Package: FuelDeep3D (via r-universe)

June 2, 2026

Type Package

Title 3D Fuel Segmentation Using Terrestrial Laser Scanning and Deep Learning

Version 0.1.1

Description Provides tools for preprocessing, feature extraction, and segmentation of three-dimensional forest point clouds derived from terrestrial laser scanning. Functions support creating height-above-ground (HAG) metrics, tiling, and sampling point clouds, generating training datasets, applying trained models to new point clouds, and producing per-point fuel classes such as stems, branches, foliage, and surface fuels. These tools support workflows for forest structure analysis, wildfire behavior modeling, and fuel complexity assessment. Deep learning segmentation relies on the PointNeXt architecture described by Qian et al. (2022) <doi:10.48550/arXiv.2206.04670>, while ground classification utilizes the Cloth Simulation Filter algorithm by Zhang et al. (2016) <doi:10.3390/rs8060501>.

Depends R (>= 4.1)

Imports stats, RColorBrewer, viridisLite, rlang

Suggests lidR, reticulate, dbscan, ggplot2, rgl, RCSF, scales

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL https://github.com/venkatasivanaga/FuelDeep3D

BugReports https://github.com/venkatasivanaga/FuelDeep3D/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

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

Date/Publication 2026-03-04 22:17:47 UTC

RemoteUrl https://github.com/venkatasivanaga/FuelDeep3D

RemoteRef HEAD

RemoteSha 98f29d11f572b1f1e53d1fde1aca604ed78bba11

Contents

`add_ground_csf` *Add a ground class using CSF post-processing*

Description

Post-processes a FuelDeep3D-predicted LAS/LAZ by detecting ground points with Cloth Simulation Filtering (CSF) and assigning them to a dedicated ground class. This is used when converting a 3-class FuelDeep3D prediction into a 4-class output where ground is encoded as class `3`.

Usage

```
add_ground_csf(in_las, out_las, csf_args = list())
```

Arguments

`in_las`	Character. Path to an input `.las` or `.laz` file that already contains FuelDeep3D predictions in the `Classification` attribute.
`out_las`	Character. Output path for the updated `.las` or `.laz`.
`csf_args`	List. Named list of arguments forwarded to `lidR::csf()` to tune CSF behavior (e.g., `rigidness`, `cloth_resolution`, `time_step`, `class_threshold`).

Details

Intended workflow

1. Run `predict` with `mode = "overwrite"` to write model predictions into the LAS `Classification` attribute (typically classes `0`, `1`, `2`).
2. Call `add_ground_csf()` to detect ground points and overwrite only those points to class `3`.

What the function does

- Reads the input LAS/LAZ from `in_las`.
- Normalizes heights with `lidR::normalize_height(..., lidR::knnidw())` so ground detection is more stable across sloped terrain and varying elevations.
- Builds a CSF ground-classification algorithm using `lidR::csf()` with parameters provided in `csf_args`.
- Runs `lidR::classify_ground()` to label ground points.
- Rewrites only the ground points to class `3`, while preserving the original FuelDeep3D predicted classes for non-ground points.
- Writes the updated LAS/LAZ to `out_las`.

Class mapping

- `0`, `1`, `2`: preserved from the FuelDeep3D prediction already stored in `Classification`.
- `3`: assigned to points detected as ground by CSF.

Dependencies

- **lidR** for I/O, height normalization, and ground classification.
- **RCSF** provides the CSF implementation used by `lidR::csf()`.

Value

Invisibly returns `out_las` (the output file path).

Examples

```
# Check if required packages are available before running
if (requireNamespace("lidR", quietly = TRUE) &&
    requireNamespace("RCSF", quietly = TRUE)) {

library(FuelDeep3D)
library(lidR)

in_file  <- system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D")
out_file <- file.path(tempdir(), "tree2_ground.laz")

add_ground_csf(
  in_las  = in_file,
  out_las = out_file,
  csf_args = list(
    rigidness = 4,
```

```
    cloth_resolution = 0.25,
    time_step = 0.65,
    class_threshold = 0.05
  )
)
}
```

---

config *Create a FuelDeep3D configuration*

Description

Constructs a named list of parameters used throughout FuelDeep3D for dataset tiling, model training, and inference. The returned configuration is consumed by `train` (to build NPZ tiles and train the Python model) and `predict` (to run inference and write a predicted LAS/LAZ).

Usage

```
config(
  las_path = system.file("extdata", "trees.las", package = "FuelDeep3D"),
  out_dir = getwd(),
  out_pred_dir = getwd(),
  model_path = system.file("extdata", "best_model.pth", package = "FuelDeep3D"),
  device = NULL,
  block_size = 6,
  stride = 1,
  sample_n = 4096,
  repeat_per_tile = 4,
  min_pts_tile = 512,
  val_split = 0.15,
  test_split = 0.1,
  seed = 42,
  batch_size = 16,
  epochs = 2,
  learning_rate = 1e-05,
  weight_decay = 1e-04,
  cell_size = 0.25,
  quantile = 0.05,
  num_classes = 3,
  csf_args = list(rigidness = 4, cloth_resolution = 0.25, time_step = 0.65,
    class_threshold = 0.05),
  delete_tiles_after_train = TRUE
)
```

