Package 'pamflow'

November 26, 2026

Title End-to-End Passive Acoustic Monitoring Workflow

Version 0.6.1

Description An end-to-end workflow for passive acoustic monitoring projects, from raw recordings to data ready for ecological analysis. Standardizes and checks field recordings, runs 'BirdNET' (Kahl et al. 2021 <doi:10.1016/j.ecoinf.2021.101236>) through its 'Python' package, cleans and validates detections, and computes soundscape indices. Builds detection histories, count histories and input for occupancy and abundance models, summarizes activity patterns and species richness, and writes 'HTML', 'PDF' and 'Excel' reports. Optional 'shiny' apps support standardization, validation and annotation, and the training of custom 'BirdNET' classifiers.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

SystemRequirements FFmpeg (audio conversion, filtering and clips); uv (creates the optional Python environment for 'BirdNET')

Depends R (>= 4.0.0)

Imports av, cli, data.table (>= 1.16.0), fs, ggplot2, glue, parallel, processx, ps, rlang, seewave, shiny, stats, tools, tuneR, utils, withr (>= 2.5.0), yaml

Suggests knitr, openxlsx, reticulate (>= 1.41), rmarkdown, rstudioapi, soundecology, tcltk, testthat (>= 3.2.0), unmarked

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

URL https://pradeeshwarrj.github.io/pamflow/, https://github.com/pradeeshwarrj/pamflow

BugReports https://github.com/pradeeshwarrj/pamflow/issues

VignetteBuilder knitr

NeedsCompilation no

Author Pradeeshwar Rajagopal James [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-1246-442X>)

Maintainer Pradeeshwar Rajagopal James <pradeeshwarrj@gmail.com>

Repository CRAN

Date/Publication 2026-11-26 09:40:02 UTC

Contents

pam_activity . . . 4
pam_add_qc . . . 5
pam_add_site_covariates . . . 6
pam_annotate . . . 7
pam_assemble_detections . . . 8
pam_as_count_history . . . 8
pam_audio_exts . . . 10
pam_audio_history . . . 10
pam_audio_info . . . 11
pam_audio_pattern . . . 12
pam_audio_summary . . . 12
pam_birdnet_benchmark . . . 13
pam_birdnet_cleanup . . . 14
pam_birdnet_doctor . . . 15
pam_birdnet_setup . . . 16
pam_birdnet_train . . . 17
pam_birdnet_train_metrics . . . 19
pam_birdnet_train_prepare . . . 20
pam_birdnet_train_repeat . . . 21
pam_build_occupancy . . . 22
pam_checkpoint_add . . . 23
pam_checkpoint_read . . . 24
pam_check_deployments . . . 24
pam_check_ffmpeg . . . 25
pam_check_model_input . . . 25
pam_clean_detections . . . 26
pam_config_read . . . 27
pam_config_validate . . . 28
pam_count_histories . . . 28
pam_count_history . . . 30
pam_cut_audio . . . 32
pam_define_seasons . . . 32
pam_detect . . . 33
pam_detection_table . . . 34
pam_example_project . . . 36
pam_extract_grid . . . 37
pam_filter_audio . . . 38
pam_filter_one . . . 39
pam_highpass_filter_string . . . 39

pam_inspect_raw . . . 40
pam_is_flac . . . 41
pam_list_audio . . . 41
pam_load_config . . . 42
pam_log . . . 42
pam_match_deployments . . . 43
pam_model_input . . . 44
pam_parse_datetime . . . 46
pam_parse_filenames . . . 47
pam_pending_batches . . . 48
pam_plot_activity . . . 48
pam_plot_richness . . . 49
pam_process_detections . . . 50
pam_read_audio . . . 51
pam_read_deployments . . . 51
pam_recall_draw . . . 52
pam_recall_export . . . 53
pam_recall_frame . . . 54
pam_recall_metrics . . . 55
pam_recall_report . . . 56
pam_recall_sample . . . 57
pam_report . . . 58
pam_richness . . . 59
pam_run_stage . . . 61
pam_scan_audio . . . 62
pam_soundscape_batch . . . 63
pam_soundscape_one . . . 63
pam_soundscape_plot_diel . . . 64
pam_soundscape_plot_phase . . . 65
pam_soundscape_report . . . 65
pam_soundscape_run . . . 66
pam_soundscape_run_vis . . . 66
pam_soundscape_summarise . . . 67
pam_species_thresholds . . . 67
pam_standardization_app . . . 69
pam_standardize . . . 70
pam_train_app . . . 71
pam_validate . . . 73
pam_validation_draw . . . 74
pam_validation_export . . . 75
pam_validation_import . . . 76
pam_validation_metrics . . . 76
pam_validation_pool . . . 78
pam_validation_report . . . 78
pam_validation_sample . . . 79
pam_wilson_ci . . . 80
soundscape . . . 81
soundscape_vis . . . 81

to_scalar_num 81
utils_audio 82

Index **83**

`pam_activity` *Activity patterns: detections per recorded hour*

Description

Counts the cleaned detections of each species in every hour of the day, day, week or month, and divides them by the time that was recorded in that period. Because the division uses recording effort, periods and sites that were recorded for different lengths of time can be compared. Use it for daily (diel) activity patterns and for seasonal patterns such as migration or breeding timing.

Usage

```
pam_activity(
  cfg,
  run_id = cfg$birdnet$run_id,
  by = c("hour", "day", "week", "month"),
  by_site = FALSE,
  species = NULL,
  thresholds = NULL,
  min_effort_hours = 0,
  write = FALSE
)
```

Arguments

`cfg`	A `pam_config` object.
`run_id`	Detection run (sub-folder of paths$detections).
`by`	The period: `"hour"` (hour of the day, pooled over all dates), `"day"`, `"week"` (weeks start on Monday) or `"month"`.
`by_site`	If `FALSE` (the default), sites are pooled: the rate is all detections divided by all recorded hours. If `TRUE`, one row per site.
`species`	Optional species to keep: scientific names, common names or species labels, in any capitalization. `NULL` keeps all.
`thresholds`	Optional per-species thresholds (a data frame, or the path of a CSV file, with `scientific_name` and `threshold`), for example from `pam_species_thresholds()`. They are applied on top of Stage 4 and can only remove more events.
`min_effort_hours`	Periods with less recorded time than this get no rate (`NA`) and `sufficient = FALSE`. Default 0 keeps every period with any recording.
`write`	If `TRUE`, also save the table as `<outputs>/<run_id>/activity_<by>.csv`. Nothing is written by default.

Details

What the rate means:

`detections_per_hour` is the number of detected events (Stage 4 merges overlapping BirdNET windows into events) per hour of recording. It is an activity index: it depends on how often and how loudly animals call, not only on how many are present, and on the confidence thresholds used. Compare it between periods and sites for the same species, and avoid comparing raw rates between species.

Time of day:

Hours are local clock hours of the project time zone. A recording is split over the hours it covers, so effort is exact for recordings that start part way through an hour or cross midnight.

Value

A data.table with one row per period and species (and per site when `by_site = TRUE`): `period` (hour 0-23, or the first day of the period), `species`, `scientific_name`, `n_detections`, `effort_hours`, `detections_per_hour`, `relative_activity` (the rate divided by the species' highest rate, so 1 marks its peak), `sufficient` and, for pooled tables, `n_sites_surveyed` and `n_sites_detected`. Periods with recordings but no detection get 0. Periods with no recording have no row. The choices are stored in the attributes `by` and `by_site`.

See Also

`pam_plot_activity()`, `pam_detection_table()`

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
act <- suppressMessages(pam_activity(ex$cfg, by = "hour"))
as.data.frame(act)[, c("period", "species", "n_detections", "detections_per_hour")]
# one row per site and day
day <- suppressMessages(pam_activity(ex$cfg, by = "day", by_site = TRUE))
head(as.data.frame(day)[, c("site_id", "period", "species", "n_detections")], 4)
```

---

pam_add_qc *Add QC flags to a master audio table*

Description

Add QC flags to a master audio table

Usage

```
pam_add_qc(master, cfg)
```

Arguments

master	data.table from pam_parse_filenames() joined with pam_audio_info().
cfg	A pam_config object (uses qc$gap_tolerance_s and audio$min_duration_s).

Details

Gaps are measured between the end of one file and the start of the next within a grid. qc_pass requires a valid name, a readable and untruncated file, and no duplicate; short files and gaps are reported but do not fail QC. Truncation checks apply to PCM WAV files (where bit depth is non-NA).

Value

master with end_time, truncated, short, duplicate, gap_s, gap_before, overlap_before and qc_pass. Sorted by grid and start time.

Examples

```
ex <- pam_example_project()
cfg <- ex$cfg
master <- pam_parse_filenames(ex$files, cfg$filenames$pattern, cfg$filenames$timezone)
master <- merge(master, pam_audio_info(ex$files), by = "file_path")
master <- pam_add_qc(master, cfg)
as.data.frame(master)[, c("file_name", "gap_s", "qc_pass")]
```

---

pam_add_site_covariates

Add site covariates to a count history

Description

Attaches covariates, such as habitat or elevation, to histories that were built without them. Rows are matched to the sites by name, so the order of your table does not matter.

Usage

```
pam_add_site_covariates(x, covariates, replace = FALSE)
```

Arguments

x	A pam_count_history or pam_count_histories.
covariates	A data frame with a site_id column and one column per covariate. Every site of x needs a row; extra rows are ignored. Without a site_id column the table needs one row per site, in the order of the sites.
replace	If FALSE (the default), a covariate that is already present is an error; if TRUE, it is replaced.

Value

x with the covariates added.

See Also

pam_as_count_history(), pam_model_input()

Examples

```
counts <- matrix(c(0, 2, 5, 1, 0, 3), nrow = 3,
                 dimnames = list(c("site_a", "site_b", "site_c"), NULL))
hist <- pam_as_count_history(counts, species = "Example species")
covs <- data.frame(site_id = c("site_c", "site_a", "site_b"), canopy = c(0.9, 0.2, 0.5))
pam_add_site_covariates(hist, covs)$site_covariates
```

---

pam_annotate *Open the ground-truth annotation app*

Description

Shows random recording segments, one at a time, in shuffled order, with a spectrogram and audio. The model's detections are never shown, so the annotation is independent of the detector. Record every species you hear; mark doubtful ones as "possibly present". Each save is written immediately to recall_annotations.csv, so you can stop and resume.

Usage

```
pam_annotate(cfg, reviewer = pam_or(cfg$recall$reviewer, ""), lang = NULL)
```

Arguments

cfg	A pam_config object.
reviewer	Annotator name pre-filled in the app.
lang	Interface language code (for example "en"); NULL chooses it as pam_train_app() does. Translations live in inst/i18n/<code>.yml.

Value

Nothing; runs until the app is closed.

Examples

```
if (interactive() && nzchar(Sys.which("ffmpeg"))) {
  ex <- pam_example_project()
  res <- suppressMessages(pam_scan_audio(ex$cfg))
  new <- suppressMessages(pam_recall_draw(ex$cfg, n = 5, run_id = NULL))
  out <- suppressMessages(pam_recall_export(ex$cfg))
  pam_annotate(ex$cfg)
}
```

pam_assemble_detections *Assemble raw detection batches into one detections table*

Description

Reads all batch files of a run, removes duplicates (from interrupted runs), joins recording metadata from the Stage 1 master table, and writes `<detections>/<run_id>/detections.csv`.

Usage

```
pam_assemble_detections(cfg, run_id = cfg$birdnet$run_id)
```

Arguments

`cfg`	A `pam_config` object.
`run_id`	Name of the detection run (a sub-folder of paths$detections). Defaults to birdnet$run_id.

Value

A data.table with `filename`, `species`, `scientific_name`, `common_name`, `confidence`, `start_time` and `end_time` (seconds into the recording), `site` (= grid), `grid`, `recorder`, `deployment_id`, `in_deployment`, `recording_date`, `recording_start` and `detection_datetime` (local time), `rel`.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
det <- suppressMessages(pam_assemble_detections(ex$cfg))
nrow(det)   # the batch file had one duplicate row, which is dropped
as.data.frame(det)[, c("filename", "species", "confidence", "start_time", "end_time")]
```

pam_as_count_history *Make a count history from your own counts*

Description

Wraps a matrix of counts (sites in rows, occasions in columns) so that it can be prepared for modeling by `pam_model_input()`. `pam_count_history()` builds the same object from the detections of a project; use this function when the counts come from somewhere else.

Usage

```
pam_as_count_history(
  y,
  site_covariates = NULL,
  effort = NULL,
  day = NULL,
  occasions = NULL,
  species = NA_character_,
  count = "events"
)
```

Arguments

y	A matrix of whole-number counts, sites in rows and occasions in columns, with NA where a site was not surveyed. Row names are the site identifiers (made up if missing).
site_covariates	Optional data frame of site covariates. With a site_id column the rows are matched to the sites by name, otherwise there must be one row per site in the same order.
effort	Optional matrix (same shape as y) of recording hours per site and occasion, usable as a detection covariate named effort.
day	Optional matrix (same shape as y) of days since the first occasion, usable as a detection covariate named day. By default the occasion number minus one.
occasions	Optional data frame describing the occasions, one row per column of y.
species	Name of the species, for labeling.
count	What the counts are: "events" or "files" (for labeling).

Value

An object of class pam_count_history: a list with y, effort, day, site_covariates, occasions, species and count.

See Also

pam_count_history(), pam_model_input()

Examples

```
counts <- matrix(c(0, 2, 5, 1, 0, 3, 4, 0, 2, 1, 1, 0), nrow = 3,
                 dimnames = list(c("site_a", "site_b", "site_c"), NULL))
hist <- pam_as_count_history(counts, species = "Example species")
hist$y
```

pam_audio_exts *Supported Audio File Extensions*

Description

Supported Audio File Extensions

Usage

```
pam_audio_exts()
```

Value

Character vector of supported audio extensions (lowercase).

Examples

```
pam_audio_exts()
```

pam_audio_history *Follow audio files through the workflow*

Description

Collects, for every audio file, what each stage recorded about it: the standardization result (Stage 0), the scan and QC flags (Stage 1), filtering (Stage 2), detection (Stage 3), cleaning (Stage 4), validation (Stage 5), ground-truth annotation (Stage 5b) and whether it went into the occupancy histories (Stage 6). It reads the files the stages already write, so it works on existing projects and changes nothing unless you ask it to write.

