# Package 'mlr3automl'

September 3, 2026

**Title** Automated Machine Learning for 'mlr3'

**Version** 0.1.0

**Description** Flexible automated machine learning (AutoML) system for the 'mlr3' ecosystem. Automatically selects a suitable machine learning algorithm and tunes its hyperparameters for a given task. Constructs preprocessing pipelines with multiple parallel branches using 'mlr3pipelines' and jointly optimizes them together with the learners using 'mlr3tuning'. The optimization is driven by asynchronous decentralized Bayesian optimization by Egele et al. (2023) <doi:10.1109/e-Science58273.2023.10254839>.

**License** LGPL-3

**URL** https://mlr3automl.mlr-org.com, https://github.com/mlr-org/mlr3automl

**BugReports** https://github.com/mlr-org/mlr3automl/issues

**Depends** mlr3 (>= 1.6.0), mlr3tuning (>= 1.6.0), R (>= 3.3.0), rush (>= 1.2.1)

**Imports** bbotk (>= 1.7.1), checkmate, data.table, lhs, mlr3learners (>= 0.14.0), mlr3mbo (>= 1.2.0), mlr3misc (>= 0.15.1), mlr3pipelines, paradox (>= 1.0.1), R6, utils

**Suggests** callr, e1071, fastai, glmnet, kknn, lgr, lightgbm, MASS, mirai, mlr3extralearners, mlr3torch (>= 0.3.3), mlr3viz, ranger, redux, reticulate, rpart, testthat (>= 3.0.0), torch, xgboost (>= 3.2.1.1)

**Additional_repositories** https://mlr-org.r-universe.dev

**Config/roxygen2/version** 8.0.0.9000

**Config/testthat/edition** 3

**Config/testthat/parallel** false

**Encoding** UTF-8

**NeedsCompilation** no

**Collate** 'mlr_auto.R' 'Auto.R' 'AutoCatboost.R' 'AutoExtraTrees.R'
'AutoFTTransformer.R' 'AutoFastai.R' 'AutoGlmnet.R'
'AutoKKNN.R' 'AutoLda.R' 'AutoLightGBM.R' 'AutoMLP.R'
'AutoRanger.R' 'AutoResNet.R' 'AutoSVM.R' 'AutoTabPFN.R'
'AutoXgboost.R' 'aaa.R' 'LearnerAuto.R' 'LearnerClassifAuto.R'
'LearnerClassifAutoCatboost.R'
'LearnerClassifAutoFTTransformer.R'
'LearnerClassifAutoFastai.R' 'LearnerClassifAutoGlmnet.R'
'LearnerClassifAutoKKNN.R' 'LearnerClassifAutoLightGBM.R'
'LearnerClassifAutoMLP.R' 'LearnerClassifAutoRanger.R'
'LearnerClassifAutoResNet.R' 'LearnerClassifAutoSVM.R'
'LearnerClassifAutoTabPFN.R' 'LearnerClassifAutoXgboost.R'
'LearnerRegrAuto.R' 'LearnerRegrAutoCatboost.R'
'LearnerRegrAutoFTTransformer.R' 'LearnerRegrAutoGlmnet.R'
'LearnerRegrAutoKKNN.R' 'LearnerRegrAutoLightGBM.R'
'LearnerRegrAutoMLP.R' 'LearnerRegrAutoRanger.R'
'LearnerRegrAutoResNet.R' 'LearnerRegrAutoSVM.R'
'LearnerRegrAutoTabPFN.R' 'LearnerRegrAutoXgboost.R' 'helper.R'
'install_python_learners.R' 'isolated_model.R'
'mlr_callbacks.R' 'sugar.R' 'train_auto.R' 'zzz.R'

**Author** Marc Becker [cre, aut, cph] (ORCID:
<https://orcid.org/0000-0002-8115-0400>),
Damir Pulatov [aut],
Baisu Zhou [aut],
Lona Koers [aut]

**Maintainer** Marc Becker <marcbecker@posteo.de>

**Repository** CRAN

**Date/Publication** 2026-09-03 12:20:03 UTC

# Contents

---

`mlr3automl-package` *mlr3automl: Automated Machine Learning for 'mlr3'*

---

## Description

Flexible automated machine learning (AutoML) system for the 'mlr3' ecosystem. Automatically selects a suitable machine learning algorithm and tunes its hyperparameters for a given task. Constructs preprocessing pipelines with multiple parallel branches using 'mlr3pipelines' and jointly optimizes them together with the learners using 'mlr3tuning'. The optimization is driven by asynchronous decentralized Bayesian optimization by Egele et al. (2023) doi:10.1109/eScience58273.2023.10254839.

## Author(s)

**Maintainer**: Marc Becker <marcbecker@posteo.de> (ORCID) [copyright holder]

Authors:

- Marc Becker <marcbecker@posteo.de> (ORCID) [copyright holder]
- Damir Pulatov <damirpolat@protonmail.com>
- Baisu Zhou <baisu.zhou@outlook.com>
- Lona Koers <lona.koers@gmail.com>

## See Also

Useful links:

- https://mlr3automl.mlr-org.com
- https://github.com/mlr-org/mlr3automl
- Report bugs at https://github.com/mlr-org/mlr3automl/issues

---

`Auto` *Auto Class*

---

## Description

This class is the base class for all autos.

## Value

Object of class R6::R6Class and `Auto`.

## Public fields

`id` (`character(1)`).

`properties` (`character()`).

`task_types` (`character()`).

`packages` (`character()`).

`devices` (`character()`).

## Methods

### Public methods:

- `Auto$new()`
- `Auto$check()`
- `Auto$graph()`
- `Auto$early_stopping_rounds()`
- `Auto$estimate_memory()`

- Auto$finalize_model()
- Auto$design_default()
- Auto$design_set()
- Auto$search_space()
- Auto$clone()

Auto$new(): Creates a new instance of this R6 class.

*Usage:*

```
Auto$new(
  id,
  properties = character(0),
  task_types = character(0),
  packages = character(0),
  devices = character(0)
)
```

*Arguments:*

id (character(1)).

properties (character()).

task_types (character()).

packages (character()).

devices (character()).

Auto$check(): Check if the auto is compatible with the task.

*Usage:*

```
Auto$check(task, memory_limit = Inf, large_data_set = FALSE, devices)
```

*Arguments:*

task (mlr3::Task).

memory_limit (integer(1)).

large_data_set (logical(1)).

devices (character())
Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

Auto$graph(): Create the graph for the auto.

