Package 'screenllm'

September 24, 2026

Title LLM-Assisted Title/Abstract Screening for Systematic Reviews

Version 0.1.0

Description A turn-key workflow for LLM-assisted systematic-review screening. The package ranks a corpus of titles and abstracts with an ensemble of open-source large language models served locally by 'Ollama', then applies the SAFE stopping rule to identify the records a human should screen. Defaults match the four-LLM mean ensemble and the SAFE configuration recommended by Spillias et al. (2026). A companion 'Shiny' app walks the human reviewer through the records above the stopping point. Complementary to the 'AIscreenR' package of Vembye et al. (2025) <doi:10.1037/met0000769>, which targets cloud-hosted 'GPT' models via the 'OpenAI' API; 'screenllm' targets locally-served open-weights ensembles with an integrated stopping rule.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports cli, digest, fs, jsonlite, tibble, dplyr, rlang, httr2, glue

Suggests shiny, bslib, DT, callr, later, writexl, readxl, testthat (>= 3.0.0), knitr, rmarkdown, readr, withr, mockery, stringdist

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/s-spillias/screenllm

BugReports https://github.com/s-spillias/screenllm/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Scott Spillias [aut, cre],
Laura Avila Turriago [aut],
Christopher Brown [aut],
Ariane Easton [aut],
Jack Roberts [aut],
Michael Sievers [aut],

Steve Swearer [aut],
Andrew Taylor [aut],
Brigette Wright [aut],
Valeriya Komyakova [aut]

Maintainer Scott Spillias <scott.spillias@csiro.au>

Repository CRAN

Date/Publication 2026-09-24 15:00:02 UTC

Contents

audit_disagreements *Surface strong LLM-human disagreements for a manual audit*

Description

Returns records where the LLM ensemble and the human reviewer disagree at high confidence. On one of the benchmark reviews reported in the paper, a similar audit caught genuine screener errors in 28\ disagreements. This is intended as a low-cost quality-control step after the main screen.

Usage

```
audit_disagreements(
  ranked,
  decisions,
  strong_fp_score = 70,
  strong_fn_score = 30
)
```

Arguments

ranked	A ranking (output of rank_records()).
decisions	A tibble with columns id and human_decision.
strong_fp_score	LLM score at or above which an accept is "confident" (default 70).
strong_fn_score	LLM score below which a reject is "confident" (default 30).

Value

A tibble of disagreement rows, one per record.

backend_mock *Mock backend for tests*

Description

Returns deterministic scores derived from a digest::digest hash of the prompt. Never hits the network. Useful for unit tests and for package-development end-to-end runs without needing Ollama.

Usage

```
backend_mock(base = 50L)
```

Arguments

base	Baseline score (integer 0-100). Added to a small prompt-derived variation.

Value

A backend object.

Examples

```
b <- backend_mock()
b$score_record("any-model", "any prompt", temperature = 0)
```

---

backend_ollama *Ollama backend*

Description

Talks to a locally-running Ollama server over HTTP. Each call sends one record's prompt to one model at the configured temperature, parses the JSON response, and returns the numeric relevance score plus the LLM's free-text explanation.

Usage

```
backend_ollama(
  ollama_url = getOption("screenllm.ollama_url"),
  timeout = 600,
  max_retries = 3L
)
```

Arguments

ollama_url	Base URL of the Ollama server.
timeout	Per-call timeout in seconds.
max_retries	Retries on transient errors.

Value

A backend object.

build_prompt *Build a screening prompt for one record*

Description

Renders the standard prompt template shipped in `inst/prompts/standard.txt` with the criteria and the record's title/abstract substituted in. Users can inspect the returned prompt string to confirm the LLM is being asked the right question, and power users can wrap or replace this function.

Usage

```
build_prompt(criteria, record)
```

Arguments

`criteria`	A `screenllm_criteria` object.
`record`	A one-row data frame or named list with `id`, `title`, `abstract`.