Usage

```
pam_audio_history(cfg, file = NULL, run_id = NULL, wide = FALSE, write = FALSE)
```

Arguments

cfg	A pam_config object.
file	Optional audio file to look up: a path relative to the standardized audio folder (such as "G0059/G0059_20260405_183000.wav"), a file name, or, for files standardized by Stage 0, the name of the raw file. NULL (the default) returns every file.
run_id	Optional detection run (or runs). NULL (the default) includes every run found in the project.
wide	If TRUE, return one row per file with one column per stage (and per run, for the stages that belong to a run) holding the status.
write	If TRUE, also save the long table to metadata/audio_history.csv. Nothing is written by default.

Details

Statuses: Stage 0 ok, warning or failed; Stage 1 pass or fail; Stage 2 and 3 ok or failed; Stage 4 kept (some events survived cleaning) or removed (none did); Stage 5 reviewed or sampled; Stage 5b annotated, partly annotated or drawn; Stage 6 included or excluded. Stages whose outputs are missing are skipped. Times are saved in UTC when write = TRUE. The history is built from the current outputs, so a stage you delete or rerun is shown as it now is.

Value

A data.table. The long table has one row per file and stage (and run) with columns rel (path relative to the standardized audio folder, NA for files Stage 0 set aside), file (file name), stage, run_id, status, detail (what happened, in words), n (a count, such as the number of detections) and time (when the stage recorded it, where the stage keeps a time, otherwise the time its output file was written). The attribute stages_found lists the stages whose outputs were found. With wide = TRUE the table has rel plus one status column per stage.

Examples

```
ex <- pam_example_project()
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))

# one row per file and stage
hist <- pam_audio_history(ex$cfg)
as.data.frame(hist)[1:3, c("file", "stage", "status", "detail")]

# one file, by name
one <- pam_audio_history(ex$cfg, file = "G0059_20260405_183004.wav")
as.data.frame(one)[, c("stage", "status", "detail")]

# one row per file, one column per stage
as.data.frame(pam_audio_history(ex$cfg, wide = TRUE))
```

---

pam_audio_info *Read Audio File Metadata (WAV or FLAC)*

Description

Reads header information (sample rate, duration, channels, bit depth) for one or more WAV/FLAC files without loading sample data into memory. Returns a data.table with readable status; errors for individual unreadable files are caught gracefully and flagged with readable = FALSE.

Usage

```
pam_audio_info(paths)

pam_audio_info_dt(paths)
```

Arguments

paths	Character vector of file paths.

Value

A data.table with columns file_path, file_size_b, sample_rate, channels, bits, duration_s, readable.

Examples

```
ex <- pam_example_project()
pam_audio_info(ex$files)
```

---

pam_audio_pattern *Regex Pattern Matching Supported Audio Files*

Description

Regex Pattern Matching Supported Audio Files

Usage

```
pam_audio_pattern()
```

Value

Character scalar regex pattern (case-insensitive).

Examples

```
grepl(pam_audio_pattern(), c("a.wav", "b.FLAC", "c.mp3"))
```

---

pam_audio_summary *Summarize the audio master table*

Description

Summarize the audio master table

Usage

```
pam_audio_summary(master, dep = NULL)
```

Arguments

master	Output of pam_scan_audio()$master.
dep	Optional deployment table, to report grids/deployments with no match.

Value

A list: `overall` (one-row data.table), `per_grid` (data.table), `grids_without_deployment`, `deployments_without_audio`.

Examples

```
ex <- pam_example_project()
res <- suppressMessages(pam_scan_audio(ex$cfg))
summ <- pam_audio_summary(res$master, res$deployments)
summ$overall
summ$per_grid
```

---

pam_birdnet_benchmark *Benchmark Stage 3 detection speed at several worker counts*

Description

Runs `pam_detect()` on the same random sample of recordings once per worker count and reports throughput, scaling efficiency, and an estimate for the whole archive. Use it to choose `birdnet$workers` for your CPU, and as the baseline to compare the new backend against.

Usage

```
pam_birdnet_benchmark(
  cfg,
  workers = c(1L, 2L, 4L, 8L),
  n_files = 8L,
  seed = 1L,
  keep_runs = FALSE
)
```

Arguments

`cfg`	A `pam_config` object.
`workers`	Integer vector of worker counts to test. Counts above the number of logical cores are dropped with a warning.
`n_files`	Files per run. Each file takes about a minute for a 60-minute recording, so keep this small (default 8).
`seed`	Seed for the file sample, identical for every run.
`keep_runs`	If FALSE (default) the temporary detection folders are deleted afterwards.

Details

The timing includes model loading in every worker, which is part of the real cost of starting a run. Real runs amortize it over many files, so the full-archive estimate is slightly pessimistic. Run on an otherwise idle machine.

Value

A data.frame with one row per worker count: `workers`, `n_done`, `n_failed`, `elapsed_s`, `files_per_hour`, `speedup`, `efficiency` and `est_hours_full`. Also written to `<logs>/birdnet_benchmark.csv`.

Examples

```
## Not run:
cfg <- pam_load_config("config.yml")
pam_birdnet_benchmark(cfg, workers = c(1, 4, 8), n_files = 8)

## End(Not run)
```

`pam_birdnet_cleanup` *Stop BirdNET worker processes that were left behind*

Description

Interrupting a run (Esc) or a crash can leave the worker processes of the Python `birdnet` package running in the background, holding memory (about 260 MB each). This finds those orphans and stops them. A worker counts as an orphan only if the process that started it no longer exists, so workers of a run that is still going are never touched.

Usage

```
pam_birdnet_cleanup(kill = TRUE, quiet = FALSE)
```

Arguments

`kill`	If FALSE, only report the orphans without stopping them.
`quiet`	If TRUE, print nothing.

Details

`pam_detect()` and the training functions call this automatically (after an interrupt, and before a run starts), so you rarely need it.

Value

Invisibly, a data.frame with `pid`, `name`, `parent_pid` and `killed` (zero rows if there were no orphans).

Examples

```
## Not run:
pam_birdnet_cleanup(kill = FALSE)   # just look
pam_birdnet_cleanup()               # stop them

## End(Not run)
```

`pam_birdnet_doctor` *Check the machine and Python environments needed for BirdNET*

Description

Inspects the R session, the machine (CPU cores, RAM, GPU) and the two pamflow BirdNET Python environments, and reports what is missing and what to do about it. Nothing is installed or changed. TensorFlow is probed in a separate process, so this never loads Python into your R session.

Usage

```
pam_birdnet_doctor(
  envs = list(inference = pam_birdnet_env_dir("inference"), training =
    pam_birdnet_env_dir("training")),
  quiet = FALSE
)
```

Arguments

`envs`	Named list with the folders of the "inference" and "training" environments. Defaults to pamflow's private environments.
`quiet`	If TRUE, return the table without printing it.

Details

Assumptions: on native Windows, TensorFlow has no GPU support, so pamflow runs BirdNET on CPU there; a detected NVIDIA GPU is reported as information only. Run `pam_birdnet_setup()` to create missing environments.

Value

Invisibly, a data.frame with columns `check`, `status` (`ok`/`warn`/`fail`/`info`) and `detail`. Class `pam_birdnet_doctor`.

Examples

```
## Not run:
pam_birdnet_doctor()

## End(Not run)
```

pam_birdnet_setup *Create a private Python environment for BirdNET*

Description

Builds a virtual environment (never touching the system Python) with a pinned Python version. Uses uv when it is installed (it downloads the requested Python itself); otherwise falls back to `reticulate`, which needs a matching Python already installed.

Usage

```
pam_birdnet_setup(
  component = c("inference", "training"),
  python_version = "3.11",
  packages = NULL,
  envdir = pam_birdnet_env_dir(component),
  recreate = FALSE,
  dry_run = FALSE,
  uv = pam_find_uv()
)
```

Arguments

component	"inference" (the `birdnet` package, used by Stage 3) or "training" (BirdNET-Analyzer, used only by the standalone training tools, which are not part of the workflow).
python_version	Python version for the environment. Default "3.11", chosen as widely supported by TensorFlow; change it if a package install reports an unsupported version.
packages	pip requirements. Defaults to `pam_birdnet_default_requirements()[[component]]`.
envdir	Environment folder. Defaults to pamflow's private location.
recreate	If TRUE, delete and rebuild an existing environment.
dry_run	If TRUE, only return the plan without creating anything.
uv	Path to the uv executable. Found automatically when uv is on PATH or in a standard install location; set it if you installed uv elsewhere.

Details

Installing TensorFlow downloads roughly a gigabyte, so this can take several minutes. In an interactive session you are asked to confirm before anything is downloaded; use `dry_run = TRUE` to see the plan first. Afterwards run `pam_birdnet_doctor()`.

Value

Invisibly, a list: `envdir`, `python`, `python_version`, `packages`.

Examples

```
## Not run:
pam_birdnet_setup("inference", dry_run = TRUE)
pam_birdnet_setup("inference")
pam_birdnet_setup("training")

## End(Not run)
```

`pam_birdnet_train` *Train a custom BirdNET classifier from R*

Description

Runs BirdNET-Analyzer's own `birdnet-train` command inside pamflow's training environment, so the trained model is the same kind of file the Analyzer GUI produces. The training runs in a separate process: an error or crash cannot take down your R session.

Usage

```
pam_birdnet_train(
  data_dir,
  output,
  params = list(),
  test_data = NULL,
  load_params = NULL,
  cache_mode = NULL,
  cache_file = NULL,
  envdir = pam_birdnet_env_dir("training"),
  live = FALSE,
  overwrite = FALSE,
  log_dir = NULL
)
```

Arguments

`data_dir`	Training folder with one sub-folder per class. Use `pam_birdnet_train_prepare()` first.
`output`	Output path prefix, e.g. `"E:/models/MyModel"`. The Analyzer writes `<output>.tflite`, `<output>_Labels.txt`, `<output>_Params.csv` and `<output>_sample_counts.csv`. pamflow adds `<output>_run.yml` (what was run) and `<output>_train.log` (everything the trainer printed) and uses `<output>_cache.npz` only if you set `cache_mode`.
`params`	Named list of training parameters. Names: epochs, val_split, learning_rate, batch_size, threads, crop_mode, overlap, fmin, fmax, audio_speed, hidden_units, dropout, focal_loss, focal_loss_gamma, focal_loss_alpha, label_smoothing, mixup, upsampling_ratio, upsampling_mode, model_save_mode, model_format ("tflite",

	"raven" or "both"), autotune, autotune_trials, autotune_executions_per_trial. Yes/no options are on/off switches: TRUE turns them on, FALSE leaves them off. Anything you do not set is left to the installed Analyzer, so a plain run uses the same defaults as that version's own command line. The values it actually used are written to <output>_Params.csv.
test_data	Optional separate test folder; if NULL the Analyzer holds out val_split of the training data.
load_params	Optional path to an earlier run's parameters file, passed as --load_params. Only Analyzer versions that provide that option accept it; otherwise pamflow stops with a clear message.
cache_mode	NULL (default), "save" or "load": passed as --cache_mode. Left off by default so a plain run behaves exactly like the Analyzer. With "load" the features are read from cache_file instead of the audio (the training folder may then be NULL).
cache_file	Cache file for cache_mode; defaults to <output>_cache.npz. A relative path is taken relative to the folder of output, not the working directory.
envdir	Training environment folder (from pam_birdnet_setup("training")).
live	If TRUE, also show the trainer's output on the console while it runs. The log file is written either way.
overwrite	If FALSE (default), refuse to overwrite an existing model.
log_dir	Optional folder for a log file (birdnet_train.log).

Details

This is a standalone tool: it is not a stage of the workflow and no workflow function calls it. Use the trained model by pointing the birdnet: section of the config at it.

This is a standalone tool, not a pipeline stage: it needs no config file and nothing in the workflow calls it. Assumptions: BirdNET-Analyzer is installed in the training environment and provides the birdnet-train command. Hyperparameter autotuning needs the Analyzer's train extra (pam_birdnet_setup("training", packages = "birdnet-analyzer[train]", recreate = TRUE)). Before running, pamflow checks that every option it passes appears in the installed birdnet-train --help and stops with a clear message if one does not.

Value

Invisibly, a list: status ("ok" or "failed"), output, files (named logical: which expected files exist), elapsed (difftime), log, run_record, args, and metrics (from pam_birdnet_train_metrics(), NULL if the log could not be read) and evaluation (with a separate test_data folder: a list with overall, classes and path read from <output>_evaluation.csv, one row per class with precision, recall, F1, AUPRC and AUROC; otherwise NULL) and, when a cache was used, cache (what the trainer reported: see pam_train_cache_notes()).

References

Kahl, S., Wood, C. M., Eibl, M. and Klinck, H. (2021). BirdNET: A deep learning solution for avian diversity monitoring. *Ecological Informatics*, 61, 101236. doi:10.1016/j.ecoinf.2021.101236

Examples

```
## Not run:
prep <- pam_birdnet_train_prepare("E:/training_clips")
res <- pam_birdnet_train("E:/training_clips", "E:/models/MyModel",
                         params = list(epochs = 100, hidden_units = 512))
# reproduce an earlier run from its parameters file
pam_birdnet_train("E:/training_clips", "E:/models/Rerun",
                  load_params = "E:/models/BirdNet_v2_custom_Params.csv")

## End(Not run)
```

---

pam_birdnet_train_metrics

Read training metrics from a `birdnet-train` *log*

Description

Extracts the per-epoch history and the final summary that the Analyzer prints, so runs can be compared (for example with the Analyzer GUI) without reading the log by hand.

Usage

```
pam_birdnet_train_metrics(log_file)
```

Arguments

`log_file`	Path to a training log (`<output>_train.log`).

Details

The log format is the one printed by BirdNET-Analyzer 2.4.0. If the trainer prints something else, the fields come back empty or NA rather than guessed. Terminal progress bars and TensorFlow messages in the log are ignored.

Value

A list: `history` (data.frame with `epoch`, `auprc`, `auroc`, `loss`, `val_auprc`, `val_auroc`, `val_loss`, `learning_rate`), `best` (list with `auprc`, `auroc`, `loss`, `epoch`, `n_epochs`; NA when not printed), `n_train`, `n_val`, `n_labels` (NA if not found) and `evaluated_on` ("validation" or "test").

Examples

```
## Not run:
m <- pam_birdnet_train_metrics("E:/models/MyModel_train.log")
m$best$auprc

## End(Not run)
```

pam_birdnet_train_prepare *Check a training dataset before training*

Description

Looks at the folder-per-class layout BirdNET-Analyzer expects and reports problems early: missing or empty classes, very small classes, files that are not audio, and a missing non-event class.

