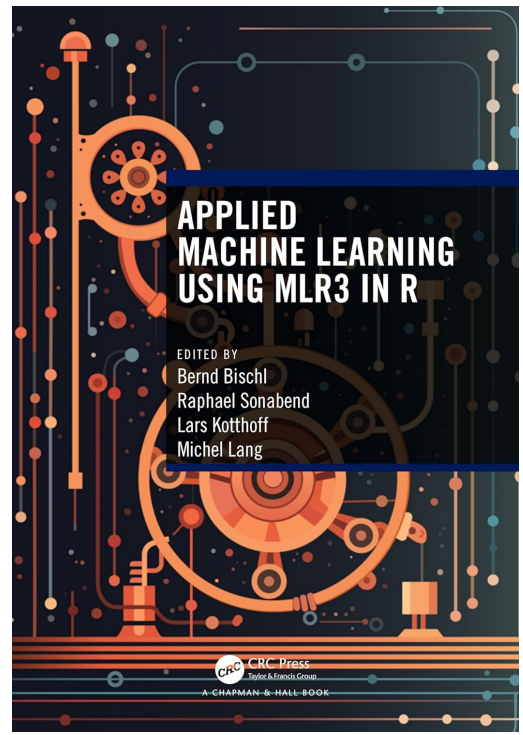
Tuning Machine Learning Algorithms with `mlr3`

- **Website:** `https://mlr-org.com/`
- **Github:** `https://github.com/mlr-org`
- **Book:** `https://mlr3book.mlr-org.com/`



April 29, 2024

Intro

Tuning

Tuning in mlr3

Parameter Transformation

AutoTuner and Nested Resampling

TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

$\Rightarrow$ We do *black box optimization* (“Try stuff and see what works”)

TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

$\Rightarrow$ We do *black box optimization* ("Try stuff and see what works")