*Usage:*

```
Auto$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

`Auto$early_stopping_rounds()`: Estimate the number of early stopping rounds (the patience) for a learner. `budget` is the maximum number of training rounds (boosting iterations or epochs) the learner may use. The patience is capped well below the budget, otherwise early stopping and validation-based internal tuning can never trigger and the learner always trains for the full budget.

*Usage:*

```
Auto$early_stopping_rounds(task, budget = Inf)
```

*Arguments:*

`task` (mlr3::Task).

`budget (integer(1))`
Maximum number of training rounds (boosting iterations or epochs) the learner may use.

`Auto$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
Auto$estimate_memory(task)
```

*Arguments:*

`task` (mlr3::Task).

`Auto$finalize_model()`: Prepare the graph learner for the final model fit. Called after tuning to undo tuning-only setup (e.g., timeout callbacks).

*Usage:*

```
Auto$finalize_model(graph_learner)
```

*Arguments:*

`graph_learner` (mlr3pipelines::GraphLearner).

`Auto$design_default()`: Default hyperparameters for the learner.

*Usage:*

```
Auto$design_default(task)
```

*Arguments:*

`task` (mlr3::Task).

`Auto$design_set()`: Get the initial hyperparameter set for the learner.

*Usage:*

```
Auto$design_set(task, measure, size)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`size (integer(1))`.

`Auto$search_space()`: Get the search space for the learner.

*Usage:*

```
Auto$search_space(task)
```

*Arguments:*

task (mlr3::Task).

`Auto$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
Auto$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

`AutoCatboost` *Catboost Auto*

---

## Description

Catboost auto.

## Value

Object of class R6::R6Class and `AutoCatboost`.

## Super class

`Auto` -> `AutoCatboost`

## Methods

### Public methods:

- `AutoCatboost$new()`
- `AutoCatboost$graph()`
- `AutoCatboost$estimate_memory()`
- `AutoCatboost$internal_measure()`
- `AutoCatboost$clone()`

`AutoCatboost$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoCatboost$new(id = "catboost")
```

*Arguments:*

id (character(1))
Identifier for the new instance.

`AutoCatboost$graph()`: Create the graph for the auto.

*Usage:*

```
AutoCatboost$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

`AutoCatboost$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoCatboost$estimate_memory(task)
```

*Arguments:*

task (mlr3::Task).

`AutoCatboost$internal_measure()`: Get the internal measure for the auto.

*Usage:*

```
AutoCatboost$internal_measure(measure, task)
```

*Arguments:*

measure (mlr3::Measure).

task (mlr3::Task).

`AutoCatboost$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoCatboost$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
auto("catboost")
```

---

`AutoExtraTrees` *Extra Trees Auto*

---

## Description

Extra Trees auto.

## Value

Object of class R6::R6Class and `AutoExtraTrees`.

## Super class

`Auto` -> `AutoExtraTrees`

## Methods

### Public methods:

- AutoExtraTrees$new()
- AutoExtraTrees$graph()
- AutoExtraTrees$estimate_memory()
- AutoExtraTrees$clone()

`AutoExtraTrees$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoExtraTrees$new(id = "extra_trees")
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`AutoExtraTrees$graph()`: Create the graph for the auto.

*Usage:*

```
AutoExtraTrees$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`n_threads` (`numeric(1)`).

`timeout` (`numeric(1)`).

`devices` (`character()`)
Devices to use. Allowed values are `"cpu"` and `"cuda"`. Default is "cpu".

`AutoExtraTrees$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoExtraTrees$estimate_memory(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoExtraTrees$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoExtraTrees$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("extra_trees")
```

AutoFastai *Fastai Auto*

## Description

Fastai auto.

## Value

Object of class R6::R6Class and AutoFastai.

## Python learners

Python learners like TabPFN and fastai run via reticulate and therefore need a Python installation with their required packages. There are two ways to provide it:

1. Do nothing and let reticulate::py_require() install the required packages into an ephemeral virtual environment automatically.
2. Point the RETICULATE_PYTHON environment variable to a Python installation that has the required packages installed.

We recommend option 2 when running on many workers, as it avoids the overhead of downloading and installing the packages on each worker. Use install_python_learners() to create a conda environment with the required packages and set RETICULATE_PYTHON to the returned Python binary.

The TabPFN learner additionally requires the TABPFN_TOKEN environment variable to download the model weights.

## Super class

Auto -> AutoFastai

## Methods

### Public methods:

- AutoFastai$new()
- AutoFastai$check()
- AutoFastai$graph()
- AutoFastai$estimate_memory()
- AutoFastai$internal_measure()
- AutoFastai$clone()

AutoFastai$new(): Creates a new instance of this R6 class.

*Usage:*

```
AutoFastai$new(id = "fastai")
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

`AutoFastai$check()`: Check if the auto is compatible with the task.

*Usage:*

```
AutoFastai$check(
  task,
  memory_limit = Inf,
  large_data_set = FALSE,
  devices = "cpu"
)
```

*Arguments:*

task (mlr3::Task).

memory_limit (integer(1)).

large_data_set (logical(1)).

devices (character())
: Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

`AutoFastai$graph()`: Create the graph for the auto.

*Usage:*

```
AutoFastai$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
: Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

`AutoFastai$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoFastai$estimate_memory(task)
```

*Arguments:*

task (mlr3::Task).

`AutoFastai$internal_measure()`: Get the internal measure for the auto.

*Usage:*

```
AutoFastai$internal_measure(measure, task)
```

*Arguments:*

measure (mlr3::Measure).

task (mlr3::Task).

`AutoFastai$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoFastai$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
auto("fastai")
```

---

`AutoFTTransformer` *FTTransformer Auto*

---

## Description

FTTransformer auto.

## Value

Object of class R6::R6Class and `AutoFTTransformer`.

## Super class

`Auto` -> `AutoFTTransformer`

## Methods

### Public methods:

- `AutoFTTransformer$new()`
- `AutoFTTransformer$graph()`
- `AutoFTTransformer$estimate_memory()`
- `AutoFTTransformer$clone()`

`AutoFTTransformer$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoFTTransformer$new(id = "ft_transformer")
```

*Arguments:*

`id (character(1))`
Identifier for the new instance.

`AutoFTTransformer$graph()`: Create the graph for the auto.

*Usage:*

```
AutoFTTransformer$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`n_threads (integer(1))`.

`timeout (integer(1))`.

`devices (character())`
Devices to use. Allowed values are `"cpu"` and `"cuda"`. Default is "cpu".

`AutoFTTransformer$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoFTTransformer$estimate_memory(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoFTTransformer$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoFTTransformer$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("ft_transformer")
```

---

`AutoGlmnet` *Glmnet Auto*

---

## Description

Glmnet auto.

