Package: stantargets (via r-universe)

August 16, 2024

Title Targets for Stan Workflows

Description Bayesian data analysis usually incurs long runtimes and
cumbersome custom code. A pipeline toolkit tailored to Bayesian
statisticians, the 'stantargets' R package leverages 'targets'
and 'cmdstanr' to ease these burdens. 'stantargets' makes it
super easy to set up scalable Stan pipelines that automatically
parallelize the computation and skip expensive steps when the
results are already up to date. Minimal custom code is
required, and there is no need to manually configure branching,
so usage is much easier than 'targets' alone. 'stantargets' can
access all of 'cmdstanr''s major algorithms (MCMC, variational
Bayes, and optimization) and it supports both single-fit
workflows and multi-rep simulation studies. For the statistical
methodology, please refer to 'Stan' documentation (Stan
Development Team 2020) <https://mc-stan.org/>.

Version 0.1.1.9000

License MIT + file LICENSE

URL https://docs.ropensci.org/stantargets/,
https://github.com/ropensci/stantargets

BugReports https://github.com/ropensci/stantargets/issues

Depends R (>= 3.5.0)

Imports cmdstanr (>= 0.5.0), fs (>= 1.5.0), fst (>= 0.9.2), posterior
(>= 1.0.1), purrr (>= 0.3.4), qs (>= 0.23.2), rlang (>=
0.4.10), secretbase (>= 0.4.0), stats, targets (>= 1.6.0),
tarchetypes (>= 0.8.0), tibble (>= 3.0.1), tidyselect, withr
(>= 2.1.2)

Suggests dplyr (>= 1.0.2), ggplot2 (>= 3.0.0), knitr (>= 1.30),
R.utils (>= 2.10.1), rmarkdown (>= 2.3), SBC (>= 0.2.0),
testthat (>= 3.0.0), tidyr (>= 1.0.0), visNetwork (>= 2.0.9)

Remotes hyunjimoon/SBC, stan-dev/cmdstanr,

SystemRequirements CmdStan >= 2.25.0

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

VignetteBuilder knitr

Config/testthat/edition 3

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

RemoteUrl https://github.com/ropensci/stantargets

RemoteRef main

RemoteSha 448518fae309ef002c7746481c621106c45c75aa

Contents

| stantargets-package | *targets: Targets Archetypes for Stan* |

Description

Bayesian data analysis usually incurs long runtimes and cumbersome custom code. A pipeline toolkit tailored to Bayesian statisticians, the stantargets R package leverages targets and cmdstanr to ease these burdens. stantargets makes it super easy to set up scalable Stan pipelines that automatically parallelize the computation and skip expensive steps when the results are already up to date. Minimal custom code is required, and there is no need to manually configure branching, so usage is much easier than targets alone. stantargets can access all of cmdstanr's major algorithms (MCMC, variational Bayes, and optimization) and it supports both single-fit workflows and multi-rep simulation studies.

See Also

https://docs.ropensci.org/stantargets/, tar_stan_mcmc()

---

tar_stan_compile   *Local Stan model compilation*

Description

`tar_stan_compile()` creates a target to compile a Stan model on the local file system and return the original Stan model file. Does not compile the model if the compilation is already up to date.

Usage

```
tar_stan_compile(
  name,
  stan_file,
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target,` `f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even

	dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.
stan_file	(string) The path to a .stan file containing a Stan program. The helper function [write_stan_file()](#) is provided for cases when it is more convenient to specify the Stan program as a string. If stan_file is not specified then exe_file must be specified.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *[Pedantic mode](#)* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the [$check_syntax()](#) method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study [Reduce Sum: A Minimal Example](#).
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	
	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
error	Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)

- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                 Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Most of the arguments are passed to the `$compile()` method of the `CmdStanModel` class. For details, visit https://mc-stan.org/cmdstanr/reference/.

Value

`tar_stan_compile()` returns a target object to compile a Stan file. The return value of this target is a character vector containing the Stan model source file and compiled executable file. A change in either file will cause the target to rerun in the next run of the pipeline. See the "Target objects" section for background.

Target objects

Most `stantargets` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
```

```
list(tar_stan_compile(compiled_model, path))
})
targets::tar_make()
})
}
```

---

tar_stan_example_data   *Simulate example data for* `tar_stan_example_file()`*.*

Description

An example dataset compatible with the model file from `tar_stan_example_file()`.

Usage

```
tar_stan_example_data(n = 10L)
```

Arguments

n   Integer of length 1, number of data points.

Format

A list with the following elements:

- n: integer, number of data points.
- x: numeric, covariate vector.
- y: numeric, response variable.
- true_beta: numeric of length 1, value of the regression coefficient beta used in simulation.
- .join_data: a list of simulated values to be appended to as a .join_data column in the output of targets generated by functions such as `tar_stan_mcmc_rep_summary()`. Contains the regression coefficient beta (numeric of length 1) and prior predictive data y (numeric vector).

Details

The tar_stan_example_data() function draws a Stan dataset from the prior predictive distribution of the model from `tar_stan_example_file()`. First, the regression coefficient beta is drawn from its standard normal prior, and the covariate x is computed. Then, conditional on the beta draws and the covariate, the response vector y is drawn from its Normal(x * beta, 1) likelihood.

Value

List, dataset compatible with the model file from `tar_stan_example_file()`.

See Also

Other examples: `tar_stan_example_file()`

Examples

```
tar_stan_example_data()
```

---

tar_stan_example_file   *Write an example Stan model file.*

Description

Overwrites the file at path with a built-in example Stan model file.

Usage

```
tar_stan_example_file(path = tempfile(pattern = "", fileext = ".stan"))
```

Arguments

path   Character of length 1, file path to write the model file.

Value

NULL (invisibly).

See Also

Other examples: [tar_stan_example_data](#)()

Examples

```
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
writeLines(readLines(path))
```

---

tar_stan_gq   *Generated quantities on an existing CmdStanFit object*

Description

tar_stan_gq() creates targets to run the generated quantities of a Stan model and save draws and summaries separately.

Usage

```
tar_stan_gq(
  name,
  stan_files,
  data = list(),
  fitted_params,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  output_dir = NULL,
  sig_figs = NULL,
  parallel_chains = getOption("mc.cores", 1),
  threads_per_chain = NULL,
  variables = NULL,
  variables_fit = NULL,
  summaries = list(),
  summary_args = list(),
  return_draws = TRUE,
  return_summary = TRUE,
  draws = NULL,
  summary = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of Stan model files. If you supply multiple files, each model will run on the one shared dataset generated by the code in data. If you supply an unnamed vector, fs::path_ext_remove(basename(stan_files)) will be used as target name suffixes. If stan_files is a named vector, the suffixed will come from names(stan_files).
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using [write_stan_json()](). See [write_stan_json()]() for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- NULL or an empty list if the Stan program has no data block.

fitted_params	Symbol, name of a CmdStanFit object computed in a previous target: for example, the *_mcmc_* target from [tar_stan_mcmc()](). Must be a subclass that $generate_quantities() can accept as fitted_params.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()]() method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *[Pedantic mode]() chapter* in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the [$check_syntax()]() method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.

cpp_options   (list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.

stanc_options   (list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.

force_recompile

                   (logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.

seed   (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

                       • If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., $save_output_files()). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
                       • If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

sig_figs   (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

parallel_chains

                   (positive integer) The *maximum* number of MCMC chains to run in parallel. If parallel_chains is not specified then the default is to look for the option "mc.cores", which can be set for an entire R session by options(mc.cores=value). If the "mc.cores" option has not been set then the default is 1.

threads_per_chain

                   (positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when using the Stan functions reduce_sum() or map_rect()). This is in contrast with parallel_chains, which specifies the number of chains to run in parallel. The

	actual number of CPU cores used is parallel_chains*threads_per_chain. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
variables	(character vector) The variables to include.
variables_fit	Character vector of variables to include in the big CmdStanFit object returned by the model fit target. The variables argument, by contrast, is for the "draws" target only. The "draws" target can only access the variables in the CmdStanFit target. Control the variables in each with the variables and variables_fit arguments.
summaries	Optional list of summary functions passed to ... in posterior::summarize_draws() through $summary() on the CmdStanFit object.
summary_args	Optional list of summary function arguments passed to .args in posterior::summarize_draws() through $summary() on the CmdStanFit object.
return_draws	Logical, whether to create a target for posterior draws. Saves posterior::as_draws_df(fit$draws()) to a compressed tibble. Convenient, but duplicates storage.
return_summary	Logical, whether to create a target for fit$summary().
draws	Deprecated on 2022-07-22. Use return_draws instead.
summary	Deprecated on 2022-07-22. Use return_summary instead.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
repository	Character of length 1, remote repository for target storage. Choices:

  - "local": file system of the local machine.
  - "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of tar_resources_aws(), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
  - "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                     Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic

files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Most of the arguments are passed to the $compile(), $generate_quantities(), and $summary() methods of the CmdStanModel class. If you previously compiled the model in an upstream `tar_stan_compile()` target, then the model should not recompile.

Value

tar_stan_gq() returns list of target objects. See the "Target objects" section for background. The target names use the name argument as a prefix, and the individual elements of stan_files appear in the suffixes where applicable. As an example, the specific target objects returned by tar_stan_gq(name = x, stan_files = "y.stan", ...) are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with the paths to the model file and compiled executable.
- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if compile = "original". Returns a character vector of lines in the model file.
- x_data: run the R expression in the data argument to produce a Stan dataset for the model. Returns a Stan data list.
- x_gq_y: run generated quantities on the model and the dataset. Returns a cmdstanr CmdStanGQ object with all the results.
- x_draws_y: extract draws from x_gq_y. Omitted if draws = FALSE. Returns a tidy data frame of draws.
- x_summary_y: extract compact summaries from x_gq_y. Returns a tidy data frame of summaries. Omitted if summary = FALSE.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other generated quantities: tar_stan_gq_rep_draws(), tar_stan_gq_rep_summary()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mcmc(
    your_model,
    stan_files = c(x = path),
    data = tar_stan_example_data(),
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  ),
  tar_stan_gq(
    custom_gq,
    stan_files = path, # Can be a different model.
    fitted_params = your_model_mcmc_x,
    data = your_model_data, # Can be a different dataset.
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
)
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_gq_rep_draws   *Multiple runs of generated quantities per model with draws*

Description

tar_stan_gq_rep_draws() creates targets to run generated quantities multiple times and save only the draws from each run.

Usage

```
tar_stan_gq_rep_draws(
  name,
  stan_files,
  data = list(),
  fitted_params,
  batches = 1L,
  reps = 1L,
  combine = FALSE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  output_dir = NULL,
  sig_figs = NULL,
  parallel_chains = getOption("mc.cores", 1),
  threads_per_chain = NULL,
  variables = NULL,
  data_copy = character(0),
  transform = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = "transient",
  garbage_collection = TRUE,
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
```

```
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using [write_stan_json()](). See [write_stan_json()]() for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- NULL or an empty list if the Stan program has no data block.

fitted_params	(multiple options) The parameter draws to use. One of the following:

- A [CmdStanMCMC]() or [CmdStanVB]() fitted model object.
- A [posterior::draws_array]() (for MCMC) or [posterior::draws_matrix]() (for VB) object returned by CmdStanR's [$draws()]() method.
- A character vector of paths to CmdStan CSV output files.

NOTE: if you plan on making many calls to $generate_quantities() then the most efficient option is to pass the paths of the CmdStan CSV output files (this avoids CmdStanR having to rewrite the draws contained in the fitted model object to CSV each time). If you no longer have the CSV files you can use [draws_to_csv()]() once to write them and then pass the resulting file paths to $generate_quantities() as many times as needed.

batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.
combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()]() method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.

stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the $check_syntax() method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	
	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
output_dir	(string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

  • If NULL (the default), then the CSV files are written to a temporary directory
    and only saved permanently if the user calls one of the $save_* methods of
    the fitted model object (e.g., $save_output_files()). These temporary

files are removed when the fitted model object is [garbage collected](manu-ally or automatically).

- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

sig_figs   (positive integer) The number of significant figures used when storing the out-put values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

parallel_chains

(positive integer) The *maximum* number of MCMC chains to run in parallel. If parallel_chains is not specified then the default is to look for the option "mc.cores", which can be set for an entire R session by options(mc.cores=value). If the "mc.cores" option has not been set then the default is 1.

threads_per_chain

(positive integer) If the model was [compiled] with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when using the Stan functions reduce_sum() or map_rect()). This is in contrast with parallel_chains, which specifies the number of chains to run in parallel. The actual number of CPU cores used is parallel_chains*threads_per_chain. For an example of using threading see the Stan case study [Reduce Sum: A Minimal Example].

variables   (character vector) Optionally, the names of the variables (parameters, trans-formed parameters, and generated quantities) to read in.

- If NULL (the default) then all variables are included.
- If an empty string (variables="") then none are included.
- For non-scalar variables all elements or specific elements can be selected:
  - variables = "theta" selects all elements of theta;
  - variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

data_copy   Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimen-sions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.

transform   Symbol or NULL, name of a function that accepts arguments data and draws and returns a data frame. Here, data is the JAGS data list supplied to the model, and draws is a data frame with one column per model parameter and one row per posterior sample. Any arguments other than data and draws must have valid default values because stantargets will not populate them. See the simulation-based calibration (SBC) section of the simulation vignette for an example.

tidy_eval   Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set pack-ages globally for all subsequent targets you define.

library	Character vector of library paths to try when loading packages.

format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().

format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().

repository   Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of tar_resources_aws(), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
         Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you

	set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit <https://books.ropensci.org/targets/crew.html>.
priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
storage	Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.
  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval	Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Most of the arguments are passed to the $compile() and $sample() methods of the CmdStanModel class. If you previously compiled the model in an upstream `tar_stan_compile()` target, then the model should not recompile.

Value

tar_stan_gq_rep_draws() returns a list of target objects. See the "Target objects" section for background. The target names use the name argument as a prefix, and the individual elements of stan_files appear in the suffixes where applicable. As an example, the specific target objects returned by tar_stan_gq_rep_draws(name = x, stan_files = "y.stan") are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with the paths to the model file and compiled executable.

- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if compile = "original". Returns a character vector of lines in the model file.

- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the data argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.

- x_y: dynamic branching target to run generated quantities once per dataset. Each dynamic branch returns a tidy data frames of draws corresponding to a batch of Stan data from x_data.

- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of draws.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and tar_option_get("seed"). This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the .seed list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other generated quantities: tar_stan_gq(), tar_stan_gq_rep_summary()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mcmc(
    your_model,
    stan_files = c(x = path),
    data = tar_stan_example_data(),
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile(),
    refresh = 0
  ),
  tar_stan_gq_rep_draws(
    generated_quantities,
    stan_files = path,
    data = tar_stan_example_data(),
    fitted_params = your_model_mcmc_x,
    batches = 2,
    reps = 2,
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
)
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_gq_rep_summary

Multiple runs of generated quantities per model with summaries

---

Description

`tar_stan_gq_rep_summaries()` creates targets to run generated quantities multiple times and save only the summaries from each run.

Usage

```
tar_stan_gq_rep_summary(
  name,
  stan_files,
  data = list(),
```

```
  fitted_params,
  batches = 1L,
  reps = 1L,
  combine = TRUE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  output_dir = NULL,
  sig_figs = NULL,
  parallel_chains = getOption("mc.cores", 1),
  threads_per_chain = NULL,
  data_copy = character(0),
  variables = NULL,
  summaries = list(),
  summary_args = list(),
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	(multiple options) The data to use for the variables specified in the data block of

the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables de-clared in the data block of the Stan program. Internally this list is then writ-ten to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- `NULL` or an empty list if the Stan program has no data block.

fitted_params   (multiple options) The parameter draws to use. One of the following:

- A CmdStanMCMC or CmdStanVB fitted model object.
- A posterior::draws_array (for MCMC) or posterior::draws_matrix (for VB) object returned by CmdStanR's `$draws()` method.
- A character vector of paths to CmdStan CSV output files.

NOTE: if you plan on making many calls to $generate_quantities() then the most efficient option is to pass the paths of the CmdStan CSV output files (this avoids CmdStanR having to rewrite the draws contained in the fitted model object to CSV each time). If you no longer have the CSV files you can use `draws_to_csv()` once to write them and then pass the resulting file paths to $generate_quantities() as many times as needed.

batches   Number of batches. Each batch is a sequence of branch targets containing mul-tiple reps. Each rep generates a dataset and runs the model on it.

reps   Number of replications per batch.

combine   Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.

compile   (logical) Do compilation? The default is `TRUE`. If `FALSE` compilation can be done later via the `$compile()` method.

quiet   (logical) Should the verbose output from CmdStan during compilation be sup-pressed? The default is `TRUE`, but if you encounter an error we recommend trying again with `quiet=FALSE` to see more of the output.

stdout   Character of length 1, file path to write the stdout stream of the model when it runs. Set to `NULL` to print to the console. Set to `R.utils::nullfile()` to suppress stdout. Does not apply to messages, warnings, or errors.

stderr   Character of length 1, file path to write the stderr stream of the model when it runs. Set to `NULL` to print to the console. Set to `R.utils::nullfile()` to suppress stderr. Does not apply to messages, warnings, or errors.

dir   (string) The path to the directory in which to store the CmdStan executable (or `.hpp` file if using `$save_hpp_file()`). The default is the same location as the Stan program.

pedantic   (logical) Should pedantic mode be turned on? The default is `FALSE`. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the `$check_syntax()` method can be used instead.

include_paths   (character vector) Paths to directories where Stan should look for files specified
                 in #include directives in the Stan program.

cpp_options   (list) Any makefile options to be used when compiling the model (STAN_THREADS,
                 STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the
                 make/local file. For an example of using threading see the Stan case study
                 Reduce Sum: A Minimal Example.

stanc_options   (list) Any Stan-to-C++ transpiler options to be used when compiling the model.
                 See the **Examples** section below as well as the stanc chapter of the CmdStan
                 Guide for more details on available options: https://mc-stan.org/docs/cmdstan-
                 guide/stanc.html.

force_recompile
                 (logical) Should the model be recompiled even if was not modified since last
                 compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile
                 option.

seed   (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of
                 multi-chain sampling the single seed will automatically be augmented by the
                 the run (chain) ID so that each chain uses a different seed. The exception is the
                 transformed data block, which defaults to using same seed for all chains so that
                 the same data is generated for all chains if RNG functions are used. The only
                 time seed should be specified as a vector (one element per chain) is if RNG
                 functions are used in transformed data and the goal is to generate *different* data
                 for each chain.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files.
                 For interactive use this can typically be left at NULL (temporary directory) since
                 CmdStanR makes the CmdStan output (posterior draws and diagnostics) avail-
                 able in R via methods of the fitted model objects. The behavior of output_dir
                 is as follows:

                     • If NULL (the default), then the CSV files are written to a temporary directory
                       and only saved permanently if the user calls one of the $save_* methods of
                       the fitted model object (e.g., $save_output_files()). These temporary
                       files are removed when the fitted model object is garbage collected (manu-
                       ally or automatically).
                     • If a path, then the files are created in output_dir with names corresponding
                       to the defaults used by $save_output_files().

sig_figs   (positive integer) The number of significant figures used when storing the out-
                 put values. By default, CmdStan represent the output values with 6 significant
                 figures. The upper limit for sig_figs is 18. Increasing this value will result in
                 larger output CSV files and thus an increased usage of disk space.

parallel_chains
                 (positive integer) The *maximum* number of MCMC chains to run in parallel.
                 If parallel_chains is not specified then the default is to look for the option
                 "mc.cores", which can be set for an entire R session by options(mc.cores=value).
                 If the "mc.cores" option has not been set then the default is 1.

threads_per_chain
                 (positive integer) If the model was compiled with threading support, the number
                 of threads to use in parallelized sections *within* an MCMC chain (e.g., when

using the Stan functions reduce_sum() or map_rect()). This is in contrast with parallel_chains, which specifies the number of chains to run in parallel. The actual number of CPU cores used is parallel_chains*threads_per_chain. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.

data_copy   Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.

variables   (character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If NULL (the default) then all variables are included.
- If an empty string (variables="") then none are included.
- For non-scalar variables all elements or specific elements can be selected:
    - variables = "theta" selects all elements of theta;
    - variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

summaries   Optional list of summary functions passed to ... in posterior::summarize_draws() through $summary() on the CmdStanFit object.

summary_args   Optional list of summary function arguments passed to .args in posterior::summarize_draws() through $summary() on the CmdStanFit object.

tidy_eval   Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

format   Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().

format_df   Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().

repository   Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of tar_resources_aws(), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends,

but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

Most of the arguments are passed to the `$compile()` and `$generate_quantities()` methods of the `CmdStanModel` class. If you previously compiled the model in an upstream [tar_stan_compile()](#) target, then the model should not recompile.

Value

`tar_stan_gq_rep_summaries()` returns a list of target objects. See the "Target objects" section for background. The target names use the name argument as a prefix, and the individual elements of `stan_files` appear in the suffixes where applicable. As an example, the specific target objects returned by `tar_stan_gq_rep_summary(name = x, stan_files = "y.stan")` returns a list of target objects:

- `x_file_y`: reproducibly track the Stan model file. Returns a character vector with the paths to the model file and compiled executable.
- `x_lines_y`: read the Stan model file for safe transport to parallel workers. Omitted if `compile = "original"`. Returns a character vector of lines in the model file.
- `x_data`: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the `data` argument. Each dynamic branch returns a batch of Stan data lists that `x_y` supplies to the model.

- x_y: dynamic branching target to run generated quantities once per dataset. Each dynamic branch returns a tidy data frames of summaries corresponding to a batch of Stan data from x_data.
- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of summaries.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and tar_option_get("seed"). This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the .seed list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other generated quantities: tar_stan_gq(), tar_stan_gq_rep_draws()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mcmc(
    your_model,
    stan_files = c(x = path),
    data = tar_stan_example_data(),
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  ),
  tar_stan_gq_rep_summary(
    generated_quantities,
```

```
      stan_files = path,
      data = tar_stan_example_data(),
      fitted_params = your_model_mcmc_x,
      batches = 2,
      reps = 2,
      stdout = R.utils::nullfile(),
      stderr = R.utils::nullfile()
    )
  )
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_mcmc   *One MCMC per model with multiple outputs*

Description

tar_stan_mcmc() creates targets to run one MCMC per model and separately save summaries
draws, and diagnostics.

Usage

```
tar_stan_mcmc(
  name,
  stan_files,
  data = list(),
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  output_basename = NULL,
  sig_figs = NULL,
  chains = 4,
  parallel_chains = getOption("mc.cores", 1),
  chain_ids = seq_len(chains),
```

```
    threads_per_chain = NULL,
    opencl_ids = NULL,
    iter_warmup = NULL,
    iter_sampling = NULL,
    save_warmup = FALSE,
    thin = NULL,
    max_treedepth = NULL,
    adapt_engaged = TRUE,
    adapt_delta = NULL,
    step_size = NULL,
    metric = NULL,
    metric_file = NULL,
    inv_metric = NULL,
    init_buffer = NULL,
    term_buffer = NULL,
    window = NULL,
    fixed_param = FALSE,
    show_messages = TRUE,
    diagnostics = c("divergences", "treedepth", "ebfmi"),
    variables = NULL,
    variables_fit = NULL,
    inc_warmup = FALSE,
    inc_warmup_fit = FALSE,
    summaries = list(),
    summary_args = list(),
    return_draws = TRUE,
    return_diagnostics = TRUE,
    return_summary = TRUE,
    draws = NULL,
    summary = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = "qs",
    format_df = "fst_tbl",
    repository = targets::tar_option_get("repository"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of Stan model files. If you supply multiple files, each model will run on the one shared dataset generated by the code in data. If you supply an unnamed vector, fs::path_ext_remove(basename(stan_files)) will be used as target name suffixes. If stan_files is a named vector, the suffixed will come from names(stan_files).
data	Code to generate the data for the Stan model.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()](#) method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *[Pedantic mode](#)* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the [$check_syntax()](#) method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study [Reduce Sum: A Minimal Example](#).
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.

seed   (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of
                      multi-chain sampling the single seed will automatically be augmented by the
                      the run (chain) ID so that each chain uses a different seed. The exception is the
                      transformed data block, which defaults to using same seed for all chains so that
                      the same data is generated for all chains if RNG functions are used. The only
                      time seed should be specified as a vector (one element per chain) is if RNG
                      functions are used in transformed data and the goal is to generate *different* data
                      for each chain.

refresh   (non-negative integer) The number of iterations between printed screen updates.
                      If refresh = 0, only error messages will be printed.

init   (multiple options) The initialization method to use for the variables declared in
                      the parameters block of the Stan program. One of the following:

                          • A real number x>0. This initializes *all* parameters randomly between [-x,x]
                            on the *unconstrained* parameter space.;
                          • The number 0. This initializes *all* parameters to 0;
                          • A character vector of paths (one per chain) to JSON or Rdump files con-
                            taining initial values for all or some parameters. See write_stan_json()
                            to write R objects to JSON files compatible with CmdStan.
                          • A list of lists containing initial values for all or some parameters. For
                            MCMC the list should contain a sublist for each chain. For other model
                            fitting methods there should be just one sublist. The sublists should have
                            named elements corresponding to the parameters for which you are speci-
                            fying initial values. See **Examples**.
                          • A function that returns a single list with names corresponding to the param-
                            eters for which you are specifying initial values. The function can take no
                            arguments or a single argument chain_id. For MCMC, if the function has
                            argument chain_id it will be supplied with the chain id (from 1 to number
                            of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics
                      (logical) Should auxiliary diagnostic information about the latent dynamics be
                      written to temporary diagnostic CSV files? This argument replaces CmdStan's
                      diagnostic_file argument and the content written to CSV is controlled by the
                      user's CmdStan installation and not CmdStanR (for some algorithms no content
                      may be written). The default is FALSE, which is appropriate for almost every use
                      case. To save the temporary files created when save_latent_dynamics=TRUE
                      see the [$save_latent_dynamics_files()](#) method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files.
                      For interactive use this can typically be left at NULL (temporary directory) since
                      CmdStanR makes the CmdStan output (posterior draws and diagnostics) avail-
                      able in R via methods of the fitted model objects. The behavior of output_dir
                      is as follows:

                          • If NULL (the default), then the CSV files are written to a temporary directory
                            and only saved permanently if the user calls one of the $save_* methods of
                            the fitted model object (e.g., [$save_output_files()](#)). These temporary
                            files are removed when the fitted model object is [garbage collected](#) (manu-
                            ally or automatically).

- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

output_basename

(string) A string to use as a prefix for the names of the output CSV files of CmdStan. If NULL (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

sig_figs   (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

chains   (positive integer) The number of Markov chains to run. The default is 4.

parallel_chains

(positive integer) The *maximum* number of MCMC chains to run in parallel. If parallel_chains is not specified then the default is to look for the option "mc.cores", which can be set for an entire R session by options(mc.cores=value). If the "mc.cores" option has not been set then the default is 1.

chain_ids   (integer vector) A vector of chain IDs. Must contain as many unique positive integers as the number of chains. If not set, the default chain IDs are used (integers starting from 1).

threads_per_chain

(positive integer) If the model was [compiled](#) with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when using the Stan functions reduce_sum() or map_rect()). This is in contrast with parallel_chains, which specifies the number of chains to run in parallel. The actual number of CPU cores used is parallel_chains*threads_per_chain. For an example of using threading see the Stan case study [Reduce Sum: A Minimal Example](#).

opencl_ids   (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with cpp_options = list(stan_opencl = TRUE) for this argument to have an effect.

iter_warmup   (positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as num_warmup.

iter_sampling   (positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as num_samples.

save_warmup   (logical) Should warmup iterations be saved? The default is FALSE.

thin   (positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.

max_treedepth   (positive integer) The maximum allowed tree depth for the NUTS engine. See the *Tree Depth* section of the CmdStan User's Guide for more details.

adapt_engaged   (logical) Do warmup adaptation? The default is TRUE. If a precomputed inverse metric is specified via the inv_metric argument (or metric_file) then, if adapt_engaged=TRUE, Stan will use the provided inverse metric just as an initial guess during adaptation. To turn off adaptation when using a precomputed inverse metric set adapt_engaged=FALSE.

adapt_delta   (real in (0,1)) The adaptation target acceptance statistic.

step_size   (positive real) The *initial* step size for the discrete approximation to continuous
                Hamiltonian dynamics. This is further tuned during warmup.

metric   (string) One of "diag_e", "dense_e", or "unit_e", specifying the geometry of
                the base manifold. See the *Euclidean Metric* section of the CmdStan User's
                Guide for more details. To specify a precomputed (inverse) metric, see the
                inv_metric argument below.

metric_file   (character vector) The paths to JSON or Rdump files (one per chain) compatible
                with CmdStan that contain precomputed inverse metrics. The metric_file
                argument is inherited from CmdStan but is confusing in that the entry in JSON
                or Rdump file(s) must be named inv_metric, referring to the *inverse* metric.
                We recommend instead using CmdStanR's inv_metric argument (see below) to
                specify an inverse metric directly using a vector or matrix from your R session.

inv_metric   (vector, matrix) A vector (if metric='diag_e') or a matrix (if metric='dense_e')
                for initializing the inverse metric. This can be used as an alternative to the
                metric_file argument. A vector is interpreted as a diagonal metric. The in-
                verse metric is usually set to an estimate of the posterior covariance. See the
                adapt_engaged argument above for details about (and control over) how speci-
                fying a precomputed inverse metric interacts with adaptation.

init_buffer   (nonnegative integer) Width of initial fast timestep adaptation interval during
                warmup.

term_buffer   (nonnegative integer) Width of final fast timestep adaptation interval during
                warmup.

window   (nonnegative integer) Initial width of slow timestep/metric adaptation interval.

fixed_param   (logical) When TRUE, call CmdStan with argument "algorithm=fixed_param".
                The default is FALSE. The fixed parameter sampler generates a new sample with-
                out changing the current state of the Markov chain; only generated quantities
                may change. This can be useful when, for example, trying to generate pseudo-
                data using the generated quantities block. If the parameters block is empty then
                using fixed_param=TRUE is mandatory. When fixed_param=TRUE the chains
                and parallel_chains arguments will be set to 1.

show_messages   (logical) When TRUE (the default), prints all output during the sampling process,
                such as iteration numbers and elapsed times. If the output is silenced then the
                [$output()](#) method of the resulting fit object can be used to display the silenced
                messages.

diagnostics   (character vector) The diagnostics to automatically check and warn about af-
                ter sampling. Setting this to an empty string "" or NULL can be used to pre-
                vent CmdStanR from automatically reading in the sampler diagnostics from
                CSV if you wish to manually read in the results and validate them yourself,
                for example using [read_cmdstan_csv()](#). The currently available diagnostics
                are "divergences", "treedepth", and "ebfmi" (the default is to check all of
                them).

                These diagnostics are also available after fitting. The [$sampler_diagnostics()](#)
                method provides access the diagnostic values for each iteration and the [$diagnostic_summary()](#)
                method provides summaries of the diagnostics and can regenerate the warning
                messages.

	Diagnostics like R-hat and effective sample size are *not* currently available via the diagnostics argument but can be checked after fitting using the $summary() method.
variables	(character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If NULL (the default) then all variables are included.
- If an empty string (variables="") then none are included.
- For non-scalar variables all elements or specific elements can be selected:
    - variables = "theta" selects all elements of theta;
    - variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

variables_fit	Character vector of variables to include in the big CmdStanFit object returned by the model fit target. The variables argument, by contrast, is for the "draws" target only. The "draws" target can only access the variables in the CmdStanFit target. Control the variables in each with the variables and variables_fit arguments.
inc_warmup	(logical) Should warmup draws be included? Defaults to FALSE. Ignored except when used with [CmdStanMCMC](#) objects.
inc_warmup_fit	Logical of length 1, whether to include warmup draws in the big MCMC object (the target with "mcmc" in the name). inc_warmup must not be TRUE if inc_warmup_fit is FALSE.
summaries	Optional list of summary functions passed to ... in posterior::summarize_draws() through $summary() on the CmdStanFit object.
summary_args	Optional list of summary function arguments passed to .args in posterior::summarize_draws() through $summary() on the CmdStanFit object.
return_draws	Logical, whether to create a target for posterior draws. Saves posterior::as_draws_df(fit$draws()) to a compressed tibble. Convenient, but duplicates storage.
return_diagnostics	
	Logical, whether to create a target for posterior::as_draws_df(fit$sampler_diagnostics()). Saves posterior::as_draws_df(fit$draws()) to a compressed tibble. Convenient, but duplicates storage.
return_summary	Logical, whether to create a target for fit$summary().
draws	Deprecated on 2022-07-22. Use return_draws instead.
summary	Deprecated on 2022-07-22. Use return_summary instead.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.

format	Character of length 1, storage format of the non-data-frame targets such as the Stan data and any CmdStanFit objects. Please choose an all=purpose format such as `"qs"` or `"aws_qs"` rather than a file format like `"file"` or a data frame format like `"parquet"`. For more on storage formats, see the help file of `targets::tar_target()`.
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as `"feather"` or `"aws_parquet"`. For more on storage formats, see the help file of `targets::tar_target()`.
repository	Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error	Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- `"null"`: The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory	Character of length 1, memory strategy. If `"persistent"`, the target stays in memory until the end of the pipeline (unless storage is `"worker"`, in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If `"transient"`, the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. `format = "file"` with `repository = "aws"`), this memory strategy applies to the temporary local copy of the file: `"persistent"` means it remains until the end of the pipeline and is then deleted, and `"transient"` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.
garbage_collection	
	Logical, whether to run `base::gc()` just before the target runs.

deployment	Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.
priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
storage	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Most of the arguments are passed to the $compile(), $sample(), and $summary() methods of the
CmdStanModel class. If you previously compiled the model in an upstream tar_stan_compile()
target, then the model should not recompile.

Value

tar_stan_mcmc() returns a list of target objects. See the "Target objects" section for background.
The target names use the name argument as a prefix, and the individual elements of stan_files
appear in the suffixes where applicable. As an example, the specific target objects returned by
tar_stan_mcmc(name = x, stan_files = "y.stan", ...) are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with paths to the
  model file and compiled executable.

- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if compile
  = "original". Returns a character vector of lines in the model file.

- x_data: run the R expression in the data argument to produce a Stan dataset for the model.
  Returns a Stan data list.

- x_mcmc_y: run MCMC on the model and the dataset. Returns a cmdstanr CmdStanMCMC
  object with all the results.

- x_draws_y: extract draws from x_mcmc_y. Omitted if draws = FALSE. Returns a tidy data
  frame of draws.

- x_summary_y: extract compact summaries from x_mcmc_y. Returns a tidy data frame of sum-
  maries. Omitted if summary = FALSE.

- x_diagnostics: extract HMC diagnostics from x_mcmc_y. Returns a tidy data frame of HMC
  diagnostics. Omitted if diagnostics = FALSE.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists
of target objects. Target objects represent skippable steps of the analysis pipeline as described
at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.
ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis
pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories
explains target factories (functions like this one which generate targets) and the design specifica-
tion at https://books.ropensci.org/targets-design/ details the structure and composition
of target objects.

See Also

Other MCMC: tar_stan_mcmc_rep_diagnostics(), tar_stan_mcmc_rep_draws(), tar_stan_mcmc_rep_summary()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mcmc(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    variables = "beta",
    summaries = list(~quantile(.x, probs = c(0.25, 0.75))),
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
)
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_mcmc_rep_diagnostics

Multiple MCMCs per model with sampler diagnostics

Description

`tar_stan_mcmc_rep_diagnostics()` creates targets to run MCMC multiple times per model and save only the sampler diagnostics from each run.

Usage

```
tar_stan_mcmc_rep_diagnostics(
  name,
  stan_files,
  data = list(),
  batches = 1L,
  reps = 1L,
  combine = FALSE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
```

```
pedantic = FALSE,
include_paths = NULL,
cpp_options = list(),
stanc_options = list(),
force_recompile = FALSE,
seed = NULL,
refresh = NULL,
init = NULL,
save_latent_dynamics = FALSE,
output_dir = NULL,
output_basename = NULL,
sig_figs = NULL,
chains = 4,
parallel_chains = getOption("mc.cores", 1),
chain_ids = seq_len(chains),
threads_per_chain = NULL,
opencl_ids = NULL,
iter_warmup = NULL,
iter_sampling = NULL,
save_warmup = FALSE,
thin = NULL,
max_treedepth = NULL,
adapt_engaged = TRUE,
adapt_delta = NULL,
step_size = NULL,
metric = NULL,
metric_file = NULL,
inv_metric = NULL,
init_buffer = NULL,
term_buffer = NULL,
window = NULL,
fixed_param = FALSE,
show_messages = TRUE,
diagnostics = c("divergences", "treedepth", "ebfmi"),
inc_warmup = FALSE,
data_copy = character(0),
tidy_eval = targets::tar_option_get("tidy_eval"),
packages = targets::tar_option_get("packages"),
library = targets::tar_option_get("library"),
format = "qs",
format_df = "fst_tbl",
repository = targets::tar_option_get("repository"),
error = targets::tar_option_get("error"),
memory = "transient",
garbage_collection = TRUE,
deployment = targets::tar_option_get("deployment"),
priority = targets::tar_option_get("priority"),
resources = targets::tar_option_get("resources"),
```

```
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	Code to generate a single replication of a simulated dataset. The workflow simulates multiple datasets, and each model runs on each dataset. To join data on to the model summaries, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read <https://docs.ropensci.org/stantargets/articles/simulation.html>.
batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.
combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()](#) method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *[Pedantic mode](#)* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the [$check_syntax()](#) method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.

cpp_options   (list) Any makefile options to be used when compiling the model (STAN_THREADS,
                   STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the
                   make/local file. For an example of using threading see the Stan case study
                   Reduce Sum: A Minimal Example.

stanc_options   (list) Any Stan-to-C++ transpiler options to be used when compiling the model.
                   See the **Examples** section below as well as the stanc chapter of the CmdStan
                   Guide for more details on available options: https://mc-stan.org/docs/cmdstan-
                   guide/stanc.html.

force_recompile
                   (logical) Should the model be recompiled even if was not modified since last
                   compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile
                   option.

seed   (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of
                   multi-chain sampling the single seed will automatically be augmented by the
                   the run (chain) ID so that each chain uses a different seed. The exception is the
                   transformed data block, which defaults to using same seed for all chains so that
                   the same data is generated for all chains if RNG functions are used. The only
                   time seed should be specified as a vector (one element per chain) is if RNG
                   functions are used in transformed data and the goal is to generate *different* data
                   for each chain.

refresh   (non-negative integer) The number of iterations between printed screen updates.
                   If refresh = 0, only error messages will be printed.

init   (multiple options) The initialization method to use for the variables declared in
                   the parameters block of the Stan program. One of the following:

                       • A real number x>0. This initializes *all* parameters randomly between [-x,x]
                         on the *unconstrained* parameter space.;
                       • The number 0. This initializes *all* parameters to 0;
                       • A character vector of paths (one per chain) to JSON or Rdump files con-
                         taining initial values for all or some parameters. See write_stan_json()
                         to write R objects to JSON files compatible with CmdStan.
                       • A list of lists containing initial values for all or some parameters. For
                         MCMC the list should contain a sublist for each chain. For other model
                         fitting methods there should be just one sublist. The sublists should have
                         named elements corresponding to the parameters for which you are speci-
                         fying initial values. See **Examples**.
                       • A function that returns a single list with names corresponding to the param-
                         eters for which you are specifying initial values. The function can take no
                         arguments or a single argument chain_id. For MCMC, if the function has
                         argument chain_id it will be supplied with the chain id (from 1 to number
                         of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics
                   (logical) Should auxiliary diagnostic information about the latent dynamics be
                   written to temporary diagnostic CSV files? This argument replaces CmdStan's
                   diagnostic_file argument and the content written to CSV is controlled by the
                   user's CmdStan installation and not CmdStanR (for some algorithms no content
                   may be written). The default is FALSE, which is appropriate for almost every use

case. To save the temporary files created when save_latent_dynamics=TRUE see the $save_latent_dynamics_files() method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., $save_output_files()). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

output_basename
                  (string) A string to use as a prefix for the names of the output CSV files of CmdStan. If NULL (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

sig_figs   (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

chains   (positive integer) The number of Markov chains to run. The default is 4.

parallel_chains
                  (positive integer) The *maximum* number of MCMC chains to run in parallel. If parallel_chains is not specified then the default is to look for the option "mc.cores", which can be set for an entire R session by options(mc.cores=value). If the "mc.cores" option has not been set then the default is 1.

chain_ids   (integer vector) A vector of chain IDs. Must contain as many unique positive integers as the number of chains. If not set, the default chain IDs are used (integers starting from 1).

threads_per_chain
                  (positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when using the Stan functions reduce_sum() or map_rect()). This is in contrast with parallel_chains, which specifies the number of chains to run in parallel. The actual number of CPU cores used is parallel_chains*threads_per_chain. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.

opencl_ids   (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with cpp_options = list(stan_opencl = TRUE) for this argument to have an effect.

iter_warmup   (positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as num_warmup.

iter_sampling   (positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as num_samples.

save_warmup	(logical) Should warmup iterations be saved? The default is FALSE.
thin	(positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.
max_treedepth	(positive integer) The maximum allowed tree depth for the NUTS engine. See the *Tree Depth* section of the CmdStan User's Guide for more details.
adapt_engaged	(logical) Do warmup adaptation? The default is TRUE. If a precomputed inverse metric is specified via the inv_metric argument (or metric_file) then, if adapt_engaged=TRUE, Stan will use the provided inverse metric just as an initial guess during adaptation. To turn off adaptation when using a precomputed inverse metric set adapt_engaged=FALSE.
adapt_delta	(real in (0,1)) The adaptation target acceptance statistic.
step_size	(positive real) The *initial* step size for the discrete approximation to continuous Hamiltonian dynamics. This is further tuned during warmup.
metric	(string) One of "diag_e", "dense_e", or "unit_e", specifying the geometry of the base manifold. See the *Euclidean Metric* section of the CmdStan User's Guide for more details. To specify a precomputed (inverse) metric, see the inv_metric argument below.
metric_file	(character vector) The paths to JSON or Rdump files (one per chain) compatible with CmdStan that contain precomputed inverse metrics. The metric_file argument is inherited from CmdStan but is confusing in that the entry in JSON or Rdump file(s) must be named inv_metric, referring to the *inverse* metric. We recommend instead using CmdStanR's inv_metric argument (see below) to specify an inverse metric directly using a vector or matrix from your R session.
inv_metric	(vector, matrix) A vector (if metric='diag_e') or a matrix (if metric='dense_e') for initializing the inverse metric. This can be used as an alternative to the metric_file argument. A vector is interpreted as a diagonal metric. The inverse metric is usually set to an estimate of the posterior covariance. See the adapt_engaged argument above for details about (and control over) how specifying a precomputed inverse metric interacts with adaptation.
init_buffer	(nonnegative integer) Width of initial fast timestep adaptation interval during warmup.
term_buffer	(nonnegative integer) Width of final fast timestep adaptation interval during warmup.
window	(nonnegative integer) Initial width of slow timestep/metric adaptation interval.
fixed_param	(logical) When TRUE, call CmdStan with argument "algorithm=fixed_param". The default is FALSE. The fixed parameter sampler generates a new sample without changing the current state of the Markov chain; only generated quantities may change. This can be useful when, for example, trying to generate pseudo-data using the generated quantities block. If the parameters block is empty then using fixed_param=TRUE is mandatory. When fixed_param=TRUE the chains and parallel_chains arguments will be set to 1.
show_messages	(logical) When TRUE (the default), prints all output during the sampling process, such as iteration numbers and elapsed times. If the output is silenced then the $output() method of the resulting fit object can be used to display the silenced messages.

diagnostics (character vector) The diagnostics to automatically check and warn about after sampling. Setting this to an empty string "" or NULL can be used to prevent CmdStanR from automatically reading in the sampler diagnostics from CSV if you wish to manually read in the results and validate them yourself, for example using `read_cmdstan_csv()`. The currently available diagnostics are "divergences", "treedepth", and "ebfmi" (the default is to check all of them).

These diagnostics are also available after fitting. The `$sampler_diagnostics()` method provides access the diagnostic values for each iteration and the `$diagnostic_summary()` method provides summaries of the diagnostics and can regenerate the warning messages.

Diagnostics like R-hat and effective sample size are *not* currently available via the diagnostics argument but can be checked after fitting using the `$summary()` method.

inc_warmup (logical) Should warmup draws be included? Defaults to FALSE. Ignored except when used with [CmdStanMCMC](#) objects.

data_copy Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read `https://docs.ropensci.org/stantargets/articles/simulation.html`.

tidy_eval Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.

packages Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library Character vector of library paths to try when loading packages.

format Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().

format_df Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().

repository Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends,

but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the `names` argument of functions like [tar_make()](#). For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

Saved diagnostics could get quite large in storage, so please use thinning if necessary.

Most of the arguments are passed to the `$compile()` and `$generate_quantities()` methods of the `CmdStanModel` class. If you previously compiled the model in an upstream [tar_stan_compile()](#) target, then the model should not recompile.

Value

`tar_stan_mcmc_rep_diagnostics()` returns a list of target objects. See the "Target objects" section for background. The target names use the `name` argument as a prefix, and the individual elements of `stan_files` appear in the suffixes where applicable. As an example, the specific target objects returned by `tar_stan_mcmc_rep_diagnostics(name = x, stan_files = "y.stan")` are as follows.

- `x_file_y`: reproducibly track the Stan model file. Returns a character vector with the paths to the model file and compiled executable.
- `x_lines_y`: read the Stan model file for safe transport to parallel workers. Omitted if `compile = "original"`. Returns a character vector of lines in the model file.

- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the data argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.
- x_y: dynamic branching target to run MCMC once per dataset. Each dynamic branch returns a tidy data frames of HMC diagnostics corresponding to a batch of Stan data from x_data.
- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of HMC diagnostics.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and tar_option_get("seed"). This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the .seed list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other MCMC: tar_stan_mcmc(), tar_stan_mcmc_rep_draws(), tar_stan_mcmc_rep_summary()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mcmc_rep_diagnostics(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    batches = 2,
    reps = 2,
    stdout = R.utils::nullfile(),
```

```
      stderr = R.utils::nullfile()
  )
 )
}, ask = FALSE)
targets::tar_make()
})
}
```

