

Package ‘tabpfn’

September 25, 2026

Title Prior-Data Fitted Network Foundational Model for Tabular Data

Version 0.4.0

Description Provides a consistent API for classification and regression models based on the 'TabPFN' model of Hollmann et al. (2025), ``Accurate predictions on small data with a tabular foundation model," Nature, 637(8045) <[doi:10.1038/s41586-024-08328-6](https://doi.org/10.1038/s41586-024-08328-6)>. The calculations are served via 'Python' to train and predict the model.

License Apache License (>= 2)

URL <https://tabpfn.tidymodels.org>,
<https://github.com/tidymodels/tabpfn>

BugReports <https://github.com/tidymodels/tabpfn/issues>

Depends R (>= 4.1.0)

Imports cli, dplyr, generics, hardhat (>= 1.4.1), jsonlite, purrr,
reticulate (>= 1.41.0.1), rlang (>= 1.1.0), tibble

Suggests covr, ggplot2, MASS, modeldata, recipes, rstudioapi,
spelling, testthat (>= 3.0.0), withr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Max Kuhn [aut, cre] (ORCID: <<https://orcid.org/0000-0003-2402-136X>>),
Edgar Ruiz [aut],
Posit Software, PBC [cph, fnd] (ROR: <<https://ror.org/03wc8by49>>)

Maintainer Max Kuhn <max@posit.co>

Repository CRAN

Date/Publication 2026-09-25 07:30:02 UTC

Contents

control_tab_pfn	2
install_tabpfn	3
is_tab_pfn_installed	5
predict.tab_pfn	6
tabpfn_download_models	8
tabpfn_initialize	8
tabpfn_list_versions	9
tab_pfn	9

Index	17
--------------	-----------

control_tab_pfn	<i>Controlling TabPFN execution</i>
-----------------	-------------------------------------

Description

Controlling TabPFN execution

Usage

```
control_tab_pfn(  
    n_preprocessing_jobs = 1L,  
    device = "auto",  
    ignore_pretraining_limits = FALSE,  
    inference_precision = "auto",  
    fit_mode = "fit_preprocessors",  
    memory_saving_mode = "auto",  
    random_state = sample.int(10^6, 1),  
    ...  
)
```

Arguments

- n_preprocessing_jobs
An integer for the number of worker processes. A value of -1L indicates all possible resources.
- device
A character value for the device used for torch (e.g., "cpu", "cuda", "mps", etc.). Th default is "auto".
- ignore_pretraining_limits
A logical, passed to the Python library, allowing data past the limits the model was pre-trained for. It covers the number of training set samples and predictors, and the much lower sample limit that applies when the fit runs on a CPU. The limit on the number of classes always applies. See the *Data limits by version* section of [tab_pfn\(\)](#).

inference_precision	A character value for the trade off between speed and reproducibility. This can be a torch dtype, "autocast" (for torch's mixed-precision autocast), or "auto".
fit_mode	A character value to control how the are preprocessed and/or cached. Values are "fit_preprocessors" (the default), "low_memory", "fit_with_cache", and "batched".
memory_saving_mode	A character string to help with out-of-memory errors. Values are either a logical or "auto".
random_state	An integer to set the random number stream.
...	Additional named arguments passed directly to the TabPFN Python constructor. Use this to supply options not covered by the named parameters above (e.g. arguments added in newer versions of the Python package).

Value

A list with extra class "control_tab_pfn" that has named elements for each of the argument values.

References

<https://github.com/PriorLabs/TabPFN/blob/main/src/tabpfn/classifier.py>, <https://github.com/PriorLabs/TabPFN/blob/main/src/tabpfn/regressor.py>

Examples

```
control_tab_pfn()
```

install_tabpfn	<i>Install the TabPFN Python environment</i>
----------------	--

Description

Sets up a persistent Python virtual environment containing the tabpfn Python library so that `tab_pfn()` and friends can use it. By default the environment is named "r-tabpfn", which reticulate automatically discovers and prefers over an ephemeral environment (see the *Environment discovery* section).