Usage

```
pam_birdnet_train_prepare(data_dir, min_clips = 20L, quiet = FALSE)
```

Arguments

data_dir	Training folder with one sub-folder per class.
min_clips	Classes (other than noise classes) with fewer audio files are flagged. A rule of thumb for your own data, not an Analyzer limit.
quiet	If TRUE, do not print the report.

Details

Audio files are counted by extension (wav, flac, mp3, ogg, m4a); clip length is not inspected. The Analyzer's documentation recommends 3-second clips and a background class named noise, other, background or silence.

Value

Invisibly, a list: classes (data.frame with class, n_audio, n_other, n_subdirs, is_noise), issues (character) and ok (TRUE if there are no issues).

Examples

```
## Not run:
pam_birdnet_train_prepare("E:/training_clips")

## End(Not run)
```

pam_birdnet_train_repeat *Train the same classifier several times and summarize the metrics*

Description

Each run uses a different random validation split and initialization, so a single score can be lucky or unlucky. Repeating the run shows how much the metrics vary, which is the yardstick for comparing settings, or pamflow against the Analyzer GUI, fairly.

Usage

```
pam_birdnet_train_repeat(data_dir, output, n = 5L, params = list(), ...)
```

Arguments

data_dir	Training folder (see pam_birdnet_train()).
output	Output path prefix; run k is written to <output>_run<kk> (two digits).
n	Number of runs.
params	Named list of training parameters (see pam_birdnet_train()).
...	Further arguments passed to pam_birdnet_train() (for example test_data, envdir, live, overwrite).

Details

This is a standalone tool: it is not a stage of the workflow and no workflow function calls it. Use the trained model by pointing the birdnet: section of the config at it.

A failed run is recorded and does not stop the others. With a fixed test_data folder every run is scored on the same clips, so the spread then reflects training randomness only.

Value

A list: runs (data.frame with run, status, auprc, auroc, loss, best_epoch, elapsed_min, output) and summary (data.frame with metric, n, mean, sd, min, max over the successful runs).

Examples

```
## Not run:
rep5 <- pam_birdnet_train_repeat("E:/clips", "E:/models/Check", n = 5)
rep5$summary

## End(Not run)
```

pam_build_occupancy *Build occupancy detection histories for all qualifying species*

Description

Reads cleaned detections and the audio master, constructs per-species detection history matrices (wide format), occasion covariates (effort and Julian date), and site covariates. Writes everything to `data/occupancy/<run_id>/`.

Usage

```
pam_build_occupancy(cfg, run_id = cfg$occupancy$run_id, force = FALSE)
```

Arguments

`cfg`	A list from `pam_config_read`.
`run_id`	Character. The detection run to use. Defaults to `cfg$occupancy$run_id`.
`force`	Logical. Rebuild even if outputs exist? Default FALSE.

Details

Occasion definition:

An occasion is a block of `occupancy$occasion_length_days` calendar days (default 1, midnight to midnight in the project timezone). Blocks start on the first day with any recording in the project and are the same for every site. Occasions with less than `min_effort_hours` of recording in total are coded NA (not surveyed). A detection counts in the occasion that contains its date. Outputs that already exist are kept unless `force = TRUE`, so rebuild after changing the occasion length.

Site definition:

Grids with a single deployment use the grid name as the site ID. Grids with multiple deployments get `grid_deployment_id`.

Confidence thresholds:

If `processing.species_thresholds_file` points to a CSV with columns `scientific_name` and `threshold`, those per-species values are used. Species not listed fall back to `processing.min_confidence`.

Species filter:

Only species with at least `occupancy.min_detections` total detections (after the confidence cut) get a history matrix.

Value

Invisibly, a list with `histories` (named list of wide matrices), `effort`, `site_covariates`, `summary`.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))  # occupancy builds on cleaned detections
occ <- suppressMessages(pam_build_occupancy(ex$cfg))
names(occ$histories)
occ$summary
# detection history of the first species: rows = sites, columns = days, NA = not surveyed
occ$histories[[1]]
```

`pam_checkpoint_add` *Record items as completed*

Description

Call this *after* the corresponding batch output has been written to disk, so an interruption never marks unwritten work as done.

Usage

```
pam_checkpoint_add(ids, path)
```

Arguments

`ids`	Character vector of IDs (e.g. file paths relative to the audio root).
`path`	Checkpoint file path.

Value

`ids`, invisibly.

Examples

```
ckpt <- tempfile(fileext = ".txt")
pam_checkpoint_add("a.wav", ckpt)
pam_checkpoint_add("b.wav", ckpt)
pam_checkpoint_read(ckpt)
unlink(ckpt)
```

pam_checkpoint_read *Read the set of already-processed item IDs*

Description

Read the set of already-processed item IDs

Usage

```
pam_checkpoint_read(path)
```

Arguments

path	Path to a plain-text checkpoint file (one ID per line).

Value

Character vector of completed IDs (empty if the file doesn't exist).

Examples

```
ckpt <- tempfile(fileext = ".txt")
pam_checkpoint_read(ckpt)            # no file yet: nothing is done
pam_checkpoint_add(c("a.wav", "b.wav"), ckpt)
pam_checkpoint_read(ckpt)
unlink(ckpt)
```

pam_check_deployments *List problems in the deployment table*

Description

List problems in the deployment table

Usage

```
pam_check_deployments(dep)
```

Arguments

dep	Table from pam_read_deployments().

Value

A data.table deployment_id, grid, issue (empty if none). Issues: unparseable start, missing/unparseable recovery, recovery before deployment, assumed recovery time, and overlap with another deployment of the same grid.

Examples

```
ex <- pam_example_project()
dep <- pam_read_deployments(ex$cfg)
pam_check_deployments(dep)   # no rows: the example deployments are consistent
```

---

pam_check_ffmpeg *Check that FFmpeg is installed and supports the filter chain*

Description

Check that FFmpeg is installed and supports the filter chain

Usage

```
pam_check_ffmpeg(ffmpeg = "ffmpeg", filter = NULL)
```

Arguments

`ffmpeg`	Executable name or full path.
`filter`	Filter string to test on 0.1 s of silence.

Value

The first line of FFmpeg's version output. Aborts with install advice if FFmpeg is missing or too old for the filter.

Examples

```
if (nzchar(Sys.which("ffmpeg"))) {
  pam_check_ffmpeg("ffmpeg", filter = pam_highpass_filter_string(250, 8))
}
```

---

pam_check_model_input *Check model input before fitting*

Description

Runs a set of consistency checks on the output of `pam_model_input()`: dimensions, site and occasion matching, the values of the response and missing values. Use it after editing the input by hand, or to confirm that input is ready before it is passed to a modeling package.

Usage

```
pam_check_model_input(x, error = TRUE)
```

Arguments

x	A pam_model_input.
error	If TRUE (the default), a failed check is an error that names the failed checks. If FALSE, the table of checks is returned whatever the result.

Details

The checks are: the response is a numeric array; sites are unique and match the site covariates (count and names); observation covariates have the shape of the response; the response holds whole numbers of 0 or more (0 or 1 for detection / non-detection); every site was surveyed at least once; observation covariates are present wherever the response is; and, for seasons, the number of seasons and occasions match the response.

Value

A data.table with one row per check: check, passed and detail (invisibly when everything passed).

See Also

pam_model_input()

Examples

```
counts <- matrix(c(0, 2, 5, 1, 0, 3), nrow = 3, dimnames = list(c("a", "b", "c"), NULL))
x <- pam_model_input(pam_as_count_history(counts))
pam_check_model_input(x)
```

---

pam_clean_detections *Apply the cleaning rules to a detections table*

Description

Pure function (no file access), so it can be tested directly. Rules run in a fixed order and each removed detection is kept with its reason.

Usage

```
pam_clean_detections(det, pc, thresholds = NULL, seasons = NULL)
```

Arguments

det	A data.table of detections as returned by pam_assemble_detections(), plus file_duration_s.
pc	The processing section of the config: min_confidence, species_whitelist, species_blacklist, date_range, drop_outside_deployment, max_overrun_s, merge_gap_s.

thresholds	Optional data.table with scientific_name and threshold; overrides min_confidence for those species.
seasons	Optional data.table with scientific_name, month_start, month_end (1-12; start > end wraps over the year end).

Value

A list: clean (one row per event, with confidence = maximum, mean_confidence, n_windows), removed (rows with reason), steps (rule, n_before, n_removed).

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
det <- suppressMessages(pam_assemble_detections(ex$cfg))
det$file_duration_s <- 2    # length of the example recordings, in seconds
cleaned <- pam_clean_detections(det, ex$cfg$processing)
cleaned$steps
cleaned$removed$reason
as.data.frame(cleaned$clean)[, c("species", "confidence", "n_windows", "start_time", "end_time")]
```

---

pam_config_read *Read and validate a pamflow configuration file*

Description

Read and validate a pamflow configuration file

Usage

```
pam_config_read(path)
```

Arguments

path	Path to a YAML config file (required). There is no default: pamflow never looks for a configuration in the working directory.

Details

Missing keys fall back to the template defaults (system.file("extdata", "config_template.yml", package = "pamflow")). A relative paths$project_root is resolved against the folder that holds the config file, not the working directory. The template uses "..", so a config saved as <project>/config/config.yml points at <project>.

Value

A named list of class pam_config, with all paths resolved against paths$project_root.

Examples

```
ex <- pam_example_project()
cfg <- pam_config_read(ex$config)
cfg$birdnet$run_id
cfg$paths$audio_standardized
```

`pam_config_validate` *Validate a pamflow configuration*

Description

Validate a pamflow configuration

Usage

```
pam_config_validate(cfg)
```

Arguments

`cfg`	A config list.

Value

`cfg`, invisibly. Aborts with a clear message on invalid values.

Examples

```
ex <- pam_example_project()
pam_config_validate(ex$cfg)          # fine: returns the config invisibly

bad <- ex$cfg
bad$birdnet$min_confidence <- 2      # must be between 0 and 1
res <- try(pam_config_validate(bad), silent = TRUE)
inherits(res, "try-error")
```

`pam_count_histories` *Count histories of many species at once*

Description

Builds a `pam_count_history()` for every species (or the species you name) in one pass over the detections. All histories share the same sites, occasions, effort and site covariates, which is what multi-species models need.

Usage

```
pam_count_histories(
  cfg,
  species = NULL,
  run_id = cfg$birdnet$run_id,
  bin_days = NULL,
  count = c("events", "files"),
  thresholds = NULL,
  min_effort_hours = NULL,
  site_covariates = NULL,
  min_total = 1
)
```

Arguments

cfg	A pam_config object.
species	Optional species to include: scientific names, common names or species labels (any capitalization). NULL (the default) takes every species in the detections.
run_id	Detection run (sub-folder of paths$detections).
bin_days	Length of an occasion in days. NULL uses occupancy$occasion_length_days (1 if not set).
count	What to count: "events" (the default; detected events, which Stage 4 forms by merging overlapping BirdNET windows) or "files" (the number of recordings with at least one detection, which cannot exceed the number of recordings and is less affected by one animal calling for a long time).
thresholds	Optional per-species thresholds (a data frame, or the path of a CSV file, with scientific_name and threshold), for example from pam_species_thresholds().
min_effort_hours	Minimum recorded hours for an occasion to count as surveyed; others are NA. NULL uses occupancy$min_effort_hours (2 if not set).
site_covariates	Optional data frame of site covariates with a site_id column (for example habitat or elevation). Every site must have a row.
min_total	Species with fewer than this many counts in total, over all sites and occasions, are left out (default 1, which drops species with no count after the thresholds).

Details

Pass the result to pam_model_input() to get a species by site by occasion array for multi-species models, or take one species with histories[["label"]].

Value

An object of class pam_count_histories: a named list (one pam_count_history() per species, named by species label).

See Also

pam_count_history(), pam_model_input()

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
hists <- suppressMessages(pam_count_histories(ex$cfg))
names(hists)
```

---

pam_count_history *Count history of one species: detections per site and occasion*

Description

Counts the detections of one species in each site and occasion of a project and returns them in the form that abundance models need: a matrix with sites in rows and occasions in columns, NA where a site was not surveyed, and matrices of recording effort and days. This is the abundance counterpart of the detection histories of Stage 6.

Usage

```
pam_count_history(
  cfg,
  species,
  run_id = cfg$birdnet$run_id,
  bin_days = NULL,
  count = c("events", "files"),
  thresholds = NULL,
  min_effort_hours = NULL,
  site_covariates = NULL
)
```

Arguments

cfg	A pam_config object.
species	The species, given as scientific name, common name or species label (any capitalization). It must identify exactly one species.
run_id	Detection run (sub-folder of paths$detections).
bin_days	Length of an occasion in days. NULL uses occupancy$occasion_length_days (1 if not set).
count	What to count: "events" (the default; detected events, which Stage 4 forms by merging overlapping BirdNET windows) or "files" (the number of recordings with at least one detection, which cannot exceed the number of recordings and is less affected by one animal calling for a long time).

`thresholds` Optional per-species thresholds (a data frame, or the path of a CSV file, with `scientific_name` and `threshold`), for example from `pam_species_thresholds()`.

`min_effort_hours` Minimum recorded hours for an occasion to count as surveyed; others are `NA`. `NULL` uses `occupancy$min_effort_hours` (2 if not set).

`site_covariates` Optional data frame of site covariates with a `site_id` column (for example habitat or elevation). Every site must have a row.

Details

Sites and occasions follow Stage 6: occasions are blocks of `bin_days` days starting on the first recorded day, shared by all sites, and only occasions with recordings at some site appear. The counts are numbers of detections, not numbers of animals, so models fitted to them give a relative abundance index. Use `pam_model_input()` to turn the history into input for abundance or occupancy models.

Value

An object of class `pam_count_history`: a list with

- `y`: integer matrix of counts (sites x occasions, `NA` = not surveyed);
- `effort`: matrix of recorded hours (`NA` where not surveyed);
- `day`: matrix of days since the first occasion;
- `site_covariates`: data frame with `site_id` and your covariates;
- `occasions`: the start and end day of each occasion;
- `species`, `count`: what was counted.

See Also

`pam_as_count_history()`, `pam_model_input()`

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
sp <- unique(proc$clean$scientific_name)[1]
hist <- suppressMessages(pam_count_history(ex$cfg, species = sp))
hist$y
```

pam_cut_audio *Cut Audio Clip Preserving Source Format*

Description

Extracts a sub-section of an audio file and writes it to disk, preserving the source file extension when possible.

