

Package: stanli (via r-universe)

October 11, 2026

Type Package

Title The Stan Language Interpreter

Version 0.20.0

Description Compile and sample Stan models without a C++ toolchain.

Stan models are lowered to a graph of operations over precompiled 'stan-math' kernels rather than translated to C++ and compiled, so preparing a model takes milliseconds instead of minutes. Provides adaptive Hamiltonian Monte Carlo with the diagnostics a Bayesian workflow needs (rank-normalized split R-hat, bulk and tail effective sample size, E-BFMI), and L-BFGS optimization. The runtime library is downloaded on first use rather than bundled; see `stanli_install()`.

License BSD_3_clause + file LICENSE

URL <https://github.com/seantalts/stanli>

BugReports <https://github.com/seantalts/stanli/issues>

Encoding UTF-8

Depends R (>= 4.0)

Imports utils, tools, methods, stats

Suggests V8, jsonlite, posterior, bayesplot, loo, rstantools, tidybayes, callr, rstan, cmdstanr, shinystan, brms, testthat (>= 3.0.0)

Additional_repositories <https://stan-dev.r-universe.dev>

NeedsCompilation yes

Config/roxygen2/version 8.1.0

Repository <https://seantalts.r-universe.dev>

Date/Publication 2026-10-11 12:10:06 UTC

RemoteUrl <https://github.com/seantalts/stanli>

RemoteRef v0.20.0

RemoteSha b92871e168db729f595725c9008eb90477f03ef6

RemoteSubdir r

Contents

as_cstanfit	2
as_draws_array	3
as_rfit	4
as_stanfit	5
cstan_model	7
laplace_model	8
log_lik	9
log_prob_grad	10
optimize_model	10
pathfinder_model	11
sample_cstan	12
sample_model	14
stanli-ecosystem	15
stanli_available	16
stanli_check_syntax	17
stanli_diagnose	17
stanli_exact_lp	18
stanli_install	18
stanli_model	19
stanli_runtime_path	20
stanli_thread_safe	20
summary.stanli_fit	21
unconstrain	21
variational_model	22
Index	24

as_cstanfit	<i>Native CmdStanR-style fit interface</i>
-------------	--

Description

Wrap stored Stanli draws in a native object with CmdStanR-style methods. Neither cmdstanr nor rstan is needed. The original fit is unchanged.

Usage

```
as_cstanfit(x)

## S3 method for class 'stanli_cstanfit'
summary(object, variables = NULL, ...)

## S3 method for class 'stanli_cstanfit'
print(x, variables = NULL, n = 10L, ...)
```

Arguments

x	A 'stanli_fit' or an existing 'stanli_cstanfit'.
object	A 'stanli_cstanfit'.
variables	Variables to summarize.
...	Arguments passed to 'posterior::summarise_draws()'.
n	Maximum number of summary rows to print (default 10).

Details

* '\$draws(variables = NULL, inc_warmup = FALSE, format = "draws_array")' returns posterior draws, including 'lp__' and excluding sampler diagnostics. Base names select all indexed elements. Supported formats are 'draws_array', 'draws_matrix', 'draws_df', 'draws_list', and 'draws_rvars'. * '\$summary(variables = NULL, ...)' delegates to 'posterior::summarise_draws()', including custom functions and formulas. It uses rank-normalized R-hat and bulk/tail ESS by default, as CmdStanR does. * '\$sampler_diagnostics(inc_warmup = FALSE, format = "draws_array")' returns the six sampler diagnostics in stable alphabetical order; CmdStanR's CSV order may differ. * '\$metadata()' includes iteration counts ***before*** thinning, 'thin', 'save_warmup', variable names, chain IDs, and sampler settings. * '\$num_chains()' returns the number of chains. * '\$time()' returns 'total' and a 'chains' data frame with 'chain_id', 'warmup', 'sampling', and 'total' seconds. Unavailable timings are 'NA'; overall wall time is unavailable, rather than inferred from chain timings. * '\$loo(variables = "log_lik", r_eff = TRUE, moment_match = FALSE, ...)' computes LOO with optional relative efficiency. Moment matching is unsupported. * '\$stanli_fit()' retrieves the original fit for native Stanli operations.

Draws and summaries require posterior; LOO also requires loo. Stored-draw methods work after serialization without a runtime or live model. The Stanli R package must remain installed. Requests for unsaved warmup fail explicitly. Model recompilation, CSV output, and CmdStan's model methods are not provided. Consumers that dispatch on CmdStanR class names must explicitly accept this class or call the methods directly.

Value

A 'stanli_cstanfit' with methods described below. This is a native list of functions, not an R6 CmdStanMCMC or RStan S4 object.

as_draws_array	<i>Convert a fit to a draws array</i>
----------------	---------------------------------------

Description

Returns an object the posterior package understands: a 'draws_array' with dimensions (iteration, chain, variable). Without posterior installed this is still a plain array with the same dimensions and dimnames, which is what most consumers actually read.