Usage

```
install_tabpfn(
    version = "default",
    envname = "r-tabpfn",
    check_latest = TRUE,
    extra_packages = NULL,
    python_version = NULL,
    method = c("auto", "virtualenv", "conda"),
    new_env = identical(envname, "r-tabpfn"),
```

```

    restart_session = TRUE,
    ...
)

```

Arguments

version	The tabpfn version to install. Use "default" (or NULL) to install the latest release, a bare version string such as "2.0.9" to pin an exact version, or a full pip specification such as ">=2.0".
envname	The name of the Python virtual environment to create or use. The default, "r-tabpfn", is auto-discovered by reticulate.
check_latest	A logical. When TRUE (the default) and no explicit version is given, an existing environment is compared against the latest release on PyPI and you are asked whether to upgrade. Set to FALSE to skip this check (useful when intentionally staying on an older version).
extra_packages	An optional character vector of additional Python packages to install alongside tabpfn. You can also use it to set the version of one of tabpfn's own dependencies, such as holding torch at an older release. See the <i>Pinning dependencies</i> section.
python_version	An optional Python version to use for the environment.
method	The installation method, passed to <code>reticulate::py_install()</code> .
new_env	A logical. When TRUE, an existing environment named envname is removed and rebuilt from scratch. Defaults to TRUE only for the canonical "r-tabpfn" environment.
restart_session	A logical. When TRUE (the default) and running in RStudio, the R session is restarted after installation so the new environment takes effect.
...	Additional arguments passed to <code>reticulate::py_install()</code> .

Value

Invisibly returns the environment name.

Environment discovery

Because the package calls `reticulate::import("tabpfn")`, reticulate will automatically use a virtual environment named "r-tabpfn" if one exists, preferring it over the ephemeral environment that is otherwise created on demand. Environments selected via `RETICULATE_PYTHON`, `VIRTUAL_ENV`, or a project-local `.venv` take precedence over "r-tabpfn".

Pinning dependencies

The Python tabpfn library pulls in other Python packages, among them torch, numpy, pandas, and scikit-learn. A new release of any one of them can break code that worked the week before, even though your R version and your tabpfn version did not change. The error usually points at tabpfn rather than at the package that changed, so pinning the suspect dependency is a quick way to confirm the cause and to keep working until a fix lands upstream.

To pin a dependency in the persistent environment that `install_tabpfn()` builds, pass it and the version you want in `extra_packages`:

```
install_tabpfn(  
  version = "2.0.9",  
  extra_packages = "torch==2.13.0",  
  new_env = TRUE  
)
```

`extra_packages` is used only when an install runs. If `envname` already exists and already has the version you asked for, this function returns early and ignores the pin. Pass `new_env = TRUE` to delete the environment and build it again with the pin.

To pin a dependency in the ephemeral environment instead, call `reticulate::py_require()` before you fit a model:

```
library(tabpfn)  
reticulate::py_require("torch==2.13.0")
```

Examples

```
## Not run:  
# Install the latest release into "r-tabpfn"  
install_tabpfn()  
  
# Pin a specific version  
install_tabpfn(version = "2.0.9")  
  
# Pin a dependency as well  
install_tabpfn(version = "2.0.9", extra_packages = "torch==2.13.0")  
  
## End(Not run)
```

is_tab_pfn_installed	<i>Check the Python package installation</i>
----------------------	--

Description

Attempts to import the Python package

Usage

```
is_tab_pfn_installed()
```

Value

A single logical

Examples

```
if (interactive()) {
  # This may take a minute
  is_tab_pfn_installed()
}
```

predict.tab_pfn	<i>Predict using TabPFN</i>
-----------------	-----------------------------

Description

Predict using TabPFN

Usage

```
## S3 method for class 'tab_pfn'
predict(object, new_data, type = NULL, quantile_levels = NULL, ...)

## S3 method for class 'tab_pfn'
augment(x, new_data, type = NULL, quantile_levels = NULL, ...)
```

Arguments

object, x	A tab_pfn object.
new_data	A data frame or matrix of new predictors.
type	The type of prediction. For classification, can be "class" or "prob". Defaults to NULL which gives all prediction types possible. For regression, can be "mean" or "quantile"; when "quantile", quantile_levels must be supplied.
quantile_levels	A numeric vector of probabilities, sorted in increasing order, at which to predict the outcome distribution. Regression only; required when type = "quantile" and must otherwise be NULL.
...	Not used, but required for extensibility.

