Package 'guildai'

July 22, 2025

Type Package

Title Track Machine Learning Experiments

Version 0.0.1

Description 'Guild AI' is an open-source tool for managing machine learning experiments. It's for scientists, engineers, and researchers who want to run scripts, compare results, measure progress, and automate machine learning workflow. 'Guild AI' is a light weight, external tool that runs locally. It works with any framework, doesn't require any changes to your code, or access to any web services. Users can easily record experiment metadata, track model changes, manage experiment artifacts, tune hyperparameters, and share results. 'Guild AI' combines features from 'Git', 'SQLite', and 'Make' to provide a lab notebook for machine learning.

URL https://guildai.github.io/guildai-r/, https://github.com/guildai/guildai-r, https://my.guild.ai/

BugReports https://github.com/guildai/guildai-r/issues

License Apache License 2.0

SystemRequirements Python (>= 3.6.0)

Imports jsonlite, rappdirs, yaml, config, rlang, rstudioapi, readr, dplyr, processx, tibble

Encoding UTF-8

RoxygenNote 7.2.3

Suggests fs, envir, rmarkdown, quarto, testthat (>= 3.0.0), withr

Config/testthat/edition 3

NeedsCompilation no

Author Tomasz Kalinowski [aut, cph, cre], Posit, PBC [cph, fnd]

Maintainer Tomasz Kalinowski <tomasz@posit.co>

Repository CRAN

Date/Publication 2023-03-06 13:50:02 UTC

Contents

`guild_merge` *Copy run files into the current project working directory*

Description

Copy run files into the current project working directory

Usage

```
guild_merge(run = NULL, ...)
```

Arguments

`run`	a run selection
`...`	Arguments passed on to `guild_merge_cli` `filter` Filter runs using a filter expression. See Filter by Expression above for details. `operation` Filter runs with operations matching `VAL`. `label` Filter runs with labels matching VAL. To show unlabeled runs, use `unlabeled`. `unlabeled` (bool) Filter runs without labels. `tag` Filter runs with TAG. `comment` Filter runs with comments matching VAL. `marked` (bool) Filter marked runs. `unmarked` (bool) Filter unmarked runs. `started` Filter runs started within RANGE. See above for valid time ranges. `digest` Filter runs with a matching source code digest. `running` (bool) Filter runs that are still running. `completed` (bool) Filter completed runs. `error` (bool) Filter runs that exited with an error.

`terminated` (bool) Filter runs terminated by the user.

`pending` (bool) Filter pending runs.

`staged` (bool) Filter staged runs.

`target_dir` Directory to merge run files to (required if project directory cannot be determined for the run).

`sourcecode` (bool) Only copy run source code. Implies use of `skip_deps`. Cannot be used with `skip_sourcecode`.

`all` (bool) Copy all run files. May be used with `skip_sourcecode`, `skip_deps`, and `exclude` to copy all but the skipped/excluded files.

`skip_sourcecode` (bool) Don't copy run source code.

`skip_deps` (bool) Don't copy project-local dependencies.

`exclude` Exclude a file or pattern (may be used multiple times).

`no_summary` (bool) Don't generate a run summary.

`summary_name` Name used for the run summary. Use '$run_id' in the name to include the run ID.

`preview` (bool) Show what would happen on a merge.

`replace` (bool) Allow replacement of existing files. Cannot be used with `no_replace`

`no_replace` (bool) Fail if any target file would be replaced, even if that file is committed to the project VCS. Cannot be used with `replace`.

Value

`NULL`, invisibly. This function is called for its side effect.