Usage

```
as_draws_array(x, include_sampler = FALSE, inc_warmup = FALSE)
```

Arguments

`x` A 'stanli_fit'.

`include_sampler` Append the seven sampler columns ('lp__', 'accept_stat__', ...), which is what a CSV reader would see.

`inc_warmup` Include saved warmup draws. Defaults to 'FALSE'.

Value

A 'draws_array', or a plain array when posterior is absent.

as_rfit	<i>Native R fit interface</i>
---------	-------------------------------

Description

Give a Stanli fit an RStan-style summary and extraction interface without loading RStan. The arrays, live model, and ecosystem integrations are kept. Existing 'stanli_fit' summary behavior is unchanged until explicitly converted.

Usage

```
as_rfit(x)

extract(
  object,
  pars = NULL,
  permuted = TRUE,
  inc_warmup = FALSE,
  include = TRUE,
  ...
)

## S3 method for class 'stanli_rfit'
summary(object, pars = NULL, probs = c(0.025, 0.25, 0.5, 0.75, 0.975), ...)

get_sampler_params(object, inc_warmup = TRUE)

get_elapsed_time(object)
```

Arguments

`x, object` A 'stanli_fit' or 'stanli_rfit'.

`pars` Parameter names or indexed element names. Defaults to all outputs, including 'lp__'. Base names select the complete vector or array.

`permuted` Merge post-warmup chains using stored per-chain permutations.

inc_warmup	Include saved warmup. Ignored for permuted extraction.
include	Select 'pars' when TRUE, or exclude them when FALSE.
...	Reserved; unsupported arguments produce an error.
probs	Quantile probabilities.

Details

'extract()' returns a named list of draw-first arrays when permuted, or an iteration-by-chain-by-parameter array otherwise. 'as_rfit()' creates per-chain permutations using R's RNG; extraction from the converted fit does not advance that RNG. All parameters share the same permutations, preserving joint draws.

'summary()' returns 'summary' and 'c_summary' matrices/arrays in RStan's layout. 'n_eff' and 'Rhat' use posterior's basic ESS and split R-hat; these are distinct from Stanli's usual bulk ESS and rank-normalized R-hat. posterior's ESS algorithm also differs from RStan's legacy ESS estimator, so 'n_eff' and 'se_mean' are not numerically identical to RStan's summary. Summary requires optional posterior but no native runtime. Sampler and timing extraction require neither posterior nor a native runtime.

This is a native S3 class, not an RStan S4 object. Calls explicitly written as 'rstan::extract()' still require RStan. Use 'stanli::extract()' instead. Packages inspecting RStan slots or requiring its C++ instance need an integration or the optional [as_stanfit()] conversion. The live model pointer does not survive serialization; stored-draw operations do.

Value

'as_rfit()' returns a 'stanli_rfit' inheriting from 'stanli_fit'. 'get_sampler_params()' returns one matrix per chain, excluding 'lp__'. 'get_elapsed_time()' returns a chain-by-phase matrix in seconds.

as_stanfit

Convert draws to an RStan fit without compilation

Description

Build a 'stanli_stanfit', an S4 subclass of 'rstan::stanfit', directly from stored arrays. Requires optional rstan but never compiles a C++ model.

Usage

```
as_stanfit(x, ...)
```

```
## S3 method for class 'stanli_fit'
```

```
as_stanfit(x, model = x$model, exclude = NULL, ...)
```

```
## S3 method for class 'stanli_stanfit'
```

```
as_stanfit(x, model, ...)
```

Arguments

<code>x</code>	A fitted object. A <code>'stanli_fit'</code> must contain the sampling metadata recorded by the current <code>[sample_model()]</code> . A <code>'stanli_stanfit'</code> is returned unchanged unless a replacement <code>'model'</code> is supplied.
<code>...</code>	Reserved for methods.
<code>model</code>	A <code>'stanli_model'</code> made with the original code and data. Defaults to the model retained by a <code>'stanli_fit'</code> . Use <code>'NULL'</code> for a draws-only result. A replacement must match the fit's output columns and parameter count.
<code>exclude</code>	Names of variables to leave out of the stored draws, as <code>'pars'</code> with <code>'include = FALSE'</code> does in <code>'rstan::sampling()'</code> . Every element of a named container is dropped. The live model is unaffected.

Details

Extraction, summaries, plots, and ordinary LOO inherit RStan's methods. With a live model, `'rstan::log_prob'`, `'grad_log_prob'`, `'constrain_pars'`, and `'unconstrain_pars'` call `stanli` directly. `'get_num_upars'` uses retained metadata and also works without a live model. LOO moment matching works with `'cores = 1'` and a live model.