Value

A character string ready to be sent as an LLM prompt.

check_setup *Verify local setup for* `screenllm`

Description

Confirms that Ollama is reachable at the configured URL, that R can talk to it, and (optionally) that the four default models are available. Returns `TRUE` invisibly on success and prints a diagnostic panel; returns `FALSE` on failure with actionable messages.

Usage

```
check_setup(
  models = .PINNED_DEFAULT_MODELS,
  ollama_url = getOption("screenllm.ollama_url")
)
```

Arguments

`models`	Character vector of model tags to require. Defaults to the four LLMs pinned by `default_ensemble()`. Pass `NULL` to skip the model-availability check.
`ollama_url`	Base URL of the local Ollama server. Defaults to the `screenllm.ollama_url` option (usually `http://localhost:11434`).

Value

Invisible logical.

Examples

```
check_setup()
check_setup(models = NULL) # just check the server, not the models
```

`clear_cache` *Clear cached LLM scores for a project*

Description

Removes score-cache files under a project's cache directory. Use this when a model returned garbage during a ranking run and you want to re-score with fresh calls; the cache would otherwise be hit on the next `rank_records()` invocation.

Usage

```
clear_cache(project, model = NULL, delete_ranked = TRUE)
```

Arguments

`project`	Project name.
`model`	Optional Ollama tag. When `NULL` (default), clears every cached score in the project. When supplied, only cached scores whose stored `$model` field matches are removed. Reading each cache file is O(n_files) but each file is tiny (~1 KB).
`delete_ranked`	If `TRUE` (default), also removes the project's `ranked.rds` artefact so a re-run starts from a clean slate.

Details

By default clears every cached score (the most conservative "start over" behaviour). Pass `model = "..."` to clear only that model's cached scores across all records / replicates. Optionally also deletes the persisted ranked artefact (`delete_ranked = TRUE`) so a stale ranking doesn't linger.

Value

Invisibly, the number of cache files removed.

Examples

```
# Operate against a throwaway data directory so the example never
# touches your real projects.
withr::with_envvar(c(R_USER_DATA_DIR = tempfile("screenllm-")), {
  # Clear one model's cached scores (a no-op on an empty project):
  clear_cache("demo-project", model = "mistral:7b")
  # Clear everything for the project:
  clear_cache("demo-project")
})
```

`custom_ensemble` *Define a custom LLM ensemble*

Description

Lets the user swap in a different set of Ollama-served models, adjust the number of replicates per model, or change the aggregation rule. The paper's findings support 3-4 comparable open-source LLMs with the *mean* aggregator; other choices are supported but not recommended.

Usage

```
custom_ensemble(
  models,
  replicates = .DEFAULT_REPLICATES,
  aggregator = c("mean", "median", "max", "topk_mean"),
  temperature = .DEFAULT_TEMPERATURE,
  backend = backend_ollama()
)
```

Arguments

`models`	Character vector of Ollama model tags (or backend-appropriate identifiers).
`replicates`	Integer >= 1. Number of replicates per model.
`aggregator`	One of "mean", "median", "max", "topk_mean".
`temperature`	Sampling temperature passed to the backend.
`backend`	A backend object (default: `backend_ollama()`).

Value

A `screenllm_ensemble` object.

default_ensemble *The paper's default LLM ensemble*

Description

Returns the four-LLM *mean* ensemble reported as the universal ranker in Spillias et al. (2026), with three replicates per LLM. This is the ensemble `rank_records()` uses when `ensemble` is not supplied.

Usage

```
default_ensemble(
  backend = backend_ollama(),
  replicates = .DEFAULT_REPLICATES,
  temperature = .DEFAULT_TEMPERATURE
)
```

Arguments

`backend`	A backend object (default: `backend_ollama()`).
`replicates`	Integer >= 1. Defaults to three.
`temperature`	Sampling temperature. Defaults to 0.7 (per the paper).