Examples

```
## Not run:
guild_merge("--help")
runs_scalars() %>%
  dplyr::slice_max("epoch_acc") %>%
  guild_merge(I("--yes --replace"))

## End(Not run)
```

---

guild_run *Launch a guild run*

Description

Launch a guild run

Usage

```
guild_run(
  opspec = "train.R",
  flags = NULL,
  ...,
  echo = TRUE,
  as_job = getOption("guildai.run_as_job", TRUE)
)
```

Arguments

opspec	typically a path to an R script, but could be any string that guild recognizes as a valid operation.
flags	flag values for the run(s). • A named list or vector like `c(noise = .3, dropout = .4)`. Lists with vectors of length greater than 1 are automatically expanded into a grid of combinations for a batch of runs. For example, `list(noise = c(.2, .3), dropout = c(.4, .5)` expands to a batch of 4 runs. • A dataframe of flags for a batch of runs, one row per run. • A scalar string like `"noise=.3 dropout=.4"`. Any flags specification accepted at the terminal is valid here as well.
...	Arguments passed on to guild_run_cli `label` Set a label for the run. `tag` Associate TAG with run. May be used multiple times. `comment` Comment associated with the run. `run_dir` Use alternative run directory DIR. Cannot be used with `stage`. `stage` (bool) Stage an operation. `start` Start a staged run or restart an existing run. Cannot be used with `proto` or `run_dir`. `restart` Start a staged run or restart an existing run. Cannot be used with `proto` or `run_dir`. `proto` Use the operation, flags and source code from RUN. Flags may be added or redefined in this operation. Cannot be used with `restart`. `force_sourcecode` (bool) Use working source code when `restart` or `proto` is specified. Ignored otherwise. `gpus` Limit availabe GPUs to DEVICES, a comma separated list of device IDs. By default all GPUs are available. Cannot beused with `no_gpus`. `no_gpus` (bool) Disable GPUs for run. Cannot be used with `gpus`. `batch_label` Label to use for batch runs. Ignored for non-batch runs. `batch_tag` Associate TAG with batch. Ignored for non-batch runs. May be used multiple times. `batch_comment` Comment associated with batch. `optimizer` Optimize the run using the specified algorithm. See Optimizing Runs for more information.

`optimize` (bool) Optimize the run using the default optimizer.

`minimize` Column to minimize when running with an optimizer. See help for compare command for details specifying a column. May not be used with `maximize`.

`maximize` Column to maximize when running with an optimizer. See help for compare command for details specifying a column. May not be used with `minimize`.

`opt_flag` Flag for OPTIMIZER. May be used multiple times.

`max_trials` Maximum number of trials to run in batch operations. Default is optimizer specific. If optimizer is not specified, default is 20.

`trials` Maximum number of trials to run in batch operations. Default is optimizer specific. If optimizer is not specified, default is 20.

`stage_trials` (bool) For batch operations, stage trials without running them.

`remote` Run the operation remotely.

`force_flags` (bool) Accept all flag assignments, even for undefined or invalid values.

`force_deps` (bool) Continue even when a required resource is not resolved.

`stop_after` Stop operation after N minutes.

`fail_on_trial_error` (bool) Stop batch operations when a trial exits with an error.

`needed` (bool) Run only if there is not an available matching run. A matching run is of the same operation with the same flag values that is not stopped due to an error.

`background` (bool) Run operation in background.

`pidfile` Run operation in background, writing the background process ID to PIDFILE.

`no_wait` (bool) Don't wait for a remote operation to complete. Ignored if run is local.

`save_trials` Saves generated trials to a CSV batch file. See BATCH FILES for more information.

`keep_run` (bool) Keep run even when configured with 'delete-on-success'.

`keep_batch` (bool) Keep batch run rather than delete it on success.

`dep` Include PATH as a dependency.

`quiet` (bool) Do not show output.

`print_cmd` (bool) Show operation command and exit.

`print_env` (bool) Show operation environment and exit.

`print_trials` (bool) Show generated trials and exit.

`help_model` (bool) Show model help and exit.

`help_op` (bool) Show operation help and exit.

`test_output_scalars` Test output scalars on output. Use '-' to read from standard intput.

`test_sourcecode` (bool) Test source code selection.

`test_flags` (bool) Test flag configuration.

echo	whether output from the run is shown in the current R console. Note, this has no effect on whether expressions are echoed in the guild run stdout log. To disable echoing of expression in the run logs, specify `#| echo: false` in the run script frontmatter.
as_job	Run the operation as an RStudio background job. This is ignored outside of the RStudio IDE.

