

Package: pnotp (via r-universe)

October 13, 2026

Title Client for the 'PnotP' Platform

Version 0.1.0

Description Access the 'PnotP' platform <<https://pnotp.ai>> from R and 'RStudio'. One account API key drives data-compliant large language model chat, where every response carries a retention receipt; model training with pause, resume and cancel; model deployments and live inference; and hosted agents. Includes 'RStudio' addins to chat with the language model from the IDE, and a bridge to 'ellmer' for tool calling and streaming.

License MIT + file LICENSE

URL <https://pnotp.ai>, <https://docs.pnotp.ai>

Depends R (>= 4.1.0)

Imports cli (>= 3.6.0), curl, graphics, httr2 (>= 1.1.0), jsonlite, rlang (>= 1.1.0), rstudioapi, utils

Suggests ellmer (>= 0.4.0), miniUI, shiny, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Vicente Alonso Gallardo Montalva [aut, cre], PnotP [cph, fnd]

Maintainer Vicente Alonso Gallardo Montalva <vicente@pnotp.ai>

Repository <https://cran.r-universe.dev>

Date/Publication 2026-10-13 14:30:02 UTC

RemoteUrl <https://github.com/cran/pnotp>

RemoteRef HEAD

RemoteSha 636a583bbe88065832d927395fb39cc8dc2ffce

Contents

plot.pnotp_job_progress	2
pnotp_account	3
pnotp_addins	4
pnotp_agent_answer	4
pnotp_agent_run	5
pnotp_agent_steps	6
pnotp_agents	6
pnotp_api_keys	7
pnotp_architectures	8
pnotp_chat	9
pnotp_client	10
pnotp_compute_options	11
pnotp_deploy	11
pnotp_deployment_keys	13
pnotp_deployments	13
pnotp_ellmer_chat	14
pnotp_errors	15
pnotp_has_key	16
pnotp_jobs	17
pnotp_llm_models	17
pnotp_llm_pricing	18
pnotp_models	18
pnotp_open_source_models	19
pnotp_predict	20
pnotp_projects	21
pnotp_request	22
pnotp_set_key	23
pnotp_toby	24
pnotp_tool	25
pnotp_train	25
pnotp_wait.pnotp_agent_run	27
Index	29

plot.pnotp_job_progress

Plot training metrics

Description

Draws each numeric per-epoch metric (train_loss, val_accuracy, val_rmse, ...) against the epoch, one panel per metric.

Usage

```
## S3 method for class 'pnotp_job_progress'
plot(x, ...)
```

Arguments

x A pnotp_job_progress from `pnotp_progress()`.
... Passed to `graphics::plot()`.

Value

x, invisibly.

pnotp_account	<i>Your account</i>
---------------	---------------------

Description

- `pnotp_me()` – who the key belongs to; the quickest "is my key working?" check.
- `pnotp_credits()` – credit balance and auto-refill state.
- `pnotp_capabilities()` – which platform features are switched on.

Usage

```
pnotp_me(client = NULL)

pnotp_credits(client = NULL)

pnotp_capabilities(client = NULL)
```

Arguments

client A `pnotp_client()`, or NULL to build one from the environment.

Value

The API response, as a list.

Examples

```
## Not run:
pnotp_me()$email
pnotp_credits()$balance_usd

## End(Not run)
```

pnotp_addins

RStudio addins

Description

Available from the 'RStudio' Addins menu once the package is installed (bind them to keys under Tools -> Modify Keyboard Shortcuts):

Usage

pnotp_addin_chat()

pnotp_addin_explain()

pnotp_addin_set_key()

Details

- **Chat with PnotP** (pnotp_addin_chat()) – a chat with the data-compliant LLM in the Viewer pane. Text selected in the editor is pre-filled as context. Each answer shows its receipt line. "Insert answer" writes the last answer on the line after your selection, never over it: its code blocks if it has any, otherwise the answer as # comments, so the script still runs. The conversation lives only in this R session.
- **Explain selection with PnotP** (pnotp_addin_explain()) – sends the selected code and prints an explanation in the console.
- **Set PnotP API key** (pnotp_addin_set_key()) – asks for the key (masked), sets it for the session and checks it against the API.

The chat needs the 'shiny' and 'miniUI' packages.

Value

pnotp_addin_chat() returns the last pnotp_chat_response (invisibly) when you press Done;
pnotp_addin_explain() returns the response invisibly.

pnotp_agent_answer

The final answer of an agent run

Description

A data-compliant run keeps a single final_answer event, which is preferred; otherwise the answer is the content of the last brain_turn.

Usage

pnotp_agent_answer(run, client = NULL)

Arguments

run	A pnotp_agent_run or run id.
client	A pnotp_client() , or NULL to build one from the environment.