Value

A `screenllm_ensemble` object.

default_ensemble_light *A "light" ensemble that runs on a laptop*

Description

Returns a four-LLM mean ensemble built from 3-4 B-class open-weights models (`gemma3:4b`, `llama3.2:3b`, `qwen3:4b`, `mistral:7b`). Fits in roughly 10 GB of disk and 8 GB of RAM at Q4 quantisation, so it runs comfortably on an 8-16 GB laptop.

Usage

```
default_ensemble_light(
  backend = backend_ollama(),
  replicates = .DEFAULT_REPLICATES,
  temperature = .DEFAULT_TEMPERATURE
)
```

Arguments

backend	A backend object (default: backend_ollama()).
replicates	Integer >= 1. Defaults to three.
temperature	Sampling temperature. Defaults to 0.7 (per the paper).

Details

Slightly less accurate than default_ensemble() (~5-10 percentage-point drop in AP on the paper benchmarks), but useful for exploration, small reviews, or machines that cannot host the paper's 65 GB ensemble.

Value

A screenllm_ensemble object.

Examples

```
light <- default_ensemble_light(backend = backend_mock())
print(light)
```

---

define_criteria *Define screening inclusion criteria*

Description

Bundles the human-readable inclusion criteria a review uses into an object that can be rendered into the paper's standard screening prompt.

Usage

```
define_criteria(
  scope,
  inclusions,
  exclusion_notes = NULL,
  clarifications = NULL
)
```

Arguments

scope	A one-sentence description of what the review is about. Used at the top of the LLM prompt so the model has orienting context.
inclusions	A character vector of numbered inclusion criteria. Order matters — criterion 1 is scored first, criterion n last.
exclusion_notes	Optional list of the same length as inclusions, giving per-criterion exclusion clarifications. NULL at any position means no notes for that criterion.

`clarifications` Optional character vector of general clarifications (e.g. definitions of terms) rendered as a block after the criteria. `NULL` (the default) omits the block.

Value

An object of class `screenllm_criteria`.

Examples

```
criteria <- define_criteria(
  scope = "Field-based coral reef restoration and performance",
  inclusions = c(
    "The study is conducted at a field-based coral reef restoration site.",
    "The study describes a project with an explicit restoration goal.",
    "The study describes an active restoration intervention.",
    "The study monitors at least one restoration-performance metric."
  )
)
print(criteria)
```

`delete_artefact` *Delete an artefact (or a whole project)*

Description

Delete an artefact (or a whole project)

Usage

```
delete_artefact(project, artefact = "all")
```

Arguments

`project`	Project name.
`artefact`	Either a canonical artefact name, or `"all"` to remove the entire project directory.

Value

Invisibly, `TRUE`.

delete_model *Delete an Ollama model to free up disk space*

Description

Wrapper around Ollama's `DELETE /api/delete` endpoint. Removes the model's blobs from the Ollama data directory. The operation is irreversible from the API side but the model can always be pulled back later with `pull_model()`.

Usage

```
delete_model(
  model,
  ollama_url = getOption("screenllm.ollama_url"),
  verbose = getOption("screenllm.verbose", TRUE)
)
```

Arguments

`model`	Model tag (e.g. `"mistral:7b"`).
`ollama_url`	Base URL of the Ollama server.
`verbose`	Show progress messages.

Value

Invisible `TRUE` on success, `FALSE` if the API returned a non-2xx status.

detect_gpu *Detect a usable GPU for Ollama inference*

Description

Probes the system for a GPU that Ollama can offload models to. Does not require Ollama itself; the checks are OS-level so users can verify their hardware is set up before pulling any models.

Usage

```
detect_gpu()
```

Details

Checks in order:

1. macOS with Apple Silicon (Metal is used automatically).
2. `nvidia-smi` returning success – an NVIDIA GPU with a working CUDA driver.
3. `rocm-smi` returning success – an AMD GPU with a working ROCm driver.

Ollama itself decides how much of a model to offload to the GPU (based on free VRAM); this function only reports whether the underlying hardware is available. To verify Ollama is actually using the GPU during a run, look at `ollama ps` in a terminal: the `SIZE (GPU)` column shows how many bytes are on the GPU.