Value

NULL, invisibly. This function is called for its side effect.

---

guild_view *Launch Guild Viewer*

Description

Launch Guild Viewer

Usage

```
guild_view(
  runs = NULL,
  ...,
  host = NULL,
  port = NULL,
  include_batch = FALSE,
  no_open = FALSE,
  stop = FALSE
)
```

Arguments

runs	an optional runs selection.
...	passed on to `guild view`.
host	Name of host interface to listen on.
port	Port to listen on.
include_batch	(bool) Include batch runs.
no_open	(bool) Don't open Guild View in a browser.
stop	Stop the existing Guild View application.

Value

The url where the Guild View application can be accessed.

The url where the View application can be accessed, invisibly.

Examples

```
## Not run:
  guild_view()

  # see all supported options
  guild_view("--help")

  # three valid ways of supplying args to the guild executable
  guild_view("--port" = "5678")
  guild_view("--port", "5678")
  guild_view(c("--port", "5678"))

## End(Not run)
```

`install_guild` *Install guildai core*

Description

This installs the `guild` executable for use by the R package. It creates an isolated python virtual environment private to the R package and installs guildai into it. Repeated calls to `install_guild()` result in a fresh installation.

Usage

```
install_guild(guildai = "guildai", python = find_python())
```

Arguments

`guildai`	Character vector of arguments passed directly to `pip install`. To install the release version of guildai, this can be `"guildai"`. Special values of `"release"` and `"dev"` are also accepted.
`python`	Path to a python binary, used to create a private isolated venv.

Details

It requires that a suitable python version is available on the system.

Value

path to the `guild` executable

Note

`install_guild()` installs guild as an isolated VM. For guild to run a python operation, the python package `guildai` must be installed in the python library where it will be used, E.g., with `pip install guildai` or `reticulate::py_install()`.

Examples

```
## Not run:
## Install release version:
install_guild()

## Install release version using a specific python
# path_to_python <- reticulate::install_python() # path to python executable
install_guild("guildai", python = path_to_python)

## Install development version
install_guild(guildai = "dev", python = path_to_python)

## Install development version from URL
 install_guild(
   guildai = "https://api.github.com/repos/guildai/guildai/tarball/HEAD",
   python = path_to_python)

## Install local development version:
install_guild(c("-e", "~/guild/guildai"))

## End(Not run)
```

---

install_guild_cli *Install guild for usage in the Terminal*

Description

This function makes available the `guild` executable installed by `install_guild()` for usage in the Terminal.

Usage

```
install_guild_cli(
  dest = "~/bin",
  completions = basename(Sys.getenv("SHELL")) %in% c("bash", "zsh", "fish")
)
```

Arguments

`dest`	Directory where to place the `guild` executable. This should be a location on the `PATH`.
`completions`	Whether to also install shell completion helpers.

Details

Note that the guild executable installed by the R function `install_guild()` is not able to run python operations. To run python operations with guild, you must install guild into the target python installation with `pip install guildai`, and ensure that the desired guild executable is on the `PATH`.

Value

path to the installed guild executable, invisibly.

`is_run_active` *Is code executing in the context of a guild run?*

Description

Is code executing in the context of a guild run?

Usage

```
is_run_active()
```

Value

Boolean

`resolve_run_ids` *Resolve run ids*

Description

This is a equivalent to `runs_info(...)$id`, implemented more efficiently.

Usage

```
resolve_run_ids(runs = NULL, ..., all = TRUE)
```

Arguments

`runs`	a runs selection. If a data.frame, the columns `id` or `run` are used as the run id. Otherwise, the arguments are transformed into a character vector of cli arguments, and passed on to `guild` as a runs filter selection. Wrap the string in `I()` to avoid quoting the argument for the shell.
`...`	Other arguments passed on to `guild`
`all`	Return all matching runs. If `FALSE`, it returns the singly most recent run matching the selection criteria.