Value

The answer as a string, or NULL if the run has none yet.

pnotp_agent_run	<i>Run a hosted agent</i>
-----------------	---------------------------

Description

Starts a run of the agent's current version. Follow it with [pnotp_wait\(\)](#), [pnotp_progress\(\)](#), [pnotp_agent_steps\(\)](#) and [pnotp_agent_answer\(\)](#).

Usage

```
pnotp_agent_run(
  agent,
  input,
  budget_usd = NULL,
  wait = FALSE,
  ...,
  client = NULL
)
```

Arguments

agent	An agent id or object.
input	The question: a string (sent as <code>list(input =)</code>), or a named list sent as-is.
budget_usd	Maximum spend for this run, in USD.
wait	If TRUE, block until the run finishes.
...	Passed to pnotp_wait() when <code>wait = TRUE</code> .
client	A pnotp_client() , or NULL to build one from the environment.

Value

A pnotp_agent_run.

pnotp_agent_steps	<i>Per-step meter of an agent run</i>
-------------------	---------------------------------------

Description

One row per brain turn or tool call, with its cost and latency.

Usage

```
pnotp_agent_steps(run, client = NULL)
```

Arguments

run	A pnotp_agent_run or run id.
client	A pnotp_client() , or NULL to build one from the environment.

Value

The API response; a list of steps comes back as a data.frame.

pnotp_agents	<i>Hosted agents</i>
--------------	----------------------

Description

A hosted agent is a brain (a reasoning model) driving a bounded tool-calling loop over the tools you attach, run on PnotP and metered against your credits. Create it, publish an immutable version with its model and tool allowlist, then run it with [pnotp_agent_run\(\)](#).

Usage

```
pnotp_agents(client = NULL)

pnotp_agent(agent, client = NULL)

pnotp_agent_create(name, client = NULL)

pnotp_agent_publish(
  agent,
  model = "pnotp-compliant",
  system_prompt = "",
  tools = character(),
  budget_usd = NULL,
  client = NULL
)
```

Arguments

client	A <code>pnotp_client()</code> , or NULL to build one from the environment.
agent	An agent id or object.
name	Agent name.
model	The brain, e.g. "pnotp-compliant".
system_prompt	The system prompt.
tools	Character vector of tool names to allow.
budget_usd	Default per-run budget, in USD.

Details

With `model = "pnotp-compliant"` a run keeps no question, no reasoning and no loop state: only metered metadata and the encrypted final answer, held for 24 hours.

Value

`pnotp_agents()` returns a data.frame; the others the API response as a list.

Examples

```
## Not run:
a <- pnotp_agent_create("Advisor")
pnotp_agent_publish(a, model = "pnotp-compliant", tools = "programs_db")
run <- pnotp_agent_run(a, "Which programs fit someone who loves maths?", wait = TRUE)
pnotp_agent_answer(run)

## End(Not run)
```

pnotp_api_keys	<i>Account API keys</i>
----------------	-------------------------

Description

Manage the `pnotp_sk_...` keys that authenticate this package. The raw key is returned only once, by `pnotp_api_key_create()`.

Usage

```
pnotp_api_keys(client = NULL)

pnotp_api_key_create(name = "API key", client = NULL)

pnotp_api_key_revoke(key, client = NULL)
```

Arguments

client	A <code>pnotp_client()</code> , or NULL to build one from the environment.
name	A label for the new key.
key	A key id or object (not the raw key).

Value

`pnotp_api_keys()` returns a data.frame; the others the API response.

`pnotp_architectures` *Architecture graphs for common models*

Description

Builders for the visual architecture graph `pnotp_model_create()` expects, so the package is usable end to end without hand-writing nodes and edges. Any hand-built graph (a nested list) works too.

Usage

```
pnotp_arch_image_classifier(num_classes = 2, channels = 3, size = 224)

pnotp_arch_tabular_classifier(n_features, num_classes = 2)
```

Arguments

num_classes	Number of classes.
channels	Image channels.
size	Image side length, in pixels.
n_features	Number of input features (columns, excluding the target).

Value

A nested list.

Examples

```
str(pnotp_arch_tabular_classifier(n_features = 4, num_classes = 3), max.level = 2)
```

pnotp_chat

*Chat with the PnotP data-compliant LLM***Description**

Calls POST /v1/llm/chat. PnotP itself stores neither the prompt nor the completion; the returned receipt is the durable audit artifact. Whether the *whole* path is zero-retention depends on the rail that served the call, and the receipt says so: read receipt\$zero_retention and receipt\$retention_basis rather than assuming it.