Value

A list with:

- `available`: logical
- `kind`: one of `"apple"`, `"nvidia"`, `"amd"`, or `"none"`
- `detail`: a short human-readable string (e.g. the GPU model name, or the reason detection failed)

Examples

```
info <- detect_gpu()
info$available
info$kind
```

---

estimate_runtime *Estimate wall-clock time for a ranking run*

Description

Rough back-of-the-envelope estimator to tell a user "this is going to take about X hours" before they hit Run. The estimate scales `n_records * n_models * n_replicates` by a per-call cost that depends on model size, and adds a small fixed overhead per model for weight loading.

Usage

```
estimate_runtime(
  n_records,
  ensemble,
  seconds_per_call = NULL,
  gpu = NULL,
  throttled = FALSE
)
```

Arguments

n_records	Number of records in the corpus.
ensemble	A screenllm_ensemble object.
seconds_per_call	Optional override for the per-call cost. When NULL (the default) a heuristic based on the largest model in the ensemble is used, adjusted for GPU / CPU / throttled hardware states.
gpu	Whether to assume GPU inference. NULL (the default) calls detect_gpu() to auto-detect. Pass TRUE or FALSE to override.
throttled	If TRUE, assume GPU cores are locked at low clock speed (see gpu_status()). Treated as CPU-equivalent for throughput. Defaults to FALSE; the Shiny app passes the live gpu_status() value.

Details

The estimate is intentionally coarse. Real wall-clock depends on hardware (GPU vs CPU), Ollama's model-swapping behaviour, prompt length, and other user-load on the machine. Treat the number as an order-of-magnitude estimate only.

Value

A screenllm_estimate object: a list with n_calls, seconds_per_call, seconds_total, human_readable, gpu, hardware (a string describing the assumed hardware profile), and a caveats character vector.

Examples

```
ens <- default_ensemble(backend = backend_mock())
estimate_runtime(n_records = 500, ensemble = ens, gpu = FALSE)
estimate_runtime(n_records = 500, ensemble = ens, gpu = TRUE)
```

---

export_report *Render the screening report as a self-contained HTML document*

Description

Knits the packaged R Markdown template with whatever project artefacts are supplied and returns the path to the rendered HTML file. Open it in a browser to view; use the browser's Print > Save as PDF to archive as PDF (this avoids the LaTeX install that a direct PDF backend would need).

Usage

```
export_report(
  output_file = NULL,
  project = NULL,
  ranked = NULL,
  plan = NULL,
```

```
  decisions = NULL,
  criteria = NULL,
  ensemble = NULL
)
```

Arguments

output_file	Path to write the HTML report to. Defaults to a file in tempdir().
project	Optional project name (for the report header).
ranked	Optional ranked corpus tibble.
plan	Optional screenllm_plan object.
decisions	Optional tibble of human decisions.
criteria	Optional screenllm_criteria object.
ensemble	Optional screenllm_ensemble object.

Details

All artefact arguments default to NULL; the template fills in "(not recorded)" for anything missing, so the same call works whether the reviewer is midway through screening or fully finished.

Value

Invisibly, the path to the rendered HTML file.

Examples

```
# Render a minimal report to a temporary HTML file.
ranked <- data.frame(
  id = "r1", title = "Example study", abstract = "A short abstract.",
  universal_best_score = 80, rank = 1
)
out <- tempfile(fileext = ".html")
export_report(output_file = out, ranked = ranked)
```

---

export_worksheet *Export the human-screening worksheet*

Description

Writes the records above the SAFE stop point to an Excel file with a blank human_decision column ready for the reviewer to complete.

Usage

```
export_worksheet(plan, path)
```

Arguments

plan	A screenllm_plan object.
path	Output path (must end in .xlsx).

Value

Invisibly, path.