Usage

```
pam_cut_audio(in_path, out_path, from, to)
```

Arguments

in_path	Character. Source audio file path.
out_path	Character. Output file path.
from	Numeric. Start time in seconds.
to	Numeric. End time in seconds.

Value

Invisibly returns out_path.

Examples

```
ex <- pam_example_project()
out <- tempfile(fileext = ".wav")
pam_cut_audio(ex$files[1], out, from = 0.5, to = 1.5)
pam_audio_info(out)$duration_s
unlink(out)
```

pam_define_seasons *Define seasons from the occasions of a history*

Description

Builds a table of seasons (one per calendar year or month) that cover the occasions of a history, for use as the seasons argument of pam_model_input(). Years or months without any occasion between the first and last are included, so the spacing of seasons is kept.

Usage

```
pam_define_seasons(history, by = c("year", "month"))
```

Arguments

history	A pam_count_history or pam_count_histories that has dates for its occasions (as built by pam_count_history()).
by	"year" (the default) or "month".

Value

A data frame with season (label), start and end (Dates).

See Also

pam_model_input()

Examples

```
dates <- as.Date(c("2025-12-20", "2026-01-05", "2026-03-10"))
counts <- matrix(c(1, 0, 2, 0, 1, 1), nrow = 2, dimnames = list(c("a", "b"), NULL))
hist <- pam_as_count_history(
  counts, occasions = data.frame(occasion = 1:3, bin_start = dates, bin_end = dates))
pam_define_seasons(hist, by = "month")
```

---

pam_detect *Stage 3: run the custom BirdNET classifier on all QC-passed recordings*

Description

Reads from the filtered folder (default) or the standardized folder (birdnet$input), runs BirdNET in parallel worker processes, and writes raw detections in batches to <detections>/<run_id>/batches/.

Usage

```
pam_detect(cfg, master = NULL, sample_n = NULL, seed = 1L, force = FALSE)
```

Arguments

cfg	A pam_config object (uses the birdnet section).
master	Optional Stage 1 master table; if NULL it is read from metadata/audio_master.csv.
sample_n	If set, process a random sample of this many files (for pilots). The same seed gives the same sample.
seed	Seed for the pilot sample.
force	If TRUE, delete this run's earlier results and start over.

Details

BirdNET runs in pamflow's Python inference environment (see `pam_birdnet_setup()`) using the `birdnet` Python package, which does its own multiprocessing: `birdnet$workers` worker processes each load the model (an unset value means half of the physical cores). `top_k` limits the species kept per 3 s segment (NULL = all above `min_confidence`). Resumable. A file is marked done (`<run_dir>/done.txt`) only after its batch file is on disk, and files with no detections count as done. If a crash happens between writing a batch and updating the checkpoint, those files are re-run, and `pam_assemble_detections()` removes the duplicates. Per-file outcomes (seconds, errors) go to `file_results.csv`; settings and the `birdnet` package version go to `run_settings_*.yml`. Run with `pam_run_stage("03_detect", pam_detect, cfg)`.

Value

Invisibly, a list: `n_pending`, `n_done`, `n_failed`, `n_detections`, `run_dir`.

References

Kahl, S., Wood, C. M., Eibl, M. and Klinck, H. (2021). BirdNET: A deep learning solution for avian diversity monitoring. *Ecological Informatics*, 61, 101236. doi:10.1016/j.ecoinf.2021.101236

Examples

```
## Not run:
# needs the BirdNET Python environment (pam_birdnet_setup("inference")) and your model
cfg <- pam_config_read("config/config.yml")
res <- pam_run_stage("03_detect", pam_detect, cfg)

## End(Not run)
```

---

pam_detection_table *Table of detections and recording effort per site, time bin and species*

Description

The shared starting point for activity patterns, richness and abundance indices. It counts the cleaned detections (Stage 4) of every species in every site and time bin, and puts the recording effort next to the counts, so rates per recorded hour can be compared between sites that were recorded for different lengths of time.

Usage

```
pam_detection_table(
  cfg,
  run_id = cfg$birdnet$run_id,
  bin_days = NULL,
  thresholds = NULL,
  min_effort_hours = NULL,
```

```
  extra_by = character(),
  complete = TRUE,
  write = FALSE
)
```

Arguments

cfg	A pam_config object.
run_id	Detection run (sub-folder of paths$detections).
bin_days	Length of a time bin in days. NULL (the default) uses occupancy$occasion_length_days from the config (1 if not set); use 7 for weekly bins.
thresholds	Optional per-species thresholds: a data frame, or the path of a CSV file, with scientific_name and threshold (for example the result of pam_species_thresholds()). They are applied to the event confidence on top of Stage 4, so they can only remove more events. To keep events below the Stage 4 cut-off, lower it there and run Stage 4 again.
min_effort_hours	Minimum recorded hours for a bin to count as surveyed. NULL uses occupancy$min_effort_hours (2 if not set).
extra_by	Optional names of further detection columns to count by, for example a call type, if your detections have one.
complete	If TRUE (the default), every surveyed site and bin gets a row for every species, with 0 where it was not detected. Set to FALSE for very large projects to keep only rows with detections.
write	If TRUE, also save the table as <outputs>/<run_id>/detection_table.csv. Nothing is written by default.

Details

Sites and effort:

Sites follow the occupancy stage: the grid name, or grid_deploymentId for grids with several deployments. Effort counts recordings that passed QC and lie inside a deployment window. Detections that fall in a site and bin with no effort (for example, in a recording that failed QC) are dropped with a warning.

Time bins:

All bins start on the first day with any recording in the project, so a bin covers the same dates at every site.

Value

A data.table with one row per site, bin and species (and extra_by column): site_id, bin_start, bin_end, species, scientific_name, n_detections (events), n_files (recordings with a detection), max_confidence, effort_hours, n_days, sufficient (the bin reached the minimum effort) and detections_per_hour. Bins below the minimum effort are kept and flagged, not dropped. The bin length and first bin day are stored in the attributes bin_days and origin.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
tab <- suppressMessages(pam_detection_table(ex$cfg))
as.data.frame(tab)[, c("site_id", "bin_start", "species", "n_detections", "effort_hours")]
# two-day bins instead of one-day bins
tab2 <- suppressMessages(pam_detection_table(ex$cfg, bin_days = 2))
unique(tab2$bin_start)
```

---

pam_example_project *Create a small example project to try pamflow on*

Description

Builds a tiny, synthetic pamflow project in a temporary folder: six two-second WAV files (pure tones, 8 kHz, mono) from two recording grids with pamflow-style file names, a deployment sheet, a configuration file, and a BirdNET-style detections file for three species. It is meant for the examples in the help pages, for tests, and for getting to know the workflow without any field data. It needs only R packages that pamflow already depends on: no FFmpeg, Python or internet connection.

Usage

```
pam_example_project(dir = tempfile("pamflow_example_"))
```

Arguments

`dir`	Folder to create the project in. The default is a new folder inside `tempdir()`, which R removes when the session ends. If the folder exists it must be empty.

Details

The project has this layout (the other folders are created when stages run):

```
<root>/
  config/config.yml
  metadata/deployments.csv
  data/standardized/G0059/ (3 files)
  data/standardized/G0110/ (3 files)
  data/detections/run01/batches/batch_001.csv
```

Grid `G0059` was recorded on 2026-04-05 at 18:30 and grid `G0110` on 2026-04-06 at 06:00; every file lies inside its deployment window, so a scan finds no problems.

The detections file looks like the batch files that `pam_detect()` writes (three species, 3-second windows). It has a few things for the cleaning stage to deal with: overlapping windows to merge,

low-confidence rows, a window past the end of its recording, and one duplicated row. The stages after the scan need the master table, so run pam_scan_audio() on ex$cfg first; the help-page examples show how.

The configuration only sets what differs from the defaults: occupancy needs no minimum recording effort and one detection per species, recall segments are 1 second long because the recordings are only 2 seconds long, and the filter stage uses a single worker.

Value

A list with:

root The project folder.

config Path to config/config.yml.

cfg The configuration, read with pam_config_read().

audio_dir Folder with the standardized audio.

files Paths of the six WAV files.

deployments Path to the deployment sheet.

detections Path to the detections batch file.

Examples

```
ex <- pam_example_project()
basename(ex$files)
res <- suppressMessages(pam_scan_audio(ex$cfg))
res$summary$overall
```

---

pam_extract_grid *Extract Grid, Recorder, or Site Identifier from Filename*

Description

Helper function to parse site/grid/recorder identifier from filename using pattern capture groups or leading token prior to first underscore.

Usage

```
pam_extract_grid(filename, pattern = NULL)
```

Arguments

filename	Character, filename or base name.
pattern	Optional regular expression pattern (e.g., cfg$filenames$pattern).

Value

Character string identifier or "unassigned".

Examples

```
pam_extract_grid("G0059_20260405_183000.wav")
pam_extract_grid("recording.wav")   # no recognisable identifier
```

`pam_filter_audio` *Stage 2: high-pass filter all QC-passed recordings*

Description

Applies a Butterworth high-pass (default 250 Hz, 48 dB/octave) with FFmpeg and writes filtered copies (WAV or FLAC) to `paths$filtered`, mirroring the folder structure and file names of `paths$audio_standardized`. Originals are never modified.

Usage

```
pam_filter_audio(cfg, master = NULL, force = FALSE)
```

Arguments

`cfg`	A `pam_config` object. Uses the `filter` section and paths.
`master`	Optional master table from `pam_scan_audio()$master`; if NULL it is read from `metadata/audio_master.csv`.
`force`	If TRUE, ignore the checkpoint and re-filter everything.

Details

Resumable: completed files are recorded in `metadata/02_filter.done` only after the output has been written and verified (readable, same duration, sample rate and channels). Per-file outcomes, including FFmpeg error text, are appended to `metadata/02_filter_results.csv`. Only files with `qc_pass == TRUE` are filtered. Format (WAV vs FLAC) is preserved from source files. Run with `pam_run_stage("02_filter", pam_filter_audio, cfg)`.

Value

Invisibly, a list: `n_pending`, `n_ok`, `n_failed`, `results_file`.

Examples

```
if (nzchar(Sys.which("ffmpeg"))) {
  ex <- pam_example_project()
  # the later stages read the master table that the scan builds
  res <- suppressMessages(pam_scan_audio(ex$cfg))
  out <- suppressMessages(pam_filter_audio(ex$cfg))
  out$n_ok
  basename(list.files(ex$cfg$paths$filtered, recursive = TRUE))
}
```

pam_filter_one *Filter one audio file with FFmpeg (WAV or FLAC)*

Description

Base R only, so it can be shipped to parallel workers without the package installed. Writes to `<output>.part` and renames on success, so an interrupted run never leaves a half-written file that looks finished. Preserves the file format (WAV or FLAC) based on output extension.

Usage

```
pam_filter_one(item, filter, ffmpeg = "ffmpeg")
```

Arguments

`item`	List with `input`, `output` (paths) and `bits` (input bit depth; 16, 24 or 32 are preserved for WAV, FLAC outputs `-c:a flac`).
`filter`	Filter string from `pam_highpass_filter_string()`.
`ffmpeg`	Executable name or path.

Value

A list: `ok` (logical) and `message` (last FFmpeg error lines, or "").

Examples

```
if (nzchar(Sys.which("ffmpeg"))) {
  ex <- pam_example_project()
  item <- list(input = ex$files[1], output = tempfile(fileext = ".wav"), bits = 16)
  res <- pam_filter_one(item, pam_highpass_filter_string(250, 8))
  res$ok
  unlink(item$output)
}
```

pam_highpass_filter_string *Build the FFmpeg filter string for a Butterworth high-pass*

Description

FFmpeg's `highpass` filter supports only 2 poles per instance, so higher orders are built as a cascade of second-order sections, each with the Q value that yields a true Butterworth response (maximally flat passband, -3 dB at the cutoff). Four identical sections would NOT give this: the cutoff would sit near -12 dB.

Usage

```
pam_highpass_filter_string(cutoff_hz = 250, order = 8L)
```

Arguments

cutoff_hz	Cutoff frequency in Hz (-3 dB point).
order	Even filter order. 8 = 48 dB/octave.

Value

A single character string for `ffmpeg -af`.

Examples

```
pam_highpass_filter_string(250, 8)
```

---

pam_inspect_raw *Inspect Raw Audio Directory*

Description

Recursively scans a raw data directory, classifies files (audio, short clips/memos, non-audio, corrupt), extracts audio properties, and parses recording datetimes.

Usage

```
pam_inspect_raw(
  raw_dir,
  recursive = TRUE,
  min_duration_s = 60,
  timezone = "Asia/Kolkata",
  pattern = NULL
)
```

Arguments

raw_dir	Path to raw data folder.
recursive	Logical, scan subfolders recursively?
min_duration_s	Minimum duration in seconds for standard audio clips (default 60s).
timezone	Timezone string for timestamp parsing.
pattern	Optional filename regex pattern.

Value

data.table containing detailed file metadata and triage categorization.

Examples

```
ex <- pam_example_project()
# the example audio stands in for a raw dump; recordings under 1 s would count as memos
insp <- suppressMessages(pam_inspect_raw(ex$audio_dir, min_duration_s = 1))
as.data.frame(insp)[, c("filename", "duration_s", "grid", "category")]
```

---

pam_is_flac *Check if a File is a FLAC File*

Description

Check if a File is a FLAC File

Usage

```
pam_is_flac(path)
```

Arguments

path	Character. File path or vector of paths.

Value

Logical scalar or vector.

Examples

```
pam_is_flac(c("a.flac", "b.wav"))
```

---

pam_list_audio *List Audio Files in a Directory (WAV + FLAC)*

Description

List Audio Files in a Directory (WAV + FLAC)

Usage

```
pam_list_audio(dir, recursive = TRUE)
```

Arguments

dir	Character. Directory to scan.
recursive	Logical. Recurse into subdirectories.

Value

Character vector of file paths.

Examples

```
ex <- pam_example_project()
basename(pam_list_audio(ex$audio_dir))
```

`pam_load_config` *Alias for pam_config_read*

Description

Alias for pam_config_read

Usage

```
pam_load_config(path)
```

Arguments

`path`	Path to config file (required).

Value

pam_config object.

Examples

