Package: layaR (via r-universe)

November 17, 2026

Title Local Laya Decision Models via Rust

Version 0.1.0

Description Run Laya System 1 decision models (typed choice, score and yes/no questions with calibrated probabilities) locally from R, through the laya-onnx Rust crate and ONNX Runtime. No Python or server is required at prediction time.

License Apache License (>= 2)

URL https://gueyenono.github.io/layaR/, https://github.com/gueyenono/layaR

BugReports https://github.com/gueyenono/layaR/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.85.0, xz, OpenSSL headers on Linux (deb: libssl-dev, rpm: openssl-devel)

Depends R (>= 4.2)

Suggests knitr, parallel, rmarkdown, testthat (>= 3.0.0), tibble, withr

Config/testthat/edition 3

VignetteBuilder knitr

Imports cli, curl (>= 5.0.0), jsonlite, stats, utils

Config/pak/sysreqs libssl-dev xz-utils libclang-dev

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

Date/Publication 2026-11-17 06:07:12 UTC

RemoteUrl https://github.com/gueyenono/layaR

RemoteRef v0.1.0

RemoteSha 2286bc68934d5b51922fe4b14dc370be61039828

Contents

as_laya_questions . . . 2
laya_annotate . . . 3
laya_bucket_keys . . . 4
laya_clean_email . . . 5
laya_default_engine . . . 6
laya_detect_language . . . 6
laya_download_model . . . 7
laya_email_state . . . 8
laya_engine . . . 9
laya_engine_info . . . 11
laya_import_model . . . 12
laya_model_root . . . 12
laya_models . . . 13
laya_predict . . . 14
laya_preset . . . 16
laya_question . . . 17
laya_questions . . . 19
laya_questions_to_json . . . 20
laya_remove_model . . . 20
laya_result . . . 21
laya_validate_questions . . . 22

Index **23**

`as_laya_questions` *Convert to a question set*

Description

Accepts a question set in the Python SDK's format, so question sets can be shared between Python and R: an R list, a JSON string, or the path to a `.json` file.

Usage

```
as_laya_questions(x, ...)
```

Arguments

`x`	A `laya_questions` object, a named list in the Python format (`list(id = list(type = , instructions = , criteria = ))`), a JSON string, or a path to a JSON file.
`...`	Unused.

Value

A `laya_questions` object.

Examples

```
as_laya_questions('{
  "urgency": {"type": "score", "instructions": "How urgent?",
              "criteria": ["low", "high"]}
}')
```

`laya_annotate` *Annotate a data frame*

Description

Asks the same questions about every row of a data frame and adds the answers as new columns, one or more per question:

Usage

```
laya_annotate(
  data,
  questions,
  state,
  engine = laya_default_engine(),
  ...,
  prefix = "",
  confidence = TRUE,
  probabilities = FALSE
)
```

Arguments

`data`	A data frame or tibble. A tibble stays a tibble.
`questions`	A question set (`laya_questions()`, a preset, or anything `as_laya_questions()` accepts).
`state`	The column or columns the model reads. With one column, its text is the state as is. With several, each row becomes a structured state such as `{"subject": ..., "body": ...}`, which keeps the field names as context for the model.
`engine`	A `laya_engine()`. Defaults to `laya_default_engine()`.
`...`	Passed to `laya_predict_batch()`: `min_confidence`, `lang`, `batch_size`, `chunk_size`, `progress` and the token budgets.
`prefix`	A prefix for the new column names, for example `"laya_"`.
`confidence`	Add a `<id>_confidence` column per question?
`probabilities`	Add a `<id>_probabilities` list-column per question, holding each option's probability as a named numeric vector?

Details

- a choice question gives a factor whose levels are the options, in order;
- a score question gives the expected level (a number from 0 to the number of levels minus one);
- a yes/no question gives the probability that the statement is true.

With `confidence = TRUE` each answer also gets a `<id>_confidence` column. With `min_confidence` (passed in `...`), each gated answer also gets a logical `<id>_abstained` column. The answer is kept either way; filter on the column to drop low-confidence answers. The gate compares the threshold with each answer's `answer_confidence` (for a choice question, the chosen option's probability), which can differ from the `<id>_confidence` column.

Value

`data` with the new columns. Rows whose state columns are all missing get `NA` answers instead of a prediction.