---

find_duplicates *Detect and remove duplicate records*

Description

Two records are considered duplicates if they share a non-missing DOI (case-insensitive) or if their normalised titles match. Title normalisation drops punctuation, lowercases, collapses whitespace, and (optionally) fuzzy-matches with stringdist if that package is available. Returns the input tibble with an added duplicate_of column: NA for unique records, otherwise the id of the earlier record that duplicates it.

Usage

```
find_duplicates(records, fuzzy = TRUE)
```

Arguments

records	A tibble of records from read_records().
fuzzy	Whether to fuzzy-match titles (Jaro-Winkler similarity of at least 0.95). Requires the stringdist package; falls back to exact normalised-title match if stringdist is not installed.

Details

Multi-database searches (Scopus, Web of Science, Google Scholar) often produce 10-20 percent duplicates; running this on the fresh corpus before ranking avoids scoring the same abstract three or four times.

Value

The input tibble with a new duplicate_of column.

Examples

```
recs <- data.frame(
  id = c("a", "b", "c"),
  title = c("Coral reefs", "Coral Reefs.", "Deep sea"),
  abstract = c("x", "x", "y")
)
find_duplicates(recs)
```

gpu_status *Live NVIDIA GPU status snapshot*

Description

Queries `nvidia-smi` for current graphics clock, memory clock, memory used, power draw and utilisation. Used to catch the "throttled" state where a laptop dGPU reports 99% utilisation but is running its cores at idle clock speeds (typically because the laptop is on battery, in a power-saver profile, or persistence mode is off). In that state throughput drops ~10x and each LLM call takes 20-30s instead of 2-3s.

Usage

```
gpu_status(throttled_clock_mhz = 800, loaded_mib = 500)
```

Arguments

`throttled_clock_mhz`	Threshold below which the graphics clock is considered throttled. Defaults to 800 MHz (a modern dGPU under real inference load should be 1500-2500 MHz).
`loaded_mib`	Threshold at which VRAM is considered "in use by a model" (avoids false-positive throttling at rest, when idle clocks are expected and normal). Defaults to 500 MiB.

Details

Returns `available = FALSE` on non-NVIDIA systems; the Metal / ROCm equivalents don't expose live clock data through a portable CLI.

Value

A list with `available`, `graphics_clock_mhz`, `memory_clock_mhz`, `memory_used_mib`, `memory_total_mib`, `power_draw_w`, `utilisation_pct`, `throttled` (logical), and `hint` (short human-readable action if throttled).

Examples

```
s <- gpu_status()
if (isTRUE(s$throttled)) message(s$hint)
```

`install_prereqs` *One-stop setup for a fresh machine*

Description

Walks a non-technical user through everything needed to run `screenllm`:

Usage

```
install_prereqs(
  preset = NULL,
  models = .PINNED_DEFAULT_MODELS,
  interactive = base::interactive(),
  wait_seconds = 60L,
  ollama_url = getOption("screenllm.ollama_url")
)
```

Arguments

`preset`	One of `"paper"` (the four ~20-30B paper models, ~65 GB), `"light"` (four ~3-7B models, ~10 GB) or `"none"` (skip the model pull). Overrides `models` when set. Defaults to `NULL`, which means "use `models`".
`models`	Character vector of Ollama model tags to ensure are installed. Ignored if `preset` is supplied. Defaults to the four-LLM paper ensemble.
`interactive`	Whether to prompt the user before running install commands or pulling large models. Defaults to `base::interactive()`. When `FALSE`, the function reports missing components but never installs or downloads anything.
`wait_seconds`	Seconds to wait for the Ollama daemon to come up after (re)starting it. Defaults to 60.
`ollama_url`	Base URL of the Ollama server.

Details

1. Detects the OS and checks whether Ollama is installed.
2. If Ollama is missing, offers to install it (`brew` on macOS, `winget` on Windows, the official install script on Linux). If the user declines or the package manager is unavailable, opens `https://ollama.com/download` in the browser.
3. Waits for the Ollama daemon to come up (up to `wait_seconds`).
4. Pulls each model in `models` that is not already installed.
5. Verifies with `check_setup()`.