## Value

Object of class R6::R6Class and `AutoGlmnet`.

## Super class

`Auto` -> `AutoGlmnet`

## Methods

### Public methods:

- `AutoGlmnet$new()`
- `AutoGlmnet$graph()`
- `AutoGlmnet$clone()`

`AutoGlmnet$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoGlmnet$new(id = "glmnet")
```

*Arguments:*

`id (character(1))`
Identifier for the new instance.

`AutoGlmnet$graph()`: Create the graph for the auto.

*Usage:*

```
AutoGlmnet$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`n_threads` (`integer(1)`).

`timeout` (`integer(1)`).

`devices` (`character()`)
Devices to use. Allowed values are `"cpu"` and `"cuda"`. Default is "cpu".

`AutoGlmnet$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoGlmnet$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("glmnet")
```

---

`AutoKKNN` *KKNN Auto*

---

## Description

Kknn auto.

## Value

Object of class R6::R6Class and `AutoKKNN`.

## Super class

`Auto` -> `AutoKKNN`

## Methods

### Public methods:

- `AutoKKNN$new()`
- `AutoKKNN$graph()`
- `AutoKKNN$search_space()`
- `AutoKKNN$clone()`

`AutoKKNN$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoKKNN$new(id = "kknn")
```

*Arguments:*

`id (character(1))`
Identifier for the new instance.

`AutoKKNN$graph()`: Create the graph for the auto.

*Usage:*

```
AutoKKNN$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`n_threads (integer(1))`.

`timeout (integer(1))`.

`devices (character())`
Devices to use. Allowed values are `"cpu"` and `"cuda"`. Default is "cpu".

`AutoKKNN$search_space()`: Get the search space for the auto.

*Usage:*

```
AutoKKNN$search_space(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoKKNN$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoKKNN$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("kknn")
```

---

`AutoLda` *Lda Auto*

---

## Description

Lda auto.

## Value

Object of class R6::R6Class and `AutoLda`.

## Super class

Auto -> AutoLda

## Methods

### Public methods:

- AutoLda$new()
- AutoLda$graph()
- AutoLda$clone()

AutoLda$new(): Creates a new instance of this R6 class.

*Usage:*

```
AutoLda$new(id = "lda")
```

*Arguments:*

id (character(1))
Identifier for the new instance.

AutoLda$graph(): Create the graph for the auto.

*Usage:*

```
AutoLda$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

AutoLda$clone(): The objects of this class are cloneable with this method.

*Usage:*

```
AutoLda$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
auto("lda")
```

AutoLightGBM *LightGBM Auto*

## Description

Lightgbm auto.

## Value

Object of class R6::R6Class and AutoLightGBM.

## Super class

Auto -> AutoLightGBM

## Methods

### Public methods:

- AutoLightGBM$new()
- AutoLightGBM$graph()
- AutoLightGBM$finalize_model()
- AutoLightGBM$estimate_memory()
- AutoLightGBM$internal_measure()
- AutoLightGBM$clone()

AutoLightGBM$new(): Creates a new instance of this R6 class.

*Usage:*

```
AutoLightGBM$new(id = "lightgbm")
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

AutoLightGBM$graph(): Create the graph for the auto.

*Usage:*

```
AutoLightGBM$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
: Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

AutoLightGBM$finalize_model(): Prepare the graph learner for the final model fit.

*Usage:*

```
AutoLightGBM$finalize_model(graph_learner)
```

*Arguments:*

`graph_learner` (mlr3pipelines::GraphLearner).

`AutoLightGBM$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoLightGBM$estimate_memory(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoLightGBM$internal_measure()`: Get the internal measure for the auto.

*Usage:*

```
AutoLightGBM$internal_measure(measure, task)
```

*Arguments:*

`measure` (mlr3::Measure).
`task` (mlr3::Task).

`AutoLightGBM$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoLightGBM$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("lightgbm")
```

---

`AutoMLP` *MLP Auto*

---

## Description

Mlp auto.

## Value

Object of class R6::R6Class and `AutoMLP`.

## Super class

`Auto` -> `AutoMLP`

## Methods

### Public methods:

- `AutoMLP$new()`
- `AutoMLP$graph()`
- `AutoMLP$estimate_memory()`
- `AutoMLP$clone()`

`AutoMLP$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoMLP$new(id = "mlp")
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`AutoMLP$graph()`: Create the graph for the auto.

*Usage:*

```
AutoMLP$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`n_threads` (`integer(1)`).

`timeout` (`integer(1)`).

`devices` (`character()`)
Devices to use. Allowed values are `"cpu"` and `"cuda"`. Default is "cpu".

`AutoMLP$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoMLP$estimate_memory(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoMLP$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoMLP$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("mlp")
```

AutoRanger *Ranger Auto*

## Description

Ranger auto.

## Value

Object of class R6::R6Class and AutoRanger.

## Super class

Auto -> AutoRanger

## Methods

### Public methods:

- AutoRanger$new()
- AutoRanger$graph()
- AutoRanger$estimate_memory()
- AutoRanger$clone()

**AutoRanger$new():** Creates a new instance of this R6 class.

*Usage:*

```
AutoRanger$new(id = "ranger")
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

**AutoRanger$graph():** Create the graph for the auto.

*Usage:*

```
AutoRanger$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
: Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

**AutoRanger$estimate_memory():** Estimate the memory for the auto.

*Usage:*

```
AutoRanger$estimate_memory(task)
```

*Arguments:*

task (mlr3::Task).

`AutoRanger$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoRanger$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
auto("ranger")
```

---

`AutoResNet` *ResNet Auto*

---

## Description

ResNet auto.

## Value

Object of class R6::R6Class and `AutoResNet`.

## Super class

`Auto` -> `AutoResNet`

## Methods

### Public methods:

- `AutoResNet$new()`
- `AutoResNet$graph()`
- `AutoResNet$estimate_memory()`
- `AutoResNet$clone()`

`AutoResNet$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoResNet$new(id = "resnet")
```

*Arguments:*

id (character(1))
Identifier for the new instance.

`AutoResNet$graph()`: Create the graph for the auto.

*Usage:*

```
AutoResNet$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`n_threads` (`integer(1)`).

`timeout` (`integer(1)`).

`devices` (`character()`)
Devices to use. Allowed values are `"cpu"` and `"cuda"`. Default is "cpu".