If using rethinking, load `rstan` before rethinking in a fresh R session. Its placeholder `'stanfit'` class can otherwise conflict with RStan's class. Native `[as_cstanfit()]` and `[as_rfit()]` do not need this load order.

Density calls support `'adjust_transform = TRUE'` (the default) and require an exact-lp runtime. `'adjust_transform = FALSE'` is rejected because the graph includes the parameter-transform Jacobian. Native RStan resampling and operations requiring its C++ model instance remain unsupported. `'constrain_pars'` includes transformed parameters and generated quantities; RNG-based quantities advance the retained model's RNG stream.

`'saveRDS()'` preserves draws, but external pointers do not survive loading. Recreate the model with the same code and data and call `'as_stanfit(saved_fit, model = model)'` to restore model-dependent methods. Serialized workers likewise cannot use live handles. No model recompiles automatically on loading or conversion.

Per-chain permutations use R's current RNG, as CSV import does. Set the same R seed before both conversions to compare permuted extraction. Stored draw order is unchanged. Empty variables cannot be inferred from column names and are omitted, as in CSV import. Packages that insist on exact class equality instead of checking inheritance may need adaptation.

Install `rstan` from a CRAN binary on supported macOS/Windows R versions to avoid a toolchain. On Linux, provision a compatible binary installation.

Value

An S4 object inheriting from `'stanfit'`, with samples, sampler statistics, timings, and parameter dimensions. Initial values and adaptation text are empty: the first saved draw is not the initial state.

A fit from `[variational_model()]`, `[pathfinder_model()]` or `[laplace_model()]` becomes a one-chain fit without warmup or sampler statistics, whose draws end with `'lp__'` and `'lp_approx__'`, and whose `'stan_args[[1]]$method'` is `"variational"`, `"pathfinder"` or `"laplace"`. This is the layout `brms`

builds from CmdStan's output for the same algorithms; unlike `[rstan::vb()]`, no row of means precedes the draws. A fixed-parameter fit keeps its chains and has 'accept_stat__' as its only sampler statistic.

Examples

```
## Not run:
m <- stanli_model(code = "parameters { real mu; } model { mu ~ normal(0, 1); }")
fit <- sample_model(m, refresh = 0)
sf <- as_stanfit(fit)
rstan::extract(sf, pars = "mu")
rstan::log_prob(sf, upars = 0, gradient = TRUE)
rstan::constrain_pars(sf, upars = 0)

## End(Not run)
```

cstan_model

Native CmdStanR-style model interface

Description

Retain Stan source in a native model object. Its `$sample()` method accepts the arguments of [sample_cstan\(\)](#) (except `model_code`) and returns the same native fit. Model preparation happens when sampling, using the supplied data and seed, including RNG calls in transformed data.

Usage

```
cstan_model(model_code, include_paths = NULL)
```

Arguments

<code>model_code</code>	A single string of Stan program source.
<code>include_paths</code>	Directories searched for Stan <code>#include</code> files; see stanli_model() . Resolved when the model object is created; files are read when <code>\$sample()</code> prepares the model.

Details

Defaults and validation are delegated to [sample_cstan\(\)](#). Neither `cmdstanr`, `rstan`, `CmdStan`, nor a C++ toolchain is needed. This object provides the sampling subset of the CmdStanR model interface, with its own class; it does not provide compilation methods, compiler options, or CSV output. Unsupported sampling arguments produce an error before model preparation.

Value

A `stanli_cstanmodel` with `$sample(...)`, `$code()`, and `$model_name()` methods. `$sample()` accepts `threads_per_chain` for native within-chain parallelism and `parallel_chains` for concurrent chains.

Examples

```
model <- cstan_model("parameters { real a; } model { a ~ normal(0, 1); }")
model$code()
if (stanli_available()) model$sample(chains = 1, iter_warmup = 20,
                                     iter_sampling = 20, seed = 1, refresh = 0)
```

laplace_model

Draws from a Laplace approximation

Description

Draws from the normal distribution, on the unconstrained scale, centred at the posterior mode with the curvature of the log density there. This is Stan's own Laplace sampling, the algorithm behind CmdStan's 'laplace' method.

Usage

```
laplace_model(
  model,
  seed = 1,
  draws = 1000,
  mode = NULL,
  jacobian = TRUE,
  iter = 2000,
  init = NULL,
  init_radius = 2,
  calculate_lp = TRUE,
  refresh = 0
)
```

Arguments

model	A 'stanli_model'.
seed	Seed.
draws	Number of draws.
mode	The mode to expand around: the result of [optimize_model()], or a point on the unconstrained scale (see [unconstrain()]). 'NULL' runs [optimize_model()] with 'seed', 'iter', 'init' and 'init_radius'.
jacobian	Must be 'TRUE'. As for [optimize_model()], the change-of-variables Jacobian is part of the compiled model, so the mode and the curvature are those of the posterior on the unconstrained scale. CmdStan's 'laplace' has the same default.
iter, init, init_radius	Settings for the optimizer when 'mode' is 'NULL'.
calculate_lp	Evaluate the log density at each draw. 'FALSE' leaves 'lp__' as 'NaN'.
refresh	Any positive value prints Stan's progress every that many draws. 0 prints nothing.