Arguments

las_path	Path to an input LAS/LAZ file.
out_dir	Directory where training tiles (NPZ) will be written.
out_pred_dir	Directory where prediction outputs will be written.
model_path	Path to a .pth model checkpoint.
device	Device to use ("cpu" or "cuda"). If NULL, the Python backend will choose automatically.
block_size	Tile size (meters).
stride	Overlap stride (meters).
sample_n	Number of points sampled per tile.
repeat_per_tile	Number of repeated samples/augmentations per tile.
min_pts_tile	Minimum number of points required to keep a tile.
val_split	Fraction of tiles used for validation.
test_split	Fraction of tiles used for testing.
seed	Random seed used for splitting/sampling.
batch_size	Batch size for training.
epochs	Number of training epochs.
learning_rate	Optimizer learning rate.
weight_decay	L2 regularization strength (weight decay).
cell_size	Grid cell size (meters) used for height normalization/statistics.
quantile	Quantile of the threshold used in metrics/filters in the Python pipeline.
num_classes	Number of output classes. Supported values are 3 or 4. If 4, ground can be added using CSF post-processing.
csf_args	Named list of arguments forwarded to RCSF::csf() inside add_ground_csf (used when num_classes = 4).
delete_tiles_after_train	Logical; if TRUE, delete generated NPZ tiles after training completes (see train).

Details

The configuration groups parameters into a few logical sections:

I/O paths

- las_path: input LAS/LAZ file used for preprocessing, training, or inference.
- out_dir: directory where NPZ tiles will be written (typically contains train/, val/, and test/ subfolders).
- out_pred_dir: directory where predicted LAS/LAZ outputs are written.
- model_path: path to a .pth model checkpoint used by predict.

Tiling / sampling (Python dataset builder)

- `block_size` and `stride` control the spatial tiling grid (meters).
- `sample_n` sets the number of points sampled per tile.
- `repeat_per_tile` controls how many repeated samples/augmentations are generated per tile.
- `min_pts_tile` drops tiles with too few points.
- `val_split`, `test_split`, and `seed` control dataset splitting.
- `cell_size` and `quantile` are forwarded to the Python pipeline for height normalization / grid-based statistics and related thresholds.

Training hyperparameters (Python trainer)

- `batch_size`, `epochs`, `learning_rate`, and `weight_decay` configure the optimizer and training loop in the Python trainer.

Device selection

- `device` can be `"cpu"` or `"cuda"`. If `NULL`, the Python backend selects CUDA when available and otherwise falls back to CPU.

Class handling: 3 vs 4 classes

- `num_classes = 3`: produces the model's 3-class predictions.
- `num_classes = 4`: after 3-class prediction, FuelDeep3D can add a ground class via CSF post-processing (see `add_ground_csf`).

Cleanup

- If `delete_tiles_after_train = TRUE`, generated NPZ tiles under `out_dir/train`, `out_dir/val`, and `out_dir/test` may be removed after training (see `train`).

Value

A named list containing all configuration parameters.

`ensure_py_env` *Ensure a Conda environment and Python dependencies for FuelDeep3D*

Description

Creates (if needed) and activates a Conda environment for FuelDeep3D, then installs the required Python dependencies into that environment using pip via `reticulate::conda_install(..., pip = TRUE)`.

Usage

```
ensure_py_env(
  envname = "pointnext",
  python_version = "3.10",
  reinstall = FALSE,
  cpu_only = TRUE,
  conda = NULL
)
```

Arguments

envname	Character. Name of the Conda environment to use/create.
python_version	Character. Python version used if a new env is created (e.g. "3.10").
reinstall	Logical. If TRUE, forces dependency installation even if key modules are present.
cpu_only	Logical. If TRUE, installs CPU-only PyTorch wheels. If FALSE, installs CUDA wheels (cu121).
conda	Path to conda binary. Default uses `reticulate::conda_binary()`.

Details

This function is intentionally opt-in. It requires:

- An existing Conda installation (Miniconda/Anaconda), discoverable by reticulate.
- Internet connection to install Python packages.

For CRAN safety, this function will not run during `R CMD check`.

Value

Invisibly TRUE if environment was ensured + activated, FALSE if skipped during check.

Examples

```
## Not run:
# Requires Conda + internet
ensure_py_env(envname = "pointnext", python_version = "3.10", cpu_only = TRUE)

# CUDA wheels (requires compatible NVIDIA drivers)
ensure_py_env(envname = "pointnext", python_version = "3.10", cpu_only = FALSE)

# Inspect reticulate Python
reticulate::py_config()

## End(Not run)
```

`evaluate_single_las` *Evaluate predictions stored in a single LAS/LAZ*

Description

Computes a confusion matrix and standard multi-class metrics when both the ground-truth labels and predictions are stored as columns in the same `lidR::LAS` object.

Usage

```
evaluate_single_las(
  las,
  truth_col = "label",
  pred_col = "Classification",
  classes = NULL,
  drop_na = TRUE,
  class_names = NULL,
  show_codes = TRUE
)
```

Arguments

`las`	A `lidR::LAS` object containing truth and prediction fields.
`truth_col`	Character. Name of the ground-truth label column (default `"label"`).
`pred_col`	Character. Name of the prediction column (default `"Classification"`).
`classes`	Optional integer vector of expected class IDs (e.g. `0:2`, `0:3`). If `NULL`, inferred from observed truth and pred values.
`drop_na`	Logical. If `TRUE` (default), rows with NA/Inf in truth or pred are dropped.
`class_names`	Optional class name mapping. Either: • Named: `c("0"="Ground","1"="Branch","2"="Leaves")` • Unnamed length-K: `c("Ground","Branch","Leaves")` (must match `classes`)
`show_codes`	Logical. If `TRUE` (default), class labels display like `"0 (Ground)"`.