See Also

`laya_predict_batch()` for the full results.

Examples

```
if (laya_model_available()) {
  tickets <- data.frame(
    subject = c("Duplicate charge", "App crashes on login"),
    body = c("We were billed twice. Refund it or we cancel.", "Since the update I can't log in.")
  )
  qs <- laya_questions(
    department = laya_choice("Which department should handle this?",
                             billing = "invoices, refunds", technical = "bugs, outages"),
    churn_risk = laya_noul("Does the customer threaten to cancel?")
  )
  laya_annotate(tickets, qs, state = c("subject", "body"))
}
```

---

laya_bucket_keys *Bucket names for* min_confidence

Description

A per-bucket `min_confidence` gives a threshold per question type and option count. These are the valid names; `"default"` also applies to every bucket not named.

Usage

```
laya_bucket_keys()
```

Value

A character vector.

Examples

```
laya_bucket_keys()
```

`laya_clean_email` *Clean email bodies*

Description

Removes quoted replies, reply headers, signatures and legal disclaimers so the model reads only the sender's message. A port of the Python SDK's `clean_email_body()`, with the same results.

Usage

```
laya_clean_email(body, max_chars = 3000)
```

Arguments

`body`	A character vector of email bodies. `NA` stays `NA`.
`max_chars`	The most characters to keep, applied after cleaning. Cleaning only looks at the first `4 * max_chars` characters, so raise it for long messages. As in Python, `0` returns `""` and a negative value trims from the end; it never means "no limit".

Value

A character vector the same length as `body`.

See Also

`laya_email_state()`

Examples

```
laya_clean_email(paste(
  "Please refund the duplicate charge.",
  "",
  "Best,",
  "Ana",
  "",
  "On Mon, 3 Mar 2025 at 10:00, Support <help@example.com> wrote:",
  "> How can we help?",
  sep = "\n"
))
```

laya_default_engine *Use the default engine*

Description

Returns the engine for `checkpoint`, loading it on first use and caching it for the rest of the session. Prediction functions use this when no `engine` is given.

Usage

```
laya_default_engine(checkpoint = getOption("layaR.checkpoint", "multilingual"))
```

Arguments

`checkpoint`	The checkpoint; defaults to the `layaR.checkpoint` option, or `"multilingual"`.

Value

A `laya_engine` object.

Examples

```
if (laya_model_available()) laya_default_engine()
```

laya_detect_language *Detect the language of states*

Description

Reads each state's text (string values only; keys are ignored) and reports its dominant script, a best-effort language guess for Latin-script text, and whether the English checkpoint can read it. Use it to pick a checkpoint: `"english"` when `is_english` is `TRUE`, `"multilingual"` otherwise. No model is needed.

Usage

```
laya_detect_language(states)
```

Arguments

`states`	A character vector of texts, a single state (a named list), a list of states, or a data frame (one state per row).

Details

A structured state that reads English as a whole is also checked field by field and line by line, so a short message in another language next to a long English stack trace still counts as non-English (`mixed_segment` shows the part that decided it).

Value

A data frame with one row per state and columns `script`, `language` (an ISO 639-1 code, or `NA` when undecided or not Latin), `is_english`, `language_undecided`, `diacritic_rate`, `non_latin_fraction`, `mixed_segment` and `script_profile` (a list-column of named numeric vectors: the share of letters in each script).

Examples

```
laya_detect_language(c(
  "Please refund the duplicate charge",
  "                  ",
  "glömt lösenord"
))
laya_detect_language(list(
  subject = "Refund",
  body = "Hola, me cobraron dos veces. \u00bfPueden devolverme el dinero?"
))
```

---

laya_download_model *Download a model*

Description

Downloads an exported ONNX checkpoint (1.3 to 1.6 GB) from a Hugging Face repository into `laya_model_path()`. Files go to a staging folder first, so an interrupted download resumes where it stopped on the next call and a partial download is never mistaken for a model.

Usage

```
laya_download_model(
  checkpoint = c("multilingual", "english", "typed-decisions"),
  repo = getOption("layaR.model_repo", "gueyenono/laya-onnx"),
  revision = "main",
  subfolder = NULL,
  token = Sys.getenv("HF_TOKEN"),
  overwrite = FALSE,
  ask = interactive()
)
```

Arguments

`checkpoint`	Which checkpoint to download.
`repo`	The Hugging Face repository, `"owner/name"`. Defaults to the `layaR.model_repo` option, or `"gueyenono/laya-onnx"`.
`revision`	Branch, tag or commit of the repository.