Details

The curvature is Stan’s finite-difference Hessian of the gradient. A point that is not a mode, or a mode with a direction of zero or positive curvature, has no normal approximation; Stan does not check for it and the draws are then meaningless or not finite.

Value

A ‘stanli_fit’ with one chain of ‘draws’ draws. Its ‘sampler’ element holds ‘lp__’, the log density at each draw, and ‘lp_approx__’, the approximation’s unnormalized log density there. ‘mode’ holds the unconstrained point that was expanded around.

log_lik	<i>Extract pointwise log likelihoods</i>
---------	--

Description

Declare the pointwise log likelihood in Stan’s generated quantities block, for example ‘vector[N] log_lik’ with one log density per observation.

Usage

```
log_lik(object, ...)

## S3 method for class 'stanli_fit'
log_lik(object, variable = "log_lik", ...)
```

Arguments

object	A fitted model.
...	Arguments passed to methods.
variable	Name of the pointwise log-likelihood variable.

Value

A numeric array (iteration, chain, observation), in Stan column order, excluding saved warmup. A scalar variable retains a singleton third axis.

log_prob_grad	<i>Log density and gradient</i>
---------------	---------------------------------

Description

Log density and gradient

Usage

```
log_prob_grad(model, q)
```

Arguments

model	A 'stanli_model'.
q	A point on the unconstrained scale.

Value

A list with 'lp' and 'grad'.

optimize_model	<i>Find the posterior mode by L-BFGS</i>
----------------	--

Description

Returns the posterior MODE. CmdStan's 'optimize' defaults to 'jacobian=0', the penalized maximum likelihood, which stanli cannot offer: the change-of-variables Jacobian is folded into the graph when the model is lowered.

Usage

```
optimize_model(model, seed = 1, iter = 2000, init = NULL, init_radius = 2)
```

Arguments

model	A 'stanli_model'.
seed, iter	Optimizer configuration.
init	Optional start on the unconstrained scale; see [unconstrain()] to build one from constrained values.
init_radius	Random start radius.

Value

A list with 'values' (named), 'unconstrained', and 'lp'.

pathfinder_model	<i>Pathfinder variational inference</i>
------------------	---

Description

Follows an L-BFGS path towards the posterior mode, builds a normal approximation at each point along it, keeps the one with the highest evidence lower bound, and draws from that. This is Stan's own single-path Pathfinder, what CmdStan's 'pathfinder' method runs with 'num_paths = 1'.

Usage

```
pathfinder_model(
  model,
  seed = 1,
  num_paths = 1,
  draws = 1000,
  max_lbfgs_iters = 1000,
  num_elbo_draws = 25,
  history_size = 5,
  init_alpha = 0.001,
  tol_obj = 1e-12,
  tol_rel_obj = 10000,
  tol_grad = 1e-08,
  tol_rel_grad = 1e+07,
  tol_param = 1e-08,
  calculate_lp = TRUE,
  init = NULL,
  init_radius = 2,
  refresh = 0
)
```

Arguments

model	A 'stanli_model'.
seed	Seed.
num_paths	Must be 1. Multi-path Pathfinder, which runs several paths and resamples their draws by importance, is not available in this version; see Details.
draws	Number of draws to take from the approximation.
max_lbfgs_iters	Maximum L-BFGS iterations.
num_elbo_draws	Draws used to compare the approximations along the path.
history_size, init_alpha, tol_obj, tol_rel_obj, tol_grad, tol_rel_grad, tol_param	L-BFGS settings, as in CmdStan.

calculate_lp	Evaluate the log density at each draw. With 'FALSE', 'lp__' is 'NaN' except for the first 'num_elbo_draws' draws, which Stan has already evaluated.
init	Optional starting point on the unconstrained scale; see [unconstrain()].
init_radius	A random start is drawn uniform(-r, r).
refresh	Any positive value prints Stan's progress every that many L-BFGS iterations. 0 prints nothing.

Details

Multi-path Pathfinder is held back because the multi-path service of Stan 2.40, the version stanli is built on, reads past the end of an array while it resamples. 'num_paths' is kept as an argument so that code written for it says what it means; any value but 1 is an error. cmdstanr's default is four paths, so its default results are not what this function returns.

A single path is one normal approximation, with no importance resampling and no Pareto k diagnostic. It can be far from the posterior and nothing in the fit says so: compare with [sample_model()] before relying on it. 'sample_model(pathfinder_init =)' uses the same algorithm to start NUTS.