Details

The function returns:

- **overall_accuracy**: $\sum diag(cm) / \sum cm$
- **class_accuracy**: per-class accuracy computed as $TP/(TP+FN)$ (same as per-class recall / producer's accuracy).
- **precision**, **recall**, **f1**: per-class metrics.
- **balanced_accuracy**: mean of `class_accuracy` across classes (macro average).

To keep the confusion matrix shape stable across files (even when some classes are missing), pass `classes = 0:2` or `classes = 0:3`.

Value

A list containing the confusion matrix and metrics.

Examples

```
if (requireNamespace("lidR", quietly = TRUE)){

library(lidR)
las <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

res <- evaluate_single_las(
  las,
  truth_col = "label",
  pred_col  = "Classification",
  classes   = 0:2,
  class_names = c("0"="Ground","1"="Branch","2"="Leaves")
)

res$class_accuracy
cat(sprintf("accuracy = %.2f%%\n", 100 * res$accuracy))
}
```

---

evaluate_two_las *Evaluate predictions stored in two LAS/LAZ objects*

Description

Computes the confusion matrix and metrics when ground truth and predictions come from two separate `lidR::LAS` objects.

Usage

```
evaluate_two_las(
  truth_las,
  pred_las,
  truth_col = "label",
  pred_col = "Classification",
  classes = NULL,
  drop_na = TRUE,
  class_names = NULL,
  show_codes = TRUE
)
```

Arguments

`truth_las`	A `lidR::LAS` containing ground-truth labels.
`pred_las`	A `lidR::LAS` containing predicted labels.
`truth_col`	Character. Name of the ground-truth label column (default `"label"`).

pred_col	Character. Name of the prediction column (default "Classification").
classes	Optional integer vector of expected class IDs (e.g. 0:2, 0:3). If NULL, inferred from observed truth and pred values.
drop_na	Logical. If TRUE (default), rows with NA/Inf in truth or pred are dropped.
class_names	Optional class name mapping. Either: • Named: c("0"="Ground","1"="Branch","2"="Leaves") • Unnamed length-K: c("Ground","Branch","Leaves") (must match classes)
show_codes	Logical. If TRUE (default), class labels display like "0 (Ground)".

Details

Important: This assumes the LAS objects are point-wise aligned (with the same points and order). If they are not aligned, metrics will be meaningless.

Value

A list containing the confusion matrix and metrics.

Examples

```
if (requireNamespace("lidR", quietly = TRUE)){

library(lidR)
truth <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))
pred  <- readLAS(system.file("extdata", "las", "tree21.laz", package = "FuelDeep3D"))

res <- evaluate_two_las(
  truth, pred,
  truth_col = "label",
  pred_col  = "Classification",
  classes   = 0:2,
  class_names = c("0"="Ground","1"="Branch","2"="Leaves")
)

res$class_accuracy
cat(sprintf("accuracy = %.2f%%\n", 100 * res$accuracy))
}
```

---

install_py_deps *Install Python dependencies into a Conda environment (FuelDeep3D)*

Description

Installs required Python packages into an existing Conda environment using pip.

Usage

```
install_py_deps(
  envname = "pointnext",
  only_if_missing = TRUE,
  cpu_only = TRUE,
  conda = NULL
)
```

Arguments

envname	Character. Name of the Conda environment.
only_if_missing	Logical. Skip install if key modules are already present.
cpu_only	Logical. If TRUE, installs CPU-only PyTorch; otherwise installs cu121 wheels.
conda	Path to conda binary. Default uses `reticulate::conda_binary()`.

Details

If `only_if_missing = TRUE`, checks for key importable modules and skips installation when they already exist.

Value

Invisibly TRUE if installation ran, FALSE if skipped.

Examples

```
## Not run:
install_py_deps(envname = "pointnext", cpu_only = TRUE)

## End(Not run)
```

---

las_class_distribution

Class distribution summary for a LAS point cloud

Description

Computes how many points belong to each class value in a given LAS attribute field (e.g., predicted classes stored in `Classification`, or original labels stored in `label`). Returns a tidy `data.frame` with counts and percentages.

Usage

```
las_class_distribution(
  las,
  field = "Classification",
  class_labels = NULL,
  include_na = TRUE,
  sort_by = c("n_points", "class", "percent"),
  decreasing = TRUE
)
```

Arguments

las	A LAS object from lidR.
field	Character. Column name in las@data containing class values (e.g., "Classification", "label").
class_labels	Optional. A named character vector mapping class values to human-readable names, e.g. c("0"="Ground","1"="Stem","2"="Crown"). Names must match the class values as characters.
include_na	Logical. If TRUE (default), includes NA as a separate row (shown as class "<NA>"). If FALSE, drops NA values.
sort_by	Character. Sort rows by "n_points" (default), "class", or "percent".
decreasing	Logical. If TRUE (default), sorts descending.

Details

This is useful after prediction to quickly inspect class balance and verify that classes look reasonable (e.g., not all points predicted as one class).