Usage

```
pnotp_chat(
  x,
  max_tokens = 256,
  temperature = 0.7,
  model = NULL,
  tools = NULL,
  tool_choice = NULL,
  file = NULL,
  image = NULL,
  audio = NULL,
  client = NULL
)
```

Arguments

x	The conversation: a single string (one user turn), a list of messages (list(role = , content =)), or a data.frame with role and content columns.
max_tokens	Maximum tokens to generate (1 to 8192).
temperature	Sampling temperature (0 to 2).
model	A public model id (see pnotp_llm_models()); NULL uses the default. A named model is never silently substituted.
tools	Tool definitions, e.g. from pnotp_tool() . When the model calls one, the response carries tool_calls.
tool_choice	OpenAI-style tool choice ("auto", "none", or a list naming a function).
file	A document to attach (PDF, txt, md, csv, json...): a path, a raw vector, or a base64 string. Parsed to text server-side.
image	An image to attach: a path, a raw vector, a base64 string, or a data:/http(s) URL. The default model sees it visually.
audio	Accepted for parity with the Python SDK; the API answers 422 until audio is enabled.
client	A pnotp_client() , or NULL to build one from the environment.

Details

Calls are metered per token against your credits; an empty balance raises `pnotp_insufficient_credits_error`. See [pnotp_llm_pricing\(\)](#) for rates.

For multi-turn tool calling and streaming, [pnotp_ellmer_chat\(\)](#) drives the same models through 'ellmer'.

Value

A `pnotp_chat_response`: a list with completion, usage, model, receipt and, when the model called a tool, `tool_calls`. `as.character()` gives the completion text.

Examples

```
## Not run:
out <- pnotp_chat("In one sentence, what is a p-value?")
out
out$receipt$zero_retention

pnotp_chat(data.frame(
  role = c("system", "user"),
  content = c("Answer in Spanish.", "What does dplyr::across() do?")
))

pnotp_chat("What is in this plot?", image = "figure.png")

## End(Not run)
```

pnotp_client

Create a PnotP client

Description

A client holds the account API key, the API base URL and the request timeout. You rarely need one: every function takes `client = NULL`, which builds a client from the environment at call time, so a key set with [pnotp_set_key\(\)](#) (or in `~/ .Renvirom`) is picked up without restarting.

Usage

```
pnotp_client(api_key = NULL, base_url = NULL, timeout = 60)
```

Arguments

<code>api_key</code>	Account API key (<code>pnotp_sk...</code>). Defaults to the <code>PNOTP_API_KEY</code> environment variable.
<code>base_url</code>	API base URL, ending in <code>/v1</code> . Defaults to the <code>PNOTP_BASE_URL</code> environment variable, then <code>https://api.pnotp.ai/v1</code> .
<code>timeout</code>	Request timeout, in seconds.

Details

A client created without a key can still call the public endpoints, such as `pnotp_llm_pricing()`. Authenticated calls fail with a `pnotp_error` that says how to set one.

Value

An object of class `pnotp_client`. Printing it masks the key.

Examples

```
cl <- pnotp_client(api_key = "pnotp_sk_example", base_url = "http://localhost:8000/v1")
cl
```

`pnotp_compute_options` *Training compute options*

Description

CPU (free) and each GPU tier with its price. Pass a choice to `pnotp_train(compute = , gpu = )`.

Usage

```
pnotp_compute_options(client = NULL)
```

Arguments

`client` A `pnotp_client()`, or `NULL` to build one from the environment.

Value

A list.

`pnotp_deploy` *Deploy a model as a live endpoint*

Description

- `pnotp_deploy_upload()` deploys a TorchScript .pt checkpoint.
- `pnotp_deploy_open_source()` deploys a catalog model (see `pnotp_open_source_models()`) with no training and no upload; extra arguments set its default inference parameters.

Usage

```

pnotp_deploy_upload(
  project,
  name,
  checkpoint,
  compliant = FALSE,
  client = NULL
)

pnotp_deploy_open_source(
  project,
  model_id,
  name = NULL,
  compliant = FALSE,
  ...,
  client = NULL
)

```

Arguments

project	A project id or object.
name	Deployment name.
checkpoint	Path to a TorchScript .pt file, or a raw vector.
compliant	Request zero-retention (data-compliant) inference.
client	A pnotp_client() , or NULL to build one from the environment.
model_id	Catalog model id, e.g. "mobilenet_v3_small".
...	Default inference parameters, validated server-side against the catalog spec (e.g. confidence_threshold = 0.5).

Details

compliant = TRUE makes the endpoint data-compliant: the platform never persists request inputs, keeps error text out of monitoring, and every [pnotp_predict\(\)](#) response carries a compliance receipt. It is a platform capability: when it is not enabled the call fails with a 409 (pnotp_api_error); check first with [pnotp_compliance_available\(\)](#).

Value

The API response: for an upload, list(deployment = , board =). Pass it straight to [pnotp_predict\(\)](#).