Tuning toolbox for `mlr3`:

```
library("bbotk")
library("mlr3tuning")
```

Intro

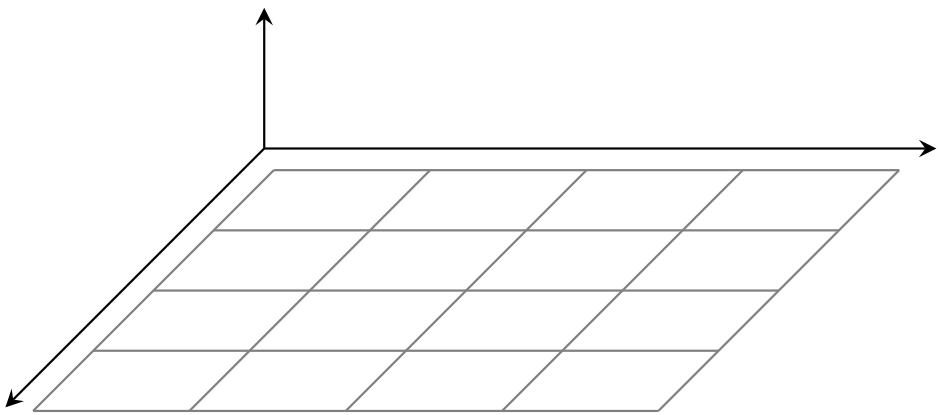
Tuning

Tuning in mlr3

Parameter Transformation

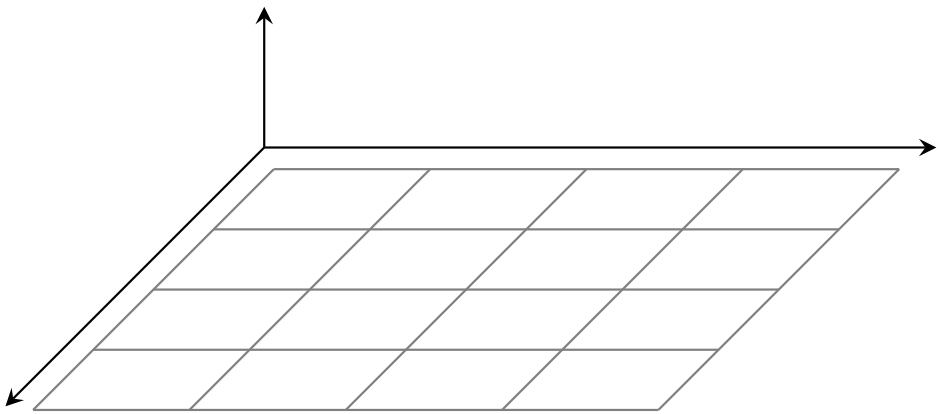
AutoTuner and Nested Resampling

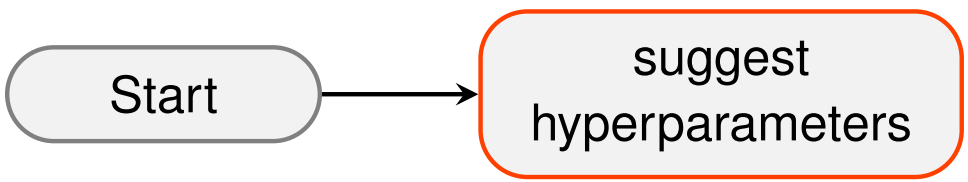
TUNING



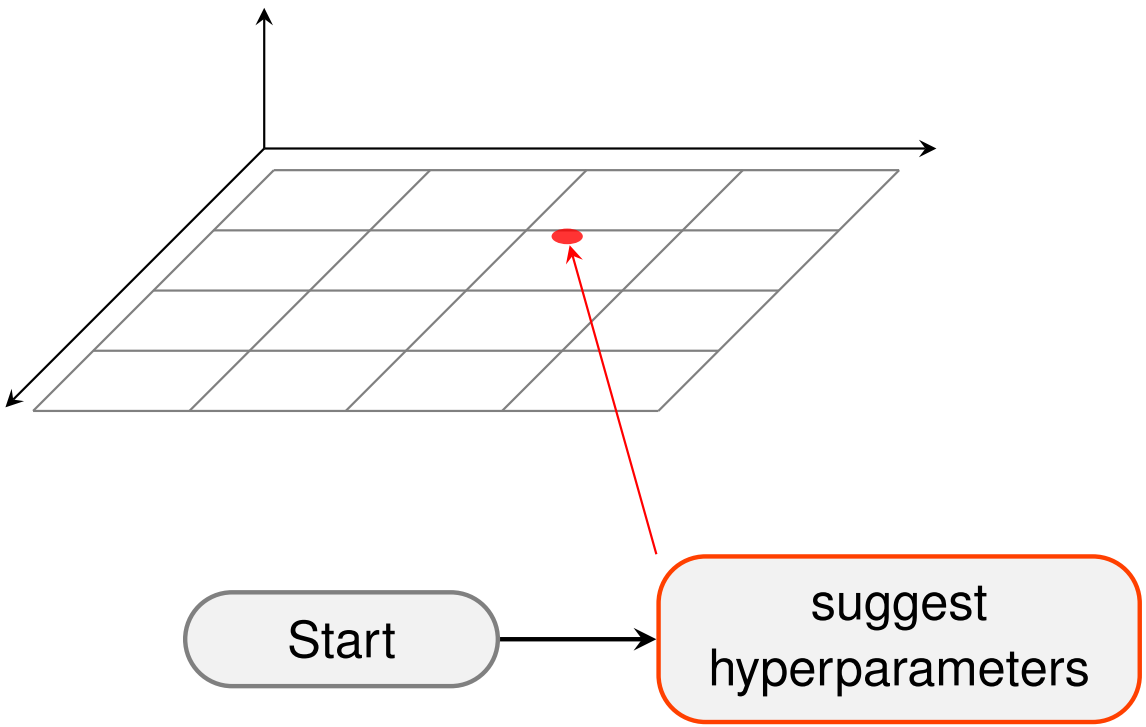
Start

TUNING

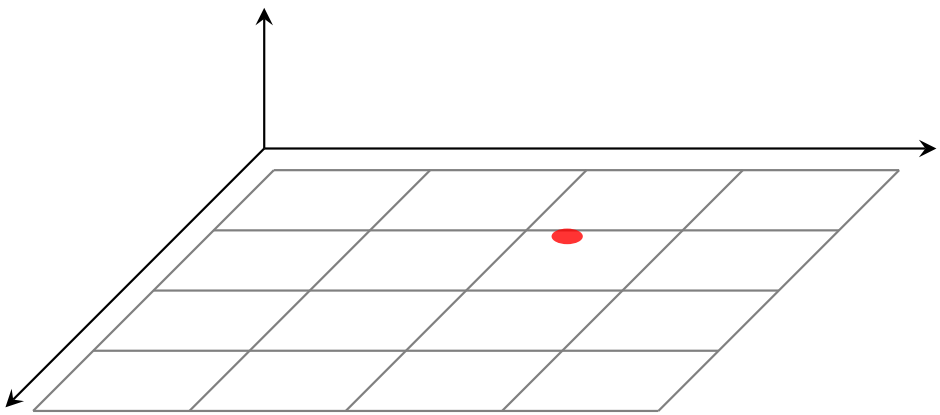


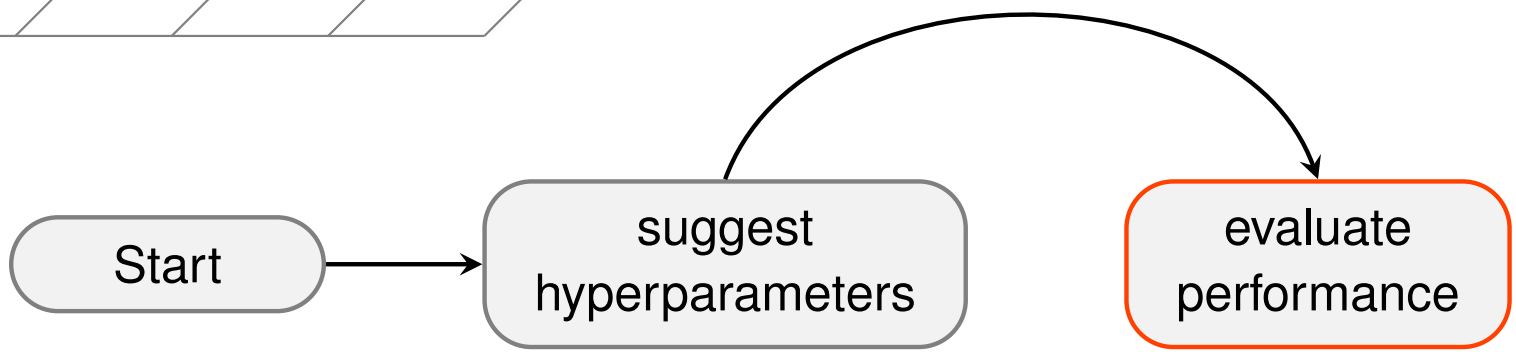


TUNING

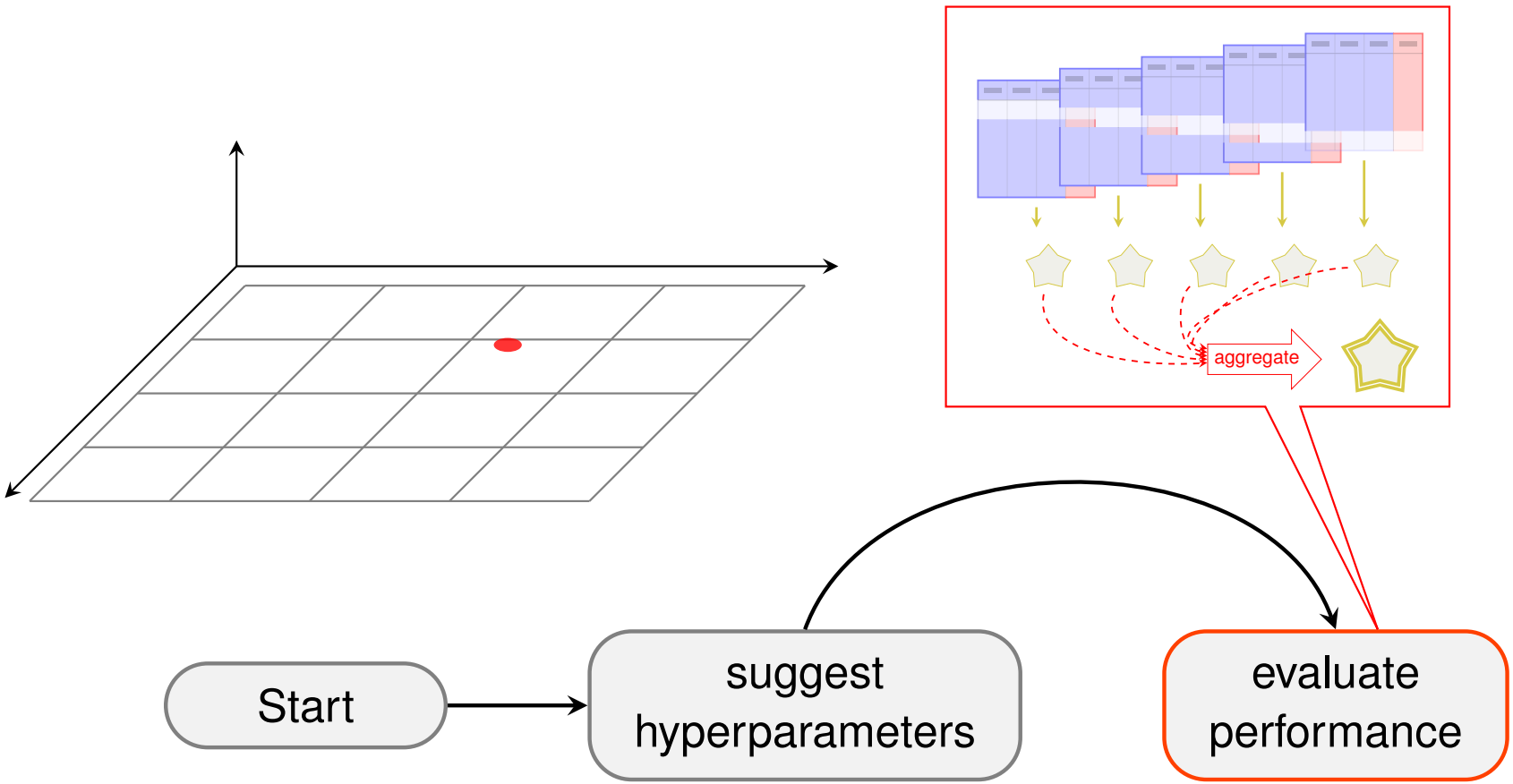


TUNING

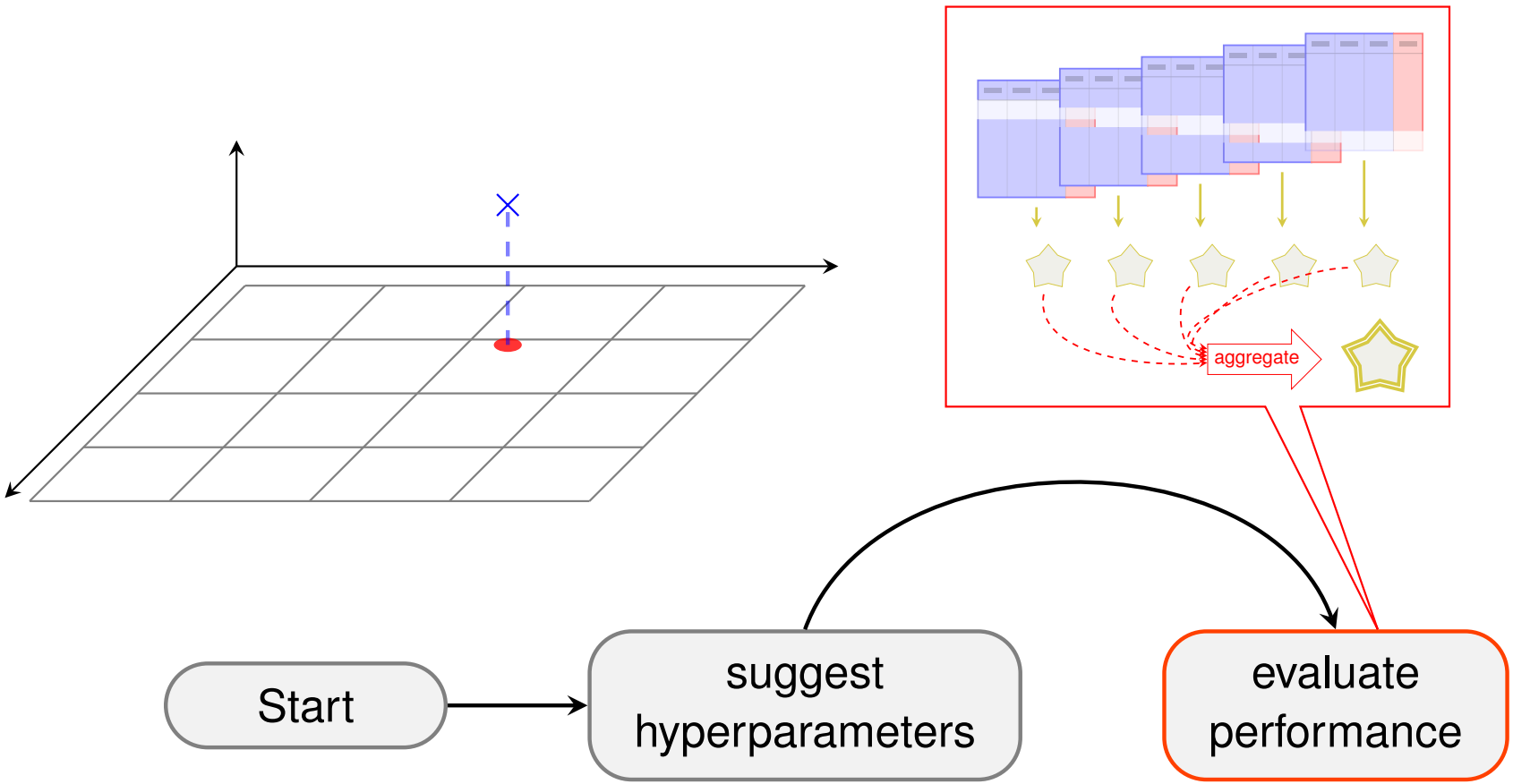




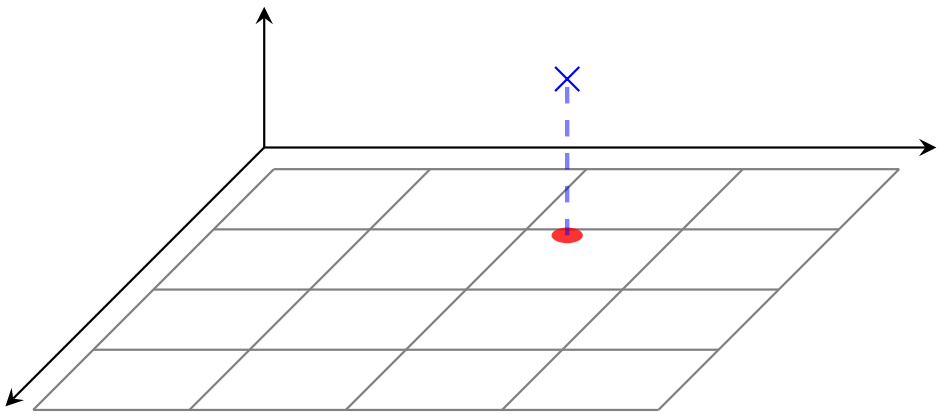
TUNING