Value

A data.frame with columns:

class Class value as character (NA shown as "<NA>")

name (Optional) human-readable name if class_labels provided

n_points Number of points in that class

percent Percent of total points in that class

Examples

```
if (requireNamespace("lidR", quietly = TRUE)){
library(lidR)
las <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

# 1) Predicted distribution (common case)
las_class_distribution(las, field = "Classification")

# 2) Raw label distribution
```

```
las_class_distribution(las, field = "label")

# 3) With human-readable names
labs <- c("0"="Ground vegetation", "1"="Foliage", "2"="Branches")
las_class_distribution(las, field = "Classification", class_labels = labs)

# 4) Drop NA rows if you don't want them
las_class_distribution(las, field = "Classification", include_na = FALSE)
}
```

---

plot_3d *Plot a 3D LAS point cloud colored by elevation*

Description

Visualize a `lidR::LAS` point cloud in 3D using **rgl**, with points colored by elevation (Z). Supports subsampling for performance, custom height color ramps, optional coordinate centering, an optional compact legend, and optional "thickness by height" (points become larger at higher Z).

Usage

```
plot_3d(
  las,
  bg = "black",
  zlim = NULL,
  height_palette = NULL,
  size = 0.6,
  max_points = 400000L,
  title = "LAS 3D View",
  center = FALSE,
  add_legend = TRUE,
  legend_height_frac = 0.5,
  legend_width_frac = 0.015,
  legend_xpad_frac = 0.03,
  legend_side = "right",
  legend_pos = c(NA_real_, NA_real_),
  legend_label_mode = c("rel_z", "norm_z", "z", "norm"),
  z_digits = 2L,
  z_unit = "m",
  size_by_height = TRUE,
  size_range = c(1, 4),
  size_power = 1.2,
  zoom = 0.7,
  theta = 0,
  phi = -90
)
```

Arguments

las	A lidR::LAS object.
bg	Background color for the **rgl** scene. (e.g., "black", "white", "#111111").
zlim	NULL or numeric length-2 vector giving the Z range used for coloring (values are clamped to this range).
height_palette	Vector of colors to define the height color ramp (e.g., c("purple","blue","cyan","yellow","red")).
size	Numeric. Base point size (thickness). If size_by_height = TRUE, this acts as a multiplier on size_range.
max_points	Integer. If LAS has more than this many points, a random subsample is plotted for speed. Use NULL to disable subsampling.
title	Character. Plot title (converted to ASCII-safe to avoid **rgl** text errors).
center	Logical. If TRUE, shifts X/Y/Z so minima become 0 (helps visualization when coordinates are large).
add_legend	Logical. If TRUE, draws a vertical colorbar and min/max labels.
legend_height_frac	Numeric in (0,1]. Visually compress legend height (e.g., 0.5 = half height).
legend_width_frac	Numeric. Legend width as a fraction of X-range.
legend_xpad_frac	Numeric. Legend x-offset as a fraction of X-range.
legend_side	Character. Either "right" or "left".
legend_pos	Numeric length-2 vector c(x, y) to override legend base position (use NA to ignore a coordinate).
legend_label_mode	Character. One of "norm_z", "z", or "norm".
z_digits	Integer. Number of digits used for legend Z labels.
z_unit	Character. Unit label appended to legend Z values (e.g., "m"). Use "" for none.
size_by_height	Logical. If TRUE, point thickness increases with height.
size_range	Numeric length-2. Min/max point size (before multiplying by size) when size_by_height = TRUE.
size_power	Numeric > 0. Controls how quickly thickness increases with height (larger = more emphasis at top).
zoom, theta, phi	Camera controls passed to rgl::view3d().

Details

Color mapping

- Elevations are optionally clamped to zlim and normalized to $[0, 1]$ for palette lookup.
- When zlim is provided, values outside the range are clamped so the color scale stays comparable.

Legend

- The legend shows the same color scale as used for points.
- When `legend_label_mode = "norm_z"`, labels are shown as `norm=0 (z = ...)` and `norm=1 (z = ...)` to clarify that 0/1 refers to the normalized scale.
- Use `legend_height_frac` to shorten the legend visually (e.g., 0.5 = half height).

CRAN / non-interactive environments

- During `R CMD check`, the function returns early (no **rgl** window is opened).
- If **rgl** is not installed, a warning is raised and the function returns invisibly.

Value

Invisibly returns NULL.

Examples

```
if (requireNamespace("lidR", quietly = TRUE) && interactive()) {
    # Your plot code here
library(lidR)

las <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

# 1) Default plot (black bg, legend on, thickness by height)
plot_3d(las)

# 2) Custom palette + white background
plot_3d(
  las,
  bg = "white",
  height_palette = c("purple","blue","cyan","yellow","red"),
  title = "Custom palette"
)

# 3) Fixed Z color scale for comparisons + no legend
plot_3d(
  las,
  zlim = c(0, 40),
  add_legend = FALSE,
  title = "Fixed Z (0-40), no legend"
)

# 4) Turn OFF thickness-by-height; use a single point size
plot_3d(
  las,
  size_by_height = FALSE,
  size = 4,
  title = "Uniform thicker points"
)

# 5) Legend on the LEFT and thicker legend bar
plot_3d(
  las,
```

```
  legend_side = "left",
  legend_width_frac = 0.05,
  title = "Legend left"
)

# 6) Make everything thicker (multiplies size_range when size_by_height=TRUE)
plot_3d(
  las,
  size = 1.8,
  size_range = c(1, 7),
  size_power = 1.2,
  title = "Thicker points by height"
)
}
```

`plot_confusion_matrix` *Plot a confusion matrix heatmap (ggplot2)*

Description

Creates a heatmap visualization of a confusion matrix. If `row_normalize = TRUE`, each row is converted to proportions so rows sum to 1.