Value

A 'stanli_fit' with one chain of 'draws' draws. Its 'sampler' element holds 'lp__', 'lp_approx__', and 'path__', which is 1 on every draw.

sample_cstan

Sample with CmdStanR-style arguments

Description

Prepare and sample a Stan model with Stanli, returning [as_cstanfit()]. This is a convenience interface; it does not install or call CmdStanR.

Usage

```
sample_cstan(
  model_code,
  data = list(),
  chains = 4,
  parallel_chains = chains,
  iter_warmup = 1000,
  iter_sampling = 1000,
  seed = NULL,
  init = NULL,
  adapt_delta = 0.8,
  max_treedepth = 10,
  thin = 1,
  save_warmup = FALSE,
  refresh = 100,
```

```

    init_radius = 2,
    pathfinder_init = NULL,
    threads_per_chain = 1,
    include_paths = NULL,
    ...
)

```

Arguments

model_code	Stan program source.
data	Named list of model data.
chains	Number of chains.
parallel_chains	Maximum concurrent chains.
iter_warmup, iter_sampling	Iterations per chain before thinning.
seed	Integer seed. If NULL, draw one from R's RNG.
init	Constrained parameter list, one list per chain, a function returning a list (optionally accepting 'chain_id'), or NULL for random initialization. A list may leave parameters out; those start at random within 'init_radius', as in CmdStan. Numeric radii and files are unsupported; use 'init_radius' for random initialization. Extra non-parameter names (such as transformed parameters) are ignored, as in CmdStanR.
adapt_delta	Target acceptance probability.
max_treedepth	Maximum NUTS tree depth.
thin	Retain every thin-th iteration, separately in warmup and sampling.
save_warmup	Retain warmup draws.
refresh	Progress interval; zero suppresses progress.
init_radius	Radius for random initialization on the unconstrained scale.
pathfinder_init	Optional Pathfinder settings passed to [sample_model()].
threads_per_chain	Threads per chain for eligible native 'reduce_sum' calls; see [sample_model()]. Defaults to 1.
include_paths	Directories searched for Stan '#include' files; see [stanli_model()].
...	Reserved. Unsupported options produce an error.

Details

Model preparation uses the sampling seed, including RNG calls in transformed data. C++ compiler options are not supported. Existing Stanli models and unconstrained initial values can be used through [sample_model()] followed by [as_cstanfit()].

Value

A native 'stanli_cstanfit'.

sample_model

*Sample a model with NUTS***Description**

Four chains by default, run in parallel: R-hat needs more than one chain, and a single-chain run cannot be checked for convergence at all. Chain ‘c’ uses CmdStan’s stream for ‘(seed, chain id c + 1)’.

Usage

```
sample_model(
  model,
  chains = 4,
  seed = 1,
  warmup = 1000,
  samples = 1000,
  thin = 1,
  delta = 0.8,
  max_depth = 10,
  save_warmup = FALSE,
  init = NULL,
  init_radius = 2,
  pathfinder_init = NULL,
  parallel_chains = NULL,
  refresh = 100,
  threads_per_chain = 1,
  fixed_param = FALSE
)
```

Arguments

model	A ‘stanli_model’.
chains, seed, warmup, samples, thin	Sampler configuration. ‘seed’ is also the model-construction seed for this run when transformed data draws from it; see [stanli_model()].
delta	Target acceptance statistic.
max_depth	Maximum treedepth.
save_warmup	Keep the warmup draws.
init	Optional starting point on the UNCONSTRAINED scale: one vector shared by every chain, or a matrix with one row per chain. Start from constrained values by passing them through [unconstrain()] first – one scale here means one contract for what a start is.
init_radius	Random inits are drawn uniform(-r, r); 0 starts at the origin.

pathfinder_init	Optional named list enabling Pathfinder-generated starts. An empty list uses defaults; supported entries are 'num_iterations' (1000), 'num_elbo_draws' (25), 'history_size' (5), and Pathfinder's own 'init_radius' (2). Uses 'seed', returns one start per chain, and cannot be combined with 'init'. Single-path Pathfinder does not perform PSIS resampling.
parallel_chains	Chains to run at once. Defaults to all of them.
refresh	Print a progress update every 'refresh' transitions within each phase, plus the first and last transition of the phase. Set to 0 to suppress all automatic sampling output.
threads_per_chain	Positive thread count per chain for eligible native 'reduce_sum' calls, default 1. Small or unsupported calls stay serial. Changing this setting prepares a new graph; reduction rounding and NUTS draws can differ across thread counts. Active chains may each use this many threads. 'fit\$model\$reduce_sum_count' and 'reduce_sum_fallbacks' report retained reductions and graph-lowering refusals (not opaque regions).
fixed_param	'TRUE' runs Stan's fixed-parameter sampler instead of NUTS: every chain keeps its parameters at its starting point and only generated quantities are drawn, which is how a model with no parameters, or one that simulates data at given values, is run. There is no warmup: 'warmup', 'delta', 'max_depth', 'save_warmup' and 'parallel_chains' are not used, and the chains run one after another. The fit's 'sampler' element then holds only 'lp__' and 'accept_stat__', both zero, and its 'report' only the time each chain took.