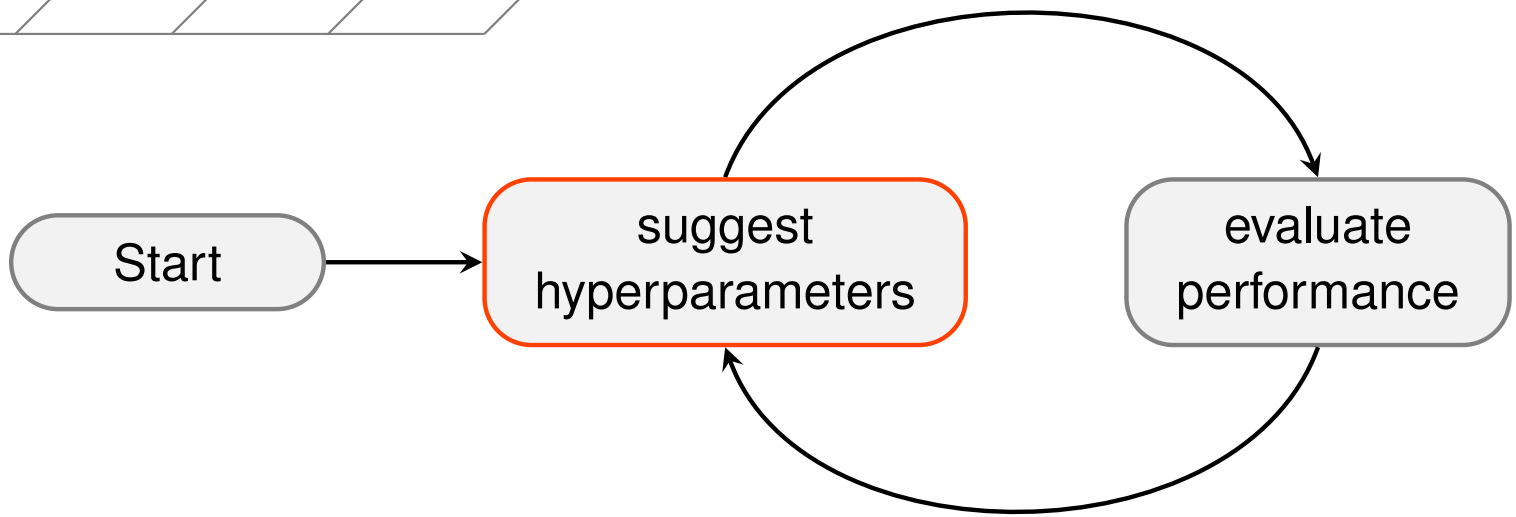

aggregate
Start
suggest hyperparameters
evaluate performance

TUNING

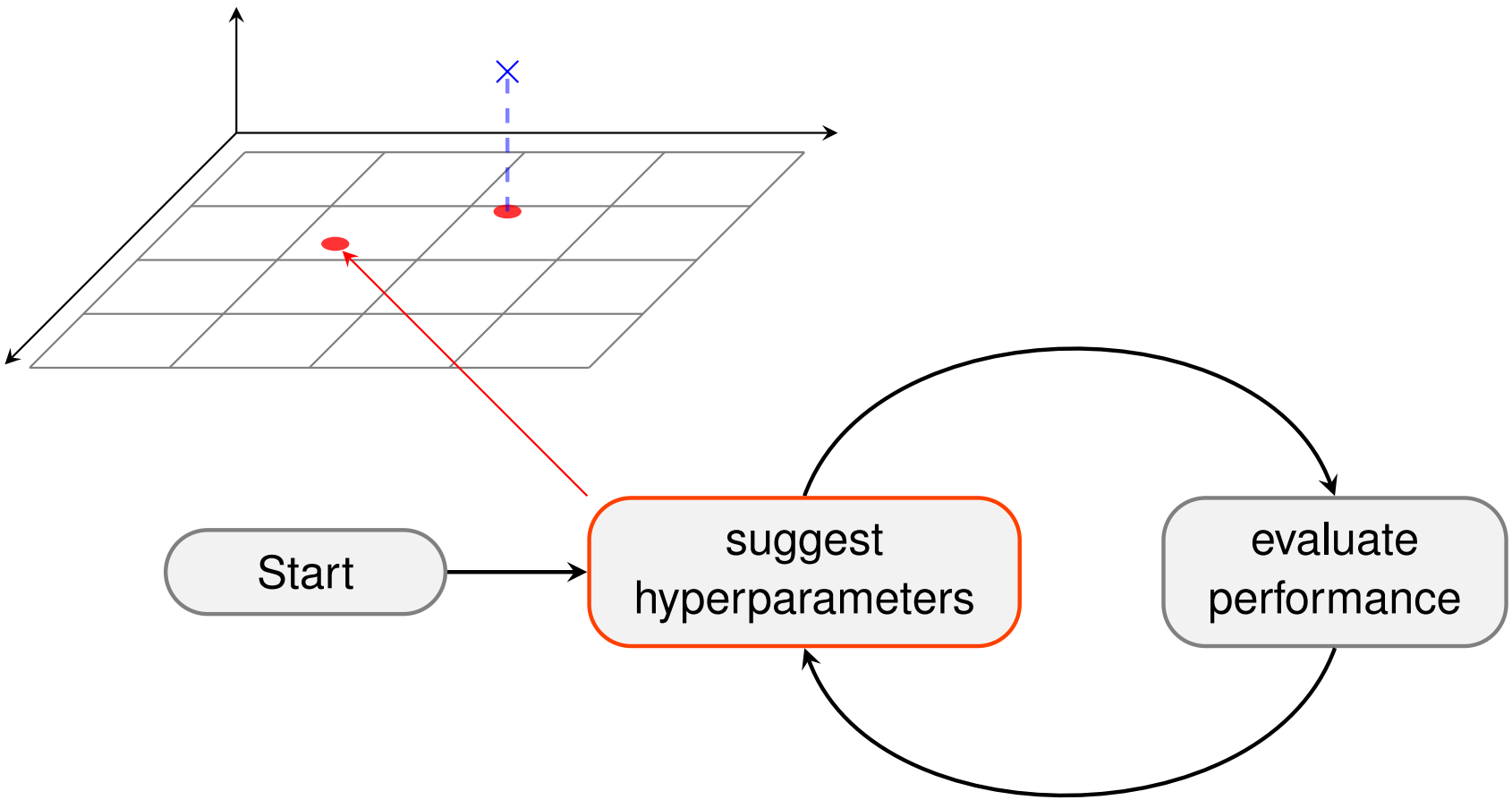

aggregate
Start
suggest hyperparameters
evaluate performance

TUNING

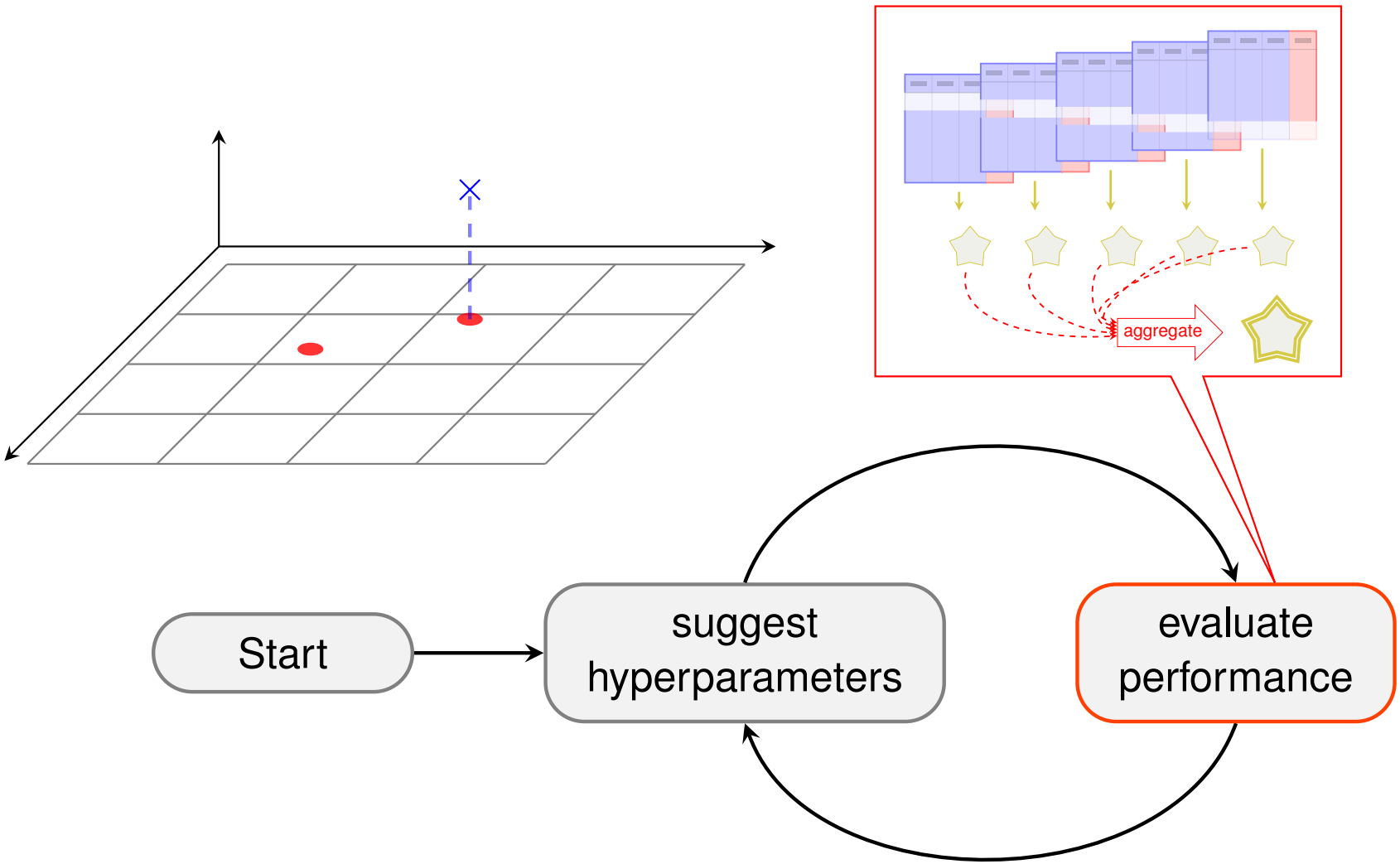




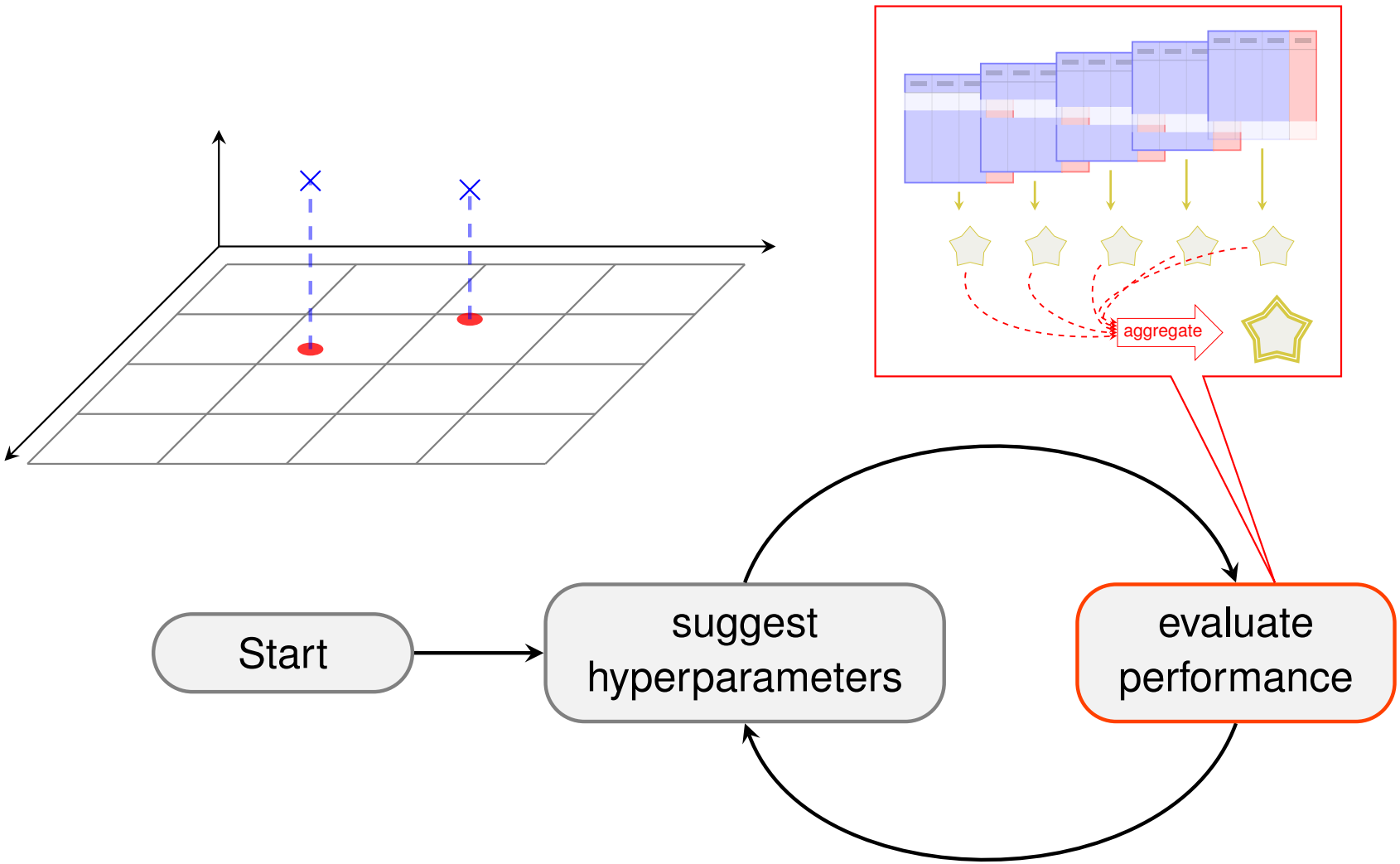
TUNING



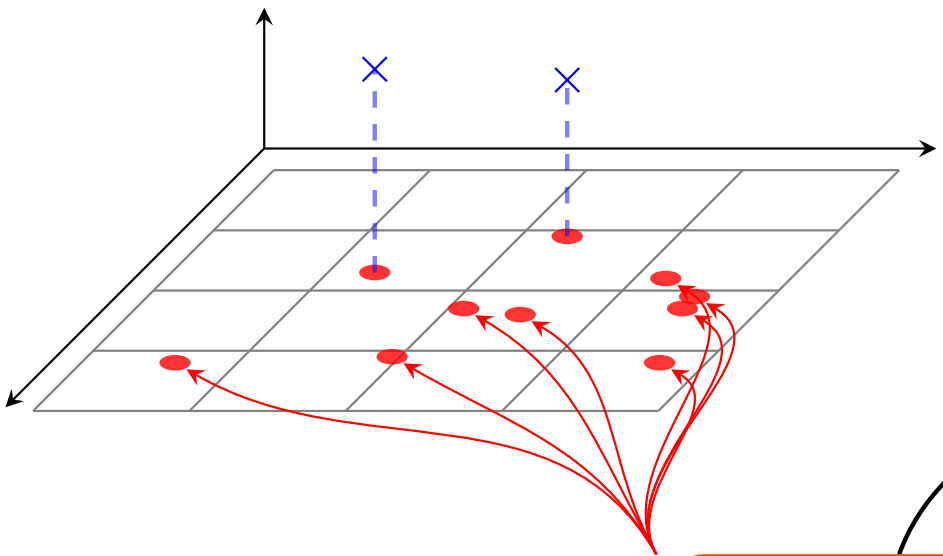
TUNING


aggregate
Start
suggest
hyperparameters
evaluate
performance

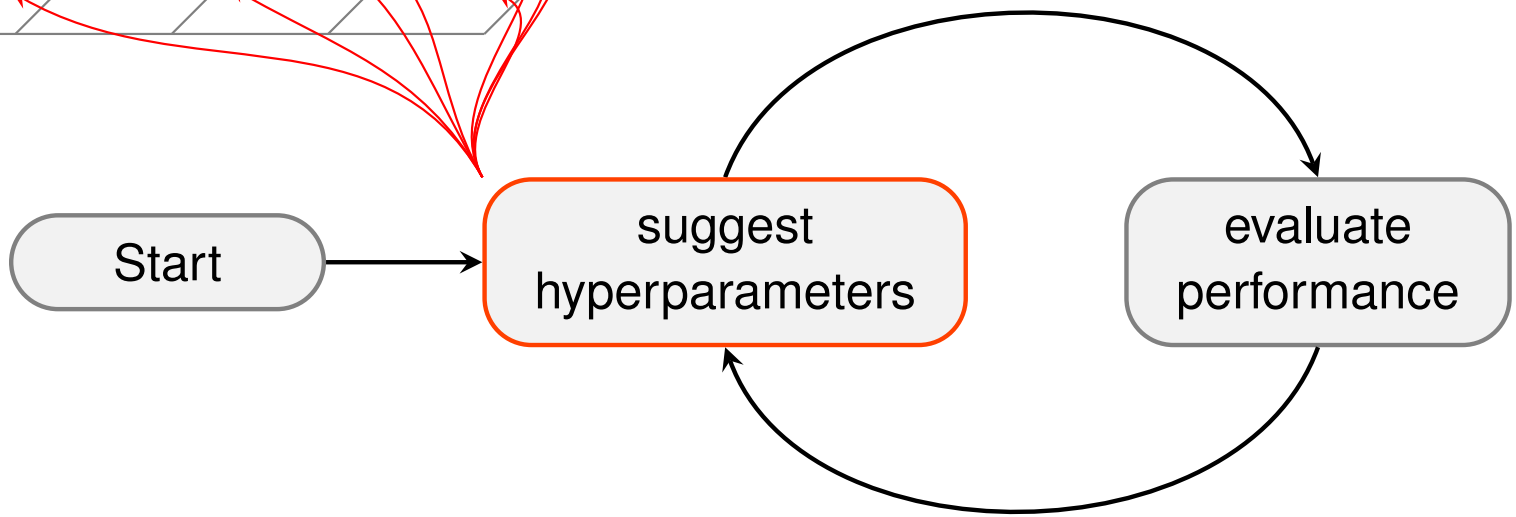
TUNING


aggregate
Start
suggest
hyperparameters
evaluate
performance

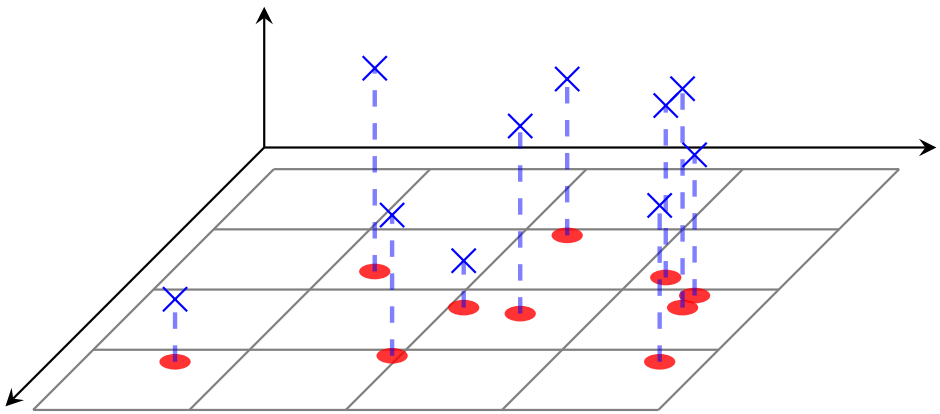
TUNING