Safe to re-run: it is a no-op if everything is already in place.

Value

Invisible logical: `TRUE` if all prerequisites are ready after the call, `FALSE` otherwise.

Examples

```
# Check for Ollama without pulling any models. In a non-interactive
# session this only reports status and installs nothing.
install_prereqs(preset = "none")

## Not run:
# Interactive setup that offers to install Ollama and pull models:
install_prereqs(preset = "light") # ~10 GB; runs on 8-16 GB RAM
install_prereqs(preset = "paper") # ~65 GB; needs a workstation

## End(Not run)
```

---

launch_app *Launch the full end-to-end Shiny app*

Description

Opens a seven-tab wizard that walks the user from Ollama setup through corpus upload, criteria definition, ensemble ranking, SAFE stopping, human screening, and reporting. Everything the app produces is persisted to the per-user data directory (`data_root()`) so a browser crash never loses more than one decision.

Usage

```
launch_app(project = NULL, launch_browser = interactive())
```

Arguments

`project`	Optional project name to open on launch. If `NULL`, the Setup tab lets the user pick or create one.
`launch_browser`	Passed to `shiny::runApp()`.

Value

Invisibly, `NULL`.

---

launch_screening_app *Launch the interactive screening Shiny app*

Description

Opens a browser-based UI that walks the reviewer through the records at or above the SAFE stop point. Each record is presented with title, abstract, LLM ensemble score, and the per-criterion justifications. Decisions are appended to `out_file` after every click so that a browser crash never loses more than one decision.

Usage

```
launch_screening_app(
  plan,
  ranked,
  out_file = file.path(tempdir(), "screening_decisions.csv"),
  launch_browser = interactive()
)
```

Arguments

plan	A screenllm_plan object.
ranked	The full ranking (needed for the per-criterion justifications).
out_file	Path where decisions are written (.csv or .xlsx). Defaults to a file in the session's temporary directory; pass an explicit path to keep the decisions after the session ends.
launch_browser	Passed to shiny::runApp().

Value

Invisibly, the path to out_file.

---

list_projects *List existing project names*

Description

List existing project names

Usage

```
list_projects()
```

Value

Character vector of project names (immediate subdirectories of <data_root()>/projects/).

list_project_artefacts *Canonical artefact filenames*

Description

Returns the mapping from artefact key to filename used inside every project directory. Exposed so power users can find files on disk without loading the package.

Usage

```
list_project_artefacts()
```

Value

Named character vector.

load_artefact *Load an artefact from a project directory*

Description

Load an artefact from a project directory

Usage

```
load_artefact(project, artefact, default = NULL)
```

Arguments

project	Project name.
artefact	One of the canonical artefact names.
default	Value to return if the artefact does not exist.

Value

The stored object, or `default`.

load_toy_cbfm_criteria *Load the ready-made criteria for the toy CBFM corpus*

Description

Reads `inst/extdata/toy_cbfm_criteria.R` (which mirrors the CBFM entry of the screening criteria used in Spillias et al. 2024, Cell Reports Sustainability) and returns a `screenllm_criteria` object ready to feed into `rank_records()`.

Usage

```
load_toy_cbfm_criteria()
```

Details

Kept as a file the user can inspect / edit rather than hard-coding the criteria in the package, so a colleague can iterate on the demo criteria without touching R source.

Value

A `screenllm_criteria` object.

Examples

```
criteria <- load_toy_cbfm_criteria()
criteria$scope
criteria$inclusions
```

ollama_catalog *Curated catalog of Ollama models useful for screening*

Description

Returns a small tibble of popular instruction-tuned models with approximate Q4_K_M-quantised disk sizes and a one-line description. Not exhaustive; the full Ollama library is at https://ollama.com/library. Users can still pull any tag with `pull_model()` regardless of whether it's in this catalog.