```
ex <- pam_example_project()
cfg <- pam_load_config(ex$config)
cfg$filenames$timezone
```

`pam_log` *Write a timestamped log line to a per-stage log file*

Description

Write a timestamped log line to a per-stage log file

Usage

```
pam_log(
  level = c("INFO", "WARN", "ERROR"),
  msg,
  stage = "pamflow",
  log_dir = NULL,
  echo = TRUE
)
```

Arguments

level	One of "INFO", "WARN", "ERROR".
msg	Character message.
stage	Stage name, used as the log file name (<stage>.log).
log_dir	Directory for log files (created if missing). If NULL, file logging is skipped.
echo	Also print to the console.

Value

The log line, invisibly.

Examples

```
log_dir <- tempfile("pamflow_logs_")
pam_log("INFO", "scan started", stage = "01_scan", log_dir = log_dir)
readLines(file.path(log_dir, "01_scan.log"))
unlink(log_dir, recursive = TRUE)
```

---

pam_match_deployments *Attach recorder and coordinates by matching grid and time window*

Description

Attach recorder and coordinates by matching grid and time window

Usage

```
pam_match_deployments(master, dep)
```

Arguments

master	Master audio table with file_path, grid, start_time.
dep	Table from pam_read_deployments().

Value

master with deployment_id, recorder, latitude, longitude and in_deployment. Recordings outside every deployment window of their grid get NA and in_deployment = FALSE (not dropped).

Examples

```
ex <- pam_example_project()
cfg <- ex$cfg
master <- pam_parse_filenames(ex$files, cfg$filenames$pattern, cfg$filenames$timezone)
master <- pam_match_deployments(master, pam_read_deployments(cfg))
as.data.frame(master)[, c("file_name", "deployment_id", "recorder", "in_deployment")]
```

pam_model_input *Prepare a count history as input for abundance and occupancy models*

Description

Turns a count history into the pieces that site-occupancy and abundance models use: the response `y` (sites by occasions), the site covariates `siteCovs` and the observation covariates `obsCovs`. It handles one species or several, and one season or several. pamflow does not fit these models; use the prepared input with the package you prefer, such as `unmarked`, `ubms` or `spOccupancy`.

Usage

```
pam_model_input(
  history,
  binary = FALSE,
  standardize = FALSE,
  drop_unsurveyed = TRUE,
  seasons = NULL,
  season_covariates = NULL,
  dir = NULL
)
```

Arguments

`history`	A `pam_count_history` from `pam_count_history()` or `pam_as_count_history()`, or a `pam_count_histories` from `pam_count_histories()`.
`binary`	If `FALSE` (the default), `y` holds the counts, as used by N-mixture (abundance) models. If `TRUE`, `y` is detection (1) or non-detection (0), as used by occupancy and Royle-Nichols models.
`standardize`	If `TRUE`, numeric covariates are centered and scaled (mean 0, standard deviation 1), which helps models converge. The center and scale are returned in `scaling` so that new data can be treated the same way. Default `FALSE`: covariates are returned as they are.
`drop_unsurveyed`	If `TRUE` (the default), sites with no surveyed occasion are left out of every piece, because model functions cannot use them.
`seasons`	Optional data frame with the columns `season` (a name), `start` and `end` (Dates) that groups the occasions into seasons (primary periods) for multi-season models; see `pam_define_seasons()`. An occasion belongs to the season that contains its first day. Occasions outside every season are left out; seasons with fewer occasions than the longest are padded with `NA`.
`season_covariates`	Optional data frame with a `season` column and one column per covariate that changes between seasons but not between sites (for example rainfall or year); needs `seasons`.
`dir`	Optional folder. If given, the pieces are also written there as CSV files, for use outside R. Nothing is written by default.

Details

Using the result with unmarked or ubms:

Both build their data frames from these pieces:

```
x <- pam_model_input(history)
umf <- unmarked::unmarkedFramePCount(y = x$y, siteCovs = x$siteCovs, obsCovs = x$obsCovs)
fit <- unmarked::pcount(~ effort ~ canopy, data = umf, K = 100)

x <- pam_model_input(history, binary = TRUE)
umf <- unmarked::unmarkedFrameOccu(y = x$y, siteCovs = x$siteCovs, obsCovs = x$obsCovs)
fit <- unmarked::occu(~ effort ~ canopy, data = umf)

# several seasons
x <- pam_model_input(history, binary = TRUE, seasons = pam_define_seasons(history))
umf <- unmarked::unmarkedMultFrame(y = x$y, siteCovs = x$siteCovs, obsCovs = x$obsCovs,
                                   numPrimary = x$num_primary)
```

The detection formula (before the second ~) may use the observation covariates `effort` and `day` and any site covariate; the other formula uses site covariates. `ubms` takes the same `unmarkedFrame*` objects.

Using the result with spOccupancy:

`spOccupancy` expects a list with the response, the occupancy covariates and the detection covariates, which are the same pieces under other names (check the argument names against its documentation):

```
x <- pam_model_input(history, binary = TRUE)          # one species
data <- list(y = x$y, occ.covs = x$siteCovs, det.covs = x$obsCovs,
             coords = coords)   # coords: your site coordinates, same row order

x <- pam_model_input(histories, binary = TRUE)        # several species
data <- list(y = x$y, occ.covs = x$siteCovs, det.covs = x$obsCovs, coords = coords)
```

For several species `y` has the species first (species x sites x occasions). With seasons use `y_array` and `obsCovs_array`. `x$site_ids` gives the site order.

About counts:

Counts are numbers of detections, not numbers of animals. A model fitted to them gives a relative abundance index, not a population size. N-mixture models also slow down when counts are large (the upper bound `K` must exceed the highest count), so `count = "files"` in `pam_count_history()` or shorter occasions are often better than raw event counts.

Site covariates with missing values cannot be used by model functions; a warning names them.

Value

An object of class `pam_model_input`. For one species and no seasons it is a list with

- `y`: integer matrix, sites in rows and occasions in columns, `NA` where a site was not surveyed;
- `siteCovs`: data frame of site covariates, one row per site (row names are the site identifiers); `NULL` if there are none;

- obsCovs: named list of matrices shaped like y: effort (recorded hours, when the history has it) and day (days since the first occasion);
- site_ids, species, count, binary;
- scaling: center and scale of every standardized covariate.

With several species, y is an array (species, sites, occasions) and species lists the species. With seasons, the list also has

- y_array: sites x seasons x occasions within season (with a leading species dimension for several species); y is then the wide matrix sites x (seasons * occasions) for one species, and NULL for several;
- obsCovs (wide matrices) and obsCovs_array (arrays), shaped like y and y_array;
- num_primary (number of seasons), n_secondary (occasions per season, including padding), seasons (with the number of occasions found) and occasion_map (which original occasion went where);
- with season_covariates: yearlySiteCovs (data frame, one row per site and season, sites first) and yearlySiteCovs_matrices (sites x seasons).

The layout of y is stored in attr(x, "layout"); pam_check_model_input() uses it.

See Also

pam_count_history(), pam_count_histories(), pam_define_seasons(), pam_check_model_input()

Examples

```
counts <- matrix(c(0, 2, 5, 1, 0, 3, 4, 0, 2, 1, 1, 0), nrow = 3,
                 dimnames = list(c("site_a", "site_b", "site_c"), NULL))
hist <- pam_as_count_history(counts, site_covariates = data.frame(canopy = c(0.2, 0.5, 0.9)),
                             effort = matrix(24, 3, 4), species = "Example species")
x <- pam_model_input(hist)
x$y
x$siteCovs
names(x$obsCovs)
# detection / non-detection instead of counts, covariates standardized
pam_model_input(hist, binary = TRUE, standardize = TRUE)$y
```

---

pam_parse_datetime *Parse Datetime from Audio Filename or Metadata*

Description

Automatically detects and extracts recording timestamp from filename (ISO formats, 14-digit, 12-digit) or falls back to file modification time. Timestamps are returned as local time strings.

Usage

```
pam_parse_datetime(filename, file_path = NULL, timezone = "Asia/Kolkata")
```

Arguments

filename	Character, filename or base name.
file_path	Optional character path to file for metadata fallback.
timezone	Character string timezone (e.g. "Asia/Kolkata").

Value

Character string timestamp in "YYYYMMDD_HHMMSS" format.

Examples

```
pam_parse_datetime("G0059_20260405_183000.wav")
pam_parse_datetime("SM4_260405183000.wav")   # 12-digit stamp: yymmddhhmmss
```

---

pam_parse_filenames *Parse grid and start time from recording filenames*

Description

Parse grid and start time from recording filenames

Usage

```
pam_parse_filenames(files, pattern, timezone)
```

Arguments

files	Character vector of file paths.
pattern	PCRE regex with named groups grid, date (YYYYMMDD) and time (HHMMSS), applied to the file name only.
timezone	Olson timezone of the filename timestamps (local time).

Value

A data.table with file_path, file_name, grid, start_time (POSIXct) and valid_name. Files that don't match get NA and valid_name = FALSE; they are never dropped.

Examples

```
ex <- pam_example_project()
# the second name does not match the pattern, so it is flagged, not dropped
pam_parse_filenames(c(ex$files[1], "not_a_recording.wav"),
                    pattern = ex$cfg$filenames$pattern,
                    timezone = "Asia/Kolkata")
```

pam_pending_batches *Split pending work into batches, skipping completed items*

Description

Split pending work into batches, skipping completed items

Usage

```
pam_pending_batches(all_ids, checkpoint_path, batch_size = 500L)
```

Arguments

all_ids	Character vector of every ID to process.
checkpoint_path	Checkpoint file path.
batch_size	Number of IDs per batch.

Value

A list of character vectors (empty list if nothing is pending).

Examples

```
ckpt <- tempfile(fileext = ".txt")
ids <- c("a.wav", "b.wav", "c.wav")
pam_pending_batches(ids, ckpt, batch_size = 2)   # all three are pending
pam_checkpoint_add("a.wav", ckpt)
pam_pending_batches(ids, ckpt, batch_size = 2)   # only b and c are left
unlink(ckpt)
```

pam_plot_activity *Plot activity patterns*

Description

Draws the result of pam_activity() with one panel per species: bars for the hour of the day, a line for days, weeks and months.

Usage

```
pam_plot_activity(activity, species = NULL, relative = FALSE)
```

Arguments

activity	A pooled table from pam_activity(by_site = FALSE).
species	Optional species (scientific or common name as in the table) to plot; NULL plots all.
relative	If TRUE, plot relative_activity (1 = the species' peak) instead of detections per recorded hour, so species can be compared by timing.

Value

A ggplot object.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
act <- suppressMessages(pam_activity(ex$cfg, by = "hour"))
p <- pam_plot_activity(act)
class(p)
```

---

pam_plot_richness *Plot species richness*

Description

Plot species richness

Usage

```
pam_plot_richness(x, type = c("accumulation", "sites"))
```

Arguments

x	The result of pam_richness().
type	"accumulation" (the default) draws the expected species accumulation over sites with the Chao2 estimate as a dashed line; "sites" draws richness per site, highest first.

Value

A ggplot object.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
rich <- suppressMessages(pam_richness(ex$cfg))
p <- pam_plot_richness(rich, "sites")
class(p)
```

---

pam_process_detections

Stage 4: clean the detections of a run

Description

Assembles the run's detections, applies `pam_clean_detections()` using the `processing` settings of the config, and writes the results to `<detections>/<run_id>/processed/`.

Usage

```
pam_process_detections(cfg, run_id = cfg$birdnet$run_id)
```

Arguments

`cfg`	A `pam_config` object.
`run_id`	Detection run (sub-folder of paths$detections). Defaults to birdnet$run_id.

Details

Writes `detections_clean.csv`, `removed_detections.csv` (with the reason for each removal) and `processing_steps.csv`. Times are written as local-time text. Run with `pam_run_stage("04_process", pam_process_detections, cfg, run_id = "...")`.

Value

A list: `clean`, `removed`, `steps` (see `pam_clean_detections()`).

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
out <- suppressMessages(pam_process_detections(ex$cfg))
out$steps
# the results are also written to the run folder
list.files(file.path(ex$cfg$paths$detections, "run01", "processed"))
```

pam_read_audio *Read Audio File as a tuneR Wave Object (WAV or FLAC)*

Description

Decodes FLAC to a temporary WAV internally when needed. Returned object is always a standard `tuneR::Wave` S4 object usable by `seewave` and `soundecology`.

Usage

```
pam_read_audio(path, from = NULL, to = NULL)
```

Arguments

`path`	Character. Audio file path.
`from`	Numeric. Optional start time in seconds.
`to`	Numeric. Optional end time in seconds.

Value

A `tuneR::Wave` S4 object.

Examples

```
ex <- pam_example_project()
w <- pam_read_audio(ex$files[1], from = 0.5, to = 1.5)
length(w@left) / w@samp.rate   # seconds read
```

pam_read_deployments *Read and clean the deployment sheet*

Description

Read and clean the deployment sheet

Usage

```
pam_read_deployments(cfg)
```

Arguments

`cfg`	A `pam_config` object. Uses `paths$deployment_sheet`, the `deployments` settings and `filenames$timezone`.

Details

Columns are matched by position (deployments$columns); everything is read as text so leading zeros survive. Repairs applied (and nothing else): "/" and "." date separators become "-", "." in times becomes ":", seconds are dropped, a blank recovery time becomes deployments$default_recovery_time, and a recovery year before 2000 (e.g. 15-04-0206) is replaced by the deployment year. Rows that still can't be parsed get NA and are listed by pam_check_deployments().

Value

A data.table with the configured columns plus deploy_start, retrieve_end (POSIXct, local time) and recovery_time_missing. grid is converted to the filename format (0059 -> "G0059").

Examples

```
ex <- pam_example_project()
dep <- pam_read_deployments(ex$cfg)
as.data.frame(dep)[, c("deployment_id", "grid", "deploy_start", "retrieve_end")]
```

---

pam_recall_draw *Draw a round of random segments for ground-truth annotation*

Description

Samples segments uniformly at random from the recordings, adds them to <paths$validation>/ground_truth/recall_se and never draws the same segment twice. The ground truth belongs to the audio, not to a detection run, so it can be reused to compare runs (for example filtered against unfiltered audio).

Usage

```
pam_recall_draw(cfg, n = cfg$recall$n_segments, run_id = cfg$birdnet$run_id)
```

Arguments

cfg	A pam_config object (uses the recall section).
n	Segments to draw in this round.
run_id	Detection run. Only recordings that run processed are sampled, because recall is measured against that run. NULL samples all recordings.