`subfolder`	Folder of the checkpoint inside the repository. `NULL` uses the upstream layout (`multilingual/`, `typed-decisions/`, and the repository root for `english`).
`token`	A Hugging Face access token for private or gated repositories; defaults to the `HF_TOKEN` environment variable. The `HF_ENDPOINT` environment variable changes the server, as in other Hugging Face tools.
`overwrite`	Replace an installed model?
`ask`	Ask for confirmation before downloading? Defaults to `TRUE` in interactive sessions.

Details

The Laya authors publish PyTorch weights only, so layaR downloads from `gueyenono/laya-onnx`, which holds ONNX exports of the three upstream checkpoints (unchanged weights, Apache-2.0). A repository without an export gives a `laya_error_model_not_published` error that explains how to export one and add it with `laya_import_model()`.

Value

The model's path, invisibly.

See Also

`laya_models()`, `laya_import_model()`

Examples

```
## Not run:
laya_download_model("multilingual")

## End(Not run)
```

`laya_email_state` *Build an email state*

Description

Builds the state the Python SDK's `email_state()` builds: `subject`, a cleaned `body`, `from` when a `sender` is given, then any extra fields. Pass the result to `laya_predict()`, or use it with the `"email"` preset (`laya_preset()`).

Usage

```
laya_email_state(
  subject,
  body,
  sender = NULL,
  ...,
  clean = TRUE,
  max_chars = 3000
)
```

Arguments

`subject`, `body`	The email's subject and body, single strings.
`sender`	Optional sender address, stored in the state's `from` field.
`...`	Extra named fields to add to the state (for example `ticket_id = 4411`). `NULL` fields are dropped, as in Python.
`clean`	Clean the body with `laya_clean_email()`? If `FALSE` the body is passed through whole.
`max_chars`	Passed to `laya_clean_email()` when `clean = TRUE`.

Value

A named list, ready to use as a state.

Examples

```
laya_email_state(
  subject = "Duplicate charge",
  body = "We were billed twice.\n\n-- \nAna Muller\nAcme Corp",
  sender = "ana@example.com",
  ticket_id = 4411L
)
```

`laya_engine` *Load a Laya engine*

Description

Loads an exported ONNX checkpoint into memory (about 1 to 1.5 GB of RAM) and returns an engine to pass to `laya_predict()` and friends. Create one and reuse it: loading takes several seconds, predictions are fast.

Usage

```
laya_engine(
  checkpoint = c("multilingual", "english", "typed-decisions"),
  model_dir = NULL,
  intra_op_threads = getOption("layaR.threads"),
  inter_op_threads = NULL,
  lang_temperatures = NULL
)
```

Arguments

`checkpoint`	Which checkpoint to load: `"multilingual"` (100+ languages, the default), `"english"` (English-optimized) or `"typed-decisions"` (fine-tuned for four specific workflows).
`model_dir`	Path to an exported checkpoint directory. Overrides the lookup by `checkpoint` (see `laya_model_path()`).
`intra_op_threads`, `inter_op_threads`	ONNX Runtime thread counts. `NULL` lets ONNX Runtime choose. The `layaR.threads` option sets the default for `intra_op_threads`.
`lang_temperatures`	Optional per-language calibration overrides, a named list such as `list(fr = list(temperature = c(1.1, 1, 0.9)))`; see the Python SDK's `lang_temperatures`. They apply to calls made with a matching `lang`.

Details

Most code never needs this: the prediction functions default to `laya_default_engine()`, which loads the checkpoint named by the `layaR.checkpoint` option (`"multilingual"` by default) on first use and keeps it for the session.

Engines can't be sent to other processes. An engine restored with `readRDS()`, or used in a PSOCK worker (`parallel::makePSOCKcluster()`, `future::multisession`), reconnects to its model directory on first use, which loads the model again in that process (another 1 to 1.5 GB each).

Engines don't work across `fork()` (`parallel::mclapply()`, `future::multicore`): ONNX Runtime's thread pools don't survive it, so a forked child could hang. Using an engine in a forked child raises a `laya_error_forked` error instead. A single engine already runs one forward pass on several cores; for more throughput use `laya_predict_batch()`, or PSOCK workers that each load their own engine.