Usage

```
plot_confusion_matrix(
  cm,
  las_name = NULL,
  title = "Confusion Matrix",
  row_normalize = FALSE,
  digits = 3,
  show_values = TRUE,
  class_names = NULL,
  show_codes = FALSE,
  flip_y = TRUE,
  palette_type = c("default", "viridis", "brewer", "gradient", "base"),
  palette_name = NULL,
  brewer_direction = 1,
  gradient_low = "white",
  gradient_high = "#132B43",
  gradient_mid = NULL,
  base_n = 256,
  na_fill = "grey90",
  label_size = 4,
  auto_label_color = TRUE,
  label_color_dark = "white",
  label_color_light = "black"
)
```

Arguments

cm	A confusion matrix (table or matrix). Rows = true class, columns = predicted class.
las_name	Optional character. A short LAS/LAZ filename label appended to the title.
title	Character. Plot title.
row_normalize	Logical. If TRUE, normalize each row so it sums to 1 (proportions).
digits	Integer. Number of decimal digits to show when row_normalize = TRUE.
show_values	Logical. If TRUE, print values inside cells.
class_names	Optional named character vector mapping class codes to readable names. Example: c("0"="Ground","1"="Leaves","2"="Branch").
show_codes	Logical. If TRUE, show both code + name on axes (e.g., "1 - Stem").
flip_y	Logical. If TRUE, reverses y-axis order (often looks more like a typical CM).
palette_type	Character. Color palette strategy for the heatmap. Common options include "viridis", "brewer", and "gradient".
palette_name	Character. Palette name used when palette_type supports named palettes (e.g., viridis option name or RColorBrewer palette name).
brewer_direction	Integer. Direction for RColorBrewer palettes. Use 1 for default direction, -1 to reverse.
gradient_low	Character. Low-end color for palette_type="gradient". Can be any valid R color (e.g., "white" or "#FFFFFF").
gradient_high	Character. High-end color for palette_type="gradient". Can be any valid R color.
gradient_mid	Character. Mid-point color for palette_type="gradient". Can be any valid R color.
base_n	Integer. Number of colors to generate for discrete palettes (used when generating a ramp for the heatmap).
na_fill	Character. Fill color used for NA cells in the heatmap.
label_size	Numeric. Text size for cell labels (counts/percentages) and/or axis labels, depending on your implementation.
auto_label_color	Logical. If TRUE, automatically chooses a readable label text color based on the cell fill (improves contrast).
label_color_dark	Character. Label text color to use on light cells when auto_label_color=TRUE.
label_color_light	Character. Label text color to use on dark cells when auto_label_color=TRUE.

Value

A ggplot object (invisibly).

Examples

```
if (requireNamespace("lidR", quietly = TRUE) &&
    requireNamespace("ggplot2", quietly = TRUE)){

library(lidR)

# Read LAS/LAZ
las <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

# Confusion matrix: True labels vs Predicted class (LAS Classification)
cm <- table(True = las@data$label, Pred = las@data$Classification)

# ------------------------------------------------------------
# 1) Row-normalized confusion matrix (Proportions)
#    - Best to understand per-class recall behavior
#    - row_normalize = TRUE is important here
# ------------------------------------------------------------
plot_confusion_matrix(
  cm,
  row_normalize = TRUE,
  las_name = "trees.laz",
  title = "Confusion Matrix (Row-normalized)",
  class_names = c("0" = "Ground", "1" = "Leaves", "2" = "Branch"),
  palette_type = "viridis",
  palette_name = "cividis"
)

# ------------------------------------------------------------
# 2) Counts confusion matrix (Raw counts)
#    - Shows absolute misclassification volume
#    - row_normalize = FALSE is important here
# ------------------------------------------------------------
plot_confusion_matrix(
  cm,
  row_normalize = FALSE,
  las_name = "trees.laz",
  title = "Confusion Matrix (Counts)",
  class_names = c("0" = "Ground", "1" = "Leaves", "2" = "Branch"),
  palette_type = "viridis",
  palette_name = "viridis"
)

# ------------------------------------------------------------
# 3) Brewer palette example (soft + classic)
#    - Works great for both normalized and counts
# ------------------------------------------------------------
plot_confusion_matrix(
  cm,
  row_normalize = TRUE,
  las_name = "trees.laz",
  title = "Confusion Matrix (Brewer Blues, Row-normalized)",
  class_names = c("0" = "Ground", "1" = "Leaves", "2" = "Branch"),
```

```
  palette_type = "brewer",
  palette_name = "Blues"
)

# ------------------------------------------------------------
# 4) Custom modern gradient (minimal + professional)
# ------------------------------------------------------------
plot_confusion_matrix(
  cm,
  row_normalize = TRUE,
  las_name = "trees.laz",
  title = "Confusion Matrix (Custom Gradient, Row-normalized)",
  class_names = c("0" = "Ground", "1" = "Leaves", "2" = "Branch"),
  palette_type = "gradient",
  gradient_low  = "white",
  gradient_high = "#2C3E50"
)

# ------------------------------------------------------------
# 5) Base palette example (if you still want them)
#    - heat / terrain / topo / cm / rainbow
# ------------------------------------------------------------
plot_confusion_matrix(
  cm,
  row_normalize = TRUE,
  las_name = "trees.laz",
  title = "Confusion Matrix (Base heat, Row-normalized)",
  class_names = c("0" = "Ground", "1" = "Leaves", "2" = "Branch"),
  palette_type = "base",
  palette_name = "heat"
)
}
```

`predict` *Predict fuel classes for a LAS/LAZ file using a pre-trained model*

Description

Runs inference on a LAS/LAZ file and writes a new LAS/LAZ with predicted fuel classes. This function uses Python inference code shipped in `inst/extdata/python` and loads a pre-trained PyTorch model via `reticulate`.