Details

guild supports a rich syntax for runs selection throughout the api. The same selection syntax is shared by the `runs_*` family of functions: `runs_info()`, `runs_scalars()`, `runs_comment()`, `runs_label()`, `runs_mark()`, `runs_tag()` `runs_delete()`,`runs_purge()`, `runs_restore()`, `runs_export()`, `runs_import()`.

Value

A character vector of run ids.

Note

You can call `Sys.setenv(GUILD_DEBUG_R = 1)` to see what system calls to the `guild` executable are made. This is useful when looking to understand how R arguments are transformed into a cli system call.

Examples

```
## Not run:
resolve_run_ids() # returns all run ids.
resolve_run_ids(1) # last run
resolve_run_ids(1:2) # last 2 runs
resolve_run_ids(1:2, operation = "train.py")

# three ways of getting ids for the currently staged or running runs
resolve_run_ids(staged = TRUE, running = TRUE)
resolve_run_ids("--staged", "--running")
resolve_run_ids(c("--staged", "--running"))
resolve_run_ids(I("--staged --running"))

# resolve_run_ids() uses the same selection rules and syntax as runs_info()
stopifnot(identical(
  resolve_run_ids(),
  runs_info()$id
))

## End(Not run)
```

---

runs_delete *Delete runs*

Description

Delete runs

Usage

```
runs_delete(runs = NULL, ...)

runs_purge(runs = NULL, ...)

runs_restore(runs = NULL, ...)
```

Arguments

runs	a runs selection
...	passed on to guild

Details

- `runs_delete()` moves runs into a guild managed "trash" directory.
- `runs_restore()` moves runs back into the main guild managed "runs" directory.
- `runs_purge()` permanently delete runs from "trash" directory. Only deleted runs can be purged.

Value

The value supplied to the `runs` argument, invisibly.

Note

To see deleted runs, do `guildai:::guild("runs list --deleted")` (`runs_info("--deleted")` supported soon)

`runs_export` *Move or copy runs*

Description

Move or copy runs

Usage

```
runs_export(runs = NULL, location, ..., move = FALSE, copy_resources = FALSE)

runs_import(runs = NULL, location, ..., move = FALSE, copy_resources = FALSE)
```

Arguments

`runs`	A runs selection
`location`	A directory where to place the runs, or find the runs.
`...`	passed on to guild
`move`	bool, whether the runs should be moved or copied by the import or export operation.
`copy_resources`	whether run resources should be also copied. If `FALSE`, (the default), run resources in the run directory will be symlinks to a guild managed storage location.

Value

The value supplied to the `runs` argument, invisibly.

runs_info *Get runs information*

Description

Returns a dataframe with information about the guild runs stored in guild home. Guild home is determined either by consulting the env var `Sys.getenv("GUILD_HOME")`, or if unset, by looking for a `.guild` directory, starting from the current working directory and walking up parent directories up to `~` or `/`.

Usage

```
runs_info(
  runs = NULL,
  ...,
  filter = NULL,
  operation = NULL,
  label = NULL,
  unlabeled = NA,
  tag = NULL,
  comment = NULL,
  marked = NA,
  unmarked = NA,
  started = NULL,
  digest = NULL,
  running = NA,
  completed = NA,
  error = NA,
  terminated = NA,
  pending = NA,
  staged = NA,
  deleted = NA,
  include_batch = NA
)
```

Arguments

`runs`	a runs specification.
`...`	passed on to `guild`.
`filter`	(character vector) Filter runs using a guild filter expression. See details section.
`operation`	(character vector) Filter runs with matching `operations`. A run is only included if any part of its full operation name matches the value.
`label`	(character vector) Filter runs with matching labels.
`unlabeled`	(bool) Filter only runs without labels.
`tag`	(character vector) Filter runs with `tag`.

comment	(character vector) Filter runs with comments matching.
marked	(bool) Filter only marked runs.
unmarked	(bool) Filter only unmarked runs.
started	(string) Filter only runs started within RANGE. See details for valid time ranges.
digest	(string) Filter only runs with a matching source code digest.
running	(bool) Filter only runs that are still running.
completed	(bool) Filter only completed runs.
error	(bool) Filter only runs that exited with an error.
terminated	(bool) Filter only runs terminated by the user.
pending	(bool) Filter only pending runs.
staged	(bool) Filter only staged runs.
deleted	(bool) Show deleted runs.
include_batch	(bool) Include batch runs.