tar_stan_mcmc_rep_draws

Multiple MCMC runs per model with draws

Description

`tar_stan_mcmc_rep_draws()` creates targets to run MCMC multiple times per model and save only the draws from each run.

Usage

```
tar_stan_mcmc_rep_draws(
  name,
  stan_files,
  data = list(),
  batches = 1L,
  reps = 1L,
  combine = FALSE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  output_basename = NULL,
  sig_figs = NULL,
  chains = 4,
  parallel_chains = getOption("mc.cores", 1),
  chain_ids = seq_len(chains),
  threads_per_chain = NULL,
```

```
  opencl_ids = NULL,
  iter_warmup = NULL,
  iter_sampling = NULL,
  save_warmup = FALSE,
  thin = NULL,
  max_treedepth = NULL,
  adapt_engaged = TRUE,
  adapt_delta = NULL,
  step_size = NULL,
  metric = NULL,
  metric_file = NULL,
  inv_metric = NULL,
  init_buffer = NULL,
  term_buffer = NULL,
  window = NULL,
  fixed_param = FALSE,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi"),
  inc_warmup = FALSE,
  variables = NULL,
  data_copy = character(0),
  transform = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = "transient",
  garbage_collection = TRUE,
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	Code to generate a single replication of a simulated dataset. The workflow simulates multiple datasets, and each model runs on each dataset. To join data on to

	the model summaries, include a `.join_data` element of your Stan data list with names and dimensions corresponding to those of the model. For details, read <https://docs.ropensci.org/stantargets/articles/simulation.html>.
batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.
combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is `TRUE`. If `FALSE` compilation can be done later via the [`$compile()`](#) method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is `TRUE`, but if you encounter an error we recommend trying again with `quiet=FALSE` to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to `NULL` to print to the console. Set to `R.utils::nullfile()` to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to `NULL` to print to the console. Set to `R.utils::nullfile()` to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or `.hpp` file if using `$save_hpp_file()`). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is `FALSE`. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *[Pedantic mode](#)* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the [`$check_syntax()`](#) method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (`STAN_THREADS`, `STAN_MPI`, `STAN_OPENCL`, etc.). Anything you would otherwise write in the `make/local` file. For an example of using threading see the Stan case study [Reduce Sum: A Minimal Example](#).
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the `stanc` chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	
	(logical) Should the model be recompiled even if was not modified since last compiled. The default is `FALSE`. Can also be set via a global `cmdstanr_force_recompile` option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the

the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.

refresh   (non-negative integer) The number of iterations between printed screen updates. If refresh = 0, only error messages will be printed.

init   (multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number x>0. This initializes *all* parameters randomly between [-x,x] on the *unconstrained* parameter space.;
- The number 0. This initializes *all* parameters to 0;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See write_stan_json() to write R objects to JSON files compatible with CmdStan.
- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument chain_id. For MCMC, if the function has argument chain_id it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the [$save_latent_dynamics_files()](save_latent_dynamics_files()) method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., [$save_output_files()](save_output_files())). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

output_basename

    (string) A string to use as a prefix for the names of the output CSV files of CmdStan. If `NULL` (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

sig_figs   (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

chains   (positive integer) The number of Markov chains to run. The default is 4.

parallel_chains

    (positive integer) The *maximum* number of MCMC chains to run in parallel. If `parallel_chains` is not specified then the default is to look for the option `"mc.cores"`, which can be set for an entire R session by `options(mc.cores=value)`. If the `"mc.cores"` option has not been set then the default is 1.

chain_ids   (integer vector) A vector of chain IDs. Must contain as many unique positive integers as the number of chains. If not set, the default chain IDs are used (integers starting from 1).

threads_per_chain

    (positive integer) If the model was [compiled](#) with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when using the Stan functions reduce_sum() or map_rect()). This is in contrast with `parallel_chains`, which specifies the number of chains to run in parallel. The actual number of CPU cores used is `parallel_chains*threads_per_chain`. For an example of using threading see the Stan case study [Reduce Sum: A Minimal Example](#).

opencl_ids   (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with `cpp_options = list(stan_opencl = TRUE)` for this argument to have an effect.

iter_warmup   (positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as `num_warmup`.

iter_sampling   (positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as `num_samples`.

save_warmup   (logical) Should warmup iterations be saved? The default is `FALSE`.

thin   (positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.

max_treedepth   (positive integer) The maximum allowed tree depth for the NUTS engine. See the *Tree Depth* section of the CmdStan User's Guide for more details.

adapt_engaged   (logical) Do warmup adaptation? The default is `TRUE`. If a precomputed inverse metric is specified via the `inv_metric` argument (or `metric_file`) then, if `adapt_engaged=TRUE`, Stan will use the provided inverse metric just as an initial guess during adaptation. To turn off adaptation when using a precomputed inverse metric set `adapt_engaged=FALSE`.

adapt_delta   (real in `(0,1)`) The adaptation target acceptance statistic.

step_size   (positive real) The *initial* step size for the discrete approximation to continuous Hamiltonian dynamics. This is further tuned during warmup.

metric	(string) One of `"diag_e"`, `"dense_e"`, or `"unit_e"`, specifying the geometry of the base manifold. See the *Euclidean Metric* section of the CmdStan User's Guide for more details. To specify a precomputed (inverse) metric, see the `inv_metric` argument below.
metric_file	(character vector) The paths to JSON or Rdump files (one per chain) compatible with CmdStan that contain precomputed inverse metrics. The `metric_file` argument is inherited from CmdStan but is confusing in that the entry in JSON or Rdump file(s) must be named `inv_metric`, referring to the *inverse* metric. We recommend instead using CmdStanR's `inv_metric` argument (see below) to specify an inverse metric directly using a vector or matrix from your R session.
inv_metric	(vector, matrix) A vector (if `metric='diag_e'`) or a matrix (if `metric='dense_e'`) for initializing the inverse metric. This can be used as an alternative to the `metric_file` argument. A vector is interpreted as a diagonal metric. The inverse metric is usually set to an estimate of the posterior covariance. See the `adapt_engaged` argument above for details about (and control over) how specifying a precomputed inverse metric interacts with adaptation.
init_buffer	(nonnegative integer) Width of initial fast timestep adaptation interval during warmup.
term_buffer	(nonnegative integer) Width of final fast timestep adaptation interval during warmup.
window	(nonnegative integer) Initial width of slow timestep/metric adaptation interval.
fixed_param	(logical) When TRUE, call CmdStan with argument `"algorithm=fixed_param"`. The default is FALSE. The fixed parameter sampler generates a new sample without changing the current state of the Markov chain; only generated quantities may change. This can be useful when, for example, trying to generate pseudo-data using the generated quantities block. If the parameters block is empty then using `fixed_param=TRUE` is mandatory. When `fixed_param=TRUE` the `chains` and `parallel_chains` arguments will be set to 1.
show_messages	(logical) When TRUE (the default), prints all output during the sampling process, such as iteration numbers and elapsed times. If the output is silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.
diagnostics	(character vector) The diagnostics to automatically check and warn about after sampling. Setting this to an empty string `""` or NULL can be used to prevent CmdStanR from automatically reading in the sampler diagnostics from CSV if you wish to manually read in the results and validate them yourself, for example using `read_cmdstan_csv()`. The currently available diagnostics are `"divergences"`, `"treedepth"`, and `"ebfmi"` (the default is to check all of them).
	These diagnostics are also available after fitting. The `$sampler_diagnostics()` method provides access the diagnostic values for each iteration and the `$diagnostic_summary()` method provides summaries of the diagnostics and can regenerate the warning messages.
	Diagnostics like R-hat and effective sample size are *not* currently available via the `diagnostics` argument but can be checked after fitting using the `$summary()` method.

inc_warmup	(logical) Should warmup draws be included? Defaults to FALSE. Ignored except when used with [CmdStanMCMC](#) objects.
variables	(character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If NULL (the default) then all variables are included.
- If an empty string (variables="") then none are included.
- For non-scalar variables all elements or specific elements can be selected:
  - variables = "theta" selects all elements of theta;
  - variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

data_copy	Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.
transform	Symbol or NULL, name of a function that accepts arguments data and draws and returns a data frame. Here, data is the JAGS data list supplied to the model, and draws is a data frame with one column per model parameter and one row per posterior sample. Any arguments other than data and draws must have valid default values because stantargets will not populate them. See the simulation-based calibration (SBC) section of the simulation vignette for an example.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](#), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target
should create a single output file. That output file is uploaded to the cloud and
tracked for changes where it exists in the cloud. The local file is deleted after
the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new tar-
  gets launch after that. (Visit https://books.ropensci.org/targets/
  debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is
  deliberately wrong so the target is not up to date for the next run of the
  pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
memory until the end of the pipeline (unless storage is "worker", in which case
targets unloads the value from memory right after storing it in order to avoid
sending copious data over a network). If "transient", the target gets unloaded
after every new target completes. Either way, the target gets automatically
loaded into memory whenever another target needs the value. For cloud-based
dynamic files (e.g. format = "file" with repository = "aws"), this memory
strategy applies to the temporary local copy of the file: "persistent" means
it remains until the end of the pipeline and is then deleted, and "transient"
means it gets deleted as soon as possible. The former conserves bandwidth, and
the latter conserves local storage.

garbage_collection
                 Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
central controlling R process. Otherwise, if deployment is "worker" and you
set up the pipeline with distributed/parallel computing, then the target runs on a
parallel worker. For more on distributed/parallel computing in targets, please
visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
when multiple competing targets are ready simultaneously. Targets with priori-
ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
computing functionality, alternative data storage formats, and other optional ca-
pabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
Must be one of the following values:

- "main": the target's return value is sent back to the host machine and
  saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the
  data needs to be loaded explicitly from another language. If you do use it,
  then the return value of the target is totally ignored when the target ends,

but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

Draws could take up a lot of storage. If storage becomes excessive, please consider thinning the draws or using `tar_stan_mcmc_rep_summary()` instead.

Most of the arguments are passed to the `$compile()` and `$sample()` methods of the `CmdStanModel` class. If you previously compiled the model in an upstream `tar_stan_compile()` target, then the model should not recompile.

Value

`tar_stan_mcmc_rep_draws()` returns a list of target objects. See the "Target objects" section for background. The target names use the `name` argument as a prefix, and the individual elements of `stan_files` appear in the suffixes where applicable. As an example, the specific target objects returned by `tar_stan_mcmc_rep_draws(name = x, stan_files = "y.stan")` are as follows.

- `x_file_y`: reproducibly track the Stan model file. Returns a character vector with the paths to the model file and compiled executable.
- `x_lines_y`: read the Stan model file for safe transport to parallel workers. Omitted if `compile = "original"`. Returns a character vector of lines in the model file.

- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the data argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.

- x_y: dynamic branching target to run MCMC once per dataset. Each dynamic branch returns a tidy data frames of draws corresponding to a batch of Stan data from x_data.

- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of draws.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and tar_option_get("seed"). This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the .seed list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other MCMC: tar_stan_mcmc(), tar_stan_mcmc_rep_diagnostics(), tar_stan_mcmc_rep_summary()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mcmc_rep_draws(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    batches = 2,
    reps = 2,
    stdout = R.utils::nullfile(),
```

```
      stderr = R.utils::nullfile()
    )
  )
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_mcmc_rep_summary
Multiple MCMCs per model with summaries

Description

Targets to run MCMC multiple times and save only the summary output from each run.

Usage