Details

Interrupting R (Ctrl-C, or the stop button in RStudio) stops every chain after its current transition and raises the usual interrupt; no fit is returned.

Value

An object of class 'stanli_fit'. Its 'report' element contains per-chain warmup and sampling times plus exact divergence and maximum-treedepth counts. With a compatible older runtime that pre-dates progress reporting, 'report\$available' is 'FALSE' and those values are 'NA'. Sampling metadata ('warmup', 'samples', 'thin', 'save_warmup', 'chains', and 'delta') records the configuration for downstream conversion; iteration counts are before thinning.

 stanli-ecosystem

R workflow integrations

Description

Optional methods for bayesplot, loo, and tidybayes. Install the relevant package to use its generic; no plotting package is needed to load stanli. Saved warmup is excluded by default.

Usage

```
nuts_params.stanli_fit(object, pars = NULL, inc_warmup = FALSE, ...)
```

```
log_posterior.stanli_fit(object, inc_warmup = FALSE, ...)
```

```
rhat.stanli_fit(object, pars = NULL, ...)
```

```
neff_ratio.stanli_fit(object, pars = NULL, ...)
```

```
loo.stanli_fit(x, ..., variable = "log_lik", r_eff = NULL)
```

```
loo_compare.stanli_fit(x, ...)
```

```
tidy_draws.stanli_fit(model, ...)
```

Arguments

<code>object, x, model</code>	A 'stanli_fit'.
<code>pars</code>	Optional variable names (base names select all indexed elements). For 'nuts_params', exact sampler names such as 'divergent__'.
<code>inc_warmup</code>	Include saved warmup in sampler tables.
<code>...</code>	Further arguments passed to the underlying package where applicable.
<code>variable</code>	Pointwise log-likelihood variable passed to [log_lik()].
<code>r_eff</code>	Optional relative effective sample sizes. By default computed from the likelihood with 'loo::relative_eff()' using the chain axis.

Details

R-hat and effective sample size use [summary.stanli_fit()]. The ESS ratio is bulk ESS divided by the number of retained post-warmup draws. Bayesplot's CmdStanMCMC method instead uses basic ESS.

Value

'nuts_params' returns a data frame with 'Chain', 'Iteration', 'Parameter' (a factor), and 'Value'; 'log_posterior' omits 'Parameter'. 'rhat' and 'neff_ratio' return named numeric vectors. 'tidy_draws' returns a tibble with draw indices. 'loo' returns a 'psis_loo' object and 'loo_compare' compares fits via their LOO results.

stanli_available	<i>Is the stanli runtime available?</i>
------------------	---

Description

Is the stanli runtime available?

Usage

```
stanli_available()
```

Value

‘TRUE’ when the runtime library is present and loadable.

stanli_check_syntax	<i>Check Stan code for syntax and type errors</i>
---------------------	---

Description

Runs the Stan compiler front end on the code and nothing else: no data is needed and no model is built.

Usage

```
stanli_check_syntax(code, include_paths = NULL)
```

Arguments

code	Stan model source.
include_paths	Directories searched for ‘#include’ files, as in [stanli_model()].

Value

‘TRUE’, invisibly. An invalid program is an error carrying the compiler’s message.

stanli_diagnose	<i>Convergence diagnostics</i>
-----------------	--------------------------------

Description

Divergent transitions, treedepth saturation, E-BFMI, R-hat and bulk/tail effective sample size – each either confirmed or reported with the number that failed and what to do about it.

Usage

```
stanli_diagnose(fit)
```

Arguments

fit	A ‘stanli_fit’ from NUTS. The other algorithms have no chains to diagnose.
-----	--

Value

The report, invisibly, after printing it.

stanli_exact_lp	<i>Does this build report CmdStan's 'lp__' exactly?</i>
-----------------	---

Description

Does this build report CmdStan's 'lp__' exactly?

Usage

```
stanli_exact_lp()
```

Value

'TRUE' unless the runtime was built with 'STANLI_LITE_LP'.

stanli_install	<i>Download the stanli runtime</i>
----------------	------------------------------------

Description

Fetches the prebuilt runtime for this platform into the user cache directory. This is a one-time step; the library is about 16 MB.

Usage

```
stanli_install(
  version = stanli_runtime_release,
  quiet = FALSE,
  overwrite = FALSE
)
```

Arguments

version	Release tag to fetch. Defaults to the release this version of the package was built against, which is the pairing its ABI check will accept. Only that release's runtime is found by [stanli_runtime_path()]; to run a different build, set 'STANLI_RUNTIME'.
quiet	Passed to [utils::download.file()].
overwrite	Re-download even if the runtime is already present.