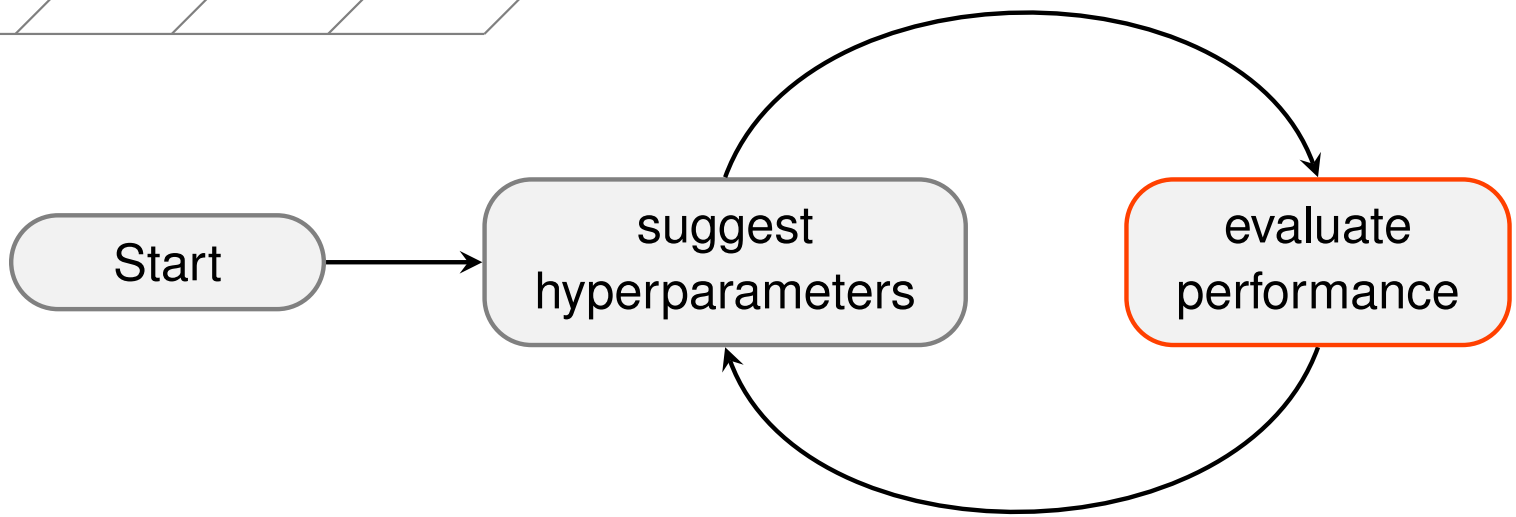
Batch evaluation



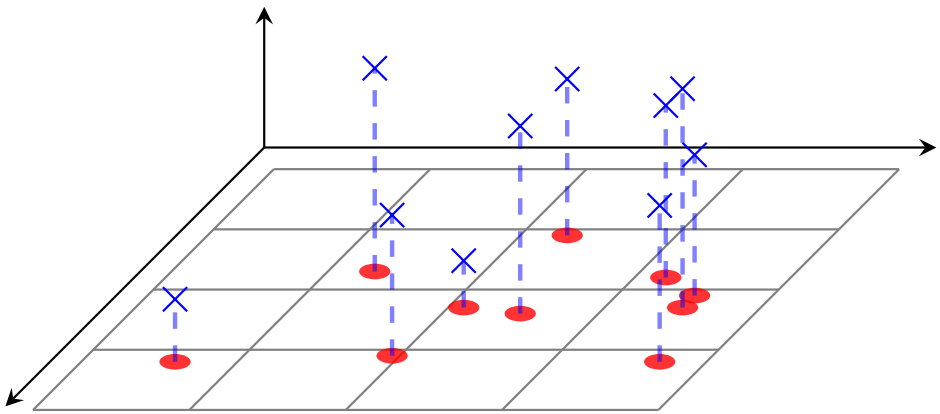
TUNING

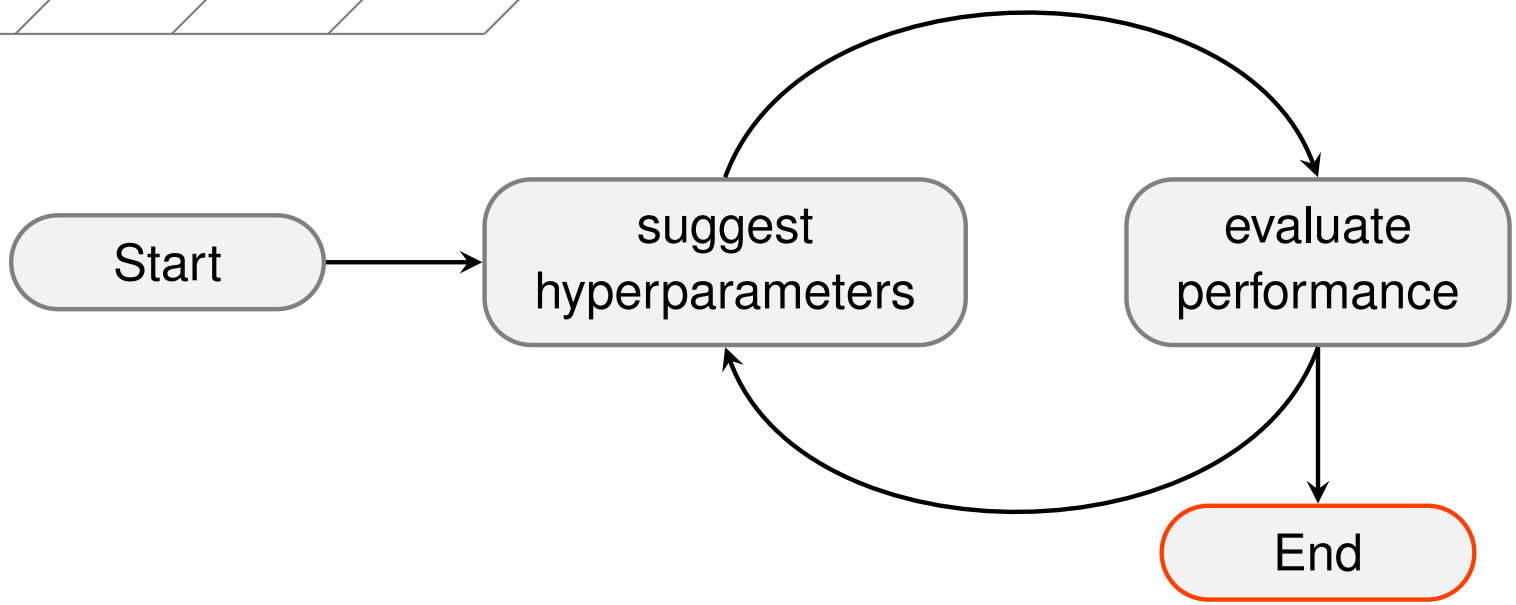


Batch evaluation

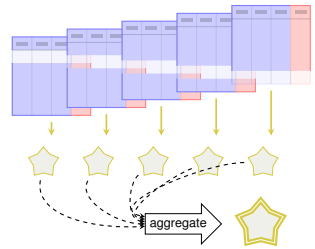


TUNING

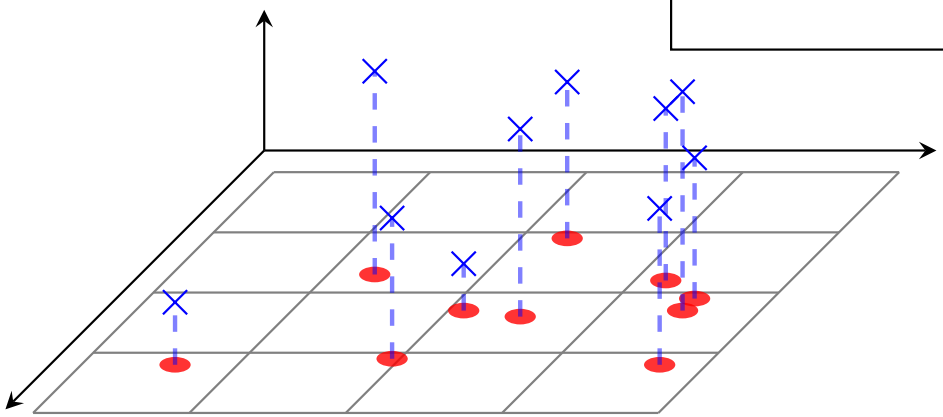


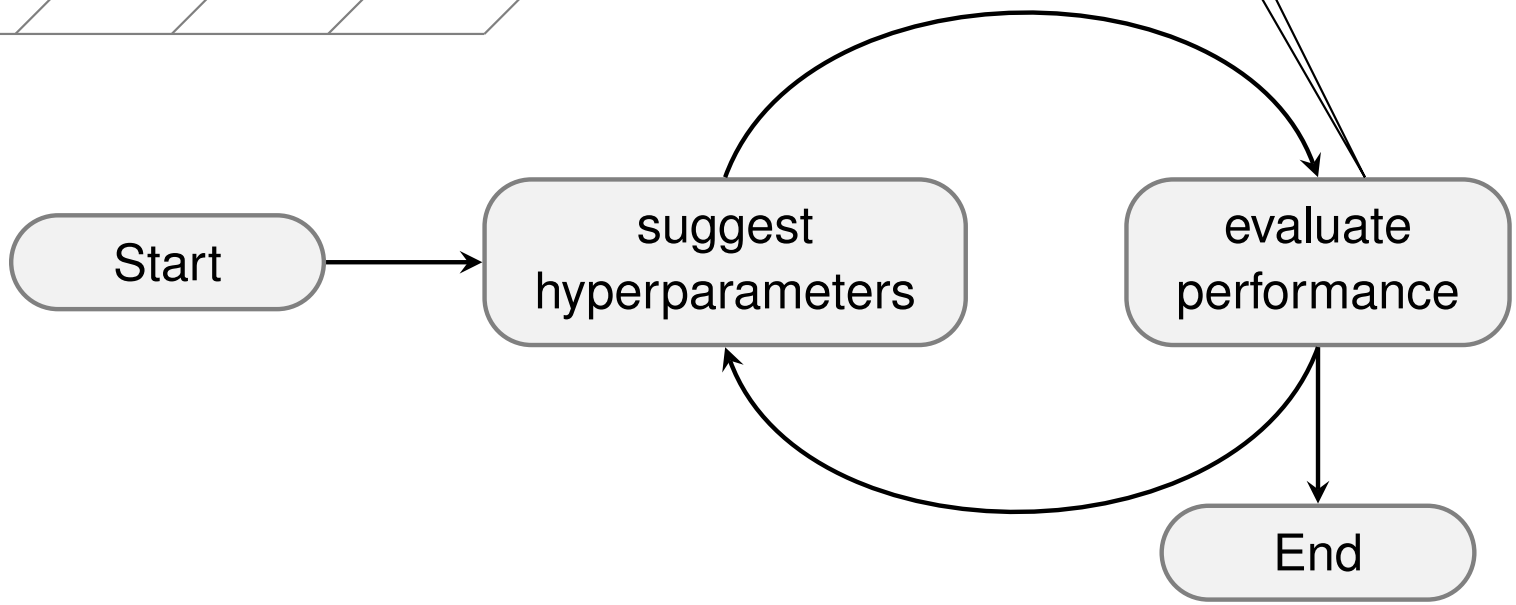


TUNING

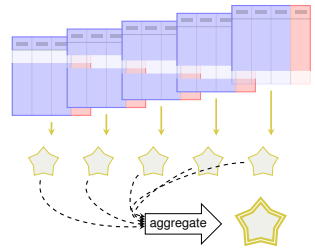


`Learner` (`lrn()`)
`Task` (`as_task_*()`)
`Resampling` (`rsmp()`)
`Measure` (`msr()`)

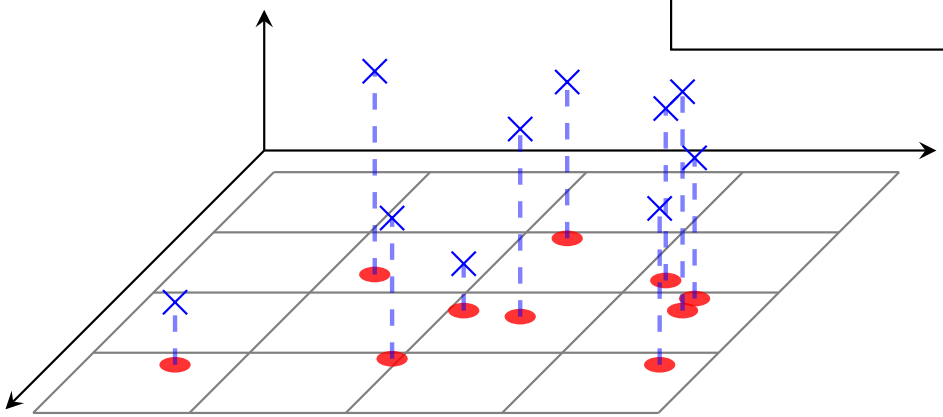




TUNING



`Learner` (`lrn()`)
`Task` (`as_task_*()`)
`Resampling` (`rsmp()`)
`Measure` (`msr()`)