Value

Invisibly, the newly added rows.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
# run_id = NULL: sample all recordings (a real run would restrict to the files it processed)
new <- suppressMessages(pam_recall_draw(ex$cfg, n = 4, run_id = NULL))
as.data.frame(new)[, c("segment_id", "rel", "seg_start", "seg_end")]
```

---

pam_recall_export *Export audio and spectrograms for the ground-truth segments*

Description

Cuts each segment from the audio chosen by `recall$audio` ("standardized" by default, so the truth does not depend on the high-pass filter) and draws a wide spectrogram. Existing files are kept.

Usage

```
pam_recall_export(cfg, force = FALSE)
```

Arguments

`cfg`	A `pam_config` object.
`force`	Recreate all files.

Value

Invisibly, list(n_ok, n_failed, failed_ids).

Examples

```
if (nzchar(Sys.which("ffmpeg"))) {
  ex <- pam_example_project()
  # the later stages read the master table that the scan builds
  res <- suppressMessages(pam_scan_audio(ex$cfg))
  new <- suppressMessages(pam_recall_draw(ex$cfg, n = 3, run_id = NULL))
  out <- suppressMessages(pam_recall_export(ex$cfg))
  out$n_ok
}
```

pam_recall_frame *All possible annotation segments of the recordings*

Description

Cuts every QC-passed, in-deployment recording into consecutive, non-overlapping segments of `segment_s` seconds (a final piece shorter than a segment is dropped).

Usage

```
pam_recall_frame(master, segment_s = 60, hours = NULL)
```

Arguments

`master`	Table with `rel`, `grid`, `recording_start` (POSIXct, local), `duration_s`, `qc_pass`, `in_deployment`.
`segment_s`	Segment length in seconds.
`hours`	Optional c(from, to): keep only segments whose local start hour is >= from and < to (e.g. c(4, 18)).

Value

A data.table with one row per segment: `rel`, `grid`, `recording_start`, `seg_start`, `seg_end` (seconds into the file), `slot_time`, `hour`, `seg_key`.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
master <- res$master
master$rel <- file.path(master$grid, master$file_name)  # path relative to the audio folder
master$recording_start <- master$start_time
# the example recordings are 2 s long, so use 1 s segments (the default is 60 s)
frame <- pam_recall_frame(master, segment_s = 1)
nrow(frame)
as.data.frame(frame)[1:3, c("rel", "seg_start", "seg_end")]
```

`pam_recall_metrics` *Segment-level recall, precision and F1 against ground truth*

Description

The unit is a segment x species: a species is "detected" in a segment if the detector has an event of that species overlapping the segment at or above the threshold, and "present" if the annotator marked it. Possibly-present pairs are left out of the counts.

Usage

```
pam_recall_metrics(
  truth,
  segments,
  events,
  classes = NULL,
  thresholds = c(0.3, 0.4, 0.5, 0.7)
)
```

Arguments

`truth`	Annotation table (`segment_id`, `present`, `possible`).
`segments`	Segment table (`segment_id`, `rel`, `seg_start`, `seg_end`).
`events`	Detector events from `pam_validation_pool()` (`rel`, `start_time`, `end_time`, `scientific_name`, `confidence`).
`classes`	Scientific names of the classifier's classes.
`thresholds`	Confidence thresholds to evaluate.

Details

With simple random sampling of segments the estimates apply to the sampled recordings and hours. A species that is rarely present in random segments has few `n_present` and an uncertain recall.

Value

A data.table with one row per species (and "ALL", summed over species) and threshold: `tp`, `fn`, `fp`, `tn`, `n_present`, `recall`, `recall_low`, `recall_high`, `precision`, `precision_low`, `precision_high`, `f1`.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
segments <- suppressMessages(pam_recall_draw(ex$cfg, n = 12, run_id = NULL))
events <- suppressMessages(pam_validation_pool(ex$cfg))
# normally the annotation app records what an expert hears; here we pretend the
```

```
# parakeet is present in every segment
truth <- data.table::data.table(segment_id = segments$segment_id,
                                present = "Psittacula longicauda_Long-tailed Parakeet",
                                possible = NA_character_)
m <- pam_recall_metrics(truth, segments, events, thresholds = c(0.3, 0.5))
m <- as.data.frame(m)
m[m$scientific_name == "ALL", c("threshold", "n_present", "tp", "fn", "fp", "recall", "precision")]
```

`pam_recall_report` *Recall report for a detection run*

Description

Compares a run's detections with the ground-truth annotations and writes `recall_by_species_threshold.csv` to `<detections>/<run_id>/validation/`. Only annotated segments from recordings the run processed are used.

Usage

```
pam_recall_report(
  cfg,
  run_id = cfg$birdnet$run_id,
  thresholds = NULL,
  pool = NULL
)
```

Arguments

`cfg`	A `pam_config` object.
`run_id`	Detection run.
`thresholds`	Confidence thresholds. Default: the band edges of `validation$bands` except the last.
`pool`	Optional pool from `pam_validation_pool()`.

Value

A list: `metrics` (see `pam_recall_metrics()`), `n_segments`, `n_no_species`, `n_other_only`.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
segments <- suppressMessages(pam_recall_draw(ex$cfg, n = 12, run_id = NULL))
gt_dir <- file.path(ex$cfg$paths$validation, "ground_truth")
# normally the annotation app writes this file; here the parakeet is "heard" everywhere
truth <- data.frame(segment_id = segments$segment_id,
                    present = "Psittacula longicauda_Long-tailed Parakeet",
```

```
                    possible = "", other = "", none_audible = FALSE, note = "",
                    reviewer = "example", reviewed_at = format(Sys.time()),
                    stringsAsFactors = FALSE)
utils::write.csv(truth, file.path(gt_dir, "recall_annotations.csv"), row.names = FALSE)
rep <- suppressMessages(pam_recall_report(ex$cfg))
rep$n_segments
```

`pam_recall_sample` *Simple random sample of segments*

Description

Simple random sample of segments

Usage

```
pam_recall_sample(frame, n, seed = 1L, exclude_keys = character())
```

Arguments

`frame`	Segments from `pam_recall_frame()`.
`n`	Number of segments.
`seed`	Random seed.
`exclude_keys`	`seg_keys` already drawn; never drawn again.

Value

A data.table of `min(n, available)` segments.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
master <- res$master
master$rel <- file.path(master$grid, master$file_name)  # path relative to the audio folder
master$recording_start <- master$start_time
frame <- pam_recall_frame(master, segment_s = 1)
pick <- pam_recall_sample(frame, n = 3, seed = 1)
as.data.frame(pick)[, c("rel", "seg_start", "seg_end")]
```

pam_report *Generate PAM workflow reports*

Description

Gathers outputs from all completed stages and renders summary reports in HTML, PDF and/or Excel format.

Usage

```
pam_report(
  cfg,
  run_id = cfg$birdnet$run_id,
  formats = cfg$report$formats,
  output_dir = cfg$paths$reports,
  force = FALSE
)
```

Arguments

cfg	A list from pam_config_read.
run_id	Character. The detection run to report on. Defaults to cfg$birdnet$run_id.
formats	Character vector. One or more of "html", "pdf", "excel". Defaults to cfg$report$formats.
output_dir	Character. Directory for the output files. Defaults to cfg$paths$reports.
force	Logical. Overwrite existing reports? Default FALSE.

Details

HTML and PDF are rendered from an R Markdown template using rmarkdown::render. PDF requires a LaTeX installation (e.g. tinytex::install_tinytex()). If PDF rendering fails, a warning is issued and the other formats are still produced.

Excel output is a multi-sheet workbook with one sheet per report section, created with openxlsx.

Besides recording effort, detections, validation and occupancy, the report includes species richness and diversity per site (with the species accumulation curve and the Chao2 estimate), activity patterns (detections per recorded hour by hour of the day and by week) and the per-species thresholds saved by pam_species_thresholds(). A section is left out when its inputs are not available.

Value

Invisibly, a character vector of generated file paths.

Examples

```
if (requireNamespace("rmarkdown", quietly = TRUE) && rmarkdown::pandoc_available()) {
  ex <- pam_example_project()
  res <- suppressMessages(pam_scan_audio(ex$cfg))
  proc <- suppressMessages(pam_process_detections(ex$cfg))
  occ <- suppressMessages(pam_build_occupancy(ex$cfg))
  files <- suppressMessages(pam_report(ex$cfg, formats = "html"))
  basename(files)
}
```

`pam_richness` *Species richness and diversity per site and occasion*

Description

Turns the cleaned detections into species richness and diversity: for every site and time bin, for every site over the whole survey, and across sites (species accumulation and an estimate of how many species were missed). The site table can be joined to habitat covariates by `site_id` to study how richness varies with the environment.

Usage

```
pam_richness(
  cfg,
  run_id = cfg$birdnet$run_id,
  bin_days = NULL,
  min_events = 1,
  abundance = c("counts", "presence"),
  q = c(1, 2),
  thresholds = NULL,
  species = NULL,
  min_effort_hours = NULL,
  write = FALSE
)
```

Arguments

`cfg`	A `pam_config` object.
`run_id`	Detection run (sub-folder of paths$detections).
`bin_days`	Length of a time bin in days. `NULL` uses `occupancy$occasion_length_days` (1 if not set).
`min_events`	A species counts as present in a site and bin only when it has at least this many events there (a whole number, default 1). A higher number removes species known only from a lone, possibly false, detection, at the cost of missing rare species.

abundance	How species are weighted in the Hill numbers: "counts" (the default; the number of events, an activity-based measure) or "presence" (every present species weighs the same, so only occurrence matters).
q	Orders of the Hill numbers to compute besides richness (default 1 and 2). Richness is the Hill number of order 0 and is always returned.
thresholds	Optional per-species thresholds (a data frame, or the path of a CSV file, with scientific_name and threshold), for example from pam_species_thresholds().
species	Optional species to keep (scientific names, common names or species labels). NULL keeps all.
min_effort_hours	Minimum recorded hours for a bin to count as surveyed. NULL uses occupancy$min_effort_hours (2 if not set).
write	If TRUE, also save the four tables to <outputs>/<run_id>/ as richness_occasions.csv, richness_sites.csv, species_accumulation.csv and richness_estimates.csv.

Details

Hill numbers:

Hill numbers describe diversity as an effective number of species. Order 1 is the number of equally common species that gives the same Shannon diversity, and order 2 the same for the Simpson index; larger orders give more weight to the dominant species. With abundance = "counts" a loud, vocal species weighs more than a quiet one, so the numbers describe the acoustic community; use "presence" when only occurrence matters. A bin with no species has richness 0 and NA for the other orders.

Accumulation and the estimate:

Sites are the sampling units. accumulation is the exact expected number of species found in n randomly chosen surveyed sites (sample-based rarefaction), so curves from different projects can be compared at the same number of sites. The Chao2 estimate uses how many species were found in only one or two sites to estimate how many remain undetected; treat it as a lower bound. A species with a false-positive detection counts as present, so validated thresholds and min_events matter for all of these numbers.

Value

A list of class pam_richness:

- occasions: one row per site and bin: richness, hill_q<q> columns, n_events (events of the species counted as present), effort_hours, n_days, sufficient. Bins below the minimum effort are kept and flagged.
- sites: one row per site, using only surveyed bins: richness, hill_q<q>, n_events, n_occasions_surveyed, effort_hours.
- accumulation: the expected number of species (richness) for n_sites = 1 to the number of surveyed sites.
- estimates: one row with the observed richness, the Chao2 estimate of the total and completeness (observed divided by estimated).
- settings: the choices that were used.

See Also

pam_plot_richness(), pam_detection_table()

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
rich <- suppressMessages(pam_richness(ex$cfg))
as.data.frame(rich$sites)[, c("site_id", "richness", "hill_q1", "hill_q2")]
rich$estimates
# count a species only when it has at least two events in a bin
strict <- suppressMessages(pam_richness(ex$cfg, min_events = 2))
strict$sites$richness
```

---

pam_run_stage *Run a workflow stage with logging and error isolation*

Description

Logs start time, end time, duration and every warning, and converts errors into a logged failure instead of terminating the whole workflow.

Usage

```
pam_run_stage(stage, fn, cfg, ...)
```

Arguments

stage	Stage name (e.g. "03_detect").
fn	Function taking cfg as its first argument.
cfg	A pam_config object.
...	Passed on to fn.

Value

A list: status ("ok" or "failed"), result, duration (difftime), warnings (character).

Examples

```
ex <- pam_example_project()
res <- suppressMessages(pam_run_stage("demo", function(cfg, x) x * 2, ex$cfg, x = 21))
res$status
res$result

# an error is logged and returned as a failed status; it does not stop your script
bad <- suppressMessages(pam_run_stage("demo_fail", function(cfg) stop("boom"), ex$cfg))
bad$status
```

`pam_scan_audio` *Stage 1: scan standardized audio and build the master metadata table*

Description

Lists WAV/FLAC files, parses filenames, reads headers (cached, so re-runs only read new files and an interrupted run resumes), joins deployments, adds QC flags, and writes the master table and summaries.

Usage

```
pam_scan_audio(cfg, force = FALSE)
```

Arguments

`cfg`	A `pam_config` object.
`force`	If TRUE, discard the header cache and re-read every file.

Details

Writes `metadata/audio_headers.csv` (cache), `metadata/audio_master.csv`, `outputs/audio_summary_per_grid.csv` and `outputs/deployment_issues.csv`. Times in CSV files are written as UTC (ISO 8601) by `data.table::fwrite`. Run with `pam_run_stage("01_scan", pam_scan_audio, cfg)` for logging.

Audio discovery uses `pam_list_audio()` (WAV + FLAC). Header reading uses `pam_audio_info()`.

Value

A list: `master`, `deployments`, `summary`.

Examples

```
ex <- pam_example_project()
res <- suppressMessages(pam_scan_audio(ex$cfg))
res$summary$overall
# the master table and summaries are also written to the project
list.files(ex$cfg$paths$metadata)
```

pam_soundscape_batch *Batch Process Audio Files for Soundscape Indices*

Description

Batch Process Audio Files for Soundscape Indices

Usage

```
pam_soundscape_batch(file_list, cfg, test_n = NULL)
```

Arguments

file_list	Character vector. File paths to audio files (WAV or FLAC).
cfg	List. Parsed configuration structure.
test_n	Integer. Optional. Process only this many files (for tests).

