parameter distributions

Description

Compute summary statistics (mean, median, confidence intervals) for posterior parameters from ABC results.

Usage

```
summarise_posterior_parameters(data, ...)

## S3 method for class 'abc'
summarise_posterior_parameters(data, ..., ci_level = 0.95)

## S3 method for class 'eam_abi_posterior_samples'
summarise_posterior_parameters(data, ..., ci_level = 0.95)
```

Arguments

data	An eam_abi_posterior_samples object (tibble) containing posterior samples with columns dataset_id and parameter columns.
...	Additional arguments for custom summary functions. Functions passed as named arguments will be applied to each parameter's posterior samples.
ci_level	Numeric; confidence interval level (default: 0.95).

Value

A data frame with summary statistics for each parameter.

See Also

summarise_posterior_parameters.abc, summarise_posterior_parameters.eam_abi_posterior_samples

Examples

```
# Load ABC output from saved file
abc_file <- system.file(
  "extdata", "rdm_minimal", "abc", "abc_rejection_model.rds",
  package = "eam"
)
abc_rejection_model <- readRDS(abc_file)

# Summarise posterior distributions
summarise_posterior_parameters(abc_rejection_model)

# Custom confidence interval level
```

```
summarise_posterior_parameters(abc_rejection_model, ci_level = 0.90)
```

`summarise_resample_medians` *Summarise resample medians*

Description

Calculate summary statistics for parameter medians across resample iterations. Returns mean, median, and confidence intervals of the median distributions.

Usage

```
summarise_resample_medians(data, ..., ci_level = 0.95)
```

Arguments

`data`	List of abc results from abc_resample
`...`	Additional custom summary functions (named functions)
`ci_level`	Confidence level for intervals (default 0.95)

Value

Data frame with summary statistics for each parameter

Examples

```
# Load ABC input data from example simulation
abc_input <- readRDS(
  system.file("extdata", "rdm_minimal", "abc", "abc_input.rds", package = "eam")
)

# Perform ABC resampling
results <- abc_resample(
  target = abc_input$target,
  param = abc_input$param,
  sumstat = abc_input$sumstat,
  n_iterations = 100,
  n_samples = 100,
  tol = 0.5,
  method = "rejection"
)

# summarise the resample medians
summary_stats <- summarise_resample_medians(results, ci_level = 0.95)
print(summary_stats)
```

update_config_from_posterior *Update a simulation config with posterior parameter values*

Description

Applies one posterior draw to an existing simulation configuration. For each name in `posterior_params`: any matching entry in `prior_params` is removed, any matching formula in `prior_formulas` is dropped, and the posterior value is appended to `prior_params` as a fixed constant.

Usage

```
update_config_from_posterior(
  config,
  posterior_params,
  n_conditions_per_chunk = NULL,
  n_conditions = config$n_conditions,
  n_trials_per_condition = config$n_trials_per_condition,
  n_items = config$n_items
)
```

Arguments

`config`	An `eam_simulation_config` object.
`posterior_params`	A named list or data frame of posterior parameter values representing exactly one posterior draw.
`n_conditions_per_chunk`	Number of conditions per processing chunk. `NULL` (default) recomputes the value via the internal heuristic.
`n_conditions`	Total number of conditions to simulate. Defaults to the value already stored in `config`.
`n_trials_per_condition`	Number of trials per condition. Defaults to the value already stored in `config`.
`n_items`	Number of items per trial. Defaults to the value already stored in `config`.

Value

A modified `eam_simulation_config` with updated `prior_params`, pruned `prior_formulas`, and the four simulation-dimension fields.

Note

This helper is intentionally conservative and mainly for teaching, demonstrations, and quick posterior predictive checks. It freezes selected top-level parameters to fixed posterior values for convenience, but it does not reconstruct or reinterpret the full dependency structure of the simulation

specification. If `config$prior_params` is a data frame with multiple rows, the single posterior draw is broadcast across those rows when inserted.

It does not re-route backend selection and does not create a new model. Parameters that appear on the left-hand side of `between_trial_formulas` or `item_formulas` cannot be replaced automatically. If you need full control and clarity over internal parameter structure, rebuild the configuration manually using `new_simulation_config`.

Examples

```
# Load example simulation output and extract its config
base_dir <- system.file("extdata", "rdm_minimal", package = "eam")
sim_output <- load_simulation_output(file.path(base_dir, "simulation"))
sim_config <- sim_output$simulation_config

# Create a simple one-draw posterior parameter data frame
posterior_params <- data.frame(
  V_beta_1 = -0.15
)

# Update the config by replacing the matching prior entry/formula
updated_config <- update_config_from_posterior(
  config = sim_config,
  posterior_params = posterior_params,
  n_conditions = 1,
  n_trials_per_condition = 500
)

# Inspect the updated fixed prior values
updated_config$prior_params
```

Index