Value

The path it was installed to, invisibly.

stanli_model

*Compile a Stan model***Description**

Compile a Stan model

Usage

```
stanli_model(
  file = NULL,
  code = NULL,
  data = NULL,
  mir = NULL,
  seed = 1,
  threads_per_chain = 1,
  include_paths = NULL,
  fast_math = FALSE
)
```

Arguments

<code>file</code>	Path to a ‘.stan’ file.
<code>code</code>	Model source, as an alternative to ‘file’.
<code>data</code>	A named list of data, or a path to a JSON data file.
<code>mir</code>	Transformed MIR text, for a build without the embedded compiler. Rarely needed.
<code>seed</code>	The model-construction seed: RNG calls in transformed data draw from it once, here, the way CmdStan’s generated constructor does. [sample_model()] and [optimize_model()] forward their own ‘seed’ here when transformed data drew from it, rebuilding the model for that run, so one seed governs a whole run as it does in CmdStan. The rebuilt model is the one the fit carries as ‘fit\$model’; the object passed in is unchanged. Models are also rebuilt when sampling changes ‘threads_per_chain’.
<code>threads_per_chain</code>	Positive thread count for eligible native ‘reduce_sum’ calls. Defaults to 1. Sampling uses its own setting; direct gradients use the model’s setting. Requires a thread-safe runtime for values above 1.
<code>include_paths</code>	Character vector of directories searched in order for Stan ‘#include’ files, followed by the directory containing ‘file’ (or the current directory for ‘code’). Nested includes use the same search path. Included source is compiled once and retained when sampling rebuilds the model, even if the files later change or move. Ignored when ‘mir’ is supplied.

`fast_math` ‘TRUE’ opts this model into fast mode: the Stan source is compiled with fused multiply-add rewriting and the runtime may use faster, less exactly rounded evaluation, so results can differ from the default in the last bits. Defaults to ‘FALSE’. With ‘mir’, use MIR compiled in fast mode.

Value

An object of class ‘stanli_model’ whose ‘columns’ name every output the way the posterior package reads them, ‘theta[1,2]’ for an indexed value. Warns, naming the part and the reason, when part of the model has no compiled path and runs through the much slower MIR interpreter; with the environment variable ‘STANLI_NO_INTERPRETER’ set that is an error instead. Either message is what to include in a bug report.

`stanli_runtime_path` *Where the stanli runtime lives*

Description

The path is taken from ‘STANLI_RUNTIME’ when that is set (which is how a development build is used), then from a runtime bundled into the package (wasm builds bundle one at build time, since a browser cannot download it), and from the user cache directory otherwise. The cache path includes the release the package pins, so a runtime downloaded by an older version of the package is never picked up by a newer one.

Usage

```
stanli_runtime_path()
```

Value

A file path, which may not exist yet.

`stanli_thread_safe` *Can this build run chains in parallel threads?*

Description

Can this build run chains in parallel threads?

Usage

```
stanli_thread_safe()
```

Value

‘TRUE’ when the runtime was built with thread support.

summary.stanli_fit	<i>Posterior summary</i>
--------------------	--------------------------

Description

Mean, MCSE, sd, quantiles, bulk and tail effective sample size, and rank-normalized split R-hat – stan’s own estimators, so the numbers agree with ‘stansummary’ rather than approximating it.

Usage

```
## S3 method for class 'stanli_fit'
summary(object, ...)
```

Arguments

object	A ‘stanli_fit’.
...	Unused.

Value

A data frame, one row per column.

unconstrain	<i>Starting values on the constrained scale, as the free vector</i>
-------------	---

Description

Unless ‘partial = TRUE’, every declared parameter must appear, at its declared size. A missing, unknown, wrong-length, or out-of-support value is an error naming the parameter. Containers are listed in Stan’s own serialization order (the first index fastest, the order a CSV column carries) and may be nested or flat – the declaration owns the shape either way.

Usage

```
unconstrain(model, values, partial = FALSE, init_radius = 2, seed = NULL)
```

Arguments

model	A ‘stanli_model’.
values	A named list of constrained starting values, or a JSON string in CmdStan’s data format.
partial	‘TRUE’ allows ‘values’ to leave parameters out. Each one left out starts where Stan would start it: at the constrained image of a point drawn uniformly in ‘(-init_radius, init_radius)’ on the unconstrained scale. Only for a list, not a JSON string.

init_radius	Radius of that draw.
seed	Seed for that draw. 'NULL' uses, and advances, R's random stream; a number leaves the stream untouched.

Details

The result is what 'sample_model(init =)' and 'optimize_model(init =)' take, so unconstraining is a step per starting point rather than a second kind of argument.