Examples

```

## Not run:
dep <- pnotp_deploy_upload(p, "prod", "model.pt", compliant = TRUE)
pnotp_predict(dep, "xray.jpg")

det <- pnotp_deploy_open_source(p, "yolox_s", confidence_threshold = 0.4)

```

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

pnotp_deployment_keys *Per-deployment inference keys*

Description

Mint, list and revoke pnp_... keys that let an external consumer call one deployment with `pnotp_predict()`. The raw key is returned only once, by `pnotp_deployment_key_create()`.

Usage

```
pnotp_deployment_keys(deployment, client = NULL)

pnotp_deployment_key_create(deployment, client = NULL)

pnotp_deployment_key_revoke(deployment, key_id, client = NULL)
```

Arguments

deployment	A deployment id or object.
client	A <code>pnotp_client()</code> , or NULL to build one from the environment.
key_id	The key's id (not the key itself).

Value

The API response, as a list.

pnotp_deployments *Manage deployments*

Description

Manage deployments

Usage

```
pnotp_deployments(project, client = NULL)

pnotp_deployment(deployment, client = NULL)

pnotp_deployment_stats(deployment, window = "24h", client = NULL)

pnotp_deployment_pause(deployment, client = NULL)
```

```

pnotp_deployment_resume(deployment, client = NULL)

pnotp_deployment_delete(deployment, client = NULL)

pnotp_deployment_set_compliant(deployment, compliant = TRUE, client = NULL)

pnotp_compliance_available(client = NULL)

pnotp_private_available(client = NULL)

```

Arguments

project	A project id or object.
client	A pnotp_client() , or NULL to build one from the environment.
deployment	A deployment id or object.
window	Stats window, e.g. "24h".
compliant	TRUE to turn data-compliant inference on (needs the platform capability; 409 otherwise), FALSE to turn it off.

Value

pnotp_deployments() returns a data.frame; the others the API response as a list. pnotp_compliance_available() and pnotp_private_available() return TRUE/FALSE, and FALSE when the capability probe itself fails.

pnotp_ellmer_chat	<i>Use PnotP models through 'ellmer'</i>
-------------------	--

Description

Returns an 'ellmer' chat object pointed at PnotP's OpenAI-compatible endpoint (POST /v1/chat/completions), authenticated with your account key. Everything 'ellmer' does then works against PnotP: multi-turn chat, streaming, tool calling with R functions (chat\$register_tool()), and structured output (chat\$chat_structured()).

Usage

```

pnotp_ellmer_chat(
  model = "pnotp-compliant",
  system_prompt = NULL,
  ...,
  client = NULL
)

```

Arguments

model	A public model id (see pnotp_llm_models()).
system_prompt	Optional system prompt.
...	Passed on to <code>ellmer::chat_openai_compatible()</code> , e.g. <code>params</code> or <code>echo</code> .
client	A pnotp_client() , or NULL to build one from the environment.

Details

The same retention terms as [pnotp_chat\(\)](#) apply: PnotP stores neither prompt nor completion. 'ellmer' keeps the conversation in your R session only. The per-call receipt the endpoint adds to its response is not surfaced by 'ellmer'; use [pnotp_chat\(\)](#) when you need to keep receipts.

Value

An 'ellmer' Chat object.

Examples

```
## Not run:
chat <- pnotp_ellmer_chat(system_prompt = "You are a terse R expert.")
chat$chat("How do I pivot a data.frame longer?")

# Tool calling with an R function
chat$register_tool(ellmer::tool(
  function() format(Sys.time()),
  name = "now",
  description = "Current date and time."
))
chat$chat("What time is it?")

## End(Not run)
```

pnotp_errors

Errors raised by pnotp

Description

Every condition raised by the package inherits from `pnotp_error`. HTTP failures inherit from `pnotp_api_error` and carry the fields the backend publishes, so you never have to parse the message text:

Details

- `status` – the HTTP status code.
- `detail` – the backend's detail (a string, or a list for validation errors).
- `code` – a stable enum on LLM endpoints, e.g. `"context_length_exceeded"`, `"rate_limit_exceeded"`, `"model_warming"`; NULL when the endpoint does not publish one.

- `retryable` – FALSE means retrying cannot succeed; NULL if unknown.
- `param` – for parameter errors, the offending field.
- `info` – the full error object (e.g. `max_context`, `tokens_in`).

Class	When
<code>pnotp_auth_error</code>	401 / 403: missing, invalid or revoked key, or not your resource
<code>pnotp_insufficient_credits_error</code>	402: the account is out of credits
<code>pnotp_not_found_error</code>	404: the resource does not exist (or you cannot see it)
<code>pnotp_api_error</code>	any other non-2xx response
<code>pnotp_connection_error</code>	the API could not be reached
<code>pnotp_timeout_error</code>	<code>pnotp_wait()</code> ran out of time

Examples

```
## Not run:
tryCatch(
  pnotp_chat("hello"),
  pnotp_insufficient_credits_error = function(e) message("Add credits in Studio"),
  pnotp_api_error = function(e) if (isTRUE(e$retryable)) message("retry later")
)

## End(Not run)
```

<code>pnotp_has_key</code>	<i>Is a PnotP API key available?</i>
----------------------------	--------------------------------------

Description

Is a PnotP API key available?