Details

Guild has support for a custom filter expression syntax. This syntax is primarily useful in the terminal, and R users will generally prefer to filter the returned dataframe directly using `dplyr::filter()` or `[`. Nevertheless, R users can supply guild filter expressions here as well.

Filter by Expression:

Use `filter` to limit runs that match a filter expressions. Filter expressions compare run attributes, flag values, or scalars to target values. They may include multiple expressions with logical operators.

For example, to match runs with flag `batch-size` equal to 100 that have `loss` less than 0.8, use:

```
runs_info(filter = "batch-size = 10 and loss < 0.8")
```

Target values may be numbers, strings or lists containing numbers and strings. Lists are defined using square braces where each item is separated by a comma.

Comparisons may use the following operators: '=', '!=', '<', '<=', '>', '>='.

Text comparisons may use 'contains' to test for case-insensitive string membership. A value may be tested for membership or not in a list using 'in' or 'not in' respectively. An value may be tested for undefined using 'is undefined' or defined using 'is not undefined'.

Logical operators include 'or' and 'and'. An expression may be negated by preceding it with 'not'. Parentheses may be used to control the order of precedence when expressions are evaluated.

If a value reference matches more than one type of run information (e.g. a flag is named 'label', which is also a run attribute), the value is read in order of run attribute, then flag value, then scalar. To disambiguate the reference, use a prefix `attr:`, `flag:`, or `scalar:` as needed. For example, to filter using a flag value named 'label', use 'flag:label'.

Other examples:

```
"operation = train and acc > 0.9"
"operation = train and (acc > 0.9 or loss < 0.3)"
"batch-size = 100 or batch-size = 200"
```

```
"batch-size in [100,200]"
"batch-size not in [400,800]"
"batch-size is undefined"
"batch-size is not undefined"
"label contains best and operation not in [test,deploy]"
"status in [error,terminated]"
```

NOTE: Comments and tags are not supported in filter expressions at this time. Use `comment` and `tag` options along with filter expressions to further refine a selection.

Filter by Run Start Time:

Use `started` to limit runs to those that have started within a specified time range.

```
runs_info(started = 'last hour')
```

You can specify a time range using several different forms:

```
"after DATETIME"
"before DATETIME"
"between DATETIME and DATETIME"
"last N minutes|hours|days"
"today|yesterday"
"this week|month|year"
"last week|month|year"
"N days|weeks|months|years ago"
```

`DATETIME` may be specified as a date in the format `YY-MM-DD` (the leading `YY-` may be omitted) or as a time in the format `HH:MM` (24 hour clock). A date and time may be specified together as `DATE TIME`.

When using `between DATETIME and DATETIME`, values for `DATETIME` may be specified in either order.

When specifying values like `minutes` and `hours` the trailing `s` may be omitted to improve readability. You may also use `min` instead of `minutes` and `hr` instead of `hours`.

Examples:

```
"after 7-1"
"after 9:00"
"between 1-1 and 4-30"
"between 10:00 and 15:00"
"last 30 min"
"last 6 hours"
"today"
"this week"
"last month"
"3 weeks ago"
```

Filter by Run Status:

Runs may also be filtered by specifying one or more status filters: `running`, `completed`, `error`, and `terminated`. These may be used together to include runs that match any of the filters. For example to only include runs that were either terminated or exited with an error, use

```
runs_info(terminated = TRUE, error = TRUE)
```

Status filters are applied before `RUN` indexes are resolved. For example, a run index of `1` (as in, `runs_info(1, terminated = TRUE, error = TRUE)` is the latest run that matches the status filters.