Start

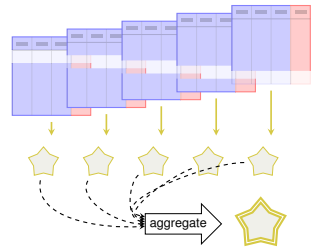
suggest hyperparameters

evaluate performance

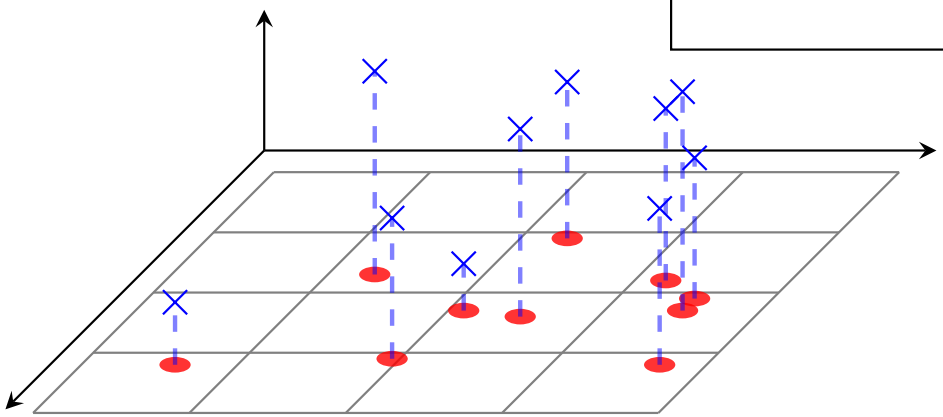
End

search space
`Tuner`

TUNING



`Learner` (`lrn()`)
`Task` (`as_task_*()`)
`Resampling` (`rsmp()`)
`Measure` (`msr()`)



Start

suggest hyperparameters

evaluate performance

search space
`Tuner`

`Terminator` (when to finish)

End

Intro

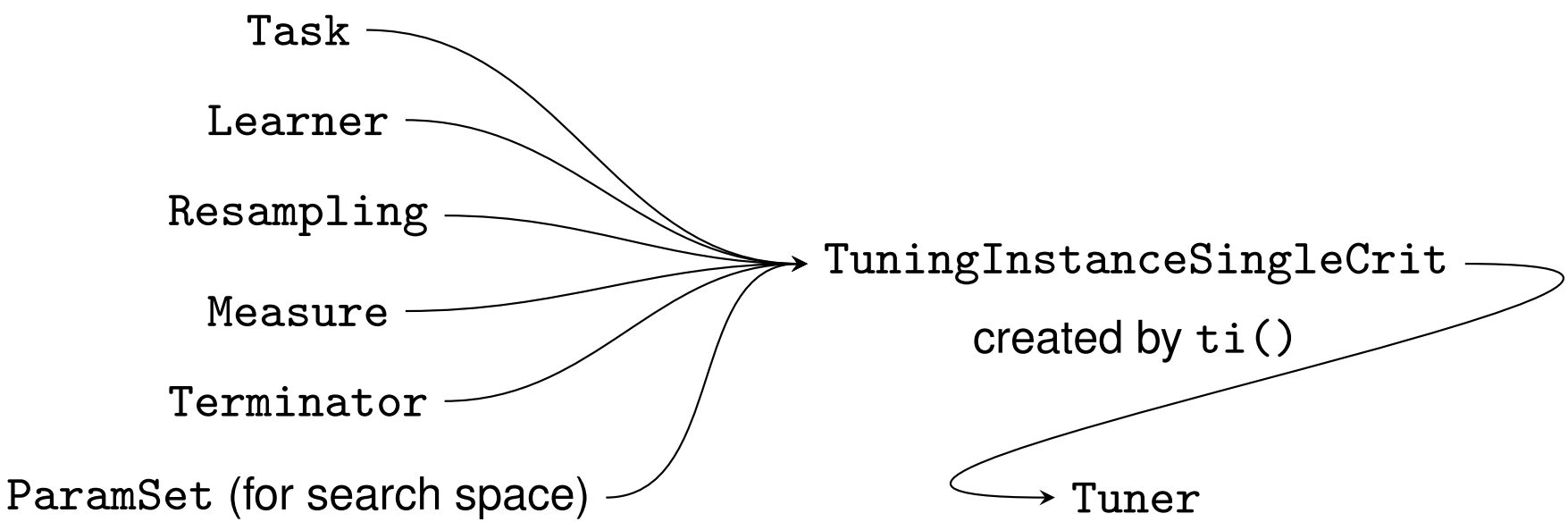
Tuning

Tuning in mlr3

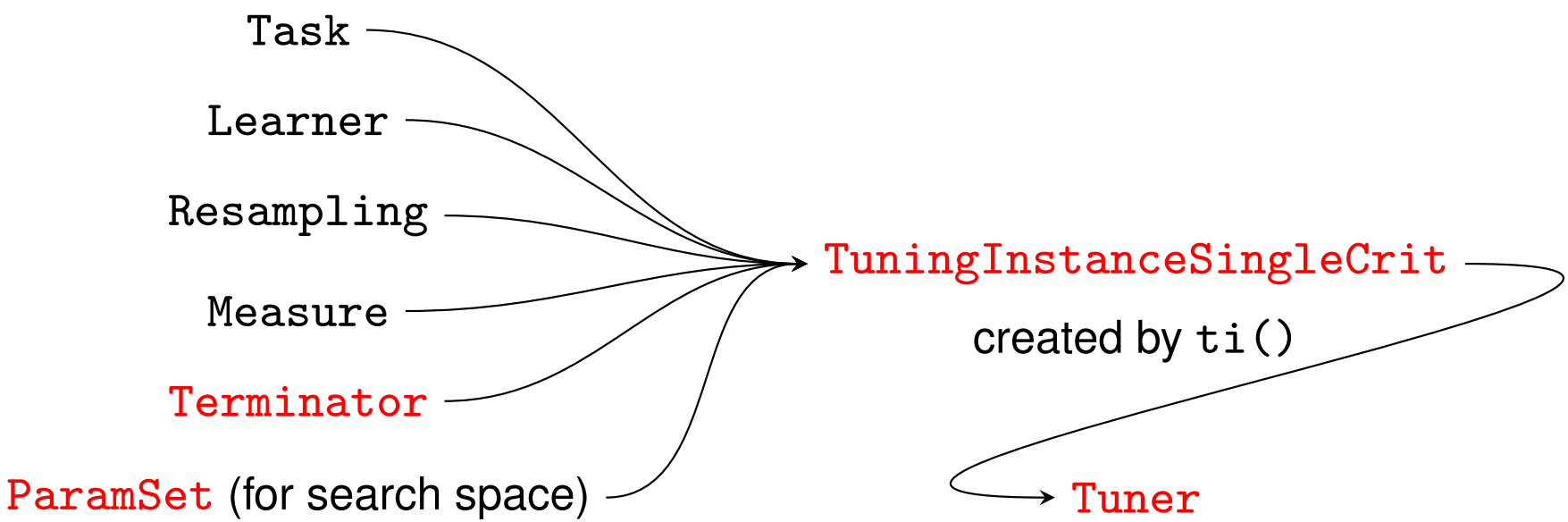
Parameter Transformation

AutoTuner and Nested Resampling

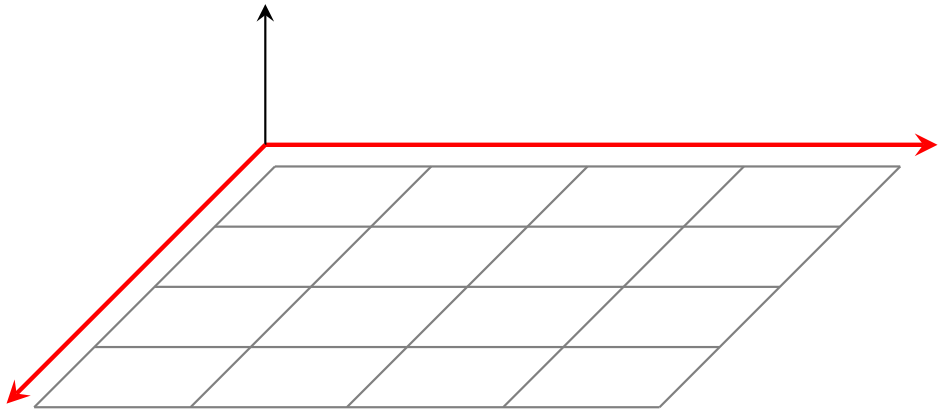
OBJECTS IN TUNING



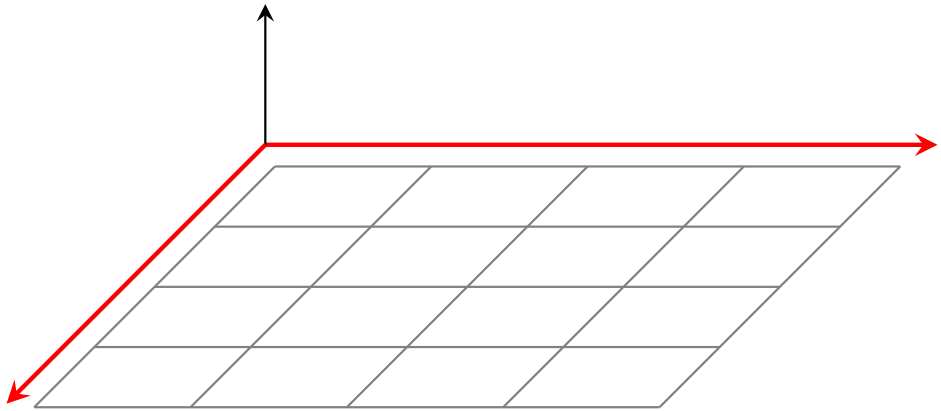
OBJECTS IN TUNING



SEARCH SPACE

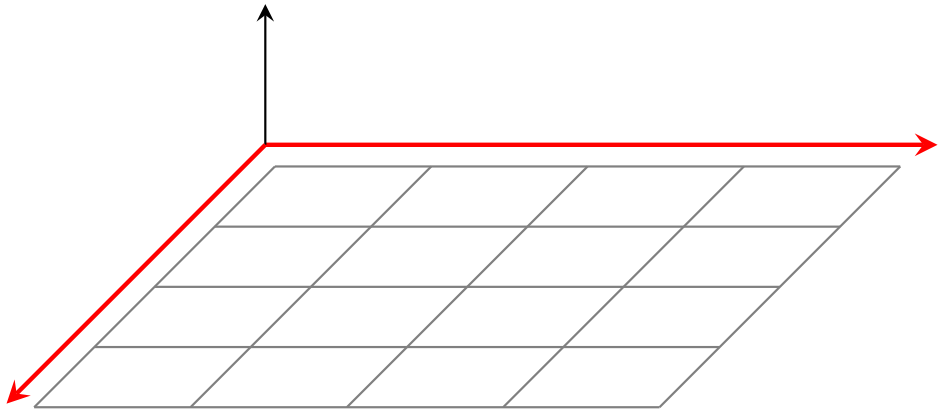


SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

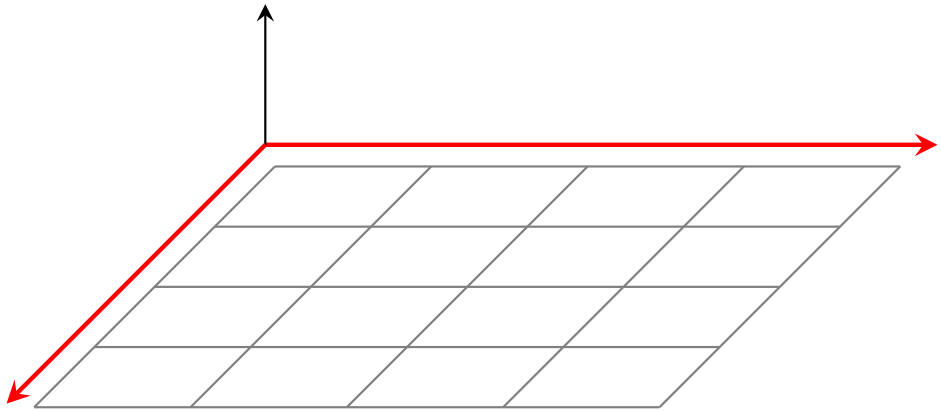
SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