Usage

```
pnotp_has_key(client = NULL)
```

Arguments

`client` A `pnotp_client()`, or NULL to build one from the environment.

Value

TRUE when the client (by default, the environment) has a key.

pnotp_jobs	<i>Training jobs</i>
------------	----------------------

Description

List, fetch and control training jobs. Pausing or cancelling a running job is cooperative: the job stops at its next epoch boundary.

Usage

```
pnotp_jobs(client = NULL)

pnotp_job(job, client = NULL)

pnotp_job_pause(job, client = NULL)

pnotp_job_resume(job, client = NULL)

pnotp_job_cancel(job, client = NULL)
```

Arguments

client	A pnotp_client() , or NULL to build one from the environment.
job	A job id or a pnotp_job.

Value

pnotp_jobs() returns a data.frame; the others a pnotp_job.

pnotp_llm_models	<i>Models the LLM endpoints serve</i>
------------------	---------------------------------------

Description

Calls the OpenAI-compatible GET /v1/models (authenticated).

Usage

```
pnotp_llm_models(client = NULL)
```

Arguments

client	A pnotp_client() , or NULL to build one from the environment.
--------	---

Value

A data.frame with one row per model: id, description, vision, default, self_hosted, ...

pnotp_llm_pricing	<i>Public LLM price list</i>
-------------------	------------------------------

Description

Calls GET /v1/llm/pricing, which needs no API key. Rates are per million tokens; metering_enabled says whether calls are billed today.

Usage

```
pnotp_llm_pricing(client = NULL)
```

Arguments

client A [pnotp_client\(\)](#), or NULL to build one from the environment.

Value

A pnotp_pricing list (the raw response). Its print method shows one row per model in USD per million tokens.

Examples

```
pnotp_llm_pricing()
```

pnotp_models	<i>Model records</i>
--------------	----------------------

Description

A model record is a versioned model definition: an architecture graph, a dataset reference and a training method. Train it with [pnotp_train\(\)](#).

Usage

```
pnotp_models(project, client = NULL)
```

```
pnotp_model(model, client = NULL)
```

```
pnotp_model_create(
  project,
  name,
  architecture,
  description = "",
  dataset_reference = NULL,
```

```

    training_method = NULL,
    client = NULL
  )

  pnotp_model_add_version(
    model,
    architecture,
    dataset_reference = NULL,
    training_method = NULL,
    client = NULL
  )

  pnotp_model_versions(model, client = NULL)

```

Arguments

project	A project id or object.
client	A pnotp_client() , or NULL to build one from the environment.
model	A model record id or object.
name	Model name.
architecture	The architecture graph, e.g. from pnotp_arch_image_classifier() .
description	Free-text description.
dataset_reference, training_method	Optional named lists, stored on the version.

Value

pnotp_model_create() and pnotp_model() return a pnotp_model_record; pnotp_models() and pnotp_model_versions() return data.frames; pnotp_model_add_version() returns the version.

Examples

```

## Not run:
m <- pnotp_model_create(p, "cnn-v1", pnotp_arch_image_classifier(num_classes = 2))
pnotp_models(p)

## End(Not run)

```

pnotp_open_source_models

The open-source model catalog

Description

Each entry carries its inference parameter spec and an available flag. The API answers 404 while the catalog is switched off.

Usage

```
pnotp_open_source_models(client = NULL)
```

Arguments

client A [pnotp_client\(\)](#), or NULL to build one from the environment.

Value

A data.frame.

pnotp_predict	<i>Run inference on a deployment</i>
---------------	--------------------------------------

Description

Uploads file to POST /v1/deployments/{id}/predict. By default the call uses your account key. Pass a per-deployment key (pnp_...) as api_key to call as an external consumer; no account key is needed then.

Usage

```
pnotp_predict(deployment, file, ..., api_key = NULL, client = NULL)
```

Arguments

deployment A deployment id, or the object returned by [pnotp_deploy_upload\(\)](#) / [pnotp_deployment\(\)](#).

file Path to the input file (e.g. an image), or a raw vector.

... Per-request inference parameter overrides for catalog deployments, e.g. confidence_threshold = 0.5.

api_key A per-deployment key, used instead of the account key.

client A [pnotp_client\(\)](#), or NULL to build one from the environment.

Details

On a compliant deployment the response carries a compliance block (compliant, retention, deployment_id, timestamp); keep it as the audit artifact.

Private (MPC) inference is not available from R yet; see the README.