Usage

```
predict(
  cfg,
  mode = c("overwrite", "extra"),
  setup_env = FALSE,
  csf_args = list()
)
```

Arguments

cfg	A configuration list created by config().
mode	Character. "overwrite" replaces the LAS Classification values with model predictions; "extra" keeps original classification and adds a new attribute for predictions (behavior depends on the Python writer).
setup_env	Logical. If TRUE, calls ensure_py_env() (or your env setup helper) before importing Python modules. Default: FALSE.
csf_args	List of arguments passed to add_ground_csf() when cfg$num_classes == 4.

Details

- **Model classes:** The shipped model is always loaded as a 3-class classifier (num_classes = 3 in the Python model). Predictions are produced for these 3 classes.
- **Output writing:** Predictions are written using the Python helper write_predictions_to_las().

  mode = "overwrite" Replaces the LAS Classification field with predictions.

  mode = "extra" Keeps the original Classification and writes predictions to an additional attribute (behavior is defined by the Python writer).
- **Optional 4th class (ground):** If cfg$num_classes == 4, the function post-processes the 3-class prediction LAS with add_ground_csf to detect ground points and assign them to class 3, producing a final 4-class LAS/LAZ.
- **Device selection:** Uses cfg$device if provided; otherwise selects "cuda" when available and falls back to "cpu".

Note: setup_env is accepted for API compatibility; if you want it to actually run the environment setup, add a call such as if (isTRUE(setup_env)) ensure_py_env() before importing.

Value

A character string giving the path to the output LAS/LAZ file.

Examples

```
if (requireNamespace("reticulate", quietly = TRUE) &&
    reticulate::py_module_available("torch")) {

library(FuelDeep3D)
library(reticulate)
use_condaenv("pointnext", required = TRUE)

cfg <- config(
  las_path     = system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"),

  # Option 2: write to a custom folder (edit this path)
  # out_pred_dir = "C:/Users/yourusername/Downloads/FuelDeep3D_predictions",
  out_pred_dir = file.path(tempdir(), "FuelDeep3D_predictions"),
 model_path   = system.file("extdata", "model", "best_model.pth", package = "FuelDeep3D"),
  num_classes  = 3
)
```

```
out_las <- predict(cfg, mode = "overwrite", setup_env = FALSE)
out_las
}
```

---

predicted_plot3d *Plot a LAS point cloud in 3D colored by a class field*

Description

Visualize a lidR LAS object in an interactive rgl window and color points by any discrete class field stored in las@data.

Usage

```
predicted_plot3d(
  las,
  field = "Classification",
  bg = "white",
  title = "LAS 3D View",
  class_colors = "default",
  class_labels = "default",
  show_all_classes = NULL,
  downsample = c("none", "voxel", "random"),
  voxel_size = 0.1,
  max_points = 200000L,
  size = 2,
  size_by_height = FALSE,
  size_range = c(0.7, 2.5),
  size_power = 1.2,
  verbose = TRUE,
  zoom = 0.7,
  theta = 0,
  phi = -90
)
```

Arguments

las	A LAS object from lidR.
field	Character. Column name in las@data used to color points (e.g., "label", "Classification").
bg	Background color of the 3D scene (passed to rgl::bg3d()). Use a dark background (e.g., "black") for bright colors, or a light background (e.g., "white") for darker colors.
title	Character. Title shown in the rgl window. Non-ASCII characters may be transliterated for compatibility.

`class_colors` Controls the class-to-color mapping. Supported forms:

- `"default"`: FuelDeep3D default palette for classes (recommended)
- `"auto"` or `NULL`: generate a distinct palette automatically
- Named character vector: explicit mapping, e.g. `c("0"="#1F77B4","1"="#8B4513","2"="#228B22`
- Unnamed character vector: assigned in the order of legend classes (i.e., `show_all_classes` if provided, otherwise `sort(unique(field))`), e.g. `c("blue","brown","green")`.

Note: color names must be valid R colors (see `grDevices::colors()`). Some CSS names (e.g. `"lime"`) are not valid in base R; use hex (e.g. `"#00FF00"`) or a valid R name like `"limegreen"`.

`class_labels` Controls the class-to-name mapping (used only in console output). Supported forms:

- `"default"`: default names for classes 0..3 (printed in the console)
- `"none"` or `NULL`: do not rename classes (print raw class values)
- Named character vector: e.g. `c("0"="Ground","1"="Stem","2"="Crown")`

`show_all_classes` Optional. A vector of class values (e.g., `c(0,1,2,3)`) to force a stable class order for color assignment and console printing, even if some classes are not present in the current LAS object.

`downsample` Downsampling mode:

- `"none"` (default): plot all points
- `"random"`: randomly sample up to `max_points`
- `"voxel"`: keep one point per voxel of size `voxel_size`

`voxel_size` Numeric. Voxel size (in LAS units) used when `downsample = "voxel"`. Smaller values retain more points; larger values reduce density more aggressively.

`max_points` Integer. Maximum number of points plotted when `downsample = "random"`.

`size` Numeric. Base point size passed to `rgl::points3d()`. Smaller values show finer detail; larger values increase thickness.

`size_by_height` Logical. If `TRUE`, point thickness increases with height (Z), implemented by binning points into layers for compatibility across platforms.

`size_range` Numeric length-2. Multipliers applied to `size` when `size_by_height = TRUE`. Example `c(0.7, 2.5)` means low points use `0.7*size` and high points use `2.5*size`.