Value

A `laya_engine` object.

See Also

`laya_predict()`, `laya_engine_info()`, `laya_engine_close()`

Examples

```
if (laya_model_available()) {
  engine <- laya_engine("multilingual", intra_op_threads = 2)
  engine
  laya_engine_close(engine)
}
```

`laya_engine_info` *Inspect or close an engine*

Description

`laya_engine_info()` reports what an engine loaded: checkpoint, ONNX layout, file paths, sequence budgets (`max_len`, `head_max_len`) and calibration temperatures. `laya_engine_close()` frees the model's memory right away instead of waiting for garbage collection; the engine can't be used afterwards.

Usage

```
laya_engine_info(engine)

laya_engine_close(engine)
```

Arguments

`engine`	A `laya_engine`.

Value

`laya_engine_info()` returns a list; `laya_engine_close()` returns `engine`, invisibly.

Examples

```
if (laya_model_available()) {
  engine <- laya_engine()
  str(laya_engine_info(engine), max.level = 1)
  laya_engine_close(engine)
}
```

laya_import_model *Import an exported model*

Description

Adds a checkpoint exported with Laya's `export_onnx.py` (or downloaded by another tool) to layaR's model root, so `laya_engine()` finds it by name.

Usage

```
laya_import_model(
  path,
  checkpoint = c("multilingual", "english", "typed-decisions"),
  method = c("symlink", "copy"),
  overwrite = FALSE
)
```

Arguments

`path`	The export's folder: it holds `model.onnx` (or `encoder.onnx` and `head.onnx`), `rl_agent_config.json` and the tokenizer.
`checkpoint`	Which checkpoint the export is.
`method`	`"symlink"` (the default) links to the folder without copying 1.3 GB; `"copy"` copies it, so the original can be deleted.
`overwrite`	Replace an installed model?

Value

The installed path, invisibly.

Examples

```
## Not run:
laya_import_model("~/exports/onnx/multilingual", "multilingual")

## End(Not run)
```

laya_model_root *Where models live*

Description

layaR keeps exported checkpoints in a model root folder with one subfolder per checkpoint (`multilingual/`, `english/`, `typed-decisions/`). The root is, in order: the `layaR.model_root` option, the `LAYA_ONNX_ROOT` environment variable (shared with Laya's Rust SDK), or the user cache folder `tools::R_user_dir("layaR", "cache")` plus `onnx/`. Downloads and imports go there.

Usage

```
laya_model_root()

laya_model_path(checkpoint = c("multilingual", "english", "typed-decisions"))

laya_model_available(
  checkpoint = getOption("layaR.checkpoint", "multilingual")
)
```

Arguments

`checkpoint`	A checkpoint name.

Details

When a checkpoint isn't in the model root, layaR also looks in the cache that Laya's other SDKs share (`~/.cache/laya/onnx`, or `%LOCALAPPDATA%/laya/onnx` on Windows), so a model exported or downloaded for the Rust or TypeScript SDK is reused. layaR never writes there. Set `options(layaR.shared_cache = FALSE)` to turn this off.

Value

`laya_model_root()` and `laya_model_path()` return paths (the latter is where layaR installs `checkpoint`); `laya_model_available()` returns `TRUE` when a complete export is found in either place.

See Also

`laya_models()`, `laya_download_model()`, `laya_import_model()`

Examples

```
laya_model_root()
laya_model_path("english")
laya_model_available()
```

`laya_models` *List checkpoints and whether they are installed*

Description

List checkpoints and whether they are installed

Usage

```
laya_models()
```

Value

A data frame with one row per checkpoint: `checkpoint`, `installed`, `path` (where it was found, or where it would be installed), `source` (`"layaR"` for the model root, `"shared"` for the cross-SDK cache, `NA` when not installed), `layout` (`"fused"` or `"split"`) and `size_gb`.

Examples

```
laya_models()
```

---

laya_predict *Answer typed questions about a state*

Description

`laya_predict()` answers every question about one state in a single forward pass. `laya_predict_batch()` does the same for many states. `laya_predict_long()` scans a state longer than the model's context in overlapping windows and keeps, per question, the most confident window.