Value

The prediction, as a list (e.g. predictions, class_names).

Examples

```
## Not run:
pnotp_predict("dep_123", "xray.jpg")
pnotp_predict(dep, "street.jpg", confidence_threshold = 0.5)
pnotp_predict(dep, "xray.jpg", api_key = Sys.getenv("PNOTP_DEPLOYMENT_KEY"))

## End(Not run)
```

pnotp_projects	<i>Projects</i>
----------------	-----------------

Description

A project groups model records and deployments.

Usage

```
pnotp_projects(client = NULL)

pnotp_project(project, client = NULL)

pnotp_project_create(
  name,
  task_type = "binary_image_classification",
  description = "",
  architecture = NULL,
  ...,
  client = NULL
)
```

Arguments

client	A pnotp_client() , or NULL to build one from the environment.
project	A project id, or an object returned by these functions.
name	Project name.
task_type	Task type, e.g. "binary_image_classification".
description	Free-text description.
architecture	Optional base architecture id (e.g. "resnet18"). The per-model network graph lives on model records; see pnotp_model_create() .
...	Extra fields sent with the create request.

Value

pnotp_projects() returns a data.frame; the others return the project as a list.

Examples

```
## Not run:
p <- pnotp_project_create("Chest X-ray", task_type = "binary_image_classification")
pnotp_projects()
pnotp_project(p)

## End(Not run)
```

pnotp_request

Call any PnotP API route

Description

The low-level escape hatch every other function is built on. It sends the account key both as Authorization: Bearer (the management API) and as X-API-Key (inference), so one key authenticates every route, and turns any HTTP error into a typed condition (see [pnotp_errors](#)).

Usage

```
pnotp_request(
  method,
  path,
  query = NULL,
  body = NULL,
  multipart = NULL,
  auth = TRUE,
  api_key = NULL,
  client = NULL
)
```

Arguments

method	HTTP method, e.g. "GET".
path	Route relative to the base URL, e.g. "projects".
query	Named list of query parameters.
body	A list, sent as a JSON body. Length-one vectors become JSON scalars; wrap a vector in I() or as.list() to force an array.
multipart	Named list of multipart fields: strings, or curl::form_file() objects.
auth	Send the API key? FALSE for public routes.
api_key	Use this key instead of the client's for this call (used for per-deployment pnp_... keys).
client	A pnotp_client() , or NULL to build one from the environment.

Value

The parsed JSON body as a list, a raw vector for non-JSON responses, or NULL for an empty body.

Examples

```
## Not run:
pnotp_request("GET", "platform/capabilities")

## End(Not run)
```

pnotp_set_key	<i>Set the PnotP API key for this R session</i>
---------------	---

Description

Stores the key in the PNOTP_API_KEY environment variable of the current session. Called with no argument in an interactive session, it asks for the key without echoing it (a masked dialog in 'RStudio').

Usage

```
pnotp_set_key(api_key = NULL)
```

Arguments

api_key An account API key (pnotp_sk_...), or NULL to be asked.

Details

To keep the key across sessions, add PNOTP_API_KEY=pnotp_sk_... to ~/.Renvirom, for example with `usethis::edit_r_environ()`. This function never writes that file itself.

Value

TRUE, invisibly.

Examples

```
## Not run:
pnotp_set_key()            # asks, masked
pnotp_set_key("pnotp_sk_...")

## End(Not run)
```

pnotp_toby

Ask Toby, PnotP's hosted assistant

Description

Asks a question in plain English about the platform, your deployments, or an error you just hit. Toby needs an API key and credits.

Usage

```
pnotp_toby(question, history = NULL, client = NULL)
```

Arguments

question	Your question.
history	A previous pnotp_toby_answer (to continue that conversation), or a list of <code>list(role = , content =)</code> messages.
client	A <code>pnotp_client()</code> , or NULL to build one from the environment.

Details

Like the Python SDK's `pnotp.help.toby()`, this never raises: without a key, without credits, or without a network, it returns an offline answer (`offline = TRUE`) that says why and points at the local help.

Toby is told the question comes from this R package. Its built-in code reference is the Python SDK's, so check function names against the R help (`?pnotp_chat`) when it shows code.

Value

A `pnotp_toby_answer`: a list with `text`, `offline`, `reason`, `proposed_action`, and `messages` (the conversation so far, to pass as `history`).

Examples

```
## Not run:
a <- pnotp_toby("How do I deploy a model and call it?")
a
pnotp_toby("And how do I make it data-compliant?", history = a)

tryCatch(pnotp_chat("hi"), pnotp_error = function(e) pnotp_toby(conditionMessage(e)))

## End(Not run)
```

pnotp_tool

Define a tool the LLM can call

Description

Builds an OpenAI-style function tool for `pnotp_chat()`. It takes care of the two JSON shapes R gets wrong by default: an empty properties is sent as `{}` (not `[]`), and required is always an array, even with one name.