SEARCH SPACE

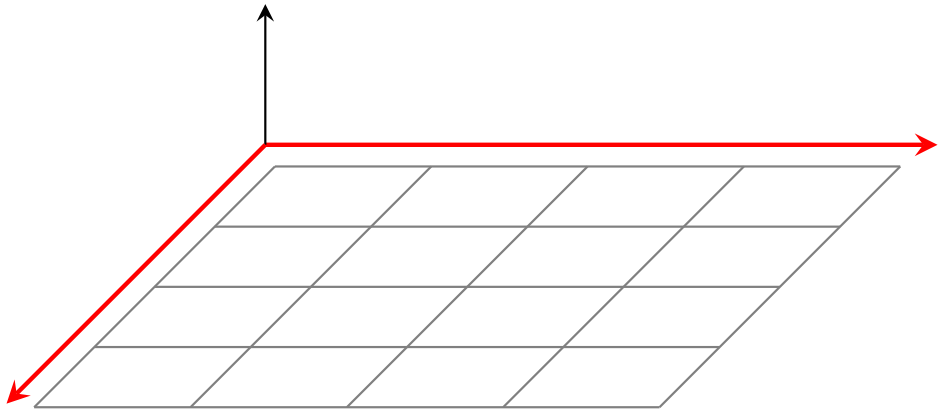


```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

```
library("paradox")
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", lower = 1, upper = 35)
))
```

SEARCH SPACE SHORT FORM



```
ps(id1 = domain1, id2 = domain2, ...)
```

Numerical parameter	`p_dbl(lower, upper)`
Integer parameter	`p_int(lower, upper)`
Discrete parameter	`p_fct(levels)`
Logical parameter	`p_lgl()`
Untyped parameter	`p_uty()`

```
library("paradox")
searchspace_knn = ps(
  "k" = p_int(lower = 1, upper = 35)
)
```

SEARCH SPACE IN LEARNER DEFINITION

Specify search space for the hyperparameter to be tuned directly when creating the learner using `to_tune`:

```
library("paradox")
learner = lrn("classif.kknn", k = to_tune(lower = 1, upper = 35))
learner$param_set

#> <ParamSet>
#>              id    class lower upper nlevels default                value
#>          <char>   <char> <num> <num>   <num>  <list>               <list>
#> 1:           k ParamInt     1   Inf     Inf       7 <RangeTuneToken[2]>
#> 2:    distance ParamDbl     0   Inf     Inf       2
#> 3:      kernel ParamFct    NA    NA      10 optimal
#> 4:       scale ParamLgl    NA    NA       2    TRUE
#> 5:     ykernel ParamUty    NA    NA     Inf
#> 6: store_model ParamLgl    NA    NA       2   FALSE
```

Note: Calling `$train()` on a learner with such a specified search space does not work.

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `mlr_terminators` dictionary:
- ```
  as.data.table(mlr_terminators)[, c("key", "label", "properties")]
  #> Key: <key>
  #>                  key                  label             properties
  #>                <char>                 <char>                 <list>
  #> 1:       clock_time             Clock Time single-crit,multi-crit
  #> 2:            combo            Combination single-crit,multi-crit
  #> 3:            evals Number of Evaluation single-crit,multi-crit
  #> 4:             none                   None single-crit,multi-crit
  #> 5:     perf_reached    Performance Level             single-crit
  #> 6:         run_time               Run Time single-crit,multi-crit
  #> 7:       stagnation             Stagnation             single-crit
  #> 8: stagnation_batch      Stagnation Batch             single-crit
  ```
  ```

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `mlr_terminators` dictionary:

```
as.data.table(mlr_terminators)[, c("key", "label", "properties")]
```

```
#> Key: <key>
#>                  key                label             properties
#>                <char>               <char>                 <list>
#> 1:       clock_time           Clock Time single-crit,multi-crit
#> 2:            combo          Combination single-crit,multi-crit
#> 3:            evals Number of Evaluation single-crit,multi-crit
#> 4:             none                 None single-crit,multi-crit
#> 5:     perf_reached  Performance Level            single-crit
#> 6:         run_time             Run Time single-crit,multi-crit
#> 7:       stagnation           Stagnation            single-crit
#> 8: stagnation_batch   Stagnation Batch            single-crit
```

- Construct `Terminator` via `trm()` short form:

```
trm("evals", n_evals = 20)
```

```
#> <TerminatorEvals>: Number of Evaluation
#> * Parameters: n_evals=20, k=0
```

TUNING METHOD

- Choose a *tuning method* via `mlr_tuners` dictionary:

```
as.data.table(mlr_tuners)[-1, c("key", "label")]
```

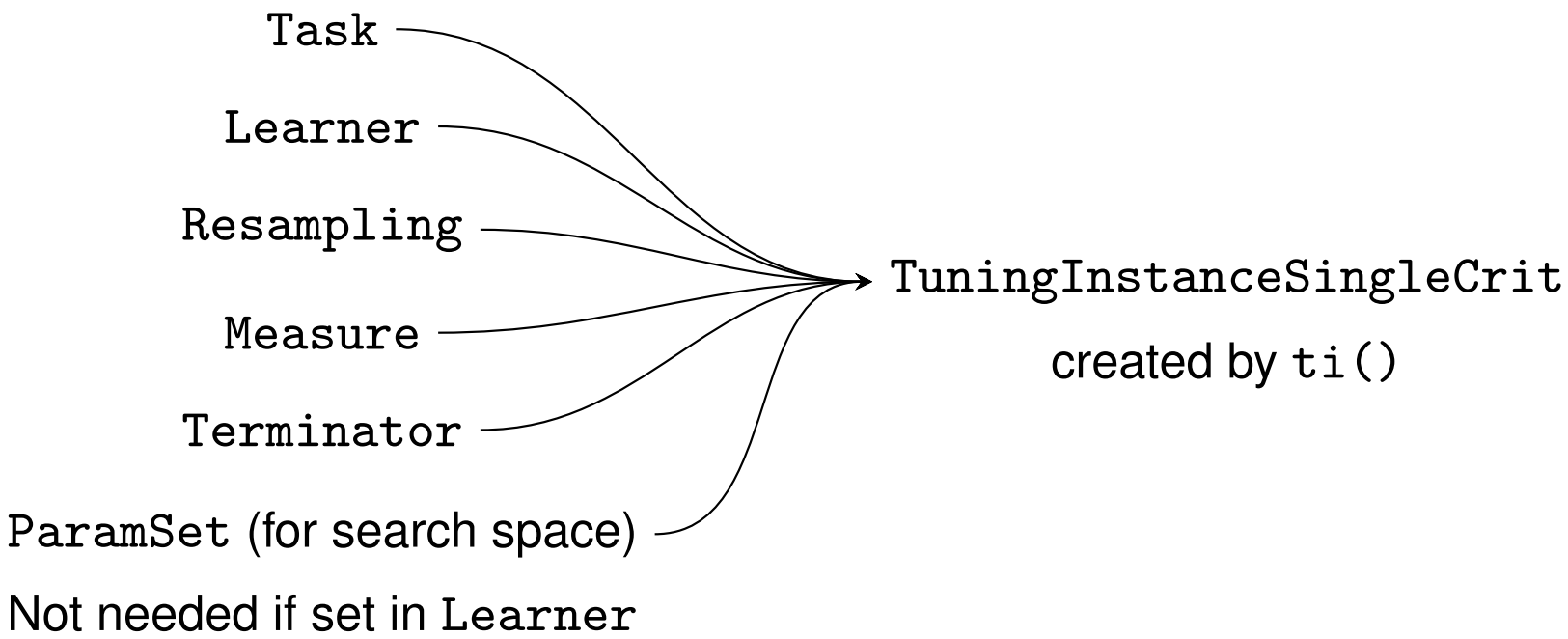
```
#> Key: <key>
#>                   key                              label
#>                <char>                             <char>
#> 1:      design_points                       Design Points
#> 2:              gensa Generalized Simulated Annealing
#> 3:        grid_search                        Grid Search
#> 4:          hyperband                          Hyperband
#> 5:              irace                     Iterated Racing
#> 6:                mbo          Model Based Optimization
#> 7:             nloptr           Non-linear Optimization
#> 8:      random_search                       Random Search
#> 9: successive_halving                 Successive Halving
```

- Construct `Tuner` via `tnr()` (`batch_size` for parallelization):

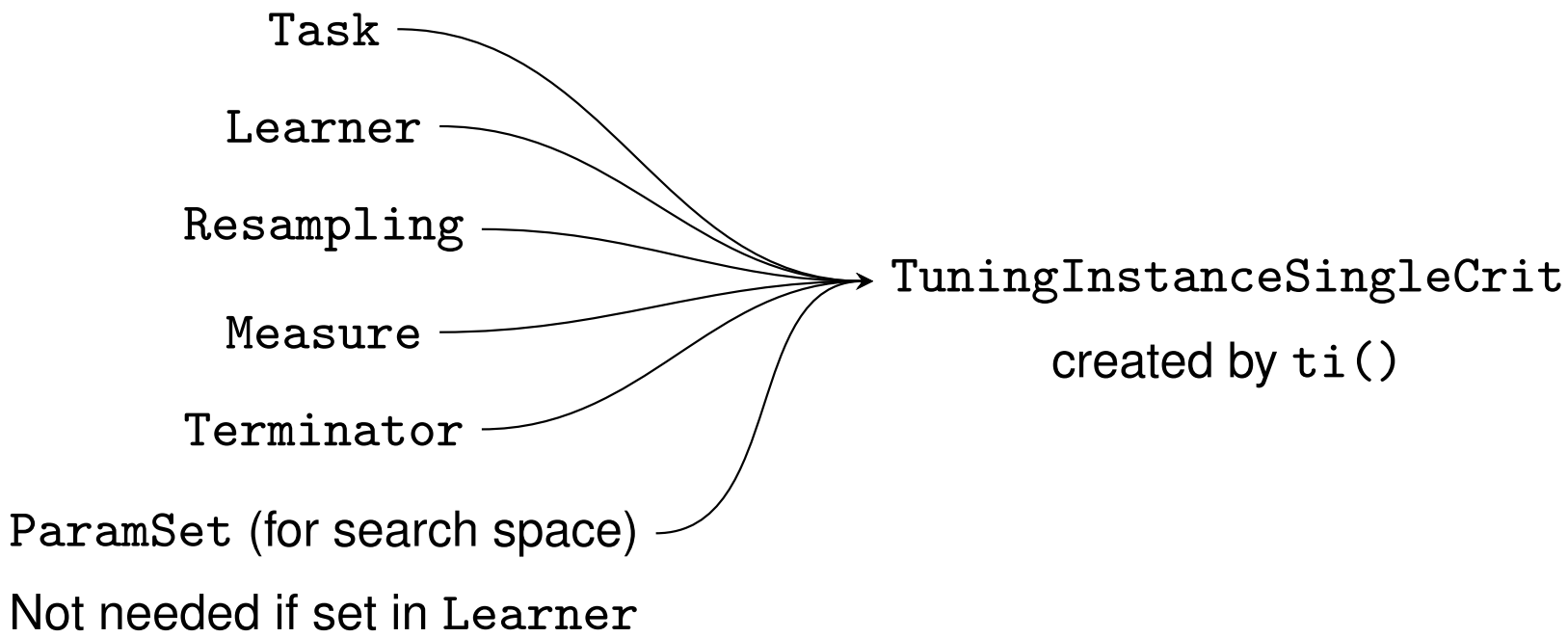
```
gsearch = tnr("grid_search", resolution = 7)
print(gsearch)
```

```
#> <TunerGridSearch>: Grid Search
#> * Parameters: resolution=7, batch_size=1
#> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
#> * Properties: dependencies, single-crit, multi-crit
#> * Packages: mlr3tuning
```

CALLING THE TUNER



CALLING THE TUNER




```
inst = ti(task = tsk("german_credit"),
  learner = lrn("classif.kknn"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = trm("evals", n_evals = 2),
  search_space = searchspace_knn
)
```

CALLING THE TUNER

```
gsearch$optimize(inst)
#> INFO  [07:05:21.749] [bbotk] Starting to optimize 1 parameter(s) with '<TunerGridSearch>
#> INFO  [07:05:21.846] [bbotk] Evaluating 1 configuration(s)
#> INFO  [07:05:24.370] [bbotk] Result of batch 1:
#> INFO  [07:05:24.377] [bbotk]   k classif.ce warnings errors runtime_learners
#> INFO  [07:05:24.377] [bbotk]  12       0.26        0      0             0.25
#> INFO  [07:05:24.377] [bbotk]                                      uhash
#> INFO  [07:05:24.377] [bbotk]  50fda423-4a96-4c6a-ba8f-dc58fb439a73
#> INFO  [07:05:24.381] [bbotk] Evaluating 1 configuration(s)
#> INFO  [07:05:24.644] [bbotk] Result of batch 2:
#> INFO  [07:05:24.651] [bbotk]  k classif.ce warnings errors runtime_learners
#> INFO  [07:05:24.651] [bbotk]  1       0.32        0      0             0.13
#> INFO  [07:05:24.651] [bbotk]                                      uhash
#> INFO  [07:05:24.651] [bbotk]  e0d8552b-bfc4-4419-855d-0bbbe4414e4c
#> INFO  [07:05:24.665] [bbotk] Finished optimizing after 2 evaluation(s)
#> INFO  [07:05:24.667] [bbotk] Result:
#> INFO  [07:05:24.672] [bbotk]      k learner_param_vals  x_domain classif.ce
#> INFO  [07:05:24.672] [bbotk]  <int>             <list>    <list>      <num>
#> INFO  [07:05:24.672] [bbotk]     12          <list[1]> <list[1]>       0.26
#>        k learner_param_vals  x_domain classif.ce
#>    <int>             <list>    <list>      <num>
#> 1:    12          <list[1]> <list[1]>       0.26
```