Value

A data.table containing raw soundscape index values.

pam_soundscape_one *Compute Ecoacoustic Soundscape Indices for a Single Audio File*

Description

Reads WAV or FLAC via pam_read_audio, optionally downsamples to 24 kHz, applies a high-pass filter at sub-window level, and computes configured acoustic indices (ACI, ADI, AEve, Bio, H, M, NDSI).

Usage

```
pam_soundscape_one(
  audio_path,
  low_stop_hz = NULL,
  cfg = NULL,
  window_sec = NULL
)
```

Arguments

audio_path	Character. Path to input audio file (.wav or .flac).
low_stop_hz	Numeric. Highpass low-stop frequency in Hz (default 500).
cfg	List. Parsed configuration structure from pam_config_read.
window_sec	Numeric. Optional window duration in seconds.

Value

A `data.table` with file/window metadata and index values.

Examples

```
if (requireNamespace("soundecology", quietly = TRUE)) {
  ex <- pam_example_project()
  # the example tones are only 2 s long and 8 kHz, so some indices come out as NA
  pam_soundscape_one(ex$files[1], cfg = ex$cfg)
}
```

---

pam_soundscape_plot_diel

Plot 24-Hour Diel Curves for Acoustic Indices

Description

Generates line plots showing index values across 0-23 hours grouped by grid site.

Usage

```
pam_soundscape_plot_diel(hourly_dt, indices = NULL, out_dir = NULL, cfg = NULL)
```

Arguments

`hourly_dt`	`data.table`. Hourly summary from `pam_soundscape_summarise`.
`indices`	Character vector. Acoustic indices to plot (e.g. `c("aci", "bio")`). If NULL, auto-detects available indices in `hourly_dt`.
`out_dir`	Character. Optional output directory to save plot images.
`cfg`	List. Optional configuration structure.

Value

A list of ggplot objects.

pam_soundscape_plot_phase *Plot Day vs. Night Comparisons for Acoustic Indices*

Description

Generates bar plots comparing mean values between Day and Night phases per grid site.

Usage

```
pam_soundscape_plot_phase(diel_dt, indices = NULL, out_dir = NULL, cfg = NULL)
```

Arguments

diel_dt	data.table. Diel summary from pam_soundscape_summarise.
indices	Character vector. Acoustic indices to plot. If NULL, auto-detects.
out_dir	Character. Optional output directory to save plot images.
cfg	List. Optional configuration structure.

Value

A list of ggplot objects.

pam_soundscape_report *Generate HTML Soundscape Diagnostic Report*

Description

Compiles an HTML diagnostic report summarizing soundscape indices, site covariates, and diel visualizations using the template at inst/rmd/soundscape_report.Rmd.

Usage

```
pam_soundscape_report(cfg, sc_results = NULL, vis_results = NULL)
```

Arguments

cfg	List. Parsed configuration structure from pam_config_read.
sc_results	List. Optional return object from pam_soundscape_run.
vis_results	List. Return object from pam_soundscape_plot_diel / pam_soundscape_plot_phase.

Value

Character. Path to the generated HTML report file.

pam_soundscape_run *Run Stage 2b Soundscape Analysis Pipeline*

Description

Discovers WAV/FLAC files via pam_list_audio, computes indices with pam_soundscape_batch, and writes hourly/diel/site summaries.

Usage

```
pam_soundscape_run(cfg, test_n = NULL)
```

Arguments

cfg	List. Parsed configuration structure from pam_config_read.
test_n	Integer. Optional. Process only this many files (for tests).

Value

A named list containing raw, hourly, diel, and site_covariates.

pam_soundscape_run_vis *Run Stage 2c Soundscape Visualization Pipeline*

Description

Takes parsed config and optional previous Stage 2b results, loads soundscape summary tables, exports diagnostic diel plots, and generates an HTML report.

Usage

```
pam_soundscape_run_vis(cfg, sc_results = NULL, generate_report = TRUE)
```

Arguments

cfg	List. Parsed configuration structure from pam_config_read.
sc_results	List. Optional return object from pam_soundscape_run.
generate_report	Logical. If TRUE, compiles an HTML diagnostic report.

Value

A named list of generated ggplot objects.

pam_soundscape_summarise

Summarize Soundscape Indices into Hourly, Diel, and Site Covariate Metrics

Description

Summarize Soundscape Indices into Hourly, Diel, and Site Covariate Metrics

Usage

```
pam_soundscape_summarise(raw_dt, master_dt, cfg = NULL)
```

Arguments

raw_dt	data.table. Raw window soundscape indices.
master_dt	data.table. Master metadata from Stage 1.
cfg	List. Parsed configuration structure.

Value

A named list containing hourly, diel, and site_covariates.

pam_species_thresholds

Choose a confidence threshold for each species from the validation results

Description

Species differ a lot in how reliable BirdNET is, so one confidence cut-off for all species is rarely right. This function reads the reviewed verdicts of Stage 5 and gives every species its own threshold, without asking you for a precision target.

Usage

```
pam_species_thresholds(
  cfg,
  run_id = cfg$birdnet$run_id,
  tolerance = 0.05,
  min_judged = 10,
  min_coverage = 0.5,
  manual = NULL,
  write = FALSE
)
```

Arguments

cfg	A pam_config object.
run_id	Detection run (sub-folder of paths$detections).
tolerance	How far below its own best estimated precision a species may fall (a number between 0 and 1, default 0.05).
min_judged	Minimum number of judged samples (correct or incorrect) a species needs; species with fewer get the default threshold.
min_coverage	A confidence level only counts when at least this share of the species' events at or above it lies in bands that were reviewed (default 0.5).
manual	Optional data frame, or path to a CSV file, with the columns scientific_name and threshold. These species use your value instead of the estimate.
write	If TRUE, save the table as <detections>/<run_id>/validation/species_thresholds.csv. Nothing is written by default.

Details

Plateau rule:

For each species, the estimated precision of all events at or above each confidence band edge comes from pam_validation_metrics(). The species' best estimated precision is found, and the **lowest** confidence whose precision is within tolerance of that best value is chosen. Going higher than that would discard detections without making them noticeably more reliable. A species with a flat precision curve therefore keeps a low threshold, and one that needs a high threshold gets one.

Defaults:

A species with too few judged samples, or no confidence level with enough reviewed events, gets processing$min_confidence and method = "default"; the reason column says why. Manual values always win (method = "manual").

Using the result:

Save it with write = TRUE and set processing$species_thresholds_file in the config to that file. Stage 4 and Stage 6 read it; the columns they do not need are ignored.

Value

A data.table with one row per species: scientific_name, threshold, method, precision_est (estimated precision of the events kept), coverage, n_judged, n_events_kept and reason.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
new <- suppressMessages(pam_validation_draw(ex$cfg, strategy = "random", n = 8))
val_dir <- file.path(ex$cfg$paths$detections, "run01", "validation")
# normally the review app writes this file; here the verdicts are made up
results <- data.frame(sample_id = new$sample_id,
```

```
                      verdict = rep_len(c("correct", "correct", "incorrect"), nrow(new)),
                       true_species = NA_character_, note = "", reviewer = "example",
                       reviewed_at = format(Sys.time()), stringsAsFactors = FALSE)
utils::write.csv(results, file.path(val_dir, "validation_results.csv"), row.names = FALSE)
thr <- suppressMessages(pam_species_thresholds(ex$cfg, min_judged = 1))
as.data.frame(thr)[, c("scientific_name", "threshold", "method")]
```

`pam_standardization_app` *Launch PAM Standardization & Data Triage Shiny App*

Description

Interactive GUI for inspecting raw acoustic data, parsing timestamps, previewing conversions, isolating non-audio/memo files, and performing standardized audio transcoding.

Usage

```
pam_standardization_app(cfg = NULL, raw_dir = NULL, lang = NULL)
```

Arguments

`cfg`	Optional configuration object from `pam_load_config()`.
`raw_dir`	Optional path override for raw audio directory.
`lang`	Interface language code (for example `"en"`); `NULL` chooses it as `pam_train_app()` does. Translations live in `inst/i18n/<code>.yml`.

Value

Runs Shiny application.

Examples

```
if (interactive()) {
  ex <- pam_example_project()
  pam_standardization_app(ex$cfg, raw_dir = ex$audio_dir)
}
```

pam_standardize *Standardize Raw PAM Audio Files (Stage 00 Engine)*

Description

Standardizes raw audio datasets by organizing folders, parsing timestamps, converting formats (WAV/FLAC), resampling, mixing channels to mono, isolating non-audio/memos, and logging results.

Usage

```
pam_standardize(
  cfg = NULL,
  raw_dir = NULL,
  standardized_dir = NULL,
  isolated_dir = NULL,
  dry_run = NULL,
  prefix = NULL,
  target_format = NULL,
  target_hz = NULL,
  channels = NULL,
  min_duration_s = NULL,
  organize_folders = NULL,
  rename_strategy = NULL,
  isolate_non_audio = NULL,
  isolate_short = NULL,
  timezone = NULL
)
```

Arguments

cfg	Optional configuration object from pam_load_config(). Supplies the folders and settings that are not given as arguments.
raw_dir	Directory containing raw audio files.
standardized_dir	Directory to save standardized audio. Required (here or in cfg) unless dry_run = TRUE; there is no default.
isolated_dir	Directory to save isolated non-audio and memo files. Required (here or in cfg) unless dry_run = TRUE; there is no default.
dry_run	Logical, if TRUE simulates actions without modifying disk.
prefix	Prefix string prepended to standardized filenames.
target_format	Output format: "flac" or "wav".
target_hz	Target sample rate (Hz). If NULL or higher than input, original rate is kept.
channels	Target audio channels (1 = mono, 2 = stereo).

min_duration_s Minimum duration threshold for valid recordings (default 60s).

organize_folders
: Subfolder structure: "grid" (G0059/), "date" (YYYY/MM/DD), "grid_date" (G0059/YYYY/MM/DD), "preserve", or "flat".

rename_strategy
: Naming strategy: "auto" (uses extracted datetime) or "keep".

isolate_non_audio
: Logical, isolate non-audio files?

isolate_short Logical, isolate short audio clips?

timezone Timezone string (default "Asia/Kolkata").

Value

data.table containing audit log of standardization operations.

Examples

```
ex <- pam_example_project()
demo <- tempfile("pamflow_std_")

# dry_run = TRUE only plans what would happen; nothing is written
plan <- suppressMessages(pam_standardize(
  ex$cfg, raw_dir = ex$audio_dir, standardized_dir = file.path(demo, "standardized"),
  isolated_dir = file.path(demo, "isolated"), dry_run = TRUE,
  target_format = "wav", min_duration_s = 1))
table(plan$action)


if (requireNamespace("av", quietly = TRUE)) {
  # convert for real: the example WAV files become FLAC
  done <- suppressMessages(pam_standardize(
    ex$cfg, raw_dir = ex$audio_dir, standardized_dir = file.path(demo, "standardized"),
   isolated_dir = file.path(demo, "isolated"), target_format = "flac", min_duration_s = 1))
  table(done$action)
}

unlink(demo, recursive = TRUE)
```

---

pam_train_app *Launch the BirdNET classifier training app*

Description

A standalone tool, not a workflow stage: a point-and-click interface for training a custom BirdNET classifier, laid out like the training tab of the BirdNET-Analyzer GUI: training data, cache, frequency and speed, cropping, autotune, training settings and the model save mode. It shows the progress and the learning curves while training runs, can repeat a training several times to show how much the scores vary, keeps a history of earlier runs, and gives the `birdnet:` config lines that use the trained model.

Usage

```
pam_train_app(
  cfg = NULL,
  data_dir = NULL,
  output_dir = NULL,
  lang = NULL,
  envdir = pam_birdnet_env_dir("training")
)
```

Arguments

cfg	Optional configuration from pam_load_config(); its birdnet$model_dir is used as the default output folder.
data_dir	Optional training folder to start with (one sub-folder per class).
output_dir	Optional output folder to start with.
lang	Interface language code (for example "en"). NULL chooses from options(pamflow.lang), the PAMFLOW_LANG variable or the system language, and falls back to English. Translations live in inst/i18n/<code>.yml; a translation only needs the texts it changes.
envdir	Folder of the training Python environment (see pam_birdnet_setup("training")).

Details

This is a standalone tool: it is not a stage of the workflow and no workflow function calls it. Use the trained model by pointing the birdnet: section of the config at it.

Training runs in a separate process, so the app stays responsive and Stop ends the training and everything it started. Closing the app also stops a running training. The settings shown are the BirdNET-Analyzer 2.4.0 defaults; each value you see is the value used, and it is recorded in the run's _run.yml. The first training downloads the base BirdNET model (about 224 MB) once.

The scores shown come from clips held out of the training folder and are therefore optimistic; give a separate test folder for a more honest estimate.

Value

A Shiny app object. Printing it at the console starts it, or pass it to shiny::runApp().

Examples

```
if (interactive()) {
  pam_train_app()
}
## Not run:
pam_train_app(data_dir = "E:/clips", output_dir = "E:/models", lang = "en")

## End(Not run)
```

pam_validate *Open the validation app*

Description

Shows the exported spectrogram and audio of each sampled event in random order and records "correct", "incorrect" (plus the true species) or "unsure". Every save is written immediately to `validation_results.csv`, so you can stop at any time and continue later.

Usage

```
pam_validate(
  cfg,
  run_id = cfg$birdnet$run_id,
  reviewer = pam_or(cfg$validation$reviewer, ""),
  show_confidence = isTRUE(cfg$validation$show_confidence),
  lang = NULL
)
```

Arguments

`cfg`	A `pam_config` object.
`run_id`	Detection run.
`reviewer`	Name pre-filled in the app.
`show_confidence`	Show the model's confidence while reviewing. Hidden by default (`validation$show_confidence`) to avoid biasing the reviewer.
`lang`	Interface language code (for example `"en"`); `NULL` chooses it as `pam_train_app()` does. Translations live in `inst/i18n/<code>.yml`.

Value

Nothing; runs until the app is closed.

Examples

```
if (interactive() && nzchar(Sys.which("ffmpeg"))) {
  ex <- pam_example_project()
  res <- suppressMessages(pam_scan_audio(ex$cfg))
  proc <- suppressMessages(pam_process_detections(ex$cfg))
  new <- suppressMessages(pam_validation_draw(ex$cfg, strategy = "random", n = 5))
 out <- suppressMessages(pam_validation_export(ex$cfg))  # clips and spectrograms to review
  pam_validate(ex$cfg)
}
```

`pam_validation_draw` *Draw a validation round and add it to the sample file*

Description

Each call adds a new round to `<detections>/<run_id>/validation/ validation_samples.csv`. Events drawn in earlier rounds are never drawn again, so you can start small and add more later.

Usage

```
pam_validation_draw(
  cfg,
  run_id = cfg$birdnet$run_id,
  strategy = c("stratified", "random", "rare"),
  n = 100L,
  species = NULL,
  n_per_cell = cfg$validation$n_per_cell,
  pool = NULL
)
```

Arguments

`cfg`	A `pam_config` object (uses the `validation` section).
`run_id`	Detection run.
`strategy`	"stratified", "random" or "rare" (see `pam_validation_sample()`).
`n`	Events for the "random" strategy.
`species`	Optional species restriction.
`n_per_cell`	Events per species x band for "stratified". Defaults to `validation$n_per_cell`.
`pool`	Optional pool from `pam_validation_pool()` (computed if NULL).

Value

Invisibly, the newly added rows.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
new <- suppressMessages(pam_validation_draw(ex$cfg, strategy = "random", n = 5))
as.data.frame(new)[, c("sample_id", "species", "confidence", "clip_file")]
# the sample file is written to the run folder
file.exists(file.path(ex$cfg$paths$detections, "run01", "validation", "validation_samples.csv"))
```

pam_validation_export *Export audio clips and spectrograms for the sampled events (WAV + FLAC)*

Description

For every row of `validation_samples.csv` whose files are missing, cuts a clip (event plus `validation$clip_pad_s` seconds of context on each side) from the audio the run read, and draws a spectrogram PNG with FFmpeg. Dynamically uses `-c:a flac` or `-c:a pcm_s16le` based on the target clip file extension. Existing files are kept, so it can be rerun after each new round.