`size_power` Numeric. Growth curve for thickness when `size_by_height = TRUE`. Values > 1 emphasize thicker points near the top; values < 1 increase thickness earlier.

`verbose` Logical. If `TRUE`, prints:

- total points vs plotted points
- the class-to-color mapping
- (optional) class display names

`zoom`, `theta`, `phi` Camera controls passed to `rgl::view3d()`.

Details

This function is designed for both *raw* and *predicted* outputs:

- `field = "label"`: color by original labels stored in `las@data$label`
- `field = "Classification"`: color by predicted classes stored in `las@data$Classification`

A fixed in-window legend overlay is intentionally **not** implemented. Instead, when `verbose = TRUE`, a class-to-color (and optional class-to-name) mapping is printed to the R console for clarity and reproducibility.

Value

Invisibly returns a list with:

mapping data.frame of class, name, and color used

n_total number of points in input LAS

n_plotted number of points actually plotted

downsample downsampling mode used

field field used for coloring

Examples

```
if (requireNamespace("lidR", quietly = TRUE) && interactive()){

library(lidR)
las <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

# 1) Predicted classes (default palette; no legend overlay)
predicted_plot3d(
  las,
  field = "Classification",
  bg = "white",
  title = "Predicted classes"
)

# 2) Raw labels
predicted_plot3d(
  las,
  field = "label",
  bg = "black",
  title = "Original labels"
)

# 3) Named custom colors (stable mapping)
my_cols <- c("0"="#1F77B4", "1"="#8B4513", "2"="#228B22")
my_labs <- c("0"="Ground vegetation", "1"="Leaves/Foliage", "2"="Branch/Stem")
predicted_plot3d(
  las,
  field = "Classification",
  class_colors = my_cols,
```

```
  class_labels = my_labs,
  bg = "white"
)

# 4) Unnamed custom colors (assigned in class order)
# If classes are 0,1,2 this maps 0->black, 1->red, 2->green.
predicted_plot3d(
  las,
  field = "Classification",
  show_all_classes = c(0,1,2),
  class_colors = c("black", "red", "#00FF00"), # hex is safest for "lime"
  bg = "white"
)

# 5) Downsample (voxel) for huge point clouds
predicted_plot3d(
  las,
  field = "Classification",
  downsample = "voxel",
  voxel_size = 0.10,
  size = 2,
  bg = "white"
)

# 6) Thickness by height
predicted_plot3d(
  las,
  field = "Classification",
  size = 1.2,
  size_by_height = TRUE,
  size_range = c(0.8, 2.8),
  size_power = 1.2
)
}
```

`print_confusion_matrix` *Print a confusion matrix (LAS/LAZ or precomputed cm)*

Description

Prints a confusion matrix as a readable table. You can pass either:

- a precomputed confusion matrix `table/matrix`, OR
- a `lidR::LAS` object (then `truth_col` and `pred_col` are used to build the matrix).

Usage

```
print_confusion_matrix(
  x,
```

```
  truth_col = "label",
  pred_col = "Classification",
  row_normalize = FALSE,
  digits = 3,
  classes = NULL,
  class_names = NULL,
  show_codes = TRUE,
  drop_na = TRUE
)
```

Arguments

x	A confusion matrix (table/matrix) OR a lidR::LAS object.
truth_col	Character. Truth label column name (used when x is LAS).
pred_col	Character. Prediction column name (used when x is LAS).
row_normalize	Logical. If TRUE, each row is converted to proportions (rows sum to 1).
digits	Integer. Number of decimal places to show when row_normalize = TRUE.
classes	Optional integer vector of expected class IDs (e.g., 0:2). Keeps matrix shape stable even if some classes are missing.
class_names	Optional mapping for nicer labels (named or unnamed).
show_codes	Logical. If TRUE (default), labels display like "0 (Ground)".
drop_na	Logical. Drop rows where truth/pred is NA/Inf (used when x is LAS).

Value

Invisibly returns the printed data frame.

Examples

```
if (requireNamespace("lidR", quietly = TRUE)){

library(lidR)
las <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

print_confusion_matrix(
  las,
  truth_col = "label",
  pred_col  = "Classification",
  classes   = 0:2,
  class_names = c("0"="Ground","1"="Branch","2"="Leaves"),
    row_normalize = TRUE,
    digits = 3
)
}
```

`print_metrics_table` *Print per-class metrics and summary averages*

Description

Formats and prints a per-class metrics table from the output of `evaluate_single_las` or `evaluate_two_las`. The table includes per-class Support, Accuracy, Precision, Recall, and F1. Optionally includes Macro and Weighted averages, and an Overall accuracy row.

Usage

```
print_metrics_table(
  results,
  digits = 4,
  include_macro = TRUE,
  include_weighted = TRUE,
  include_overall_accuracy = FALSE
)
```

Arguments

`results`	List returned by `evaluate_single_las()` or `evaluate_two_las()`.
`digits`	Integer. Number of decimal places to round numeric metrics.
`include_macro`	Logical. If TRUE, include a "Macro avg" row.
`include_weighted`	Logical. If TRUE, include a "Weighted avg" row.
`include_overall_accuracy`	Logical. If TRUE, include an "Overall accuracy" row.

Details

- **Support**: number of true points in each class (row sum of confusion matrix).
- **Accuracy**: per-class accuracy here means **Recall** (TP / (TP + FN)). This is sometimes called *producer's accuracy*.
- **Macro avg**: unweighted mean across classes (ignores class imbalance).
- **Weighted avg**: mean across classes weighted by Support (reflects imbalance).
- **Overall accuracy**: sum(diag(cm)) / sum(cm). This is optional because it often duplicates the weighted recall/accuracy in multi-class settings.