```
tar_stan_mcmc_rep_summary(
  name,
  stan_files,
  data = list(),
  batches = 1L,
  reps = 1L,
  combine = TRUE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  output_basename = NULL,
  sig_figs = NULL,
  chains = 4,
  parallel_chains = getOption("mc.cores", 1),
  chain_ids = seq_len(chains),
  threads_per_chain = NULL,
  opencl_ids = NULL,
```

```
    iter_warmup = NULL,
    iter_sampling = NULL,
    save_warmup = FALSE,
    thin = NULL,
    max_treedepth = NULL,
    adapt_engaged = TRUE,
    adapt_delta = NULL,
    step_size = NULL,
    metric = NULL,
    metric_file = NULL,
    inv_metric = NULL,
    init_buffer = NULL,
    term_buffer = NULL,
    window = NULL,
    fixed_param = FALSE,
    show_messages = TRUE,
    diagnostics = c("divergences", "treedepth", "ebfmi"),
    data_copy = character(0),
    variables = NULL,
    summaries = NULL,
    summary_args = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = "qs",
    format_df = "fst_tbl",
    repository = targets::tar_option_get("repository"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	Code to generate a single replication of a simulated dataset. The workflow simulates multiple datasets, and each model runs on each dataset. To join data on to the model summaries, include a `.join_data` element of your Stan data list with

	names and dimensions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.
batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.
combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the $compile() method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the $check_syntax() method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the

transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.

refresh   (non-negative integer) The number of iterations between printed screen updates. If refresh = 0, only error messages will be printed.

init   (multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number x>0. This initializes *all* parameters randomly between [-x,x] on the *unconstrained* parameter space.;
- The number 0. This initializes *all* parameters to 0;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See write_stan_json() to write R objects to JSON files compatible with CmdStan.
- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument chain_id. For MCMC, if the function has argument chain_id it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the [$save_latent_dynamics_files()](http://...) method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., [$save_output_files()](http://...)). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

output_basename

> (string) A string to use as a prefix for the names of the output CSV files of CmdStan. If `NULL` (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

sig_figs

> (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

chains

> (positive integer) The number of Markov chains to run. The default is 4.

parallel_chains

> (positive integer) The *maximum* number of MCMC chains to run in parallel. If `parallel_chains` is not specified then the default is to look for the option `"mc.cores"`, which can be set for an entire R session by `options(mc.cores=value)`. If the `"mc.cores"` option has not been set then the default is 1.

chain_ids

> (integer vector) A vector of chain IDs. Must contain as many unique positive integers as the number of chains. If not set, the default chain IDs are used (integers starting from 1).

threads_per_chain

> (positive integer) If the model was [compiled] with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when using the Stan functions reduce_sum() or map_rect()). This is in contrast with `parallel_chains`, which specifies the number of chains to run in parallel. The actual number of CPU cores used is `parallel_chains*threads_per_chain`. For an example of using threading see the Stan case study [Reduce Sum: A Minimal Example].

opencl_ids

> (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with `cpp_options = list(stan_opencl = TRUE)` for this argument to have an effect.

iter_warmup

> (positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as `num_warmup`.

iter_sampling

> (positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as `num_samples`.

save_warmup

> (logical) Should warmup iterations be saved? The default is `FALSE`.

thin

> (positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.

max_treedepth

> (positive integer) The maximum allowed tree depth for the NUTS engine. See the *Tree Depth* section of the CmdStan User's Guide for more details.

adapt_engaged

> (logical) Do warmup adaptation? The default is `TRUE`. If a precomputed inverse metric is specified via the `inv_metric` argument (or `metric_file`) then, if `adapt_engaged=TRUE`, Stan will use the provided inverse metric just as an initial guess during adaptation. To turn off adaptation when using a precomputed inverse metric set `adapt_engaged=FALSE`.

adapt_delta

> (real in `(0,1)`) The adaptation target acceptance statistic.

step_size

> (positive real) The *initial* step size for the discrete approximation to continuous Hamiltonian dynamics. This is further tuned during warmup.

metric   (string) One of "diag_e", "dense_e", or "unit_e", specifying the geometry of
                        the base manifold. See the *Euclidean Metric* section of the CmdStan User's
                        Guide for more details. To specify a precomputed (inverse) metric, see the
                        inv_metric argument below.

metric_file   (character vector) The paths to JSON or Rdump files (one per chain) compatible
                        with CmdStan that contain precomputed inverse metrics. The metric_file
                        argument is inherited from CmdStan but is confusing in that the entry in JSON
                        or Rdump file(s) must be named inv_metric, referring to the *inverse* metric.
                        We recommend instead using CmdStanR's inv_metric argument (see below) to
                        specify an inverse metric directly using a vector or matrix from your R session.

inv_metric   (vector, matrix) A vector (if metric='diag_e') or a matrix (if metric='dense_e')
                        for initializing the inverse metric. This can be used as an alternative to the
                        metric_file argument. A vector is interpreted as a diagonal metric. The in-
                        verse metric is usually set to an estimate of the posterior covariance. See the
                        adapt_engaged argument above for details about (and control over) how speci-
                        fying a precomputed inverse metric interacts with adaptation.

init_buffer   (nonnegative integer) Width of initial fast timestep adaptation interval during
                        warmup.

term_buffer   (nonnegative integer) Width of final fast timestep adaptation interval during
                        warmup.

window   (nonnegative integer) Initial width of slow timestep/metric adaptation interval.

fixed_param   (logical) When TRUE, call CmdStan with argument "algorithm=fixed_param".
                        The default is FALSE. The fixed parameter sampler generates a new sample with-
                        out changing the current state of the Markov chain; only generated quantities
                        may change. This can be useful when, for example, trying to generate pseudo-
                        data using the generated quantities block. If the parameters block is empty then
                        using fixed_param=TRUE is mandatory. When fixed_param=TRUE the chains
                        and parallel_chains arguments will be set to 1.

show_messages   (logical) When TRUE (the default), prints all output during the sampling process,
                        such as iteration numbers and elapsed times. If the output is silenced then the
                        $output() method of the resulting fit object can be used to display the silenced
                        messages.

diagnostics   (character vector) The diagnostics to automatically check and warn about af-
                        ter sampling. Setting this to an empty string "" or NULL can be used to pre-
                        vent CmdStanR from automatically reading in the sampler diagnostics from
                        CSV if you wish to manually read in the results and validate them yourself,
                        for example using read_cmdstan_csv(). The currently available diagnostics
                        are "divergences", "treedepth", and "ebfmi" (the default is to check all of
                        them).

                        These diagnostics are also available after fitting. The $sampler_diagnostics()
                        method provides access the diagnostic values for each iteration and the $diagnostic_summary()
                        method provides summaries of the diagnostics and can regenerate the warning
                        messages.

                        Diagnostics like R-hat and effective sample size are *not* currently available via
                        the diagnostics argument but can be checked after fitting using the $summary()
                        method.

data_copy	Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.
variables	(character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If NULL (the default) then all variables are included.
- If an empty string (variables="") then none are included.
- For non-scalar variables all elements or specific elements can be selected:
  - variables = "theta" selects all elements of theta;
  - variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

summaries	Optional list of summary functions passed to ... in posterior::summarize_draws() through $summary() on the CmdStanFit object.
summary_args	Optional list of summary function arguments passed to .args in posterior::summarize_draws() through $summary() on the CmdStanFit object.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic

files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Most of the arguments are passed to the $compile() and $sample() methods of the CmdStanModel class. If you previously compiled the model in an upstream [tar_stan_compile()](#) target, then the model should not recompile.

Value

tar_stan_mcmc_rep_summary() returns a list of target objects. See the "Target objects" section for background. The target names use the name argument as a prefix, and the individual elements of stan_files appear in the suffixes where applicable. As an example, the specific target objects returned by tar_stan_mcmc_rep_summary(name = x, stan_files = "y.stan") are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with the paths to the model file and compiled executable.
- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if compile = "original". Returns a character vector of lines in the model file.
- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the data argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.
- x_y: dynamic branching target to run MCMC once per dataset. Each dynamic branch returns a tidy data frames of summaries. corresponding to a batch of Stan data from x_data.
- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of summaries.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and `tar_option_get("seed")`. This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the `.seed` list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most `stantargets` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other MCMC: `tar_stan_mcmc()`, `tar_stan_mcmc_rep_diagnostics()`, `tar_stan_mcmc_rep_draws()`

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mcmc_rep_summary(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    batches = 2,
    reps = 2,
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
)
}, ask = FALSE)
targets::tar_make()
})
}
```

Description

`tar_stan_mle()` creates targets to optimize a Stan model once per model and separately save draws-like output and summary-like output.

Usage

```
tar_stan_mle(
  name,
  stan_files,
  data = list(),
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  algorithm = NULL,
  init_alpha = NULL,
  iter = NULL,
  tol_obj = NULL,
  tol_rel_obj = NULL,
  tol_grad = NULL,
  tol_rel_grad = NULL,
  tol_param = NULL,
  history_size = NULL,
  sig_figs = NULL,
  variables = NULL,
  variables_fit = NULL,
  summaries = list(),
  summary_args = list(),
  return_draws = TRUE,
  return_summary = TRUE,
  draws = NULL,
  summary = NULL,
```

```
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of Stan model files. If you supply multiple files, each model will run on the one shared dataset generated by the code in data. If you supply an unnamed vector, fs::path_ext_remove(basename(stan_files)) will be used as target name suffixes. If stan_files is a named vector, the suffixed will come from names(stan_files).
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

  - A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using [write_stan_json()](). See [write_stan_json()]() for details on the conversions performed on R objects before they are passed to Stan.
  - A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
  - NULL or an empty list if the Stan program has no data block. |
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()]() method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.

stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the $check_syntax() method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
refresh	(non-negative integer) The number of iterations between printed screen updates. If refresh = 0, only error messages will be printed.
init	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number x>0. This initializes *all* parameters randomly between [-x,x] on the *unconstrained* parameter space.;
- The number 0. This initializes *all* parameters to 0;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See write_stan_json() to write R objects to JSON files compatible with CmdStan.

- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument chain_id. For MCMC, if the function has argument chain_id it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the [$save_latent_dynamics_files()](#) method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., [$save_output_files()](#)). These temporary files are removed when the fitted model object is [garbage collected](#) (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

algorithm   (string) The optimization algorithm. One of "lbfgs", "bfgs", or "newton". The control parameters below are only available for "lbfgs" and "bfgs. For their default values and more details see the CmdStan User's Guide. The default values can also be obtained by running cmdstanr_example(method="optimize")$metadata().

init_alpha   (positive real) The initial step size parameter.

iter   (positive integer) The maximum number of iterations.

tol_obj   (positive real) Convergence tolerance on changes in objective function value.

tol_rel_obj   (positive real) Convergence tolerance on relative changes in objective function value.

tol_grad   (positive real) Convergence tolerance on the norm of the gradient.

tol_rel_grad   (positive real) Convergence tolerance on the relative norm of the gradient.

tol_param   (positive real) Convergence tolerance on changes in parameter value.

history_size   (positive integer) The size of the history used when approximating the Hessian. Only available for L-BFGS.

sig_figs	(positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
variables	(character vector) The variables to include.
variables_fit	Character vector of variables to include in the big `CmdStanFit` object returned by the model fit target. The `variables` argument, by contrast, is for the `"draws"` target only. The `"draws"` target can only access the variables in the `CmdStanFit` target. Control the variables in each with the `variables` and `variables_fit` arguments.
summaries	Optional list of summary functions passed to `...` in `posterior::summarize_draws()` through `$summary()` on the `CmdStanFit` object.
summary_args	Optional list of summary function arguments passed to `.args` in `posterior::summarize_draws()` through `$summary()` on the `CmdStanFit` object.
return_draws	Logical, whether to create a target for posterior draws. Saves `posterior::as_draws_df(fit$draws())` to a compressed `tibble`. Convenient, but duplicates storage.
return_summary	Logical, whether to create a target for `fit$summary()`.
draws	Deprecated on 2022-07-22. Use `return_draws` instead.
summary	Deprecated on 2022-07-22. Use `return_summary` instead.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as `"feather"` or `"aws_parquet"`. For more on storage formats, see the help file of `targets::tar_target()`.
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as `"feather"` or `"aws_parquet"`. For more on storage formats, see the help file of `targets::tar_target()`.
repository	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [`tar_resources_aws()`](), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. • `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target
should create a single output file. That output file is uploaded to the cloud and
tracked for changes where it exists in the cloud. The local file is deleted after
the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new tar-
  gets launch after that. (Visit https://books.ropensci.org/targets/
  debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is
  deliberately wrong so the target is not up to date for the next run of the
  pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                 memory until the end of the pipeline (unless storage is "worker", in which case
                 targets unloads the value from memory right after storing it in order to avoid
                 sending copious data over a network). If "transient", the target gets unloaded
                 after every new target completes. Either way, the target gets automatically
                 loaded into memory whenever another target needs the value. For cloud-based
                 dynamic files (e.g. format = "file" with repository = "aws"), this memory
                 strategy applies to the temporary local copy of the file: "persistent" means
                 it remains until the end of the pipeline and is then deleted, and "transient"
                 means it gets deleted as soon as possible. The former conserves bandwidth, and
                 the latter conserves local storage.

garbage_collection
                 Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                 central controlling R process. Otherwise, if deployment is "worker" and you
                 set up the pipeline with distributed/parallel computing, then the target runs on a
                 parallel worker. For more on distributed/parallel computing in targets, please
                 visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                 when multiple competing targets are ready simultaneously. Targets with priori-
                 ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
                 computing functionality, alternative data storage formats, and other optional ca-
                 pabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                 Must be one of the following values:

- "main": the target's return value is sent back to the host machine and
  saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the
  data needs to be loaded explicitly from another language. If you do use it,
  then the return value of the target is totally ignored when the target ends,

but each downstream target still attempts to load the data file (except when
`retrieval = "none"`).