Usage

```
pnotp_tool(name, description, properties = list(), required = character())
```

Arguments

name	Function name.
description	What the tool does; the model reads this.
properties	Named list of JSON Schema properties, e.g. <code>list(city = list(type = "string", description = "City name"))</code> .
required	Character vector of required property names.

Value

A list ready to pass in `tools = list(...)`.

Examples

```
weather <- pnotp_tool(
  "get_weather", "Current weather for a city.",
  properties = list(city = list(type = "string")),
  required = "city"
)
str(weather)
```

pnotp_train

Train a model

Description

Submits a training run for a model record. Image models train on a .zip of class-named folders; tabular models train on a CSV, and you can pass a data.frame directly (it is written to a temporary CSV, uploaded, and deleted). Modality is detected from the model's architecture.

Usage

```
pnotp_train(
  model,
  dataset,
  version_id = NULL,
  epochs = 3,
  batch_size = 16,
  lr = 0.001,
  image_size = 64,
  target_column = "target",
  compute = NULL,
  gpu = NULL,
  wait = FALSE,
  client = NULL
)
```

Arguments

model	A model record id or object (from pnotp_model_create()).
dataset	A path to a .zip / .csv, a raw vector, or a data.frame.
version_id	Version to train; NULL uses the record's current version.
epochs, batch_size, lr, image_size	Training hyperparameters.
target_column	For tabular data, the column holding the labels.
compute	"cpu" (free) or "gpu" (billed per second of training); NULL uses the platform default. See pnotp_compute_options() .
gpu	GPU tier, e.g. "auto" or "24gb".
wait	If TRUE, block until the job finishes (see pnotp_wait()).
client	A pnotp_client() , or NULL to build one from the environment.

Value

A pnotp_job.

Examples

```
## Not run:
m <- pnotp_model_create(p, "iris", pnotp_arch_tabular_classifier(n_features = 4, num_classes = 3))
job <- pnotp_train(m, iris, target_column = "Species", epochs = 20)
job <- pnotp_wait(job)
plot(pnotp_progress(job))

## End(Not run)
```

`pnotp_wait.pnotp_agent_run`*Wait for, refresh, or inspect a long-running object*

Description

Generics for training jobs (`pnotp_train()`) and hosted agent runs (`pnotp_agent_run()`).

Usage

```
## S3 method for class 'pnotp_agent_run'
pnotp_wait(
  x,
  poll_interval = 1,
  timeout = 120,
  on_progress = NULL,
  quiet = FALSE,
  client = NULL,
  ...
)
```

```
## S3 method for class 'pnotp_agent_run'
pnotp_refresh(x, client = NULL, ...)
```

```
## S3 method for class 'pnotp_agent_run'
pnotp_progress(x, client = NULL, ...)
```

```
pnotp_wait(x, ...)
```

```
pnotp_refresh(x, ...)
```

```
pnotp_progress(x, ...)
```

```
## S3 method for class 'pnotp_job'
pnotp_wait(
  x,
  poll_interval = 2,
  timeout = NULL,
  on_progress = NULL,
  quiet = FALSE,
  client = NULL,
  ...
)
```

```
## S3 method for class 'pnotp_job'
pnotp_refresh(x, client = NULL, ...)
```

```
## S3 method for class 'pnotp_job'  
pnotp_progress(x, client = NULL, ...)
```

Arguments

<code>x</code>	A <code>pnotp_job</code> or <code>pnotp_agent_run</code> .
<code>poll_interval</code>	Seconds between polls.
<code>timeout</code>	Give up after this many seconds (NULL waits forever for jobs; agent runs default to 120). Raises <code>pnotp_timeout_error</code> .
<code>on_progress</code>	Optional function called with the refreshed object after every poll.
<code>quiet</code>	Suppress status messages.
<code>client</code>	A pnotp_client() , or NULL to build one from the environment.
<code>...</code>	Unused.

Details

- `pnotp_refresh()` re-fetches the object.
- `pnotp_wait()` polls until the status is succeeded, failed or cancelled, reporting status changes. Press Esc to stop waiting; the job itself keeps running.
- `pnotp_progress()` returns the live progress snapshot. For a training job, `epochs` is a `data.frame` with one row per finished epoch, and `plot()` draws its metrics.

Value

The refreshed object (`pnotp_wait()`, `pnotp_refresh()`), or a progress list (`pnotp_progress()`).