`AutoResNet$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoResNet$estimate_memory(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoResNet$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoResNet$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("resnet")
```

---

`AutoSVM` *SVM Auto*

---

## Description

Svm auto.

## Value

Object of class R6::R6Class and `AutoSVM`.

## Super class

`Auto` -> `AutoSVM`

## Methods

### Public methods:

- AutoSVM$new()
- AutoSVM$graph()
- AutoSVM$design_default()
- AutoSVM$clone()

`AutoSVM$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoSVM$new(id = "svm")
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`AutoSVM$graph()`: Create the graph for the auto.

*Usage:*

```
AutoSVM$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

`task` (mlr3::Task).

`measure` (mlr3::Measure).

`n_threads` (`integer(1)`).

`timeout` (`integer(1)`).

`devices` (`character()`)
Devices to use. Allowed values are `"cpu"` and `"cuda"`. Default is "cpu".

`AutoSVM$design_default()`: Default hyperparameters for the learner.

*Usage:*

```
AutoSVM$design_default(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoSVM$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoSVM$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("svm")
```

AutoTabPFN *TabPFN Auto*

## Description

Tabpfn auto.

## Value

Object of class R6::R6Class and AutoTabPFN.

## Python learners

Python learners like TabPFN and fastai run via reticulate and therefore need a Python installation with their required packages. There are two ways to provide it:

1. Do nothing and let reticulate::py_require() install the required packages into an ephemeral virtual environment automatically.
2. Point the RETICULATE_PYTHON environment variable to a Python installation that has the required packages installed.

We recommend option 2 when running on many workers, as it avoids the overhead of downloading and installing the packages on each worker. Use install_python_learners() to create a conda environment with the required packages and set RETICULATE_PYTHON to the returned Python binary.

The TabPFN learner additionally requires the TABPFN_TOKEN environment variable to download the model weights.

## Super class

Auto -> AutoTabPFN

## Methods

### Public methods:

- AutoTabPFN$new()
- AutoTabPFN$check()
- AutoTabPFN$graph()
- AutoTabPFN$estimate_memory()
- AutoTabPFN$design_default()
- AutoTabPFN$search_space()
- AutoTabPFN$clone()

AutoTabPFN$new(): Creates a new instance of this R6 class.

*Usage:*

```
AutoTabPFN$new(id = "tabpfn")
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

`AutoTabPFN$check()`: Check if the auto is compatible with the task.

*Usage:*

```
AutoTabPFN$check(
  task,
  memory_limit = Inf,
  large_data_set = FALSE,
  devices = "cpu"
)
```

*Arguments:*

task (mlr3::Task).

memory_limit (integer(1)).

large_data_set (logical(1)).

devices (character())
: Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

`AutoTabPFN$graph()`: Create the graph for the auto.

*Usage:*

```
AutoTabPFN$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
: Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

`AutoTabPFN$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoTabPFN$estimate_memory(task)
```

*Arguments:*

task (mlr3::Task).

`AutoTabPFN$design_default()`: Default hyperparameters for the learner.

*Usage:*

```
AutoTabPFN$design_default(task)
```

*Arguments:*

task (mlr3::Task).

`AutoTabPFN$search_space()`: Get the search space for the auto.

*Usage:*

```
AutoTabPFN$search_space(task)
```

*Arguments:*

`task` (mlr3::Task).

`AutoTabPFN$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoTabPFN$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
auto("tabpfn")
```

---

`AutoXgboost` *Xgboost Auto*

---

## Description

Xgboost auto.

## Value

Object of class R6::R6Class and `AutoXgboost`.

## Super class

`Auto` -> `AutoXgboost`

## Methods

### Public methods:

- `AutoXgboost$new()`
- `AutoXgboost$graph()`
- `AutoXgboost$finalize_model()`
- `AutoXgboost$estimate_memory()`
- `AutoXgboost$internal_measure()`
- `AutoXgboost$clone()`

`AutoXgboost$new()`: Creates a new instance of this R6 class.

*Usage:*

```
AutoXgboost$new(id = "xgboost")
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

`AutoXgboost$graph()`: Create the graph for the auto.

*Usage:*

```
AutoXgboost$graph(task, measure, n_threads, timeout, devices)
```

*Arguments:*

task (mlr3::Task).

measure (mlr3::Measure).

n_threads (integer(1)).

timeout (integer(1)).

devices (character())
: Devices to use. Allowed values are "cpu" and "cuda". Default is "cpu".

`AutoXgboost$finalize_model()`: Prepare the graph learner for the final model fit.

*Usage:*

```
AutoXgboost$finalize_model(graph_learner)
```

*Arguments:*

graph_learner (mlr3pipelines::GraphLearner).

`AutoXgboost$estimate_memory()`: Estimate the memory for the auto.

*Usage:*

```
AutoXgboost$estimate_memory(task)
```

*Arguments:*

task (mlr3::Task).

`AutoXgboost$internal_measure()`: Get the internal measure for the auto.

*Usage:*

```
AutoXgboost$internal_measure(measure, task)
```

*Arguments:*

measure (mlr3::Measure).

task (mlr3::Task).

`AutoXgboost$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AutoXgboost$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
auto("xgboost")
```

install_python_learners *Install Python Learners*

## Description

Creates a conda environment and installs the packages required by the Python learners. Installs the dependencies of the fastai auto learner and the tabpfn auto learner. The learners share a single environment, so one `RETICULATE_PYTHON` covers both.

## Usage

```
install_python_learners(
  learners = c("fastai", "tabpfn"),
  envname = file.path(getwd(), ".conda", "mlr3automl-python"),
  python_version = "3.12"
)
```

## Arguments

| | |
|---|---|
| `learners` | (`character()`)<br>Learners to install the Python dependencies for. One or more of `"fastai"` and `"tabpfn"`. |
| `envname` | (`character(1)`)<br>Path to the conda environment directory. Defaults to `.conda/mlr3automl-python` in the current working directory. |
| `python_version` | (`character(1)`)<br>Python version to use. Pinned to `"3.12"` by default so the environment is reproducible across machines. |

## Details

The environment is created in the project directory by default. The function downloads and installs Python packages and must therefore be called explicitly by the user. It is never called by any other function of the package.

## Value

Invisibly returns the path to the Python binary in the environment.

## Examples

```
# The example is wrapped in \dontrun{} because it cannot be executed during checks.
# It downloads and installs conda, a Python interpreter, and several Python packages,
# which requires network access, several GB of disk space, and a few minutes of runtime.
## Not run:
python = install_python_learners()
```

```
Sys.setenv(RETICULATE_PYTHON = python)

## End(Not run)
```

---

`LearnerClassifAuto` *Classification AutoML Learner*

---

## Description

The LearnerClassifAuto is an automated machine learning (AutoML) system for classification tasks. It combines preprocessing, a switch between multiple learners, and hyperparameter tuning to find the best model for the given task.

## Value

Object of class R6::R6Class and `LearnerClassifAuto`.