If you select `storage = "none"`, then the return value of the target's com-
mand is ignored, and the data is not saved automatically. As with dynamic
files (`format = "file"`) it is the responsibility of the user to write to the
data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically
try to load the data from the data store as a dependency. As a corollary,
`storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`.
               Must be one of the following values:

               • `"main"`: the target's dependencies are loaded on the host machine and sent
                 to the worker before the target runs.
               • `"worker"`: the worker loads the targets dependencies.
               • `"none"`: the dependencies are not loaded at all. This choice is almost never
                 recommended. It is only for niche situations, e.g. the data needs to be
                 loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether
               the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
               target. Descriptions appear as target labels in functions like `tar_manifest()`
               and `tar_visnetwork()`, and they let you select subsets of targets for the `names`
               argument of functions like `tar_make()`. For example, tar_manifest(names =
               tar_described_as(starts_with("survival model"))) lists all the targets
               whose descriptions start with the character string `"survival model"`.

Details

Most of the arguments are passed to the `$compile()`, `$optimize()`, and `$summary()` methods of
the `CmdStanModel` class. If you previously compiled the model in an upstream `tar_stan_compile()`
target, then the model should not recompile.

Value

`tar_stan_mle()` returns a list of target objects. See the "Target objects" section for background.
The target names use the `name` argument as a prefix, and the individual elements of `stan_files`
appear in the suffixes where applicable. As an example, the specific target objects returned by
tar_stan_mle(name = x, stan_files = "y.stan", ...) are as follows.

   • `x_file_y`: reproducibly track the Stan model file. Returns a character vector with paths to the
     model file and compiled executable.

   • `x_lines_y`: read the Stan model file for safe transport to parallel workers. Omitted if `compile
     = "original"`. Returns a character vector of lines in the model file.

   • `x_data`: run the R expression in the `data` argument to produce a Stan dataset for the model.
     Returns a Stan data list.

- x_mle_y: run generated quantities on the model and the dataset. Returns a `cmdstanr CmdStanGQ` object with all the results.

- x_draws_y: extract maximum likelihood estimates from x_mle_y in draws format. Omitted if draws = FALSE. Returns a wide data frame of MLEs.

- x_summary_y: extract MLEs from from x_mle_y in summary format. Returns a long data frame of MLEs. Omitted if summary = FALSE.

Target objects

Most `stantargets` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other optimization: tar_stan_mle_rep_draws(), tar_stan_mle_rep_summary()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mle(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
)
}, ask = FALSE)
targets::tar_make()
})
}
```

tar_stan_mle_rep_draws

Multiple optimization runs per model with draws

Description

tar_stan_mle_rep_draws() creates targets to run maximum likelihood multiple times per model and save the MLEs in a wide-form draws-like data frame.

Usage

```
tar_stan_mle_rep_draws(
  name,
  stan_files,
  data = list(),
  batches = 1L,
  reps = 1L,
  combine = TRUE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  algorithm = NULL,
  init_alpha = NULL,
  iter = NULL,
  sig_figs = NULL,
  tol_obj = NULL,
  tol_rel_obj = NULL,
  tol_grad = NULL,
  tol_rel_grad = NULL,
  tol_param = NULL,
  history_size = NULL,
  data_copy = character(0),
  variables = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
```

```
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:
	• A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan.
	• A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
	• NULL or an empty list if the Stan program has no data block.
batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.
combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the `$compile()` method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.

stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the $check_syntax() method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
refresh	(non-negative integer) The number of iterations between printed screen updates. If refresh = 0, only error messages will be printed.
init	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number x>0. This initializes *all* parameters randomly between [-x,x] on the *unconstrained* parameter space.;
- The number 0. This initializes *all* parameters to 0;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See write_stan_json() to write R objects to JSON files compatible with CmdStan.

- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument chain_id. For MCMC, if the function has argument chain_id it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics
(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the [$save_latent_dynamics_files()](#) method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., [$save_output_files()](#)). These temporary files are removed when the fitted model object is [garbage collected](#) (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

algorithm   (string) The optimization algorithm. One of "lbfgs", "bfgs", or "newton". The control parameters below are only available for "lbfgs" and "bfgs. For their default values and more details see the CmdStan User's Guide. The default values can also be obtained by running cmdstanr_example(method="optimize")$metadata().

init_alpha   (positive real) The initial step size parameter.

iter   (positive integer) The maximum number of iterations.

sig_figs   (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

tol_obj   (positive real) Convergence tolerance on changes in objective function value.

tol_rel_obj   (positive real) Convergence tolerance on relative changes in objective function value.

tol_grad   (positive real) Convergence tolerance on the norm of the gradient.

tol_rel_grad   (positive real) Convergence tolerance on the relative norm of the gradient.

tol_param	(positive real) Convergence tolerance on changes in parameter value.
history_size	(positive integer) The size of the history used when approximating the Hessian. Only available for L-BFGS.
data_copy	Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.
variables	(character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If NULL (the default) then all variables are included.
- If an empty string (variables="") then none are included.
- For non-scalar variables all elements or specific elements can be selected:
  - variables = "theta" selects all elements of theta;
  - variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error	Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Most of the arguments are passed to the $compile() and $optimize() methods of the CmdStanModel class. If you previously compiled the model in an upstream [tar_stan_compile()](#) target, then the model should not recompile.

Value

tar_stan_mle_rep_draws() returns a list of target objects. See the "Target objects" section for background. The target names use the name argument as a prefix, and the individual elements of stan_files appear in the suffixes where applicable. As an example, the specific target objects returned by tar_stan_mcmc_rep_draws(name = x, stan_files = "y.stan") are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with paths to the model file and compiled executable.
- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if compile = "original". Returns a character vector of lines in the model file.
- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the data argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.
- x_y: dynamic branching target to run maximum likelihood once per dataset. Each dynamic branch returns a tidy data frames of maximum likelihood estimates corresponding to a batch of Stan data from x_data.
- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of maximum likelihood estimates.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and `tar_option_get("seed")`. This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the `.seed` list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most `stantargets` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other optimization: `tar_stan_mle()`, `tar_stan_mle_rep_summary()`

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mle_rep_draws(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    batches = 2,
    reps = 2,
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
)
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_mle_rep_summary

Multiple optimization runs per model with summaries

Description

tar_stan_mle_rep_summaries() creates targets to run maximum likelihood multiple times per model and save the MLEs in a long-form summary-like data frame.

Usage

```
tar_stan_mle_rep_summary(
  name,
  stan_files,
  data = list(),
  batches = 1L,
  reps = 1L,
  combine = TRUE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  algorithm = NULL,
  init_alpha = NULL,
  iter = NULL,
  tol_obj = NULL,
  tol_rel_obj = NULL,
  tol_grad = NULL,
  tol_rel_grad = NULL,
  tol_param = NULL,
  history_size = NULL,
  sig_figs = NULL,
  data_copy = character(0),
  variables = NULL,
  summaries = list(),
  summary_args = list(),
```

```
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = "qs",
    format_df = "fst_tbl",
    repository = targets::tar_option_get("repository"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using [write_stan_json()](). See [write_stan_json()]() for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- NULL or an empty list if the Stan program has no data block.

batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.
combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()]() method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.

stderr   Character of length 1, file path to write the stderr stream of the model when
                   it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to
                   suppress stderr. Does not apply to messages, warnings, or errors.

dir   (string) The path to the directory in which to store the CmdStan executable (or
                   .hpp file if using $save_hpp_file()). The default is the same location as the
                   Stan program.

pedantic   (logical) Should pedantic mode be turned on? The default is FALSE. Pedantic
                   mode attempts to warn you about potential issues in your Stan program beyond
                   syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference
                   Manual. **Note:** to do a pedantic check for a model without compiling it or for
                   a model that is already compiled the $check_syntax() method can be used
                   instead.

include_paths   (character vector) Paths to directories where Stan should look for files specified
                   in #include directives in the Stan program.

cpp_options   (list) Any makefile options to be used when compiling the model (STAN_THREADS,
                   STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the
                   make/local file. For an example of using threading see the Stan case study
                   Reduce Sum: A Minimal Example.

stanc_options   (list) Any Stan-to-C++ transpiler options to be used when compiling the model.
                   See the **Examples** section below as well as the stanc chapter of the CmdStan
                   Guide for more details on available options: https://mc-stan.org/docs/cmdstan-
                   guide/stanc.html.

force_recompile
                   (logical) Should the model be recompiled even if was not modified since last
                   compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile
                   option.

seed   (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of
                   multi-chain sampling the single seed will automatically be augmented by the
                   the run (chain) ID so that each chain uses a different seed. The exception is the
                   transformed data block, which defaults to using same seed for all chains so that
                   the same data is generated for all chains if RNG functions are used. The only
                   time seed should be specified as a vector (one element per chain) is if RNG
                   functions are used in transformed data and the goal is to generate *different* data
                   for each chain.

refresh   (non-negative integer) The number of iterations between printed screen updates.
                   If refresh = 0, only error messages will be printed.

init   (multiple options) The initialization method to use for the variables declared in
                   the parameters block of the Stan program. One of the following:

                   • A real number x>0. This initializes *all* parameters randomly between [-x,x]
                     on the *unconstrained* parameter space.;
                   • The number 0. This initializes *all* parameters to 0;
                   • A character vector of paths (one per chain) to JSON or Rdump files con-
                     taining initial values for all or some parameters. See write_stan_json()
                     to write R objects to JSON files compatible with CmdStan.

- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.

- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument chain_id. For MCMC, if the function has argument chain_id it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the $save_latent_dynamics_files() method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., $save_output_files()). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).

- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

algorithm   (string) The optimization algorithm. One of "lbfgs", "bfgs", or "newton". The control parameters below are only available for "lbfgs" and "bfgs. For their default values and more details see the CmdStan User's Guide. The default values can also be obtained by running cmdstanr_example(method="optimize")$metadata().

init_alpha   (positive real) The initial step size parameter.

iter   (positive integer) The maximum number of iterations.

tol_obj   (positive real) Convergence tolerance on changes in objective function value.

tol_rel_obj   (positive real) Convergence tolerance on relative changes in objective function value.

tol_grad   (positive real) Convergence tolerance on the norm of the gradient.

tol_rel_grad   (positive real) Convergence tolerance on the relative norm of the gradient.

tol_param   (positive real) Convergence tolerance on changes in parameter value.

history_size   (positive integer) The size of the history used when approximating the Hessian. Only available for L-BFGS.

sig_figs   (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

data_copy   Character vector of names of scalars in `data`. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a `.join_data` element of your Stan data list with names and dimensions corresponding to those of the model. For details, read `https://docs.ropensci.org/stantargets/articles/simulation.html`.

variables   (character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If `NULL` (the default) then all variables are included.
- If an empty string (`variables=""`) then none are included.
- For non-scalar variables all elements or specific elements can be selected:
  - `variables = "theta"` selects all elements of `theta`;
  - `variables = c("theta[1]", "theta[3]")` selects only the 1st and 3rd elements.

summaries   Optional list of summary functions passed to `...` in `posterior::summarize_draws()` through `$summary()` on the `CmdStanFit` object.

summary_args   Optional list of summary function arguments passed to `.args` in `posterior::summarize_draws()` through `$summary()` on the `CmdStanFit` object.

tidy_eval   Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading `packages`.

format   Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as `"feather"` or `"aws_parquet"`. For more on storage formats, see the help file of `targets::tar_target()`.

format_df   Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as `"feather"` or `"aws_parquet"`. For more on storage formats, see the help file of `targets::tar_target()`.

repository   Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.

Note: if repository is not "local" and format is "file" then the target
should create a single output file. That output file is uploaded to the cloud and
tracked for changes where it exists in the cloud. The local file is deleted after
the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new tar-
  gets launch after that. (Visit https://books.ropensci.org/targets/
  debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is
  deliberately wrong so the target is not up to date for the next run of the
  pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                  memory until the end of the pipeline (unless storage is "worker", in which case
                  targets unloads the value from memory right after storing it in order to avoid
                  sending copious data over a network). If "transient", the target gets unloaded
                  after every new target completes. Either way, the target gets automatically
                  loaded into memory whenever another target needs the value. For cloud-based
                  dynamic files (e.g. format = "file" with repository = "aws"), this memory
                  strategy applies to the temporary local copy of the file: "persistent" means
                  it remains until the end of the pipeline and is then deleted, and "transient"
                  means it gets deleted as soon as possible. The former conserves bandwidth, and
                  the latter conserves local storage.

garbage_collection
                  Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                  central controlling R process. Otherwise, if deployment is "worker" and you
                  set up the pipeline with distributed/parallel computing, then the target runs on a
                  parallel worker. For more on distributed/parallel computing in targets, please
                  visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                  when multiple competing targets are ready simultaneously. Targets with priori-
                  ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
                  computing functionality, alternative data storage formats, and other optional ca-
                  pabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                  Must be one of the following values:

- "main": the target's return value is sent back to the host machine and
  saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the
  data needs to be loaded explicitly from another language. If you do use it,
  then the return value of the target is totally ignored when the target ends,

but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string `"survival model"`.

Details

Most of the arguments are passed to the `$compile()` and `$optimize()` methods of the `CmdStanModel` class. If you previously compiled the model in an upstream `tar_stan_compile()` target, then the model should not recompile.

Value

`tar_stan_mle_rep_summaries()` returns a list of target objects. See the "Target objects" section for background. The target names use the `name` argument as a prefix, and the individual elements of `stan_files` appear in the suffixes where applicable. The specific target objects returned by tar_stan_mle_rep_summary(name = x, , stan_files = "y.stan") are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with paths to the model file and compiled executable.
- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if compile = "original". Returns a character vector of lines in the model file.
- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the data argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.

- x_y: dynamic branching target to run maximum likelihood once per dataset. Each dynamic branch returns a tidy data frames of maximum likelihood estimates corresponding to a batch of Stan data from x_data.

- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of maximum likelihood estimates.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and tar_option_get("seed"). This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the .seed list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other optimization: tar_stan_mle(), tar_stan_mle_rep_draws()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_mle_rep_summary(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    batches = 2,
    reps = 2,
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
```

```
)
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_summary   *One summary of a* CmdStanFit *object*

Description

Create a target to run the $summary() method of a CmdStanFit object.

Usage