Index

`as.list()`, 22

`curl::form_file()`, 22

`graphics::plot()`, 3

`I()`, 22

`plot.pnotp_job_progress`, 2

`pnotp_account`, 3

`pnotp_addin_chat (pnotp_addins)`, 4

`pnotp_addin_explain (pnotp_addins)`, 4

`pnotp_addin_set_key (pnotp_addins)`, 4

`pnotp_addins`, 4

`pnotp_agent (pnotp_agents)`, 6

`pnotp_agent_answer`, 4

`pnotp_agent_answer()`, 5

`pnotp_agent_create (pnotp_agents)`, 6

`pnotp_agent_publish (pnotp_agents)`, 6

`pnotp_agent_run`, 5

`pnotp_agent_run()`, 6, 27

`pnotp_agent_steps`, 6

`pnotp_agent_steps()`, 5

`pnotp_agents`, 6

`pnotp_api_key_create (pnotp_api_keys)`, 7

`pnotp_api_key_revoke (pnotp_api_keys)`, 7

`pnotp_api_keys`, 7

`pnotp_arch_image_classifier`
(`pnotp_architectures`), 8

`pnotp_arch_image_classifier()`, 19

`pnotp_arch_tabular_classifier`
(`pnotp_architectures`), 8

`pnotp_architectures`, 8

`pnotp_capabilities (pnotp_account)`, 3

`pnotp_chat`, 9

`pnotp_chat()`, 15, 25

`pnotp_client`, 10

`pnotp_client()`, 3, 5–9, 11–22, 24, 26, 28

`pnotp_compliance_available`
(`pnotp_deployments`), 13

`pnotp_compliance_available()`, 12

`pnotp_compute_options`, 11

`pnotp_compute_options()`, 26

`pnotp_credits (pnotp_account)`, 3

`pnotp_deploy`, 11

`pnotp_deploy_open_source`
(`pnotp_deploy`), 11

`pnotp_deploy_upload (pnotp_deploy)`, 11

`pnotp_deploy_upload()`, 20

`pnotp_deployment (pnotp_deployments)`, 13

`pnotp_deployment()`, 20

`pnotp_deployment_delete`
(`pnotp_deployments`), 13

`pnotp_deployment_key_create`
(`pnotp_deployment_keys`), 13

`pnotp_deployment_key_revoke`
(`pnotp_deployment_keys`), 13

`pnotp_deployment_keys`, 13

`pnotp_deployment_pause`
(`pnotp_deployments`), 13

`pnotp_deployment_resume`
(`pnotp_deployments`), 13

`pnotp_deployment_set_compliant`
(`pnotp_deployments`), 13

`pnotp_deployment_stats`
(`pnotp_deployments`), 13

`pnotp_deployments`, 13

`pnotp_ellmer_chat`, 14

`pnotp_ellmer_chat()`, 10

`pnotp_errors`, 15, 22

`pnotp_has_key`, 16

`pnotp_job (pnotp_jobs)`, 17

`pnotp_job_cancel (pnotp_jobs)`, 17

`pnotp_job_pause (pnotp_jobs)`, 17

`pnotp_job_resume (pnotp_jobs)`, 17

`pnotp_jobs`, 17

`pnotp_llm_models`, 17

`pnotp_llm_models()`, 9, 15

`pnotp_llm_pricing`, 18

`pnotp_llm_pricing()`, [10](#), [11](#)
`pnotp_me (pnotp_account)`, [3](#)
`pnotp_model (pnotp_models)`, [18](#)
`pnotp_model_add_version (pnotp_models)`,
 [18](#)
`pnotp_model_create (pnotp_models)`, [18](#)
`pnotp_model_create()`, [8](#), [21](#), [26](#)
`pnotp_model_versions (pnotp_models)`, [18](#)
`pnotp_models`, [18](#)
`pnotp_open_source_models`, [19](#)
`pnotp_open_source_models()`, [11](#)
`pnotp_predict`, [20](#)
`pnotp_predict()`, [12](#), [13](#)
`pnotp_private_available`
 (`pnotp_deployments`), [13](#)
`pnotp_progress`
 (`pnotp_wait.pnotp_agent_run`),
 [27](#)
`pnotp_progress()`, [3](#), [5](#)
`pnotp_project (pnotp_projects)`, [21](#)
`pnotp_project_create (pnotp_projects)`,
 [21](#)
`pnotp_projects`, [21](#)
`pnotp_refresh`
 (`pnotp_wait.pnotp_agent_run`),
 [27](#)
`pnotp_request`, [22](#)
`pnotp_set_key`, [23](#)
`pnotp_set_key()`, [10](#)
`pnotp_toby`, [24](#)
`pnotp_tool`, [25](#)
`pnotp_tool()`, [9](#)
`pnotp_train`, [25](#)
`pnotp_train()`, [18](#), [27](#)
`pnotp_wait`
 (`pnotp_wait.pnotp_agent_run`),
 [27](#)
`pnotp_wait()`, [5](#), [16](#), [26](#)
`pnotp_wait.pnotp_agent_run`, [27](#)