TUNING RESULTS

```
inst = ti(task = tsk("german_credit"), learner = lrn("classif.kknn"),
  resampling = rsmp("holdout"), measure = msr("classif.ce"),
  terminator = trm("none"), search_space = searchspace_knn)
gsearch = tnr("grid_search", resolution = 7)
gsearch$optimize(inst)

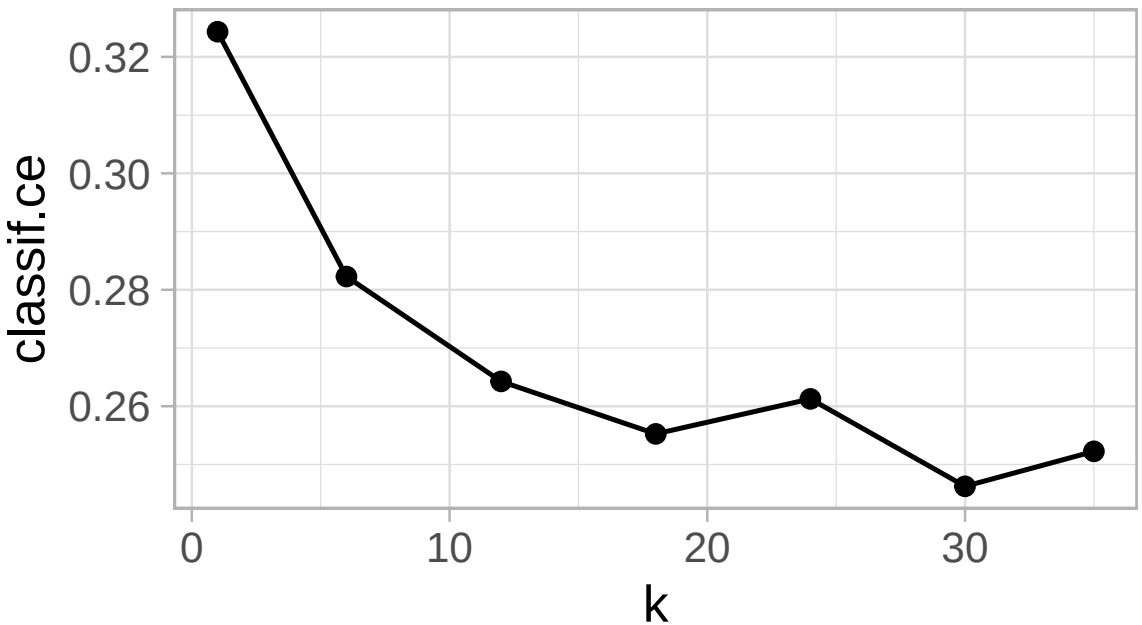
#>        k learner_param_vals  x_domain classif.ce
#>    <int>             <list>    <list>      <num>
#> 1:    30          <list[1]> <list[1]>       0.25

inst$archive # results are stored in tuning instance

#> <ArchiveTuning> with 7 evaluations
#>        k classif.ce batch_nr warnings errors
#>    <int>      <num>    <int>    <int>  <int>
#> 1:     6       0.28        1        0      0
#> 2:    24       0.26        2        0      0
#> 3:    18       0.26        3        0      0
#> 4:    30       0.25        4        0      0
#> 5:    12       0.26        5        0      0
#> 6:    35       0.25        6        0      0
#> 7:     1       0.32        7        0      0
```

TUNING RESULTS - VISUALIZATION

```
ggplot(as.data.table(inst$archive), aes(x = k, y = classif.ce)) +
  geom_line() + geom_point()
```



RECAP

1. Create a `Task`, `Learner`, `Resampling`, `Measure`, `Terminator` (defines when to stop), and a `ParamSet` (defines the search space):

```
task = tsk("german_credit")
learner = lrn("classif.kknn")
resampling = rsmp("holdout")
measure = msr("classif.ce")
terminator = trm("none")
searchspace_knn =  ps("k" = p_int(lower = 1, upper = 35))
```

2. Create the `TuningInstanceSingleCrit` object:

```
inst = ti(task, learner, resampling, measure,
  terminator, searchspace_knn)
```

3. Create the `Tuner` (tuning method) and optimize the learner by passing over the previously created instance to the `$optimize` method:

```
gsearch = tnr("grid_search", resolution = 7)
gsearch$optimize(inst)
```

```
#>        k learner_param_vals  x_domain classif.ce
#>    <int>             <list>    <list>      <num>
#> 1:    24          <list[1]> <list[1]>       0.24
```

Intro

Tuning

Tuning in mlr3

Parameter Transformation

AutoTuner and Nested Resampling

PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Use transformations (part of `ParamSet`)

PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Use transformations (part of `ParamSet`)

Example:

1. optimize from $\log(1) \ldots \log(100)$
2. transform by $\exp()$ in `trafo` function
3. don't forget to `round` (k must be integer)

```
searchspace_knn_trafo = ps(
  "k" = p_dbl(log(1), log(35), trafo = function(x) round(exp(x)))
)
```

PARAMETER TRANSFORMATION

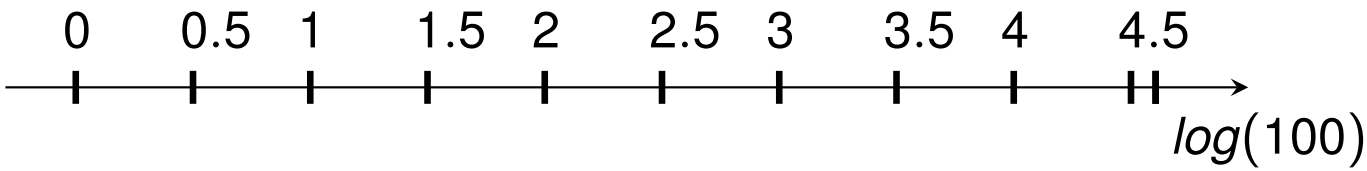
What is our transformation doing?

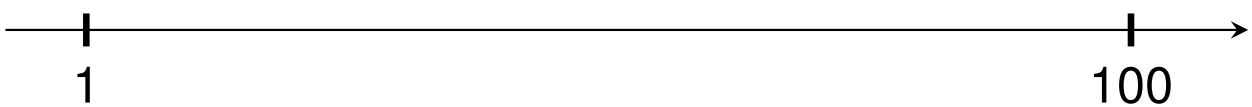
PARAMETER TRANSFORMATION

What is our transformation doing?

PARAMETER TRANSFORMATION

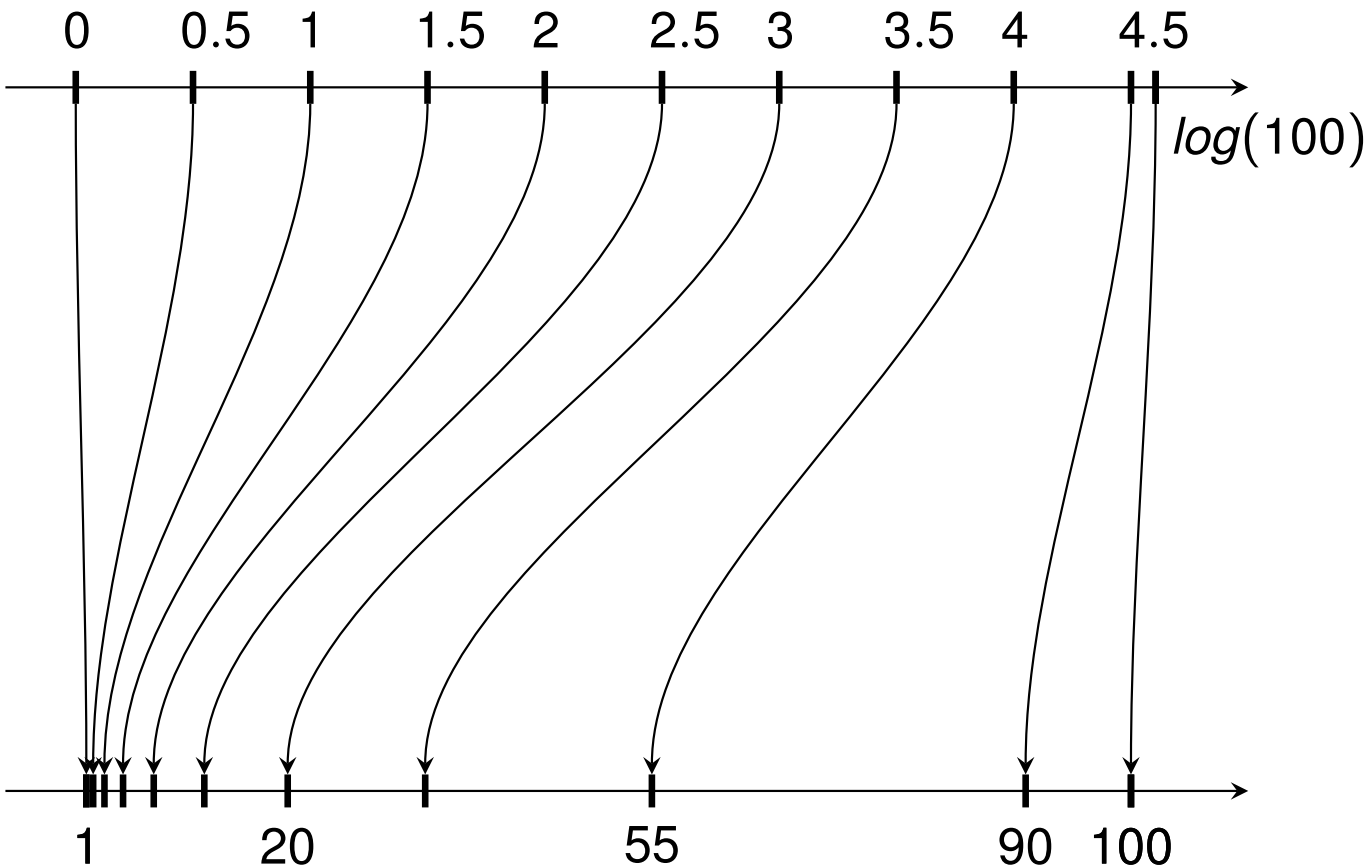
What is our transformation doing?





PARAMETER TRANSFORMATION

What is our transformation doing?



PARAMETER TRANSFORMATION

Tune again with `searchspace_knn_trafo` ...

```
inst_trafo = ti(task, learner, resampling, measure,
    terminator, searchspace_knn_trafo)
gsearch$optimize(inst_trafo)

#>        k learner_param_vals  x_domain classif.ce
#>    <num>             <list>    <list>      <num>
#> 1:   3.6          <list[1]> <list[1]>       0.21
```