Usage

```
ollama_catalog()
```

Details

The list is intentionally short and opinionated; it favours models that behave well on screening prompts. Update this in the package as the Ollama ecosystem evolves.

Value

A tibble with columns `tag`, `family`, `size_gb`, `description`.

Examples

```
catalog <- ollama_catalog()
head(catalog)
# Filter to models that fit in 8 GB of VRAM.
catalog[catalog$size_gb <= 8, ]
```

`ollama_health` *Ping the Ollama server*

Description

Returns `TRUE` if the Ollama HTTP API responds within a short timeout. Used by `check_setup()` and callable directly by backends.

Usage

```
ollama_health(
  ollama_url = getOption("screenllm.ollama_url"),
  quiet = !interactive()
)
```

Arguments

`ollama_url`	Base URL of the Ollama server.
`quiet`	If `FALSE` (the default when interactive), print status.

Value

Logical.

`ollama_installed_models_detail` *Installed Ollama models with disk-space metadata*

Description

Like `ollama_installed_models()`, but returns a data.frame with one row per installed model and columns `name` and `size_bytes` – suitable for a "manage models / free up disk space" UI.

Usage

```
ollama_installed_models_detail(ollama_url = getOption("screenllm.ollama_url"))
```

Arguments

ollama_url	Base URL of the Ollama server.

Value

A data.frame; zero rows if Ollama is unreachable or has no models installed.

---

plan_screening *Plan the human screening set with the SAFE stopping rule*

Description

Applies the SAFE rule from Spillias et al. (2026) to a ranking produced by rank_records(). Walks the ranked corpus from highest score to lowest, and returns the position where SAFE fires along with the subset of records the human should screen (everything at or above that position). Defaults reproduce the paper's advance-choosable recommended setting: minimum coverage 50\

Usage

```
plan_screening(
  ranked,
  target_recall = .DEFAULT_TARGET_RECALL,
  safe_min_cover = .DEFAULT_SAFE_MIN_COVER,
  safe_run_length = .DEFAULT_SAFE_RUN_LENGTH,
  spot_check_n = .DEFAULT_SPOT_CHECK_N,
  spot_check_labels = NULL,
  seed = 1L
)
```

Arguments

ranked	A screenllm_ranking object (output of rank_records()).
target_recall	Target recall (default 0.95).
safe_min_cover	Minimum-coverage fraction (default 0.50).
safe_run_length	Consecutive-negatives run length (default 50).
spot_check_n	Number of records in the SAFE spot-check (default 200).
spot_check_labels	Optional named vector of accept/reject decisions for the spot-check records (names = record ids, values in c("Accept", "Reject")). If NULL, the plan uses a placeholder estimate and reports the stop-point conservatively.
seed	Random seed for the spot-check draw.

Details

Because SAFE's spot-check gate depends on a random sample, the returned plan is only deterministic when seed is set.

Value

A screenllm_plan object.

Examples

```
# A minimal example. In practice, `ranked` comes from `rank_records()`.
ranked <- data.frame(
  id = paste0("r", 1:100),
  universal_best_score = sort(runif(100, 0, 100), decreasing = TRUE),
  rank = 1:100
)
plan <- plan_screening(ranked, safe_run_length = 10, spot_check_n = 20)
plan
```

---

pull_model *Pull an Ollama model to the local server*

Description

Wrapper around Ollama's HTTP pull endpoint. Streams progress messages if verbose = TRUE. Blocks until the pull completes.

Usage

```
pull_model(
  model,
  ollama_url = getOption("screenllm.ollama_url"),
  verbose = getOption("screenllm.verbose", TRUE)
)
```

Arguments

model	The model tag (e.g. "gemma3:27b").
ollama_url	Base URL of the Ollama server.
verbose	Logical.

Value

Invisible TRUE on success.

rank_records *Rank a corpus with an LLM ensemble*

Description

Sends every record to every model in the ensemble at every replicate, aggregates the resulting scores per record, and returns the corpus with the aggregated score and rank attached. Progress is displayed with `cli`. All intermediate scores are cached to disk so an interrupted run can be resumed by calling `rank_records()` again with the same `cache_dir`.