Value

`predict()` returns a tibble of predictions and `augment()` appends the columns in `new_data`. In either case, the number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`.

For regression data, the prediction is in the column `.pred`. For classification, the class predictions are in `.pred_class` and the probability estimates are in columns with the pattern `.pred_{level}` where `level` is the levels of the outcome factor vector.

When `quantile_levels` is given, regression results also have a `.pred_quantile` column of `hardhat::quantile_pred()` values.

Examples

```
## Not run:
if (rlang::is_installed(c("MASS", "ggplot2")) &
    is_tab_pfn_installed() &
    interactive()) {
  library(ggplot2)

  motorcycles <- MASS::mcycle
  in_tr <- seq(1, nrow(motorcycles), by = 2)
  mcycle_tr <- motorcycles[in_tr, ]
  mcycle_te <- motorcycles[-in_tr, ]

  mcycle_grid <-
    dplyr::tibble(
      times = seq(min(motorcycles$times), max(motorcycles$times), length.out = 200)
    )
  mcycle_grid$row <- seq_len(nrow(mcycle_grid))

  fit <- tab_pfn(accel ~ times, data = mcycle_tr)

  # -----
  # Predict mean acceleration

  mean_pred <- augment(fit, mcycle_grid)

  mean_p <-
    mean_pred |>
    ggplot(aes(times)) +
    geom_point(data = mcycle_te, aes(y = accel), alpha = 1 / 2) +
    geom_line(aes(y = .pred))

  #-----Predict 5 %, 50%
  # Predict 5%, 50%, and 90% quantiles of acceleration

  q_pred <-
    predict(fit,
      mcycle_grid,
      type = "quantile",
      quantile_levels = c(0.1, 0.5, 0.9))
  q_pred$row <- seq_len(nrow(q_pred))

  q_pred_longer <-
    q_pred$.pred_quantile |>
    dplyr::as_tibble() |>
    dplyr::full_join(mcycle_grid, by = ".row") |>
    dplyr::mutate(level = format(.quantile_levels))

  mean_p +
    geom_line(
      data = q_pred_longer,
      aes(y = .pred_quantile, col = level, group = level)
    )
}
```

```
}
## End(Not run)
```

```
tabpfn_download_models
```

Download all TabPFN pre-trained model checkpoints

Description

As of 2026-05-05, there are 36 pre-trained models equaling roughly 1.2 GB of storage. Each model is trained on various synthetic & real datasets tailored to classification & regression. This function routine will require you to sign a one-time license for both 2.5 & 2.6 model varieties. Downloading all models will take some time.

Usage

```
tabpfn_download_models(cache_dir = NULL)
```

Arguments

cache_dir an option to override the default cache directory

Value

Invisibly returns NULL. Called for its side effect of downloading model files.

Examples

```
tabpfn_download_models()
```

```
tabpfn_initialize
```

Eagerly initialize the TabPFN Python library

Description

Forces the Python tabpfn library (and its PyTorch dependency) to load now instead of on first use. Because PyTorch bundles its own OpenMP runtime, loading it before any other package that uses OpenMP avoids the segmentation fault described in <https://github.com/tidymodels/tabpfn/issues/34>.

For this to work, call it as the very first thing in your session, using `tabpfn::tabpfn_initialize()` (with the `::` prefix so it runs before `library(tabpfn)` and before any other package that might load OpenMP, such as **recipes**):

```
tabpfn::tabpfn_initialize()
library(tabpfn)
suppressPackageStartupMessages(library(recipes))
fit_obj <- tab_pfn(mpg ~ ., data = mtcars)
```

Usage

```
tabpfn_initialize()
```

Value

NULL, invisibly. Called for its side effect of loading the Python library.