Value

Invisibly returns a `data.frame` with per-class metrics and optional summary rows.

Examples

```
if (requireNamespace("lidR", quietly = TRUE)){
library(lidR)
las <-  readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

res <- evaluate_single_las(
  las,
  truth_col = "label",
  pred_col  = "Classification",
  classes   = 0:2,
  class_names = c("0"="Ground","1"="Branch","2"="Leaves")
)

print_metrics_table(res, include_overall_accuracy = TRUE)
}
```

`remove_noise_sor` *Remove sparse outlier points using Statistical Outlier Removal (SOR)*

Description

Applies a k-nearest-neighbor Statistical Outlier Removal (SOR) filter to points above a user-defined height threshold. Points at or below the threshold are preserved unchanged.

Usage

```
remove_noise_sor(las, height_thresh = 5, k = 20, zscore = 2.5)
```

Arguments

`las`	A `lidR::LAS` object.
`height_thresh`	Numeric. Height (meters) above which filtering is applied.
`k`	Integer. Number of nearest neighbors used by the SOR filter.
`zscore`	Numeric. Standard deviation multiplier controlling outlier rejection.

Details

The filter is applied only to points with `Z` is greater than `height_thresh`. For each of these points, the mean distance to its `k` nearest neighbors is computed in 3D XYZ space. A point is kept if:

$$d_i < mean(d) + zscore * sd(d)$$

where `d_i` is the mean kNN distance for point i.

To keep behavior stable and safe, `k` is automatically capped so that `k < n`, where `n` is the number of points being filtered.

This function preserves the original point order by computing a global keep mask and subsetting once.

Value

A filtered `lidR::LAS` object.

Examples

```
# Check for both lidR and dbscan before running
if (requireNamespace("lidR", quietly = TRUE) &&
    requireNamespace("dbscan", quietly = TRUE)) {

 las <- lidR::readLAS(system.file("extdata", "las", "tree2.laz", package = "FuelDeep3D"))

  if (!lidR::is.empty(las)) {
    las_small <- las[seq_len(min(20000, lidR::npoints(las)))]
    las_clean <- remove_noise_sor(las_small, height_thresh = 1, k = 10, zscore = 2.5)

    # Optional: Print to console to show it worked
    print(lidR::npoints(las_small))
    print(lidR::npoints(las_clean))
  }
}
```

`train` *Train the FuelDeep3D model (build NPZ tiles if missing)*

Description

Trains the FuelDeep3D point-cloud model using the Python training pipeline shipped with the package (under `inst/extdata/python`) and executed via **reticulate**. If preprocessed NPZ tiles are not found in `file.path(cfg$out_dir, "train")`, the function automatically runs the dataset builder to generate `train/val/test` NPZ tiles from the input LAS/LAZ before starting training.

Usage

```
train(cfg, setup_env = FALSE)
```

Arguments

`cfg`	A list created by `config()`.
`setup_env`	Logical; if `TRUE`, calls `ensure_py_env()` to create/use a venv.

Details

Training proceeds in two stages:

1. **Dataset preparation (optional):** If no `.npz` files are found in `cfg$out_dir/train`, the function calls the Python function `dataset.build_dataset_from_las()` to create NPZ tiles for training, validation, and testing. Dataset tiling behavior is controlled by fields in `cfg` (e.g., `block_size`, `stride`, `sample_n`, `repeat_per_tile`, `min_pts_tile`, `cell_size`, `quantile`, splits, and `seed`).

2. **Model training:** The function calls `train.train_model(cfg)` in the shipped Python code. The returned object (e.g., best metrics and checkpoint path) is converted back to R with `reticulate::py_to_r()`.

Environment: This function requires a working Python environment with the required dependencies installed (e.g., PyTorch). If `setup_env = TRUE`, it calls `ensure_py_env` before importing Python modules.

Safety during checks: `train()` is intentionally disabled during `R CMD check` to avoid running long, non-deterministic computations on CRAN.

Optional cleanup: If `cfg$delete_tiles_after_train` is `TRUE`, the generated NPZ directories `train/`, `val/`, and `test/` under `cfg$out_dir` are deleted after training completes.

Value

A list with training outputs (e.g., best metrics and checkpoint path), returned from the Python trainer.

Examples

```
if (requireNamespace("reticulate", quietly = TRUE) &&
    reticulate::py_module_available("torch")) {

library(FuelDeep3D)
library(reticulate)
use_condaenv("pointnext", required = TRUE)

cfg <- config(
  las_path     = system.file("extdata", "las", "trees.laz", package = "FuelDeep3D"),
  out_dir      = system.file("extdata", "npz_files", package = "FuelDeep3D"),
  out_pred_dir = system.file("extdata", "output_directory", package = "FuelDeep3D"),
 model_path  = system.file("extdata", "model", "best_model.pth", package = "FuelDeep3D"),
  epochs       = 2, batch_size = 16,
  learning_rate = 1e-5, weight_decay = 1e-4,
  block_size = 6, stride = 1, sample_n = 4096,
  repeat_per_tile = 4, min_pts_tile = 512,
  cell_size = 0.25, quantile = 0.05,
  delete_tiles_after_train = TRUE
)

res <- train(cfg, setup_env = FALSE)        # trains & saves best .pth

}
```

Index