Usage

```
laya_predict(
  state,
  questions,
  engine = laya_default_engine(),
  ...,
  min_confidence = NULL,
  lang = NULL,
  max_len = NULL,
  head_max_len = NULL
)

laya_predict_batch(
  states,
  questions,
  engine = laya_default_engine(),
  ...,
  batch_size = getOption("layaR.batch_size", 1L),
  sort_by_length = FALSE,
  min_confidence = NULL,
  lang = NULL,
  max_len = NULL,
  head_max_len = NULL,
  chunk_size = 64L,
  progress = interactive()
)

laya_predict_long(
```

```
  state,
  questions,
  engine = laya_default_engine(),
  ...,
  window = NULL,
  stride = NULL,
  batch_size = NULL,
  lang = NULL
)

## S3 method for class 'laya_engine'
predict(object, state, questions, ...)
```

Arguments

`state`	A string, a named list, or a one-row data frame.
`questions`	A question set from `laya_questions()` or `laya_preset()`, or anything `as_laya_questions()` accepts.
`engine`	A `laya_engine()`. Defaults to `laya_default_engine()`.
`...`	Must be empty; the options below are named.
`min_confidence`	Optional abstention gate: a threshold in `[0, 1]`, or a named numeric vector of thresholds per option-count bucket (names from `laya_bucket_keys()`, plus `"default"`). The threshold is compared with each answer's `answer_confidence` (for a choice question, the chosen option's probability), as in the Python SDK. Every answer then reports `abstention`: `"passed"`, `"abstained"` or `"unevaluated"`.
`lang`	Optional language code (e.g. `"fr"`) selecting per-language calibration temperatures, when the engine has them.
`max_len`, `head_max_len`	Optional per-call token budgets overriding the checkpoint's: the whole sequence, and the part spent on a question's options.
`states`	A character vector, a list of states, or a data frame (each row becomes a named list).
`batch_size`	States per forward pass. Defaults to the `layaR.batch_size` option, or 1: on a CPU, one state per pass is usually fastest, because a batch pads every state to the longest one (see `vignette("models-and-performance", package = "layaR")`). Larger batches can pay off on other hardware. `NULL` sends each chunk in one pass, as the Python SDK does by default. Results are the same either way.
`sort_by_length`	Group similarly sized states within a chunk to reduce padding (only with a `batch_size` above 1 and smaller than the chunk).
`chunk_size`	States sent to the engine per call. Between chunks R can be interrupted and the progress bar updates.

`progress`	Show a progress bar? Defaults to `TRUE` in interactive sessions with more than one chunk.
`window`, `stride`	Optional scan window and stride, in tokens. By default the window is the room the questions leave in one sequence and the stride is half the window.
`object`	A `laya_engine`, for the `stats::predict()` method.

Details

A state is either a single string, used verbatim, or a named list, which the model reads as a JSON object (field order is kept). A one-row data frame is read as a named list.

Value

`laya_predict()` and `laya_predict_long()` return a `laya_result`; `laya_predict_batch()` returns a `laya_batch_result` (a list of `laya_result`s, in input order). See laya_result for how to read them.

Examples

```
if (laya_model_available()) {
  qs <- laya_questions(
    department = laya_choice("Which department should handle this?",
                             billing = "invoices, refunds",
                             technical = "bugs, outages"),
    churn_risk = laya_noul("Does the user threaten to cancel?")
  )
  res <- laya_predict("I was billed twice. Refund me or I cancel.", qs)
  res
  as.data.frame(res)

  laya_predict_batch(c("Refund please", "The app crashes"), qs)
}
```

---

laya_preset *Built-in question presets*

Description

Ready-made question sets ported from the Python SDK's presets. Each preset's instructions refer to one field of the state, so pass the state as a named list with that field (see `laya_preset_fields()`).

Usage

```
laya_preset(
  name = c("triage", "email", "guard", "moderation", "router"),
  categories = NULL
)

laya_preset_fields()
```

Arguments

`name`	The preset name.
`categories`	For the `email` preset only: the routing categories, as a named character vector (`c(label = "description")`) in offer order. `NULL` uses the defaults (billing, technical, sales, security, hr, other).