## Debugging

Set `options(bbotk.debug = TRUE)` to run the tuning in the main session. Set `encapsulate_learner = FALSE` to remove encapsulation of the learner. Set `encapsulate_mbo = FALSE` to catch no errors in mbo.

## Parameters

**learner_timeout** (`integer(1)`)
: Timeout for training and predicting with a single learner.

**n_threads** (`integer(1)`)
: Number of threads used for training a single learner.

**memory_limit** (`integer(1)`)
: Memory limit for training a single learner in MB. The limit is shared across the parallel workers, i.e. divided by the number of workers.

**devices** (`character()`)
: Devices to use for model training. Possible values are `"cpu"` and `"cuda"`. If `"cuda"`, the learner will be trained on a GPU.

**large_data_size** (`integer(1)`)
: Threshold for the data set size (number of rows times number of columns) above which large-data rules apply. Beyond this threshold the number of parallel workers is reduced and each remaining worker is given proportionally more threads and memory.

**small_data_size** (`integer(1)`)
: Threshold value for the data set size from which special rules apply.

**small_data_resampling** (mlr3::Resampling)
: Resampling strategy to use for model training on small data sets.

**initial_design_default** (`logical(1)`)
: Whether to use the default design of the learner.

**initial_design_set** (`integer(1)`)
Number of points to use for the initial design set.

**initial_design_size** (`integer(1)`)
Size of the random, sobol or lhs initial design.

**initial_design_type** (`character(1)`)
Type of the initial design used for mbo. Possible values are `"lhs"`, `"sobol"`, `"random"`. `"lhs"` uses a Latin Hypercube Sampling design. `"sobol"` uses a Sobol sequence design. `"random"` uses a random design.

**initial_design_fraction** (`numeric(1)`)
Fraction of the budget to use for the initial design.

**resampling** (mlr3::Resampling)
Resampling strategy used for tuning.

**terminator** (bbotk::Terminator)
Terminator criterion for tuning.

**measure** (mlr3::Measure)
Measure used for tuning.

**callbacks** (mlr3tuning::CallbackAsyncTuning)
Callbacks used for tuning.

**store_benchmark_result** (`logical(1)`)
Whether to store the benchmark result.

**store_models** (`logical(1)`)
Whether to store the models.

**encapsulate_learner** (`logical(1)`)
Whether to encapsulate the learner. Change to `FALSE` to debug.

**encapsulate_mbo** (`logical(1)`)
Whether to encapsulate the tuning. Change to `FALSE` to debug.

**check_learners** (`logical(1)`)
Whether to check if the learners are compatible with the task. Change to `FALSE` to debug.

## Python learners

Python learners like `TabPFN` and `fastai` run via `reticulate` and therefore need a Python installation with their required packages. There are two ways to provide it:

1. Do nothing and let `reticulate::py_require()` install the required packages into an ephemeral virtual environment automatically.
2. Point the `RETICULATE_PYTHON` environment variable to a Python installation that has the required packages installed.

We recommend option 2 when running on many workers, as it avoids the overhead of downloading and installing the packages on each worker. Use `install_python_learners()` to create a conda environment with the required packages and set `RETICULATE_PYTHON` to the returned Python binary.

The `TabPFN` learner additionally requires the `TABPFN_TOKEN` environment variable to download the model weights.

## Super classes

mlr3::Learner -> LearnerAuto -> LearnerClassifAuto

## Methods

### Public methods:

- LearnerClassifAuto$new()
- LearnerClassifAuto$clone()

LearnerClassifAuto$new(): Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAuto$new(id = "classif.auto", learner_ids, rush = NULL)
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

learner_ids (character())
: Learner that should be used.

rush rush::Rush
: Rush instance.

LearnerClassifAuto$clone(): The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAuto$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
packages = c(
  "mlr3extralearners", "catboost", "ranger", "callr", "mlr3torch",
  "glmnet", "kknn", "MASS", "lightgbm", "e1071", "xgboost"
)
if (mlr3misc::require_namespaces(packages, quietly = TRUE)) {
  learner = lrn("classif.auto")
  learner
}
```

`LearnerClassifAutoCatboost` *Classification Gradient Boosted Decision Trees Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoCatboost`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoCatboost`

## Methods

### Public methods:

- `LearnerClassifAutoCatboost$new()`
- `LearnerClassifAutoCatboost$clone()`

`LearnerClassifAutoCatboost$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoCatboost$new(id = "classif.auto_catboost", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoCatboost$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoCatboost$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces(c("mlr3extralearners", "catboost"), quietly = TRUE)) {
  learner = lrn("classif.auto_catboost")
  learner
}
```

`LearnerClassifAutoFastai` *Classification Fastai Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoFastai`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoFastai`

## Methods

### Public methods:

- `LearnerClassifAutoFastai$new()`
- `LearnerClassifAutoFastai$clone()`

`LearnerClassifAutoFastai$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoFastai$new(id = "classif.auto_fastai", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoFastai$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoFastai$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces(c("mlr3extralearners", "callr"), quietly = TRUE)) {
  learner = lrn("classif.auto_fastai")
  learner
}
```

`LearnerClassifAutoFTTransformer` *Classification FT-Transformer Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoFTTransformer`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoFTTransformer`

## Methods

### Public methods:

- `LearnerClassifAutoFTTransformer$new()`
- `LearnerClassifAutoFTTransformer$clone()`

`LearnerClassifAutoFTTransformer$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoFTTransformer$new(
  id = "classif.auto_ft_transformer",
  rush = NULL
)
```

*Arguments:*

`id` (`character(1)`)
: Identifier for the new instance.

`rush` rush::Rush
: Rush instance.