PARAMETER TRANSFORMATION

Tune again with `searchspace_knn_trafo` ...

```
inst_trafo = ti(task, learner, resampling, measure,
    terminator, searchspace_knn_trafo)
gsearch$optimize(inst_trafo)

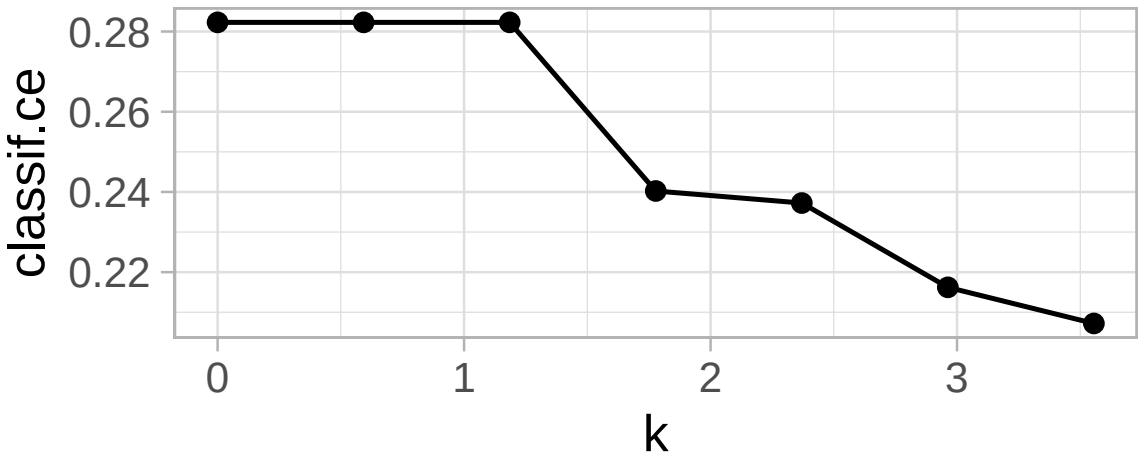
#>        k learner_param_vals  x_domain classif.ce
#>    <num>             <list>    <list>      <num>
#> 1:   3.6          <list[1]> <list[1]>       0.21
```

```
arv_trafo = as.data.table(inst_trafo$archive)
arv_trafo[, 1:4]

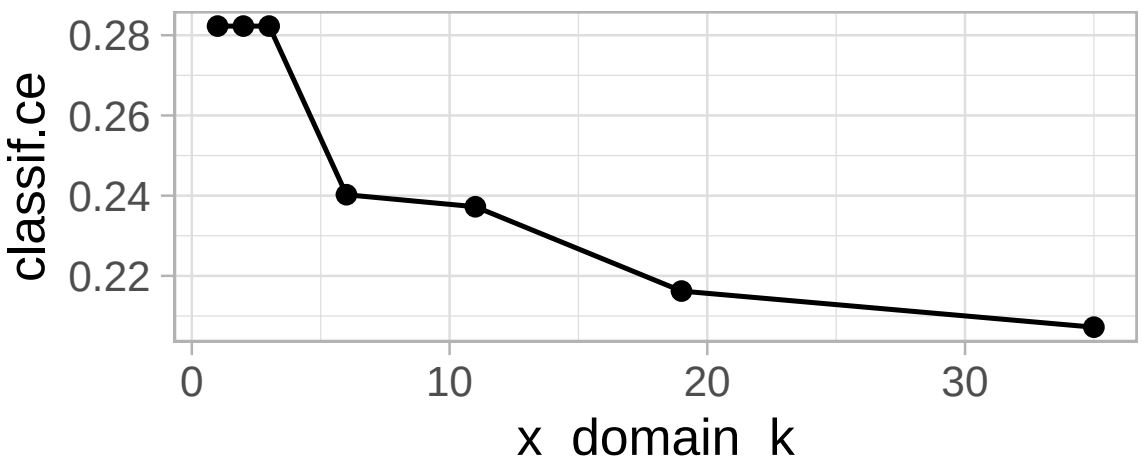
#>        k classif.ce x_domain_k runtime_learners
#>    <num>      <num>      <num>             <num>
#> 1:  2.96       0.22         19              0.13
#> 2:  3.56       0.21         35              0.19
#> 3:  1.78       0.24          6              0.11
#> 4:  1.19       0.28          3              0.17
#> 5:  0.59       0.28          2              0.11
#> 6:  0.00       0.28          1              0.12
#> 7:  2.37       0.24         11              0.12
```

PARAMETER TRANSFORMATION

```
ggplot(data = arv_trafo, aes(x = k, y = classif.ce)) +
  geom_line() + geom_point()
```




```
ggplot(data = arv_trafo, aes(x = x_domain_k, y = classif.ce)) +
  geom_line() + geom_point()
```



Intro

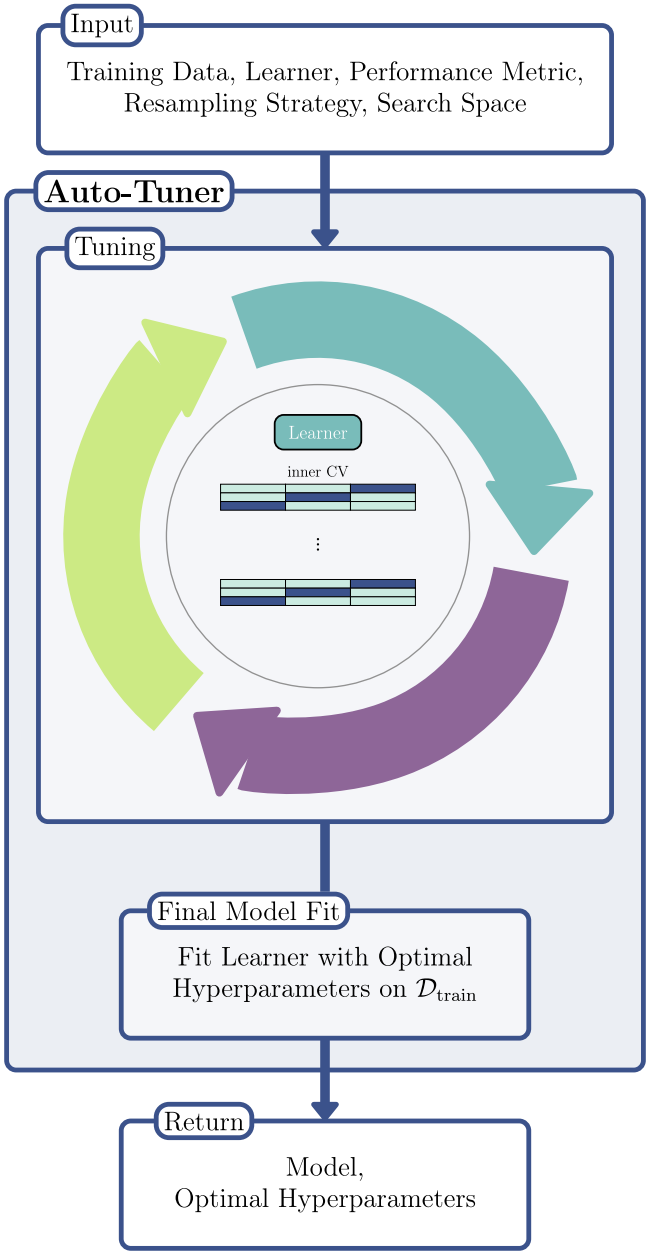
Tuning

Tuning in mlr3

Parameter Transformation

AutoTuner and Nested Resampling

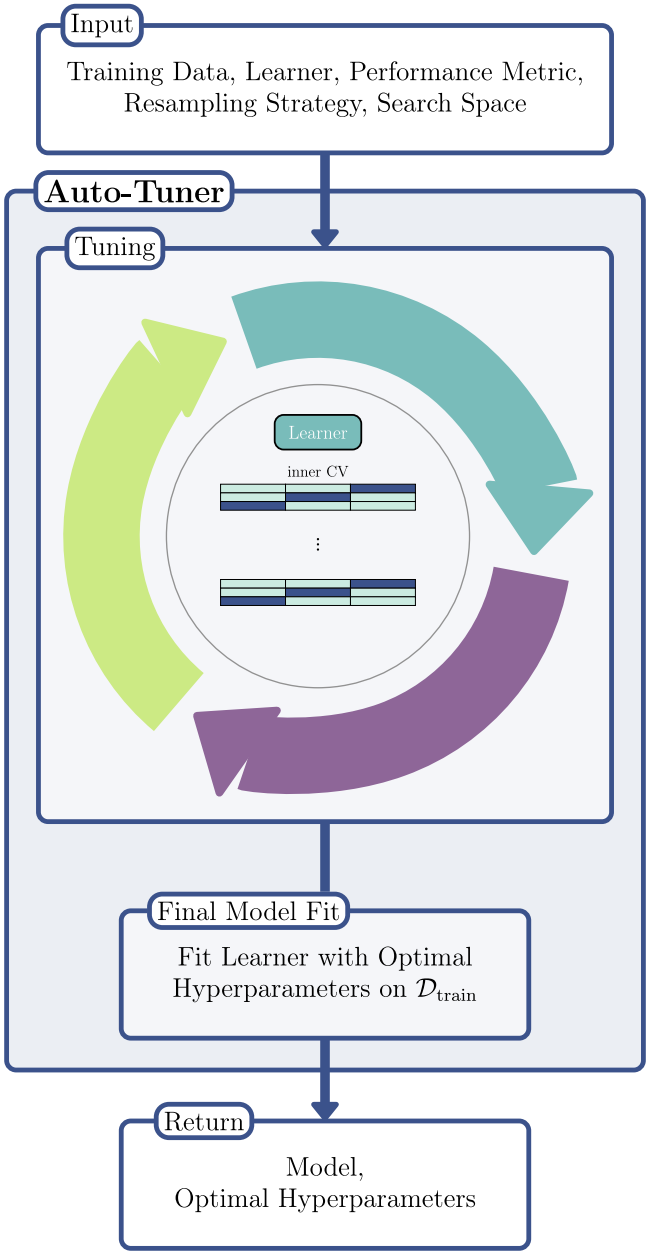
AUTOTUNER



Treat tuning as part of the learning process!

- Training:
  1. Tune model using (inner) resampling
  2. Train final model with optimal parameters on all data
- Predicting: Use final model

AUTOTUNER



Treat tuning as part of the learning process!

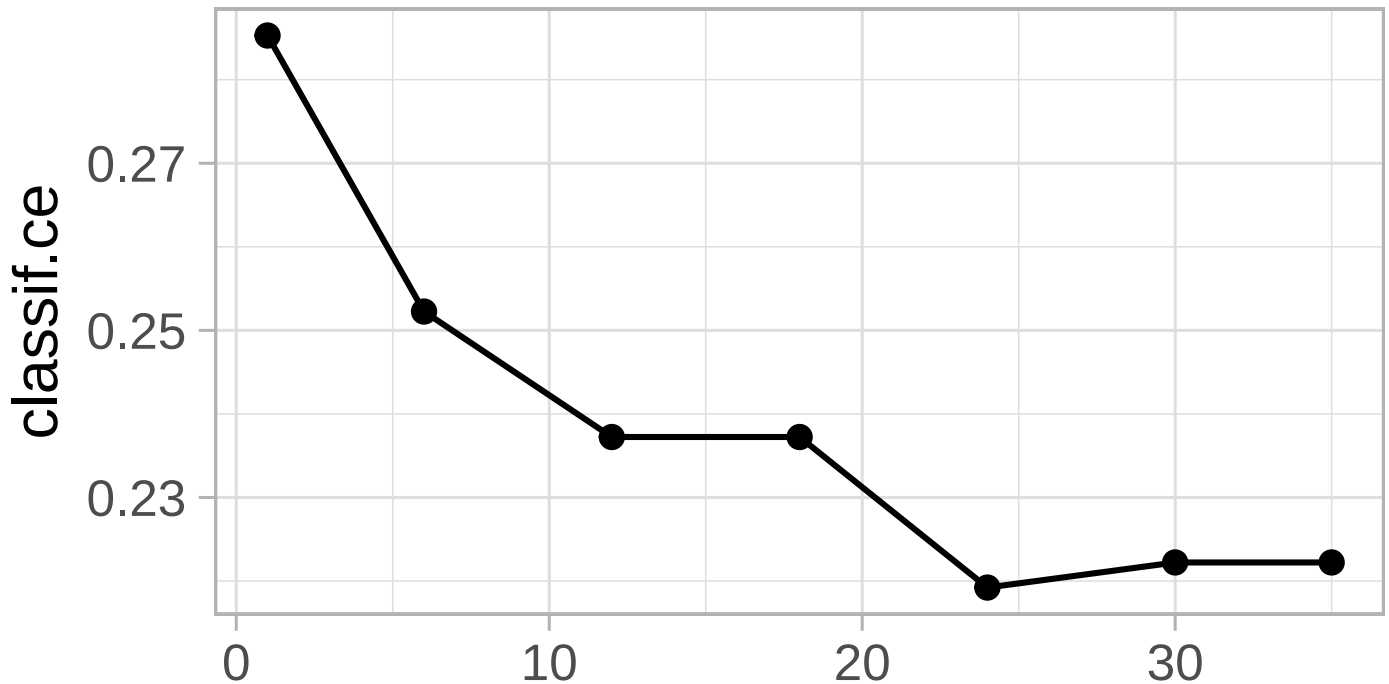
- Training:
  1. Tune model using (inner) resampling
  2. Train final model with optimal parameters on all data
- Predicting: Use final model

```
optlrn = auto_tuner(
  tuner = tnr("grid_search", resolution = 7),
  learner = lrn("classif.kknn", k = to_tune(1, 35)),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = trm("none"))
optlrn$train(tsk("german_credit"))
optlrn$learner # learner with best found HP

#> <LearnerClassifKKNN:classif.kknn>: k-Nearest-Neighbor
#> * Model: list
#> * Parameters: k=24
#> * Packages: mlr3, mlr3learners, kknn
#> * Predict Types:  [response], prob
#> * Feature Types: logical, integer, numeric,
#>   factor, ordered
#> * Properties: multiclass, twoclass
```

AUTOTUNER - ANALYZE TUNING RESULTS

```
arv = as.data.table(optlrn$tuning_instance$archive)
ggplot(arv, aes(x = k, y = classif.ce)) +
  geom_line() + geom_point() + xlab("")
```



AUTOTUNER - NESTED RESAMPLING

To estimate **tuned learner** performance, an outer resampling is required (performs nested resampling):

```
rr = resample(task = tsk("german_credit"), learner = optlrn,
  resampling = rsmp("cv", folds = 2), store_models = TRUE)
arv1 = as.data.table(rr$learners[[1]]$tuning_instance$archive)
arv2 = as.data.table(rr$learners[[2]]$tuning_instance$archive)
ggplot() +
  geom_line(data = arv1, aes(x = k, y = classif.ce)) +
  geom_line(data = arv2, aes(x = k, y = classif.ce), linetype = 2)
```