```
tar_stan_summary(
  name,
  fit,
  data = NULL,
  variables = NULL,
  summaries = NULL,
  summary_args = NULL,
  format = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
fit	Symbol, name of a CmdStanFit object or an upstream target that returns a CmdStanFit object.
data	Code to generate the data for the Stan model.
variables	(character vector) The variables to include.
summaries	Optional list of summary functions passed to ... in posterior::summarize_draws() through $summary() on the CmdStanFit object.

summary_args   Optional list of summary function arguments passed to .args in posterior::summarize_draws()
                  through $summary() on the CmdStanFit object.

format   Character of length 1, storage format of the data frame of posterior summaries.
                  We recommend efficient data frame formats such as "feather" or "aws_parquet".
                  For more on storage formats, see the help file of targets::tar_target().

repository   Character of length 1, remote repository for target storage. Choices:

                  • "local": file system of the local machine.
                  • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a
                    non-AWS S3 bucket using the endpoint argument of tar_resources_aws(),
                    but versioning capabilities may be lost in doing so. See the cloud stor-
                    age section of https://books.ropensci.org/targets/data.html for
                    details for instructions.
                  • "gcp": Google Cloud Platform storage bucket. See the cloud storage sec-
                    tion of https://books.ropensci.org/targets/data.html for details for
                    instructions.

                  Note: if repository is not "local" and format is "file" then the target
                  should create a single output file. That output file is uploaded to the cloud and
                  tracked for changes where it exists in the cloud. The local file is deleted after
                  the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

                  • "stop": the whole pipeline stops and throws an error.
                  • "continue": the whole pipeline keeps going.
                  • "abridge": any currently running targets keep running, but no new tar-
                    gets launch after that. (Visit https://books.ropensci.org/targets/
                    debugging.html to learn how to debug targets using saved workspaces.)
                  • "null": The errored target continues and returns NULL. The data hash is
                    deliberately wrong so the target is not up to date for the next run of the
                    pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                  memory until the end of the pipeline (unless storage is "worker", in which case
                  targets unloads the value from memory right after storing it in order to avoid
                  sending copious data over a network). If "transient", the target gets unloaded
                  after every new target completes. Either way, the target gets automatically
                  loaded into memory whenever another target needs the value. For cloud-based
                  dynamic files (e.g. format = "file" with repository = "aws"), this memory
                  strategy applies to the temporary local copy of the file: "persistent" means
                  it remains until the end of the pipeline and is then deleted, and "transient"
                  means it gets deleted as soon as possible. The former conserves bandwidth, and
                  the latter conserves local storage.

garbage_collection
                  Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                  central controlling R process. Otherwise, if deployment is "worker" and you
                  set up the pipeline with distributed/parallel computing, then the target runs on a
                  parallel worker. For more on distributed/parallel computing in targets, please
                  visit https://books.ropensci.org/targets/crew.html.

priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).
resources	Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.
storage	Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
- `"worker"`: the worker saves/uploads the value.
- `"none"`: almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).
  If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.
  The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval	Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

`tar_stan_mcmc()` etc. with `summary = TRUE` already gives you a target with output from the `$summary()` method. Use `tar_stan_summary()` to create additional specialized summaries.

Value

tar_stan_summary() returns target object to summarize a CmdStanFit object. The return value of
the target is a tidy data frame of summaries returned by the $summary() method of the CmdStanFit
object. See the "Target objects" section for background.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists
of target objects. Target objects represent skippable steps of the analysis pipeline as described
at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.
ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis
pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories
explains target factories (functions like this one which generate targets) and the design specifica-
tion at https://books.ropensci.org/targets-design/ details the structure and composition
of target objects.

Examples

```
# First, write your Stan model file, e.g. model.stan.
# Then in _targets.R, write a pipeline like this:
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
# Running inside a temporary directory to avoid
# modifying the user's file space. The file "model.stan"
# created below lives in a temporary directory.
# This satisfies CRAN policies.
tar_stan_example_file("model.stan")
targets::tar_script({
library(stantargets)
list(
  # Run a model and produce default summaries.
  tar_stan_mcmc(
    your_model,
    stan_files = "model.stan",
    data = tar_stan_example_data()
  ),
  # Produce a more specialized summary
  tar_stan_summary(
    your_summary,
    fit = your_model_mcmc_model,
    data = your_model_data_model,
    variables = "beta",
    summaries = list(~quantile(.x, probs = c(0.25, 0.75)))
  )
)}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_vb   *One variational Bayes run per model with multiple outputs*

Description

Targets to run a Stan model once with variational Bayes and save multiple outputs.

Usage

```
tar_stan_vb(
  name,
  stan_files,
  data = list(),
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  algorithm = NULL,
  iter = NULL,
  grad_samples = NULL,
  elbo_samples = NULL,
  eta = NULL,
  adapt_engaged = NULL,
  adapt_iter = NULL,
  tol_rel_obj = NULL,
  eval_elbo = NULL,
  output_samples = NULL,
  sig_figs = NULL,
  variables = NULL,
  variables_fit = NULL,
  summaries = list(),
  summary_args = list(),
  return_draws = TRUE,
  return_summary = TRUE,
  draws = NULL,
  summary = NULL,
```

```
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of Stan model files. If you supply multiple files, each model will run on the one shared dataset generated by the code in data. If you supply an unnamed vector, fs::path_ext_remove(basename(stan_files)) will be used as target name suffixes. If stan_files is a named vector, the suffixed will come from names(stan_files).
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using [write_stan_json()](). See [write_stan_json()]() for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- NULL or an empty list if the Stan program has no data block.

compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()]() method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.

stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the $check_syntax() method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
refresh	(non-negative integer) The number of iterations between printed screen updates. If refresh = 0, only error messages will be printed.
init	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number x>0. This initializes *all* parameters randomly between [-x,x] on the *unconstrained* parameter space.;
- The number 0. This initializes *all* parameters to 0;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See write_stan_json() to write R objects to JSON files compatible with CmdStan.

- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.

- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument chain_id. For MCMC, if the function has argument chain_id it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the $save_latent_dynamics_files() method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., $save_output_files()). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).

- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

algorithm   (string) The algorithm. Either "meanfield" or "fullrank".

iter   (positive integer) The *maximum* number of iterations.

grad_samples   (positive integer) The number of samples for Monte Carlo estimate of gradients.

elbo_samples   (positive integer) The number of samples for Monte Carlo estimate of ELBO (objective function).

eta   (positive real) The step size weighting parameter for adaptive step size sequence.

adapt_engaged   (logical) Do warmup adaptation? The default is TRUE. If a precomputed inverse metric is specified via the inv_metric argument (or metric_file) then, if adapt_engaged=TRUE, Stan will use the provided inverse metric just as an initial guess during adaptation. To turn off adaptation when using a precomputed inverse metric set adapt_engaged=FALSE.

adapt_iter   (positive integer) The *maximum* number of adaptation iterations.

tol_rel_obj   (positive real) Convergence tolerance on the relative norm of the objective.

eval_elbo   (positive integer) Evaluate ELBO every Nth iteration.

output_samples	(positive integer) Use draws argument instead. output_samples will be deprecated in the future.
sig_figs	(positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
variables	(character vector) The variables to include.
variables_fit	Character vector of variables to include in the big CmdStanFit object returned by the model fit target. The variables argument, by contrast, is for the "draws" target only. The "draws" target can only access the variables in the CmdStanFit target. Control the variables in each with the variables and variables_fit arguments.
summaries	Optional list of summary functions passed to ... in posterior::summarize_draws() through $summary() on the CmdStanFit object.
summary_args	Optional list of summary function arguments passed to .args in posterior::summarize_draws() through $summary() on the CmdStanFit object.
return_draws	Logical, whether to create a target for posterior draws. Saves posterior::as_draws_df(fit$draws()) to a compressed tibble. Convenient, but duplicates storage.
return_summary	Logical, whether to create a target for fit$summary().
draws	Deprecated on 2022-07-22. Use return_draws instead.
summary	Deprecated on 2022-07-22. Use return_summary instead.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
repository	Character of length 1, remote repository for target storage. Choices: • "local": file system of the local machine. • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of tar_resources_aws(), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. • "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target
should create a single output file. That output file is uploaded to the cloud and
tracked for changes where it exists in the cloud. The local file is deleted after
the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new tar-
  gets launch after that. (Visit https://books.ropensci.org/targets/
  debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is
  deliberately wrong so the target is not up to date for the next run of the
  pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                     memory until the end of the pipeline (unless storage is "worker", in which case
                     targets unloads the value from memory right after storing it in order to avoid
                     sending copious data over a network). If "transient", the target gets unloaded
                     after every new target completes. Either way, the target gets automatically
                     loaded into memory whenever another target needs the value. For cloud-based
                     dynamic files (e.g. format = "file" with repository = "aws"), this memory
                     strategy applies to the temporary local copy of the file: "persistent" means
                     it remains until the end of the pipeline and is then deleted, and "transient"
                     means it gets deleted as soon as possible. The former conserves bandwidth, and
                     the latter conserves local storage.

garbage_collection
                     Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                     central controlling R process. Otherwise, if deployment is "worker" and you
                     set up the pipeline with distributed/parallel computing, then the target runs on a
                     parallel worker. For more on distributed/parallel computing in targets, please
                     visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                     when multiple competing targets are ready simultaneously. Targets with priori-
                     ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
                     computing functionality, alternative data storage formats, and other optional ca-
                     pabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                     Must be one of the following values:

- "main": the target's return value is sent back to the host machine and
  saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the
  data needs to be loaded explicitly from another language. If you do use it,
  then the return value of the target is totally ignored when the target ends,

but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval
Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue
An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description
Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

Most of the arguments are passed to the `$compile()`, `$variational()`, and `$summary()` methods of the `CmdStanModel` class. If you previously compiled the model in an upstream `tar_stan_compile()` target, then the model should not recompile.

Value

`tar_stan_vb()` returns a list of target objects. See the "Target objects" section for background. The target names use the `name` argument as a prefix, and the individual elements of `stan_files` appear in the suffixes where applicable. As an example, the specific target objects returned by `tar_stan_vb(name = x, stan_files = "y.stan", ...)` are as follows.

- `x_file_y`: reproducibly track the Stan model file. Returns a character vector with paths to the model file and compiled executable.

- `x_lines_y`: read the Stan model file for safe transport to parallel workers. Omitted if `compile = "original"`. Returns a character vector of lines in the model file.

- `x_data`: run the R expression in the `data` argument to produce a Stan dataset for the model. Returns a Stan data list.

- x_vb_y: run variational Bayes on the model and the dataset. Returns a `cmdstanr CmdStanVB` object with all the results.

- x_draws_y: extract draws from x_vb_y. Omitted if `draws = FALSE`. Returns a tidy data frame of draws.

- x_summary_y: extract compact summaries from x_vb_y. Returns a tidy data frame of summaries. Omitted if `summary = FALSE`.

Target objects

Most `stantargets` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other variational Bayes: `tar_stan_vb_rep_draws()`, `tar_stan_vb_rep_summary()`

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_vb(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    variables = "beta",
    summaries = list(~quantile(.x, probs = c(0.25, 0.75))),
    stdout = R.utils::nullfile(),
    stderr = R.utils::nullfile()
  )
)
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_vb_rep_draws   *Multiple variational Bayes runs per model with draws*

Description

tar_stan_vb_rep_draws() creates targets to run variational Bayes multiple times per model and save only the draws from each run.