`LearnerClassifAutoFTTransformer$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoFTTransformer$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("mlr3torch", quietly = TRUE)) {
  learner = lrn("classif.auto_ft_transformer")
  learner
}
```

`LearnerClassifAutoGlmnet` *Classification GLM with Elastic Net Regularization Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoGlmnet`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoGlmnet`

## Methods

### Public methods:

- `LearnerClassifAutoGlmnet$new()`
- `LearnerClassifAutoGlmnet$clone()`

`LearnerClassifAutoGlmnet$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoGlmnet$new(id = "classif.auto_glmnet", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoGlmnet$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoGlmnet$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("glmnet", quietly = TRUE)) {
  learner = lrn("classif.auto_glmnet")
  learner
}
```

`LearnerClassifAutoKKNN` *Classification k-Nearest-Neighbor Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoKKNN`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoKKNN`

## Methods

### Public methods:

- `LearnerClassifAutoKKNN$new()`
- `LearnerClassifAutoKKNN$clone()`

`LearnerClassifAutoKKNN$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoKKNN$new(id = "classif.auto_kknn", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoKKNN$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoKKNN$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("kknn", quietly = TRUE)) {
  learner = lrn("classif.auto_kknn")
  learner
}
```

LearnerClassifAutoLightGBM

*Classification LightGBM Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and LearnerClassifAutoLightGBM.

## Super classes

mlr3::Learner -> LearnerAuto -> LearnerClassifAuto -> LearnerClassifAutoLightGBM

## Methods

### Public methods:

- LearnerClassifAutoLightGBM$new()
- LearnerClassifAutoLightGBM$clone()

LearnerClassifAutoLightGBM$new(): Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoLightGBM$new(id = "classif.auto_lightgbm", rush = NULL)
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

rush rush::Rush
: Rush instance.

LearnerClassifAutoLightGBM$clone(): The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoLightGBM$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces(c("mlr3extralearners", "lightgbm"), quietly = TRUE)) {
  learner = lrn("classif.auto_lightgbm")
  learner
}
```

`LearnerClassifAutoMLP` *Classification MLP Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoMLP`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoMLP`

## Methods

### Public methods:

- `LearnerClassifAutoMLP$new()`
- `LearnerClassifAutoMLP$clone()`

`LearnerClassifAutoMLP$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoMLP$new(id = "classif.auto_mlp", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoMLP$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoMLP$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("mlr3torch", quietly = TRUE)) {
  learner = lrn("classif.auto_mlp")
  learner
}
```

LearnerClassifAutoRanger

*Classification Ranger Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and LearnerClassifAutoRanger.

## Super classes

mlr3::Learner -> LearnerAuto -> LearnerClassifAuto -> LearnerClassifAutoRanger

## Methods

### Public methods:

- LearnerClassifAutoRanger$new()
- LearnerClassifAutoRanger$clone()

LearnerClassifAutoRanger$new(): Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoRanger$new(id = "classif.auto_ranger", rush = NULL)
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

rush rush::Rush
: Rush instance.

LearnerClassifAutoRanger$clone(): The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoRanger$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("ranger", quietly = TRUE)) {
  learner = lrn("classif.auto_ranger")
  learner
}
```

`LearnerClassifAutoResNet` *Classification ResNet Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoResNet`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoResNet`

## Methods

### Public methods:

- `LearnerClassifAutoResNet$new()`
- `LearnerClassifAutoResNet$clone()`

`LearnerClassifAutoResNet$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoResNet$new(id = "classif.auto_resnet", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoResNet$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoResNet$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("mlr3torch", quietly = TRUE)) {
  learner = lrn("classif.auto_resnet")
  learner
}
```

`LearnerClassifAutoSVM` *Classification SVM Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoSVM`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoSVM`

## Methods

### Public methods:

- `LearnerClassifAutoSVM$new()`
- `LearnerClassifAutoSVM$clone()`

`LearnerClassifAutoSVM$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoSVM$new(id = "classif.auto_svm", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoSVM$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoSVM$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("e1071", quietly = TRUE)) {
  learner = lrn("classif.auto_svm")
  learner
}
```

`LearnerClassifAutoTabPFN` *Classification TabPFN Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoTabPFN`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoTabPFN`

## Methods

### Public methods:

- `LearnerClassifAutoTabPFN$new()`
- `LearnerClassifAutoTabPFN$clone()`

`LearnerClassifAutoTabPFN$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoTabPFN$new(id = "classif.auto_tabpfn", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoTabPFN$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoTabPFN$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces(c("mlr3extralearners", "callr"), quietly = TRUE)) {
  learner = lrn("classif.auto_tabpfn")
  learner
}
```

`LearnerClassifAutoXgboost` *Classification XGBoost Auto Learner*

## Description

Classification auto learner.

## Value

Object of class R6::R6Class and `LearnerClassifAutoXgboost`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerClassifAuto` -> `LearnerClassifAutoXgboost`

## Methods

### Public methods:

- `LearnerClassifAutoXgboost$new()`
- `LearnerClassifAutoXgboost$clone()`

`LearnerClassifAutoXgboost$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifAutoXgboost$new(id = "classif.auto_xgboost", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerClassifAutoXgboost$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifAutoXgboost$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("xgboost", quietly = TRUE)) {
  learner = lrn("classif.auto_xgboost")
  learner
}
```

`LearnerClassifFastaiIsolated` *Fastai Learner Isolated*

## Description

A subclass of mlr3extralearners::LearnerClassifFastai that isolates the Python environment in a callr session.

## Value

Object of class R6::R6Class and `LearnerClassifFastaiIsolated`.

## Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `mlr3extralearners::LearnerClassifFastai` -> `LearnerClassifFastaiIsolated`

## Methods

### Public methods:

- `LearnerClassifFastaiIsolated$new()`
- `LearnerClassifFastaiIsolated$clone()`

`LearnerClassifFastaiIsolated$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifFastaiIsolated$new()
```

`LearnerClassifFastaiIsolated$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifFastaiIsolated$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

`LearnerClassifTabPFNIsolated` *TabPFN Learner Isolated*

## Description

A subclass of mlr3extralearners::LearnerClassifTabPFN that isolates the Python environment in a callr session.

## Value

Object of class R6::R6Class and `LearnerClassifTabPFNIsolated`.

## Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `mlr3extralearners::LearnerClassifTabPFN` -> `LearnerClassifTabPFNIsolated`

## Methods

### Public methods:

- `LearnerClassifTabPFNIsolated$new()`
- `LearnerClassifTabPFNIsolated$clone()`

`LearnerClassifTabPFNIsolated$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerClassifTabPFNIsolated$new()
```

`LearnerClassifTabPFNIsolated$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerClassifTabPFNIsolated$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

`LearnerRegrAuto` *Regression AutoML Learner*

## Description

The LearnerRegrAuto is an automated machine learning (AutoML) system for regression tasks. It combines preprocessing, a switch between multiple learners, and hyperparameter tuning to find the best model for the given task.

## Value

Object of class R6::R6Class and `LearnerRegrAuto`.

## Debugging

Set `options(bbotk.debug = TRUE)` to run the tuning in the main session. Set `encapsulate_learner = FALSE` to remove encapsulation of the learner. Set `encapsulate_mbo = FALSE` to catch no errors in mbo.

## Parameters

**learner_timeout** (`integer(1)`)
Timeout for training and predicting with a single learner.

**n_threads** (`integer(1)`)
Number of threads used for training a single learner.

**memory_limit** (`integer(1)`)
Memory limit for training a single learner in MB. The limit is shared across the parallel workers, i.e. divided by the number of workers.