Details

Preset	State field	Questions
`triage`	`message`	intent, is_urgent, frustration, refund_requested, churn_risk
`email`	`body`	category, is_spam, is_phishing, urgency, needs_reply
`guard`	`prompt`	jailbreak, prompt_injection, sensitive_data, harm_severity, topic
`moderation`	`post`	toxic, harassment, threat, spam, severity
`router`	`request`	difficulty, domain, needs_tools, is_sensitive

Value

A `laya_questions` object.

Examples

```
laya_preset("triage")
laya_preset("email", categories = c(orders = "order status, shipping",
                                    returns = "returns and exchanges"))
laya_preset_fields()
```

`laya_question` *Build Laya questions*

Description

Laya answers three kinds of typed question about a state in one forward pass:

Usage

```
laya_choice(instructions, ..., .options = NULL)

laya_score(instructions, levels)

laya_noul(instructions, if_false = NULL, if_true = NULL)
```

Arguments

`instructions`	The question, as a string. A list is also accepted and is serialized to JSON, as the Python SDK does for structured instructions.
`...`	For `laya_choice()`, the options: either named arguments (`label = "description"`) or unnamed labels without descriptions. Option order matters: the model scores one marker per option, in order.
`.options`	Alternatively, the options as a named character vector or list (labels with descriptions), or an unnamed character vector (labels only).
`levels`	For `laya_score()`, the levels from lowest to highest, as a character vector or a list of structured levels.
`if_false`, `if_true`	For `laya_noul()`, optional descriptions of what the false and true sides mean.

Details

- `laya_choice()`: pick one label from a fixed set of options.
- `laya_score()`: place the state on an ordinal scale (levels lowest first); the answer is the expected level, a number between 0 and the number of levels minus one.
- `laya_noul()`: a yes/no statement; the answer is the calibrated probability that it holds.

Combine questions into a set with `laya_questions()`.

Value

A `laya_question` object.

See Also

`laya_questions()`, `laya_preset()`

Examples

```
laya_choice(
  "Which department should handle this request?",
  billing = "invoices, payments, refunds",
  technical = "bugs, outages, system errors",
  other = "everything else"
)
laya_choice("What is this about?", "coding", "writing", "other")
```

```
laya_score("How urgent is this request?", c("not urgent", "soon", "blocking"))
laya_noul("Is this email a phishing attempt?",
          if_false = "a legitimate email", if_true = "phishing or fraud")
```

`laya_questions` *Combine questions into a question set*

Description

Combine questions into a question set

Usage

```
laya_questions(...)
```

Arguments

`...`	Named questions built with `laya_choice()`, `laya_score()` or `laya_noul()`. The names are the question ids used in the results.

Value

A `laya_questions` object: a named list of questions in the Python SDK's format.

See Also

`as_laya_questions()` to import a Python question set, and `laya_questions_to_json()` to export one.

Examples

```
qs <- laya_questions(
  department = laya_choice("Which department?", billing = "refunds", technical = "bugs"),
  urgency = laya_score("How urgent?", c("low", "high")),
  churn_risk = laya_noul("Does the user threaten to cancel?")
)
qs
```

laya_questions_to_json *Write a question set as JSON*

Description

The output is in the Python SDK's format.

Usage

```
laya_questions_to_json(questions, path = NULL, pretty = TRUE)
```

Arguments

questions	A laya_questions object.
path	Optional file path. If given, the JSON is written there.
pretty	Pretty-print the JSON?

Value

The JSON string (invisibly when path is given).

Examples

```
laya_questions_to_json(laya_questions(spam = laya_noul("Is this spam?")))
```

laya_remove_model *Remove an installed model*

Description

Deletes a checkpoint from layaR's model root, freeing about 1.3 GB. A model imported as a symlink only loses the link. Models in the shared cross-SDK cache are never touched.

Usage

```
laya_remove_model(
  checkpoint = c("multilingual", "english", "typed-decisions"),
  ask = interactive()
)
```

Arguments

checkpoint	Which checkpoint to remove.
ask	Ask for confirmation first? Defaults to TRUE in interactive sessions.

Value

TRUE if a model was removed, invisibly.