Examples

```
## Not run:
tabpfn::tabpfn_initialize()
library(tabpfn)

## End(Not run)
```

tabpfn_list_versions	<i>List available TabPFN model versions</i>
----------------------	---

Description

Returns a character vector of valid model version strings accepted by `tab_pfn()`'s version argument. The available model versions are queried directly from the currently installed Python tabpfn library, not hard-coded in this package, so results may differ across Python library versions.

Usage

```
tabpfn_list_versions()
```

Value

A character vector of model version strings.

tab_pfn	<i>Fit a TabPFN model.</i>
---------	----------------------------

Description

`tab_pfn()` applies data to a pre-estimated deep learning model defined by Hollmann *et al* (2025). This model emulates Bayesian inference for regression and classification models.

Usage

```
tab_pfn(x, ...)

## Default S3 method:
tab_pfn(x, ...)

## S3 method for class 'data.frame'
tab_pfn(
  x,
  y,
  num_estimators = 8L,
  softmax_temperature = 0.9,
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = Inf,
  version = NULL,
  control = control_tab_pfn(),
  ...
)

## S3 method for class 'matrix'
tab_pfn(
  x,
  y,
  num_estimators = 8L,
  softmax_temperature = 0.9,
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = Inf,
  version = NULL,
  control = control_tab_pfn(),
  ...
)

## S3 method for class 'formula'
tab_pfn(
  formula,
  data,
  num_estimators = 8L,
  softmax_temperature = 0.9,
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = Inf,
  version = NULL,
  control = control_tab_pfn(),
  ...
)
```

```
## S3 method for class 'recipe'
tab_pfn(
  x,
  data,
  num_estimators = 8L,
  softmax_temperature = 0.9,
  balance_probabilities = FALSE,
  average_before_softmax = FALSE,
  training_set_limit = Inf,
  version = NULL,
  control = control_tab_pfn(),
  ...
)
```

Arguments

x	Depending on the context: • A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>.
...	Not currently used, but required for extensibility.
y	When x is a data frame or matrix , y is the outcome specified as: • A data frame with 1 numeric column. • A matrix with 1 numeric column. • A numeric vector for regression or a factor for classification.
num_estimators	An integer for the ensemble size. Default is 8L.
softmax_temperature	An adjustment factor that is a divisor in the exponents of the softmax function (see Details below). Defaults to 0.9.
balance_probabilities	A logical to adjust the prior probabilities in cases where there is a class imbalance. Default is FALSE. Classification only.
average_before_softmax	A logical. For cases where <code>num_estimators > 1</code> , should the average be done before using the softmax function or after? Default is FALSE.
training_set_limit	An integer greater than 2L, or Inf (the default) to use every row. Anything smaller samples the training set down to that many rows, stratified by class for classification and by quartile for regression. Use it to speed up a fit, or to make one possible at all on a machine that cannot hold the whole training set.
version	The model version, such as "v2.5" or "v3.5". A bare number works too: 2.5, "2.5", and "v2.5" are equivalent. Call <code>tabpfn_list_versions()</code> for the versions your installed Python library offers. When NULL (the default), the Python library's current default version is used. When set, the model is initialized via <code>create_default_for_version()</code> with the corresponding <code>ModelVersion</code> enum value.

control	A list of options produced by <code>control_tab_pfn()</code> .
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as: • A data frame containing both the predictors and the outcome.

Details

Computing Requirements:

This model can be used with or without a graphics processing unit (GPU). However, it is fairly limited when used with a CPU (and no GPU). There might be additional data size limitation warnings with CPU computations, and, understandably, the execution time is much longer. CPU computations can also consume a significant amount of system memory, depending on the size of your data.

GPUs using CUDA (Compute Unified Device Architecture) are most effective. Limited testing with others has shown that GPUs with Metal Performance Shaders (MPS) instructions (e.g., Apple GPUs) have limited utility for these specific computations and might be slower than the CPU for some data sets.

License Requirements:

Starting with version 2.5, using TabPFN requires accepting the model license and obtaining a token from PriorLabs. Every version from 2.5 onwards has its own license, and you must accept each one on its own. Accepting the license for one version does not cover the others.