**devices** (`character()`)
Devices to use for model training. Possible values are `"cpu"` and `"cuda"`. If `"cuda"`, the learner will be trained on a GPU.

**large_data_size** (`integer(1)`)
Threshold for the data set size (number of rows times number of columns) above which large-data rules apply. Beyond this threshold the number of parallel workers is reduced and each remaining worker is given proportionally more threads and memory.

**small_data_size** (`integer(1)`)
Threshold value for the data set size from which special rules apply.

**small_data_resampling** (mlr3::Resampling)
Resampling strategy to use for model training on small data sets.

**initial_design_default** (`logical(1)`)
Whether to use the default design of the learner.

**initial_design_set** (`integer(1)`)
Number of points to use for the initial design set.

**initial_design_size** (`integer(1)`)
Size of the random, sobol or lhs initial design.

**initial_design_type** (`character(1)`)
Type of the initial design used for mbo. Possible values are `"lhs"`, `"sobol"`, `"random"`. `"lhs"` uses a Latin Hypercube Sampling design. `"sobol"` uses a Sobol sequence design. `"random"` uses a random design.

**initial_design_fraction** (`numeric(1)`)
Fraction of the budget to use for the initial design.

**resampling** (mlr3::Resampling)
Resampling strategy used for tuning.

**terminator** (bbotk::Terminator)
Terminator criterion for tuning.

**measure** (mlr3::Measure)
Measure used for tuning.

**callbacks** (mlr3tuning::CallbackAsyncTuning)
Callbacks used for tuning.

**store_benchmark_result** (`logical(1)`)
Whether to store the benchmark result.

**store_models** (`logical(1)`)
Whether to store the models.

**encapsulate_learner** (`logical(1)`)
Whether to encapsulate the learner. Change to `FALSE` to debug.

**encapsulate_mbo** (`logical(1)`)
Whether to encapsulate the tuning. Change to `FALSE` to debug.

**check_learners** (`logical(1)`)
Whether to check if the learners are compatible with the task. Change to `FALSE` to debug.

## Python learners

Python learners like `TabPFN` and `fastai` run via `reticulate` and therefore need a Python installation with their required packages. There are two ways to provide it:

1. Do nothing and let `reticulate::py_require()` install the required packages into an ephemeral virtual environment automatically.
2. Point the `RETICULATE_PYTHON` environment variable to a Python installation that has the required packages installed.

We recommend option 2 when running on many workers, as it avoids the overhead of downloading and installing the packages on each worker. Use `install_python_learners()` to create a conda environment with the required packages and set `RETICULATE_PYTHON` to the returned Python binary.

The `TabPFN` learner additionally requires the `TABPFN_TOKEN` environment variable to download the model weights.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerRegrAuto`

## Methods

### Public methods:

- `LearnerRegrAuto$new()`
- `LearnerRegrAuto$clone()`

`LearnerRegrAuto$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAuto$new(id = "regr.auto", learner_ids, rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`learner_ids` (`character()`)
Learner that should be used.

`rush` rush::Rush
Rush instance.

`LearnerRegrAuto$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAuto$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
packages = c(
  "mlr3extralearners", "catboost", "ranger", "callr", "mlr3torch",
  "glmnet", "kknn", "MASS", "lightgbm", "e1071", "xgboost"
)
if (mlr3misc::require_namespaces(packages, quietly = TRUE)) {
  learner = lrn("regr.auto")
  learner
}
```

---

# LearnerRegrAutoCatboost

*Regression Gradient Boosted Decision Trees Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoCatboost`.

### Super classes

mlr3::Learner -> LearnerAuto -> LearnerRegrAuto -> LearnerRegrAutoCatboost

### Methods

#### Public methods:

- LearnerRegrAutoCatboost$new()
- LearnerRegrAutoCatboost$clone()

LearnerRegrAutoCatboost$new(): Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoCatboost$new(id = "regr.auto_catboost", rush = NULL)
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

rush rush::Rush
: Rush instance.

LearnerRegrAutoCatboost$clone(): The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoCatboost$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

### Examples

```
if (mlr3misc::require_namespaces(c("mlr3extralearners", "catboost"), quietly = TRUE)) {
  learner = lrn("regr.auto_catboost")
  learner
}
```

---

## LearnerRegrAutoFTTransformer

*Regression FT-Transformer Auto Learner*

---

### Description

Regression auto learner.

### Value

Object of class R6::R6Class and LearnerRegrAutoFTTransformer.

### Super classes

mlr3::Learner -> LearnerAuto -> LearnerRegrAuto -> LearnerRegrAutoFTTransformer

## Methods

### Public methods:

- `LearnerRegrAutoFTTransformer$new()`
- `LearnerRegrAutoFTTransformer$clone()`

`LearnerRegrAutoFTTransformer$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoFTTransformer$new(id = "regr.auto_ft_transformer", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoFTTransformer$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoFTTransformer$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("mlr3torch", quietly = TRUE)) {
  learner = lrn("regr.auto_ft_transformer")
  learner
}
```

---

# LearnerRegrAutoGlmnet *Regression GLM with Elastic Net Regularization Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoGlmnet`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerRegrAuto` -> `LearnerRegrAutoGlmnet`

## Methods

### Public methods:

- LearnerRegrAutoGlmnet$new()
- LearnerRegrAutoGlmnet$clone()

`LearnerRegrAutoGlmnet$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoGlmnet$new(id = "regr.auto_glmnet", rush = NULL)
```

*Arguments:*

`id (character(1))`
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoGlmnet$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoGlmnet$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("glmnet", quietly = TRUE)) {
  learner = lrn("regr.auto_glmnet")
  learner
}
```

---

`LearnerRegrAutoKKNN` *Regression k-Nearest-Neighbor Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoKKNN`.