Examples

```
## Not run:
laya_remove_model("english")

## End(Not run)
```

`laya_result` *Laya results*

Description

A `laya_result` is a list with `model`, `answers` (one entry per question, in question order) and `usage` (token accounting and truncation flags), in the Python SDK's shape. Each answer has `type`, `confidence`, `answer_confidence` and `action$act_probability`, plus:

Usage

```
## S3 method for class 'laya_result'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

laya_probabilities(x)

laya_answer(x, id)

## S3 method for class 'laya_batch_result'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A `laya_result` or `laya_batch_result`.
`row.names`, `optional`	Ignored; part of the `as.data.frame()` generic.
`...`	Unused.
`id`	A question id.

Details

- choice: `choice` (the top label) and `probabilities` (per label);
- score: `score` (the expected level, from 0 to levels - 1), `legend` and `probabilities` (per level, keyed `"0"`, `"1"`, ...);

- noul: `noul` (the probability the statement holds).

With `min_confidence`, answers also carry `abstention` and `abstention_threshold`; from `laya_predict_long()`, `window`.

`as.data.frame()` gives one row per question; `laya_probabilities()` one row per option; `laya_answer()` extracts one answer. A `laya_batch_result` (from `laya_predict_batch()`) is a list of `laya_results` whose data-frame forms carry a `state_id` column.

Value

`as.data.frame()` returns a data frame with columns `id`, `type`, `answer` (the label, the score or `TRUE`/`FALSE`, as text), `value` (the chosen label's probability, the score, or P(true)), `confidence`, `answer_confidence`, `act_probability` and `abstention`. `laya_probabilities()` returns columns `id`, `type`, `option` and `probability` (for noul questions, options `"false"` and `"true"`).

Examples

```
if (laya_model_available()) {
  res <- laya_predict("Refund me now!", laya_preset("triage"))
  as.data.frame(res)
  laya_probabilities(res)
  laya_answer(res, "intent")$choice
}
```

---

laya_validate_questions

Validate a question set without a model

Description

Runs the Rust engine's own question validation, so errors surface before a model is loaded.

Usage

```
laya_validate_questions(questions)
```

Arguments

`questions`	A `laya_questions` object, or anything `as_laya_questions()` accepts.

Value

`questions` as a `laya_questions` object, invisibly. Errors with a `laya_error_invalid_question` (or `laya_error_not_supported`) condition.

Examples

```
laya_validate_questions(laya_questions(spam = laya_noul("Is this spam?")))
```

Index

as.data.frame(), *21*
as.data.frame.laya_batch_result (*laya_result*), 21
as.data.frame.laya_result (*laya_result*), 21
as_laya_questions, 2
as_laya_questions(), *3*, *15*, *19*, *22*

laya_annotate, 3
laya_answer (*laya_result*), 21
laya_bucket_keys, 4
laya_bucket_keys(), *15*
laya_choice (*laya_question*), 17
laya_choice(), *19*
laya_clean_email, 5
laya_clean_email(), *9*
laya_default_engine, 6
laya_default_engine(), *3*, *15*
laya_detect_language, 6
laya_download_model, 7
laya_download_model(), *13*
laya_email_state, 8
laya_email_state(), *5*
laya_engine, 9
laya_engine(), *3*, *12*, *15*
laya_engine_close (*laya_engine_info*), 11
laya_engine_close(), *10*
laya_engine_info, 11
laya_engine_info(), *10*
laya_import_model, 12
laya_import_model(), *8*, *13*
laya_model_available (*laya_model_root*), 12
laya_model_path (*laya_model_root*), 12
laya_model_path(), *7*, *10*
laya_model_root, 12
laya_models, 13
laya_models(), *8*, *13*
laya_noul (*laya_question*), 17
laya_noul(), *19*
laya_predict, 14
laya_predict(), *8–10*
laya_predict_batch (*laya_predict*), 14
laya_predict_batch(), *3*, *4*, *10*, *22*
laya_predict_long (*laya_predict*), 14
laya_predict_long(), *22*
laya_preset, 16
laya_preset(), *8*, *15*, *18*
laya_preset_fields (*laya_preset*), 16
laya_probabilities (*laya_result*), 21
laya_question, 17
laya_questions, 19
laya_questions(), *3*, *15*, *18*
laya_questions_to_json, 20
laya_questions_to_json(), *19*
laya_remove_model, 20
laya_result, *16*, 21
laya_score (*laya_question*), 17
laya_score(), *19*
laya_validate_questions, 22

predict.laya_engine (*laya_predict*), 14

stats::predict(), *16*