Usage

```
rank_records(
  records,
  criteria,
  ensemble = default_ensemble(),
  cache_dir = getOption("screenllm.cache_dir"),
  verbose = getOption("screenllm.verbose", TRUE),
  max_workers = getOption("screenllm.max_workers", 1L),
  on_score = NULL
)
```

Arguments

`records`	A tibble of records (produced by `read_records()`).
`criteria`	A `screenllm_criteria` object.
`ensemble`	A `screenllm_ensemble` object. Defaults to `default_ensemble()`.
`cache_dir`	Directory to write per-call cache files. If `NULL`, a temporary directory is used and the results are not persisted between R sessions.
`verbose`	Show progress messages and bars.
`max_workers`	Not yet used; reserved for future concurrency.
`on_score`	Optional callback invoked once per (record, model, replicate) tuple after each score is available (whether fresh or restored from cache). Signature: `function(id, model, replicate,` Used by the Shiny app's ranking module to stream partial scores into the UI as they land; safe to leave `NULL`.

Value

The input tibble with added columns: `universal_best_score`, `rank`, `per_model_scores` (list-column), `justifications` (list-column of per-criterion rationales as returned by the LLM).

Examples

```
# A no-Ollama, in-R demo using the mock backend.
records <- data.frame(
  id = c("a", "b"),
  title = c("Coral reef restoration outcomes",
            "Deep-sea mining impacts on benthic fauna"),
  abstract = c("We monitored transplanted corals for 12 months...",
               "This modelling study estimates long-term sediment plumes...")
)
criteria <- define_criteria(
  scope = "Field-based coral reef restoration and performance",
  inclusions = c("Study is field-based.", "Study describes an intervention.")
)
ens <- default_ensemble(backend = backend_mock())
ranked <- rank_records(records, criteria, ensemble = ens, verbose = FALSE)
ranked[, c("id", "universal_best_score", "rank")]
```

`read_decisions` *Read completed screening decisions*

Description

Reads a worksheet the reviewer has completed offline (an Excel file produced by `export_worksheet()`) and returns a tibble of decisions.

Usage

```
read_decisions(path)
```

Arguments

`path`	Path to the completed worksheet.

Value

A tibble with columns `id`, `human_decision`, `note` (if present).

read_records *Read a corpus of records from disk*

Description

Accepts a data frame directly, or a path to a CSV, TSV, RIS, BibTeX, or Excel file. Returns a tibble with the columns `screenllm` expects: `id`, `title`, `abstract`, plus any additional bibliographic columns the input carries.

Usage

```
read_records(source, id_column = NULL)
```

Arguments

`source`	Either a data frame with (at minimum) columns `title` and `abstract`, or a file path.
`id_column`	Optional name of an existing column to use as the record id. If `NULL`, an `id` column is generated as `record_<zero-padded index>`.

Value

A tibble with columns `id`, `title`, `abstract`, plus any extras.

save_artefact *Save an artefact into a project directory*

Description

Writes an object under a canonical filename inside the project directory so all downstream tabs / modules can find it.

Usage

```
save_artefact(project, artefact, x)
```

Arguments

`project`	Project name.
`artefact`	One of the canonical artefact names (see `list_project_artefacts()`).
`x`	The object to save.

Value

Invisibly, the path written to.

summarise_screening *Summarise a completed screening session*

Description

Combines the LLM ranking with a set of human decisions and reports the realised recall (against the decisions the human made above the stop point), the workload actually incurred, and diagnostic per-record counts.

Usage

```
summarise_screening(ranked, decisions, plan = NULL)
```

Arguments

ranked	A ranking (output of rank_records()).
decisions	A tibble with columns id and human_decision (values "Accept" or "Reject").
plan	Optional screenllm_plan used to identify the stop point; when supplied, summarise_screening() also reports the workload fraction the plan intended.

Value

A screenllm_report object.

Index