## Super classes

mlr3::Learner -> LearnerAuto -> LearnerRegrAuto -> LearnerRegrAutoKKNN

## Methods

### Public methods:

- `LearnerRegrAutoKKNN$new()`
- `LearnerRegrAutoKKNN$clone()`

`LearnerRegrAutoKKNN$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoKKNN$new(id = "regr.auto_kknn", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoKKNN$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoKKNN$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("kknn", quietly = TRUE)) {
  learner = lrn("regr.auto_kknn")
  learner
}
```

---

# LearnerRegrAutoLightGBM

*Regression LightGBM Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoLightGBM`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerRegrAuto` -> `LearnerRegrAutoLightGBM`

## Methods

### Public methods:

- `LearnerRegrAutoLightGBM$new()`
- `LearnerRegrAutoLightGBM$clone()`

`LearnerRegrAutoLightGBM$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoLightGBM$new(id = "regr.auto_lightgbm", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoLightGBM$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoLightGBM$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces(c("mlr3extralearners", "lightgbm"), quietly = TRUE)) {
  learner = lrn("regr.auto_lightgbm")
  learner
}
```

---

`LearnerRegrAutoMLP` *Regression MLP Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoMLP`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerRegrAuto` -> `LearnerRegrAutoMLP`

## Methods

### Public methods:

- LearnerRegrAutoMLP$new()
- LearnerRegrAutoMLP$clone()

LearnerRegrAutoMLP$new(): Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoMLP$new(id = "regr.auto_mlp", rush = NULL)
```

*Arguments:*

id (character(1))
: Identifier for the new instance.

rush rush::Rush
: Rush instance.

LearnerRegrAutoMLP$clone(): The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoMLP$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("mlr3torch", quietly = TRUE)) {
  learner = lrn("regr.auto_mlp")
  learner
}
```

---

# LearnerRegrAutoRanger *Regression Ranger Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and LearnerRegrAutoRanger.

## Super classes

mlr3::Learner -> LearnerAuto -> LearnerRegrAuto -> LearnerRegrAutoRanger

## Methods

### Public methods:

- `LearnerRegrAutoRanger$new()`
- `LearnerRegrAutoRanger$clone()`

`LearnerRegrAutoRanger$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoRanger$new(id = "regr.auto_ranger", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoRanger$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoRanger$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("ranger", quietly = TRUE)) {
  learner = lrn("regr.auto_ranger")
  learner
}
```

---

# LearnerRegrAutoResNet *Regression ResNet Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoResNet`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerRegrAuto` -> `LearnerRegrAutoResNet`

## Methods

### Public methods:

- LearnerRegrAutoResNet$new()
- LearnerRegrAutoResNet$clone()

`LearnerRegrAutoResNet$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoResNet$new(id = "regr.auto_resnet", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoResNet$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoResNet$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("mlr3torch", quietly = TRUE)) {
  learner = lrn("regr.auto_resnet")
  learner
}
```

---

# LearnerRegrAutoSVM *Regression SVM Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoSVM`.

## Super classes

mlr3::Learner -> LearnerAuto -> LearnerRegrAuto -> LearnerRegrAutoSVM

## Methods

### Public methods:

- LearnerRegrAutoSVM$new()
- LearnerRegrAutoSVM$clone()

`LearnerRegrAutoSVM$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoSVM$new(id = "regr.auto_svm", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoSVM$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoSVM$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("e1071", quietly = TRUE)) {
  learner = lrn("regr.auto_svm")
  learner
}
```

---

`LearnerRegrAutoTabPFN` *Regression TabPFN Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoTabPFN`.

## Super classes

`mlr3::Learner` -> `LearnerAuto` -> `LearnerRegrAuto` -> `LearnerRegrAutoTabPFN`

## Methods

### Public methods:

- LearnerRegrAutoTabPFN$new()
- LearnerRegrAutoTabPFN$clone()

`LearnerRegrAutoTabPFN$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoTabPFN$new(id = "regr.auto_tabpfn", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoTabPFN$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoTabPFN$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces(c("mlr3extralearners", "callr"), quietly = TRUE)) {
  learner = lrn("regr.auto_tabpfn")
  learner
}
```

---

# LearnerRegrAutoXgboost

*Regression XGBoost Auto Learner*

---

## Description

Regression auto learner.

## Value

Object of class R6::R6Class and `LearnerRegrAutoXgboost`.

## Super classes

mlr3::Learner -> LearnerAuto -> LearnerRegrAuto -> LearnerRegrAutoXgboost

## Methods

### Public methods:

- `LearnerRegrAutoXgboost$new()`
- `LearnerRegrAutoXgboost$clone()`

`LearnerRegrAutoXgboost$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrAutoXgboost$new(id = "regr.auto_xgboost", rush = NULL)
```

*Arguments:*

`id` (`character(1)`)
Identifier for the new instance.

`rush` rush::Rush
Rush instance.

`LearnerRegrAutoXgboost$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrAutoXgboost$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
if (mlr3misc::require_namespaces("xgboost", quietly = TRUE)) {
  learner = lrn("regr.auto_xgboost")
  learner
}
```

---

`LearnerRegrTabPFNIsolated`
*TabPFN Regressor Learner Isolated*

---

## Description

A subclass of mlr3extralearners::LearnerRegrTabPFN that isolates the Python environment in a callr session.

## Value

Object of class R6::R6Class and `LearnerRegrTabPFNIsolated`.

## Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `mlr3extralearners::LearnerRegrTabPFN` -> `LearnerRegrTabPFNIsolated`

## Methods

### Public methods:

- `LearnerRegrTabPFNIsolated$new()`
- `LearnerRegrTabPFNIsolated$clone()`

`LearnerRegrTabPFNIsolated$new()`: Creates a new instance of this R6 class.

*Usage:*

```
LearnerRegrTabPFNIsolated$new()
```

`LearnerRegrTabPFNIsolated$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
LearnerRegrTabPFNIsolated$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`mlr3automl.encapsulation_daemon`
*Encapsulation Daemon Callback*

---

## Description

This mlr3misc::Callback starts a persistent **mirai** daemon on each rush worker that is reused for the `"mirai"` encapsulation of the tuned learners. Reusing a single daemon avoids the overhead of starting a new daemon for every learner evaluation. The daemon is started when the worker begins and its liveness is checked before every evaluation. If the daemon has died, it is restarted.

## Parameters

- `n_daemons` :: `integer(1)`
  Number of daemons to start on each worker. Defaults to `1`.

## Examples

```
clbk("mlr3automl.encapsulation_daemon", n_daemons = 1)
```

## mlr3automl.initial_design_runtime *Initial Design Runtime Limit Callback*

### Description

This mlr3misc::Callback drops the remaining tasks of the initial design from the queue when a configurable fraction (`initial_design_fraction`) of the runtime limit is reached. The `initial_design_fraction` is set to `0.25` by default.

### Examples

```
clbk("mlr3automl.initial_design_runtime", initial_design_fraction = 0.5)
```

## mlr_auto *Dictionary of Auto Objects*

### Description

A dictionary of Auto objects.

Sugar function to retrieve Auto objects from mlr_auto.

### Usage

```
mlr_auto

auto(.key, ...)
```

### Arguments

| | |
|---|---|
| `.key` | (`character(1)`)<br>Key of the object to retrieve. If missing, the dictionary itself is returned. |
| `...` | (named `list()`)<br>Additional arguments passed to the constructor. |

### Value

Auto

### Examples

```
auto("catboost")
```

# Index