Usage

```
pam_validation_export(cfg, run_id = cfg$birdnet$run_id, force = FALSE)
```

Arguments

`cfg`	A `pam_config` object.
`run_id`	Detection run.
`force`	Recreate all files.

Value

Invisibly, list(n_ok, n_failed, failed_ids).

Examples

```
if (nzchar(Sys.which("ffmpeg"))) {
  ex <- pam_example_project()
  # the later stages read the master table that the scan builds
  res <- suppressMessages(pam_scan_audio(ex$cfg))
  proc <- suppressMessages(pam_process_detections(ex$cfg))
  new <- suppressMessages(pam_validation_draw(ex$cfg, strategy = "random", n = 3))
  out <- suppressMessages(pam_validation_export(ex$cfg))
  out$n_ok
}
```

pam_validation_import *Join the reviewed verdicts to the sample table*

Description

Join the reviewed verdicts to the sample table

Usage

```
pam_validation_import(cfg, run_id = cfg$birdnet$run_id)
```

Arguments

cfg	A pam_config object.
run_id	Detection run.

Value

A data.table: one row per reviewed sample, with verdict ("correct", "incorrect", "unsure"), true_species, note, reviewer, reviewed_at added to the sample columns.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
new <- suppressMessages(pam_validation_draw(ex$cfg, strategy = "random", n = 8))
val_dir <- file.path(ex$cfg$paths$detections, "run01", "validation")
# normally the review app writes this file; here the verdicts are made up
results <- data.frame(sample_id = new$sample_id,
                      verdict = rep_len(c("correct", "correct", "incorrect"), nrow(new)),
                       true_species = NA_character_, note = "", reviewer = "example",
                       reviewed_at = format(Sys.time()), stringsAsFactors = FALSE)
utils::write.csv(results, file.path(val_dir, "validation_results.csv"), row.names = FALSE)
val <- pam_validation_import(ex$cfg)
as.data.frame(val)[, c("sample_id", "species", "confidence", "verdict")]
```

pam_validation_metrics *Precision by species and confidence band, and by confidence threshold*

Description

Precision by species and confidence band, and by confidence threshold

Usage

```
pam_validation_metrics(val, pool, bands)
```

Arguments

val	Reviewed samples from pam_validation_import().
pool	Pool of all events from pam_validation_pool(); its cell sizes weight the threshold estimates, because sampling is stratified.
bands	Confidence band edges used when sampling.

Details

Recall, false negatives and F1 cannot be estimated from reviewing detections alone: they need a set of recordings in which every call has been annotated independently of the detector.

Value

A list:

- cells: per species x band: n_pool, n_reviewed, n_correct, n_incorrect, n_unsure, n_judged, precision and its 95% Wilson interval (unsure verdicts are excluded from precision).
- thresholds: for each species (and "ALL") and each band edge, the estimated precision of all events at or above that confidence, weighted by band size, plus coverage (share of those events that sit in bands with at least one judged sample).
- confusion: counts of predicted species vs the true species entered for incorrect verdicts.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
new <- suppressMessages(pam_validation_draw(ex$cfg, strategy = "random", n = 8))
val_dir <- file.path(ex$cfg$paths$detections, "run01", "validation")
# normally the review app writes this file; here the verdicts are made up
results <- data.frame(sample_id = new$sample_id,
                      verdict = rep_len(c("correct", "correct", "incorrect"), nrow(new)),
                       true_species = NA_character_, note = "", reviewer = "example",
                       reviewed_at = format(Sys.time()), stringsAsFactors = FALSE)
utils::write.csv(results, file.path(val_dir, "validation_results.csv"), row.names = FALSE)
val <- pam_validation_import(ex$cfg)
pool <- suppressMessages(pam_validation_pool(ex$cfg))
metrics <- pam_validation_metrics(val, pool, bands = unlist(ex$cfg$validation$bands))
as.data.frame(metrics$cells)[, c("scientific_name", "band", "n_pool", "n_reviewed", "precision")]
```

pam_validation_pool *Build the pool of candidate events for validation*

Description

Takes the run's assembled detections through the Stage 4 rules (deployment window, species lists, merging of windows into events) but with the confidence cutoff set to the lowest stored confidence and no species thresholds or seasons, so that events below your intended cutoff can also be reviewed.

Usage

```
pam_validation_pool(cfg, run_id = cfg$birdnet$run_id)
```

Arguments

cfg	A pam_config object.
run_id	Detection run (sub-folder of paths$detections).

Value

A data.table of events with event_id (rel|species|start).

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
pool <- suppressMessages(pam_validation_pool(ex$cfg))
as.data.frame(pool)[, c("species", "confidence", "n_windows", "start_time")]
```

pam_validation_report *Validation report: precision tables written to disk*

Description

Imports the reviewed verdicts, computes pam_validation_metrics() and writes precision_by_cell.csv, precision_by_threshold.csv and confusion_incorrect.csv to <detections>/<run_id>/validation/.

Usage

```
pam_validation_report(cfg, run_id = cfg$birdnet$run_id, pool = NULL)
```

Arguments

cfg	A pam_config object.
run_id	Detection run.
pool	Optional pool from pam_validation_pool().

Value

The metrics list (see `pam_validation_metrics()`).

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
proc <- suppressMessages(pam_process_detections(ex$cfg))
new <- suppressMessages(pam_validation_draw(ex$cfg, strategy = "random", n = 8))
val_dir <- file.path(ex$cfg$paths$detections, "run01", "validation")
# normally the review app writes this file; here the verdicts are made up
results <- data.frame(sample_id = new$sample_id,
                      verdict = rep_len(c("correct", "correct", "incorrect"), nrow(new)),
                       true_species = NA_character_, note = "", reviewer = "example",
                       reviewed_at = format(Sys.time()), stringsAsFactors = FALSE)
utils::write.csv(results, file.path(val_dir, "validation_results.csv"), row.names = FALSE)
metrics <- suppressMessages(pam_validation_report(ex$cfg))
list.files(val_dir, pattern = "^precision")
```

---

pam_validation_sample *Draw a validation sample from a pool of events*

Description

Pure function (no file access). Strategies:

- `"stratified"`: up to `n_per_cell` events per species x confidence band; restrict to some species with `species`.
- `"random"`: `n` events drawn at random from the whole pool (or from `species`).
- `"rare"`: species with at most `rare_max_events` events in the pool; up to `rare_n` events of each.

Usage

```
pam_validation_sample(
  pool,
  strategy = c("stratified", "random", "rare"),
  bands,
  n_per_cell = 25L,
  n = 100L,
  species = NULL,
  rare_max_events = 30L,
  rare_n = 50L,
  seed = 1L,
  exclude_ids = character()
)
```

Arguments

pool	Events from pam_validation_pool().
strategy	One of "stratified", "random", "rare".
bands	Increasing confidence band edges.
n_per_cell, n, rare_max_events, rare_n	See above.
species	Optional names (scientific, common or "Scientific_Common").
seed	Random seed.
exclude_ids	event_ids already sampled; never drawn again.

Value

A data.table: the sampled rows with band_idx and band.

Examples

```
ex <- pam_example_project()
# the later stages read the master table that the scan builds
res <- suppressMessages(pam_scan_audio(ex$cfg))
pool <- suppressMessages(pam_validation_pool(ex$cfg))
bands <- unlist(ex$cfg$validation$bands)
# at most one event per species and confidence band
smp <- pam_validation_sample(pool, "stratified", bands = bands, n_per_cell = 1)
as.data.frame(smp)[, c("species", "confidence", "band")]

# five events at random from the whole pool
rnd <- pam_validation_sample(pool, "random", bands = bands, n = 5, seed = 42)
nrow(rnd)
```

---

pam_wilson_ci *Wilson score confidence interval for a proportion*

Description

Wilson score confidence interval for a proportion

Usage

```
pam_wilson_ci(x, n, conf = 0.95)
```

Arguments

x	Number of successes (vector).
n	Number of trials (vector, same length).
conf	Confidence level.

Value

A data.table with `low` and `high` (NA where `n` is 0).

Examples

```
pam_wilson_ci(c(18, 5), c(25, 5))
```

`soundscape` *Stage 2b: Soundscape Analysis (Indices & Summaries)*

Description

Implementation of ecoacoustic soundscape indices following Bradfer-Lawrence et al. (2019) guidelines. Computes core acoustic indices with 1-minute sub-windowing, checkpointing, multi-core parallel processing, and timezone-aware hourly/diel summarization. Supports WAV and FLAC via `pam_read_audio` / `pam_list_audio`.

`soundscape_vis` *Stage 2c: Soundscape Visualization & Diagnostics*

Description

Generates 24-hour diel profiles, day/night comparison plots, and summary visualizations for ecoacoustic soundscape indices.

`to_scalar_num` *Safely Extract a Single Numeric Value from Soundecology/Seewave Returns*

Description

Safely Extract a Single Numeric Value from Soundecology/Seewave Returns

Usage

```
to_scalar_num(x)
```

Arguments

`x`	An object returned by a soundecology/seewave function.

Value

A length-1 numeric, or NA_real_.

utils_audio *Audio I/O Utilities (WAV + FLAC)*

Description

Centralized helpers for cross-format audio reading, metadata inspection, and lossless clip extraction. All downstream pamflow stages route through this module so WAV and FLAC are handled uniformly.

Index

pam_activity, 4
pam_activity(), *48*
pam_add_qc, 5
pam_add_site_covariates, 6
pam_annotate, 7
pam_as_count_history, 8
pam_as_count_history(), *7*, *31*, *44*
pam_assemble_detections, 8
pam_audio_exts, 10
pam_audio_history, 10
pam_audio_info, 11
pam_audio_info(), *62*
pam_audio_info_dt *(*pam_audio_info*)*, 11
pam_audio_pattern, 12
pam_audio_summary, 12
pam_birdnet_benchmark, 13
pam_birdnet_cleanup, 14
pam_birdnet_doctor, 15
pam_birdnet_setup, 16
pam_birdnet_train, 17
pam_birdnet_train_metrics, 19
pam_birdnet_train_prepare, 20
pam_birdnet_train_repeat, 21
pam_build_occupancy, 22
pam_check_deployments, 24
pam_check_ffmpeg, 25
pam_check_model_input, 25
pam_check_model_input(), *46*
pam_checkpoint_add, 23
pam_checkpoint_read, 24
pam_clean_detections, 26
pam_config_read, 27, *63*, *65, 66*
pam_config_read(), *37*
pam_config_validate, 28
pam_count_histories, 28
pam_count_histories(), *44*, *46*
pam_count_history, 30
pam_count_history(), *8, 9*, *28–30*, *33*, *44–46*
pam_cut_audio, 32
pam_define_seasons, 32
pam_define_seasons(), *44*, *46*
pam_detect, 33
pam_detect(), *36*
pam_detection_table, 34
pam_detection_table(), *5*, *61*
pam_example_project, 36
pam_extract_grid, 37
pam_filter_audio, 38
pam_filter_one, 39
pam_highpass_filter_string, 39
pam_inspect_raw, 40
pam_is_flac, 41
pam_list_audio, 41, *66*, *81*
pam_list_audio(), *62*
pam_load_config, 42
pam_log, 42
pam_match_deployments, 43
pam_model_input, 44
pam_model_input(), *7–9*, *25, 26*, *29–33*
pam_parse_datetime, 46
pam_parse_filenames, 47
pam_pending_batches, 48
pam_plot_activity, 48
pam_plot_activity(), *5*
pam_plot_richness, 49
pam_plot_richness(), *61*
pam_process_detections, 50
pam_read_audio, 51, *63*, *81*
pam_read_deployments, 51
pam_recall_draw, 52
pam_recall_export, 53
pam_recall_frame, 54
pam_recall_metrics, 55
pam_recall_report, 56
pam_recall_sample, 57
pam_report, 58
pam_richness, 59

pam_richness(), *49*
pam_run_stage, 61
pam_scan_audio, 62
pam_scan_audio(), *37*
pam_soundscape_batch, 63, *66*
pam_soundscape_one, 63
pam_soundscape_plot_diel, 64, *65*
pam_soundscape_plot_phase, *65*, 65
pam_soundscape_report, 65
pam_soundscape_run, *65, 66*, 66
pam_soundscape_run_vis, 66
pam_soundscape_summarise, *64, 65*, 67
pam_species_thresholds, 67
pam_standardization_app, 69
pam_standardize, 70
pam_train_app, 71
pam_validate, 73
pam_validation_draw, 74
pam_validation_export, 75
pam_validation_import, 76
pam_validation_metrics, 76
pam_validation_pool, 78
pam_validation_report, 78
pam_validation_sample, 79
pam_wilson_ci, 80

soundscape, 81
soundscape_vis, 81

tempdir(), *36*
to_scalar_num, 81

utils_audio, 82