Value

A numeric vector of length 'model\$*n_unconstrained*'.

variational_model	<i>Variational inference with ADVI</i>
-------------------	--

Description

Fits a normal approximation on the unconstrained scale by stochastic gradient ascent on the evidence lower bound, then draws from it. This is Stan's own ADVI, the algorithm behind CmdStan's 'variational' method and 'rstan::vb()', with their defaults.

Usage

```
variational_model(
  model,
  algorithm = c("meanfield", "fullrank"),
  seed = 1,
  iter = 10000,
  grad_samples = 1,
  elbo_samples = 100,
  eta = 1,
  adapt_engaged = TRUE,
  adapt_iter = 50,
  tol_rel_obj = 0.01,
  eval_elbo = 100,
  draws = 1000,
  init = NULL,
  init_radius = 2,
  refresh = 100
)
```

Arguments

model	A 'stanli_model'.
algorithm	"meanfield" for independent normals, "fullrank" for a normal with a full covariance matrix.

seed	Seed. Also the model-construction seed for this run when transformed data draws from it; see [stanli_model()].
iter	Maximum number of iterations.
grad_samples, elbo_samples	Monte Carlo draws for each gradient and for each evaluation of the lower bound.
eta	Step size. Used as given when 'adapt_engaged' is 'FALSE'.
adapt_engaged	Choose the step size by a short search first.
adapt_iter	Iterations for each step size the search tries.
tol_rel_obj	Relative change in the lower bound below which the fit is taken as converged.
eval_elbo	Evaluate the lower bound every this many iterations.
draws	Number of draws to take from the approximation.
init	Optional starting point on the unconstrained scale; see [unconstrain()].
init_radius	Random starts are drawn uniform(-r, r); 0 starts at the origin.
refresh	Any positive value prints Stan's progress, including the table of lower-bound values. 0 prints nothing.

Details

The approximation is not checked against the posterior. ADVI can stop early at a poor optimum, and a mean-field fit ignores correlation by construction, so compare with [sample_model()] before relying on the spread of the draws. Reaching 'iter' without converging is reported as a warning and the fit reached so far is returned.

Stan measures the lower bound with every constant of the log density included. stanli evaluates the density 'lp__' reports, which leaves out the constants of '~' statements, so for a model written with them the lower bound printed here differs from CmdStan's by a constant. The gradients and the random stream are the same, but the stopping rule is a relative change of the bound, so the fit can stop on a different iteration than CmdStan's with the same seed. A model that uses 'target += ' throughout, as brms models do, is unaffected.

Value

A 'stanli_fit' with one chain of 'draws' draws. Its 'sampler' element holds 'lp__', the log density at each draw, and 'lp_approx__', the approximation's unnormalized log density there. 'mean' holds every column at the mean of the approximation, and 'report\$sampling_seconds' the wall time of the run.

Index

`as_cstanfit`, [2](#)
`as_draws_array`, [3](#)
`as_rfit`, [4](#)
`as_stanfit`, [5](#)

`cstan_model`, [7](#)

`extract` (`as_rfit`), [4](#)

`get_elapsed_time` (`as_rfit`), [4](#)
`get_sampler_params` (`as_rfit`), [4](#)

`laplace_model`, [8](#)
`log_lik`, [9](#)
`log_posterior.stanli_fit`
 (`stanli-ecosystem`), [15](#)
`log_prob_grad`, [10](#)
`loo.stanli_fit` (`stanli-ecosystem`), [15](#)
`loo_compare.stanli_fit`
 (`stanli-ecosystem`), [15](#)

`neff_ratio.stanli_fit`
 (`stanli-ecosystem`), [15](#)
`nuts_params.stanli_fit`
 (`stanli-ecosystem`), [15](#)

`optimize_model`, [10](#)

`pathfinder_model`, [11](#)
`print.stanli_cstanfit` (`as_cstanfit`), [2](#)

`rhat.stanli_fit` (`stanli-ecosystem`), [15](#)

`sample_cstan`, [12](#)
`sample_cstan()`, [7](#)
`sample_model`, [14](#)
`stanli-ecosystem`, [15](#)
`stanli_available`, [16](#)
`stanli_check_syntax`, [17](#)
`stanli_diagnose`, [17](#)
`stanli_exact_lp`, [18](#)

`stanli_install`, [18](#)
`stanli_model`, [19](#)
`stanli_model()`, [7](#)
`stanli_runtime_path`, [20](#)
`stanli_thread_safe`, [20](#)
`summary.stanli_cstanfit` (`as_cstanfit`), [2](#)
`summary.stanli_fit`, [21](#)
`summary.stanli_rfit` (`as_rfit`), [4](#)

`tidy_draws.stanli_fit`
 (`stanli-ecosystem`), [15](#)

`unconstrain`, [21](#)

`variational_model`, [22](#)