Value

A dataframe (tibble) of runs

Examples

```
## Not run:
withr::with_package("dplyr", {

runs_info() # get the full set of runs
runs_info(1) # get the most recent run
runs_info(1:3) # get the last 3 runs

# some other examples for passing filter expressions
runs_info(staged = TRUE) # list only staged runs
runs_info(tag = c("convnet", "keras"), started = "last hour")
runs_info(error = TRUE)

runs <- runs_info()

# filter down the runs list to ones of interest
runs <- runs %>%
  filter(exit_status == 0) %>% # run ended without an error code
  filter(scalars$test_accuracy > .8) %>%
  filter(flags$epochs > 10) %>%
  arrange(scalars$test_loss) %>%
  select(id, flags, scalars)

# retrieve full scalars history from the runs of interest
runs$id %>%
  runs_scalars()

# export the best run
best_runs_dir <- tempfile()
dir.create(best_runs_dir)
runs %>%
  slice_max(scalars$test_accuracy) %>%
  runs_tag("best") %>%
  runs_export(best_runs_dir)

})

## End(Not run)
```

`runs_label` *Annotate runs*

Description

Annotate runs

Usage

```
runs_label(runs = NULL, label = NULL, ..., clear = FALSE)

runs_tag(runs = NULL, add = NULL, ..., remove = NULL, clear = FALSE)

runs_mark(runs = NULL, ..., clear = FALSE)

runs_comment(runs = NULL, comment = NULL, ..., delete = NULL, clear = FALSE)
```

Arguments

`runs`	a runs selection
`label`, `comment`	a string
`...`	passed on to `guild`. Pass `"--help"` to see all options.
`clear`	bool, whether to clear the existing tags/comments/label.
`add`, `remove`	a character vector of tags to add or remove
`delete`	integer vector, which comment(s) to delete, corresponding to the row number(s) in the dataframe found at `runs_info()$comments`.

Details

Annotation types and their recommended uses:

- **labels**: short, single line descriptions tailored for readability, not programmatic consumption. Labels are presented prominently in `guild_view()` and other run views.
- **tags**: short single-token strings. Tags can be used for organizing, grouping, and filtering runs.
- **comments**: longer (potentially multi-paragraph) descriptions of the run. Guild stores and presents run comments as log entries, complete with timestamps and author info.
- **marks**: A boolean attribute of a run (a run can be marked or unmarked). Marked runs are primarily used to declare a run as the preferred source for resolving an operation dependency. If a operation declares a dependency on another operation, and one of the dependent operation runs is marked, the marked run is used rather than the latest run for resolving the dependency. Marks can also be a convenient mechanism for ad-hoc filtering operations, but in general, tags are preferred over marks for this.

Value

The value supplied to the `runs` argument, invisibly.

Note

`runs_comment()` will open up an editor if `comment` is not supplied.

Examples

```
## Not run:
runs_info(1) %>% runs_tag(clear = TRUE)
runs_info(1) %>% runs_tag("foo")
runs_info(1)$tags
runs_info(1) %>% runs_tag("bar")
runs_info(1)$tags
runs_info(1) %>% runs_tag(remove = "foo")
runs_info(1)$tags
runs_info(1) %>% runs_tag("baz", clear = TRUE)
runs_info(1)$tags

## pass through options to `guild tag` cli subcommand
runs_tag("--help")

## End(Not run)
```

`runs_scalars` *Get full set of runs scalars*

Description

Get full set of runs scalars

Usage

```
runs_scalars(runs = NULL, ...)
```

Arguments

`runs`	a runs selection
`...`	passed on go `guild`

Value

A dataframe (tibble) of runs

Examples

```
## Not run:
runs_scalars(1) # scalars from most recent run
runs_scalars(1:2) # scalars form two most recent runs

# pass in a dataframe of runs
runs_info() %>%
```

```
  filter(flags$epochs > 5) %>%
  runs_scalars()

## End(Not run)
```

Index