Usage

```
tar_stan_vb_rep_draws(
  name,
  stan_files,
  data = list(),
  batches = 1L,
  reps = 1L,
  combine = FALSE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  algorithm = NULL,
  iter = NULL,
  grad_samples = NULL,
  elbo_samples = NULL,
  eta = NULL,
  adapt_engaged = NULL,
  adapt_iter = NULL,
  tol_rel_obj = NULL,
  eval_elbo = NULL,
  output_samples = NULL,
  sig_figs = NULL,
  data_copy = character(0),
  variables = NULL,
  transform = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
```

```
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = "qs",
  format_df = "fst_tbl",
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = "transient",
  garbage_collection = TRUE,
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using [write_stan_json()](). See [write_stan_json()]() for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- NULL or an empty list if the Stan program has no data block.

batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.
combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the [$compile()]() method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.

stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the $check_syntax() method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
refresh	(non-negative integer) The number of iterations between printed screen updates. If refresh = 0, only error messages will be printed.
init	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following: • A real number x>0. This initializes *all* parameters randomly between [-x,x] on the *unconstrained* parameter space.; • The number 0. This initializes *all* parameters to 0; • A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See write_stan_json() to write R objects to JSON files compatible with CmdStan.

- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument chain_id. For MCMC, if the function has argument chain_id it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the [$save_latent_dynamics_files()](#) method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., [$save_output_files()](#)). These temporary files are removed when the fitted model object is [garbage collected](#) (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

algorithm   (string) The algorithm. Either "meanfield" or "fullrank".

iter   (positive integer) The *maximum* number of iterations.

grad_samples   (positive integer) The number of samples for Monte Carlo estimate of gradients.

elbo_samples   (positive integer) The number of samples for Monte Carlo estimate of ELBO (objective function).

eta   (positive real) The step size weighting parameter for adaptive step size sequence.

adapt_engaged   (logical) Do warmup adaptation?

adapt_iter   (positive integer) The *maximum* number of adaptation iterations.

tol_rel_obj   (positive real) Convergence tolerance on the relative norm of the objective.

eval_elbo   (positive integer) Evaluate ELBO every Nth iteration.

output_samples (positive integer) Use draws argument instead. output_samples will be deprecated in the future.

sig_figs   (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant

	figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
data_copy	Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.
variables	(character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

  • If NULL (the default) then all variables are included.
  • If an empty string (variables="") then none are included.
  • For non-scalar variables all elements or specific elements can be selected:
    – variables = "theta" selects all elements of theta;
    – variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

transform	Symbol or NULL, name of a function that accepts arguments data and draws and returns a data frame. Here, data is the JAGS data list supplied to the model, and draws is a data frame with one column per model parameter and one row per posterior sample. Any arguments other than data and draws must have valid default values because stantargets will not populate them. See the simulation-based calibration (SBC) section of the simulation vignette for an example.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
repository	Character of length 1, remote repository for target storage. Choices:

  • "local": file system of the local machine.
  • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of tar_resources_aws(), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
  • "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                       Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends,

but each downstream target still attempts to load the data file (except when
retrieval = "none").

If you select storage = "none", then the return value of the target's com-
mand is ignored, and the data is not saved automatically. As with dynamic
files (format = "file") it is the responsibility of the user to write to the
data store from inside the target.

The distinguishing feature of storage = "none" (as opposed to format =
"file") is that in the general case, downstream targets will automatically
try to load the data from the data store as a dependency. As a corollary,
storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#).
                 Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent
  to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never
  recommended. It is only for niche situations, e.g. the data needs to be
  loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether
                 the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
                 target. Descriptions appear as target labels in functions like [tar_manifest()](#)
                 and [tar_visnetwork()](#), and they let you select subsets of targets for the names
                 argument of functions like [tar_make()](#). For example, tar_manifest(names =
                 tar_described_as(starts_with("survival model"))) lists all the targets
                 whose descriptions start with the character string "survival model".

Details

Draws could take up a lot of storage. If storage becomes excessive, please consider thinning the
draws or using tar_stan_vb_rep_summaries() instead.

Most of the arguments are passed to the $compile() and $variational() methods of the CmdStanModel
class. If you previously compiled the model in an upstream [tar_stan_compile()](#) target, then the
model should not recompile.

Value

tar_stan_vb_rep_summaries() returns a list of target objects. See the "Target objects" section
for background. The target names use the name argument as a prefix, and the individual elements
of stan_files appear in the suffixes where applicable. As an example, the specific target objects
returned by tar_stan_vb_rep_draws(name = x, stan_files = "y.stan") are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with paths to the
  model file and compiled executable.
- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if compile
  = "original". Returns a character vector of lines in the model file.

- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the data argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.

- x_y: dynamic branching target to run variational Bayes once per dataset. Each dynamic branch returns a tidy data frames of draws corresponding to a batch of Stan data from x_data.

- x: combine all branches of x_y into a single non-dynamic target. Suppressed if combine is FALSE. Returns a long tidy data frame of draws.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the seed argument, batch, rep, parent target name, and tar_option_get("seed"). This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the .seed list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most stantargets functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other variational Bayes: tar_stan_vb(), tar_stan_vb_rep_summary()

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_vb_rep_draws(
    your_model,
    stan_files = path,
    data = tar_stan_example_data(),
    batches = 2,
    reps = 2,
    stdout = R.utils::nullfile(),
```

```
      stderr = R.utils::nullfile()
    )
  )
}, ask = FALSE)
targets::tar_make()
})
}
```

---

tar_stan_vb_rep_summary

Multiple iterations per model of variational Bayes with summaries

---

Description

`tar_stan_vb_rep_summaries()` creates targets to run variational Bayes multiple times and save only the summary output from each run.

Usage

```
tar_stan_vb_rep_summary(
  name,
  stan_files,
  data = list(),
  batches = 1L,
  reps = 1L,
  combine = TRUE,
  compile = c("original", "copy"),
  quiet = TRUE,
  stdout = NULL,
  stderr = NULL,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = FALSE,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = NULL,
  algorithm = NULL,
  iter = NULL,
  grad_samples = NULL,
  elbo_samples = NULL,
  eta = NULL,
  adapt_engaged = NULL,
```

```
    adapt_iter = NULL,
    tol_rel_obj = NULL,
    eval_elbo = NULL,
    output_samples = NULL,
    sig_figs = NULL,
    data_copy = character(0),
    variables = NULL,
    summaries = list(),
    summary_args = list(),
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = "qs",
    format_df = "fst_tbl",
    repository = targets::tar_option_get("repository"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, base name for the collection of targets. Serves as a prefix for target names.
stan_files	Character vector of paths to known existing Stan model files created before running the pipeline.
data	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- NULL or an empty list if the Stan program has no data block.

batches	Number of batches. Each batch is a sequence of branch targets containing multiple reps. Each rep generates a dataset and runs the model on it.
reps	Number of replications per batch.

combine	Logical, whether to create a target to combine all the model results into a single data frame downstream. Convenient, but duplicates data.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the `$compile()` method.
quiet	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is TRUE, but if you encounter an error we recommend trying again with quiet=FALSE to see more of the output.
stdout	Character of length 1, file path to write the stdout stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stdout. Does not apply to messages, warnings, or errors.
stderr	Character of length 1, file path to write the stderr stream of the model when it runs. Set to NULL to print to the console. Set to R.utils::nullfile() to suppress stderr. Does not apply to messages, warnings, or errors.
dir	(string) The path to the directory in which to store the CmdStan executable (or .hpp file if using $save_hpp_file()). The default is the same location as the Stan program.
pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the `$check_syntax()` method can be used instead.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
cpp_options	(list) Any makefile options to be used when compiling the model (STAN_THREADS, STAN_MPI, STAN_OPENCL, etc.). Anything you would otherwise write in the make/local file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
stanc_options	(list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the stanc chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.
force_recompile	
	(logical) Should the model be recompiled even if was not modified since last compiled. The default is FALSE. Can also be set via a global cmdstanr_force_recompile option.
seed	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single seed will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time seed should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
refresh	(non-negative integer) The number of iterations between printed screen updates. If refresh = 0, only error messages will be printed.

init   (multiple options) The initialization method to use for the variables declared in
                         the parameters block of the Stan program. One of the following:

- A real number x>0. This initializes *all* parameters randomly between [-x,x]
  on the *unconstrained* parameter space.;
- The number 0. This initializes *all* parameters to 0;
- A character vector of paths (one per chain) to JSON or Rdump files con-
  taining initial values for all or some parameters. See write_stan_json()
  to write R objects to JSON files compatible with CmdStan.
- A list of lists containing initial values for all or some parameters. For
  MCMC the list should contain a sublist for each chain. For other model
  fitting methods there should be just one sublist. The sublists should have
  named elements corresponding to the parameters for which you are speci-
  fying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the param-
  eters for which you are specifying initial values. The function can take no
  arguments or a single argument chain_id. For MCMC, if the function has
  argument chain_id it will be supplied with the chain id (from 1 to number
  of chains) when called to generate the initial values. See **Examples**.

save_latent_dynamics

                         (logical) Should auxiliary diagnostic information about the latent dynamics be
                         written to temporary diagnostic CSV files? This argument replaces CmdStan's
                         diagnostic_file argument and the content written to CSV is controlled by the
                         user's CmdStan installation and not CmdStanR (for some algorithms no content
                         may be written). The default is FALSE, which is appropriate for almost every use
                         case. To save the temporary files created when save_latent_dynamics=TRUE
                         see the $save_latent_dynamics_files() method.

output_dir   (string) A path to a directory where CmdStan should write its output CSV files.
                         For interactive use this can typically be left at NULL (temporary directory) since
                         CmdStanR makes the CmdStan output (posterior draws and diagnostics) avail-
                         able in R via methods of the fitted model objects. The behavior of output_dir
                         is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory
  and only saved permanently if the user calls one of the $save_* methods of
  the fitted model object (e.g., $save_output_files()). These temporary
  files are removed when the fitted model object is garbage collected (manu-
  ally or automatically).
- If a path, then the files are created in output_dir with names corresponding
  to the defaults used by $save_output_files().

algorithm   (string) The algorithm. Either "meanfield" or "fullrank".

iter   (positive integer) The *maximum* number of iterations.

grad_samples   (positive integer) The number of samples for Monte Carlo estimate of gradients.

elbo_samples   (positive integer) The number of samples for Monte Carlo estimate of ELBO
                         (objective function).

eta   (positive real) The step size weighting parameter for adaptive step size sequence.

adapt_engaged   (logical) Do warmup adaptation?

adapt_iter	(positive integer) The *maximum* number of adaptation iterations.
tol_rel_obj	(positive real) Convergence tolerance on the relative norm of the objective.
eval_elbo	(positive integer) Evaluate ELBO every Nth iteration.
output_samples	(positive integer) Use draws argument instead. output_samples will be deprecated in the future.
sig_figs	(positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
data_copy	Character vector of names of scalars in data. These values will be inserted as columns in the output data frame for each rep. To join more than just scalars, include a .join_data element of your Stan data list with names and dimensions corresponding to those of the model. For details, read https://docs.ropensci.org/stantargets/articles/simulation.html.
variables	(character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If NULL (the default) then all variables are included.
- If an empty string (variables="") then none are included.
- For non-scalar variables all elements or specific elements can be selected:
    - variables = "theta" selects all elements of theta;
    - variables = c("theta[1]", "theta[3]") selects only the 1st and 3rd elements.

summaries	Optional list of summary functions passed to ... in posterior::summarize_draws() through $summary() on the CmdStanFit object.
summary_args	Optional list of summary function arguments passed to .args in posterior::summarize_draws() through $summary() on the CmdStanFit object.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Character of length 1, storage format of the data frame of posterior summaries. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
format_df	Character of length 1, storage format of the data frame targets such as posterior draws. We recommend efficient data frame formats such as "feather" or "aws_parquet". For more on storage formats, see the help file of targets::tar_target().
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.

- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

  Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

                    Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to [`tar_make_clustermq()`](#) and [`tar_make_future()`](#). Must be one of the following values:

- `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
- `"worker"`: the worker saves/uploads the value.
- `"none"`: almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

  If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to [`tar_make_clustermq()`](#) and [`tar_make_future()`](#). Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [`tar_manifest()`](#) and [`tar_visnetwork()`](#), and they let you select subsets of targets for the `names` argument of functions like [`tar_make()`](#). For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

Most of the arguments are passed to the `$compile()` and `$variational()` methods of the `CmdStanModel` class. If you previously compiled the model in an upstream [`tar_stan_compile()`](#) target, then the model should not recompile.

Value

`tar_stan_vb_rep_summaries()` returns a list of target objects. See the "Target objects" section for background. The target names use the `name` argument as a prefix, and the individual elements of `stan_files` appear in the suffixes where applicable. As an example, the specific target objects returned by `tar_stan_vb_rep_summary(name = x, stan_files = "y.stan")` are as follows.

- x_file_y: reproducibly track the Stan model file. Returns a character vector with paths to the model file and compiled executable.
- x_lines_y: read the Stan model file for safe transport to parallel workers. Omitted if `compile = "original"`. Returns a character vector of lines in the model file.
- x_data: use dynamic branching to generate multiple datasets by repeatedly running the R expression in the `data` argument. Each dynamic branch returns a batch of Stan data lists that x_y supplies to the model.
- x_y: dynamic branching target to run variational Bayes once per dataset. Each dynamic branch returns a tidy data frames of summaries corresponding to a batch of Stan data from x_data.
- x: combine all branches of x_y into a single non-dynamic target. Suppressed if `combine` is `FALSE`. Returns a long tidy data frame of summaries.

Seeds

Rep-specific random number generator seeds for the data and models are automatically set based on the `seed` argument, batch, rep, parent target name, and `tar_option_get("seed")`. This ensures the rep-specific seeds do not change when you change the batching configuration (e.g. 40 batches of 10 reps each vs 20 batches of 20 reps each). Each data seed is in the `.seed` list element of the output, and each Stan seed is in the .seed column of each Stan model output.

Target objects

Most `stantargets` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other variational Bayes: `tar_stan_vb()`, `tar_stan_vb_rep_draws()`

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
library(stantargets)
# Do not use temporary storage for stan files in real projects
# or else your targets will always rerun.
path <- tempfile(pattern = "", fileext = ".stan")
tar_stan_example_file(path = path)
list(
  tar_stan_vb_rep_summary(
    your_model,
```

```
        stan_files = path,
        data = tar_stan_example_data(),
        batches = 2,
        reps = 2,
        stdout = R.utils::nullfile(),
        stderr = R.utils::nullfile()
    )
  )
}, ask = FALSE)
targets::tar_make()
})
}
```

Index