To set up access:

1. Visit <https://ux.priorlabs.ai> and create an account.
2. Go to the **License** tab and accept the license for each model version you intend to use.
3. Obtain your token from your account page.
4. Set the TABPFN_TOKEN environment variable. The easiest way is to add it to your .Renviro file:

```
TABPFN_TOKEN=your_token_value
```

The `usethis` function `edit_r_environ()` can be very helpful here.

Users who already have TABPFN_TOKEN set can use TabPFN v2 without any additional steps.

Python Installation:

You will need a working Python virtual environment with the correct packages to use these modeling functions.

There are at least two ways to proceed.

Ephemeral uv Install:

The first approach, which we *strongly suggest*, is to simply load this package and attempt to run a model. This will prompt **reticulate** to create an ephemeral environment and automatically install the required packages. That process would look like this:

```
> library(tabpfn)
>
> predictors <- mtcars[, -1]
```

```

> outcome <- mtcars[, 1]
>
> # XY interface
> mod <- tab_pfn(predictors, outcome)
Downloading uv...Done!
Downloading cpython-3.12.12 (download) (15.9MiB)
  Downloading cpython-3.12.12 (download)
Downloading setuptools (1.1MiB)
Downloading scikit-learn (8.2MiB)
Downloading numpy (4.9MiB)

```

<downloading and installing more packages>

```

  Downloading llvmlite
  Downloading torch
Installed 58 packages in 350ms
> mod
TabPFN Regression Model

```

```

Training set
i 32 data points
i 10 predictors

```

The location of the environment can be found at `tools::R_user_dir("reticulate", "cache")`. See the documentation for [reticulate::py_require\(\)](#) to learn more about this method.

Persistent Environment with `install_tabpfn()`:

Alternatively, [install_tabpfn\(\)](#) creates a persistent virtual environment named "r-tabpfn" and installs the Python tabpfn library into it:

```

library(tabpfn)

# Install the latest release
install_tabpfn()

# Or pin a specific version for reproducibility
install_tabpfn(version = "2.0.9")

```

You do not need to call `use_virtualenv()` afterwards: because this package imports the Python module "tabpfn", **reticulate** automatically discovers and prefers the "r-tabpfn" environment over the ephemeral one. Run `install_tabpfn()` before **tabpfn** has initialized Python (i.e., before fitting a model); if Python is already loaded, restart R first.

Data:

Each model version was pre-trained on data up to a certain size, and those sizes have grown a great deal across versions. The *Data limits by version* section below has the numbers.

These limits are enforced by the Python library, which raises when data exceeds them. **tabpfn** does not check them itself, so the error you see names the model actually loaded.

Predictors do not require preprocessing; missing values and factor vectors are allowed.

Model Selection:

By default, TabPFN uses the Python library's current default model version. There are two ways to override this.

Selecting a model version:

Use the version argument to select a specific released model version:

```
mod <- tab_pfn(predictors, outcome, version = "v2.5")
```

```
# A bare number works too
```

```
mod <- tab_pfn(predictors, outcome, version = 3.5)
```

New model versions are released from time to time, so rather than listing them here, call `tabpfn_list_versions()` to see what your installed Python library offers:

```
> tabpfn_list_versions()
[1] "v2"      "v2.5"    "v2.6"    "v3"      "v3.5"    "v3.5-fast"
```

Pointing to a local model file:

If you have a model file on disk (e.g., downloaded for offline use), pass its path via `control_tab_pfn(model_path = ...)`:

```
ctrl <- control_tab_pfn(model_path = "/path/to/model_file.ckpt")
```

```
mod <- tab_pfn(predictors, outcome, control = ctrl)
```

Note that version and model_path are mutually exclusive: if version is set, it overwrites any model_path supplied through control.

Calculations:

For the softmax_temperature value, the softmax terms are:

```
exp(value / softmax_temperature)
```

A value of softmax_temperature = 1 results in a plain softmax value.

Value

A tab_pfn object with elements:

- fit: the python object containing the model.
- levels: a character string of class levels (or NULL for regression)
- training: a vector with the training set dimensions.
- version: the underlying TabPFN model version (or "unknown" if it cannot be determined).
- device: the device(s) the model was fitted on, e.g. "cpu", "mps", or "cuda:0" (or "unknown" if it cannot be determined).
- logging: any R or python messages produced by the computations.
- blueprint: an object produced by `hardhat::mold()` used to process new data during prediction.

Data limits by version

Version	Rows (GPU)	Rows (CPU)	Predictors	Classes
"v3.5-fast"	1M	5K	20K	160

"v3.5"	1M	5K	20K	160
"v3"	1M	5K	2K	160
"v2.6"	100K	1K	2K	10
"v2.5"	50K	1K	2K	10
"v2"	10K	1K	500	10

The CPU column is not advice. TabPFN refuses a CPU fit above that many rows, whatever the version's own limit says, so "v3.5" stops at 5,000 rows on a machine without a GPU. Set `ignore_pretraining_limits = TRUE` in `control_tab_pfn()`, or the `TABPFN_ALLOW_CPU_LARGE_DATASET` environment variable, to lift it. The fit then runs, slowly.

Every limit here is enforced by the Python library, which raises an error naming the count and the limit. Use `training_set_limit` to fit on a sample instead.

The row and predictor maxima trade off against each other, so you cannot always reach both at once. The ceiling is not a promise either: for "v3.5", PriorLabs recommends up to 6,000 predictors even though the model tops out at 20,000. See <https://docs.priorlabs.ai/models>.

References

Hollmann, Noah, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeyer, and Frank Hutter. "Accurate predictions on small data with a tabular foundation model." *Nature* 637, no. 8045 (2025): 319-326.

Hollmann, Noah, Samuel Müller, Katharina Eggenberger, and Frank Hutter. "Tabpfn: A transformer that solves small tabular classification problems in a second." *arXiv preprint arXiv:2207.01848* (2022).

Müller, Samuel, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. "Transformers can do Bayesian inference." *arXiv preprint arXiv:2112.10510* (2021).

Grinsztajn, Léo, et al. "Tabpfn-3: Technical report." *arXiv preprint arXiv:2605.13986* (2026).

Jäger, Benjamin, et al. "TabPFN-3.5: Technical Report." *arXiv preprint arXiv:2609.17895* (2026).

See Also

`control_tab_pfn()`, `predict.tab_pfn()`

Examples

```
predictors <- mtcars[, -1]
outcome <- mtcars[, 1]

## Not run:
if (is_tab_pfn_installed() & interactive()) {
  # XY interface
  mod <- tab_pfn(predictors, outcome)

  # Formula interface
  mod2 <- tab_pfn(mpg ~ ., mtcars)

  # Recipes interface
```

```
if (rlang::is_installed("recipes")) {  
  suppressPackageStartupMessages(library(recipes))  
  rec <-  
    recipe(mpg ~ ., mtcars) %>%  
    step_log(displacement)   
  
  mod3 <- tab_pfn(rec, mtcars)  
  mod3  
}  
}  
  
## End(Not run)
```

Index

`augment()`, [6](#)
`augment.tab_pfn(predict.tab_pfn)`, [6](#)

`control_tab_pfn`, [2](#)
`control_tab_pfn()`, [12](#), [15](#)

`hardhat::mold()`, [14](#)
`hardhat::quantile_pred()`, [6](#)

`install_tabpfn`, [3](#)
`install_tabpfn()`, [13](#)
`is_tab_pfn_installed`, [5](#)

`predict()`, [6](#)
`predict.tab_pfn`, [6](#)
`predict.tab_pfn()`, [15](#)

`recipes::recipe()`, [11](#)
`reticulate::py_install()`, [4](#)
`reticulate::py_require()`, [5](#), [13](#)

`tab_pfn`, [9](#)
`tab_pfn()`, [2](#), [3](#), [9](#)
`tabpfn_download_models`, [8](#)
`tabpfn_initialize`, [8](#)
`tabpfn_list_versions`, [9](#)
`tabpfn_list_versions()`, [11](#), [14](#)