Methodology for applying the Laserfarm workflow to calculate LiDAR vegetation metrics across seven demonstration sites from five European countries

W. Daniel Kissling, Wessel Mulder, Jinhu Wang & Yifang Shi

Institute for Biodiversity and Ecosystem Dynamics (IBED), University of Amsterdam, P.O. Box 94240, 1090 GE Amsterdam, The Netherlands

This document describes the methodology of applying the Laserfarm workflow (Kissling et al. 2022) for calculating LiDAR vegetation metrics across several European demonstration sites from the EU-project MAMBO (Modern Approaches to the Monitoring of Biodiversity; Høye et al. 2023). The following pages cover the details of the methodology.

Background on Laserfarm workflow

Laserfarm is a high-throughput workflow for generating geospatial data products of ecosystem structure using LiDAR point clouds from national or regional airborne laser scanning (ALS) surveys (Figure 1). It is a free and open-source end-to-end workflow which enables the efficient, scalable and distributed processing of multi-terabyte LiDAR point clouds. It has been developed to obtain high-resolution raster layers (GeoTIFF files) of LiDAR vegetation metrics at a national or regional extent (Figure 1). The workflow has been applied to process the third Dutch national airborne laser scanning flight campaign (AHN3, Actueel Hoogtebestand Nederland) which contains ~700 billion points and ~ 16 TB uncompressed LiDAR point clouds with a point density of ~10(–20) points/m^2 across the whole Netherlands (Kissling et al. 2023).

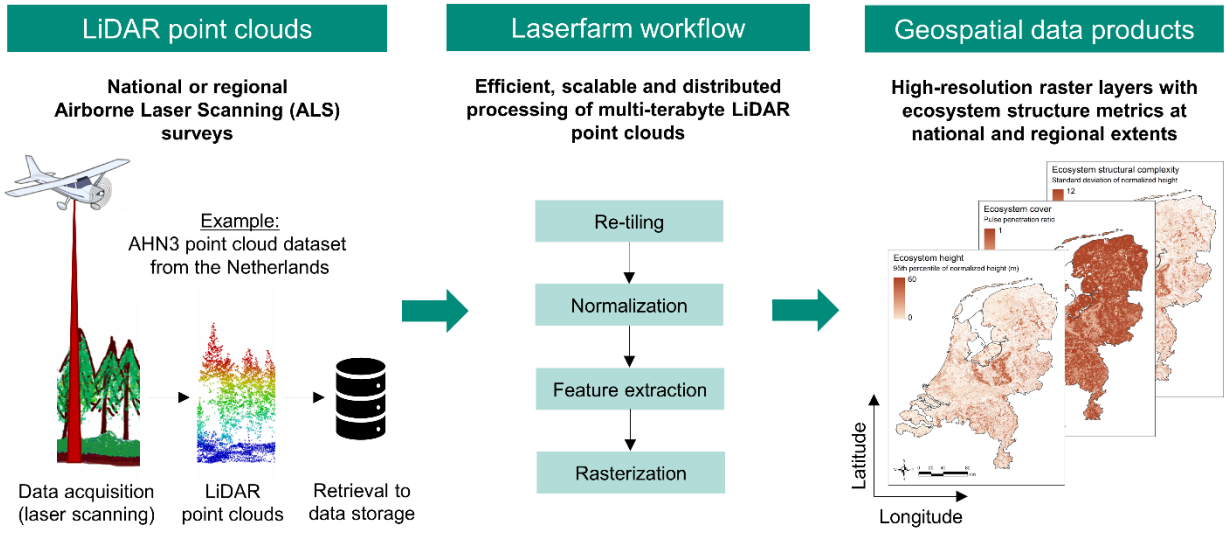


Figure 1: The Laserfarm workflow enables the processing of LiDAR point clouds from national or regional airborne laser scanning surveys into geospatial data products of ecosystem structure (from Kissling et al. 2022).

The Laserfarm workflow contains modular pipelines for (1) re-tiling, (2) normalization, (3) feature extraction and (4) rasterization of point cloud information from ALS and other LiDAR surveys (Figure 2). It is implemented in Python, available as Jupyter Notebooks, and builds on

the Laserchicken software for extracting statistical properties of flexibly defined subsets of point cloud data (Meijer et al. 2020). Laserchicken is a free open source software (FOSS) that has been specifically designed for extracting statistical properties of point cloud data from multi-terabyte datasets (Meijer et al. 2020).

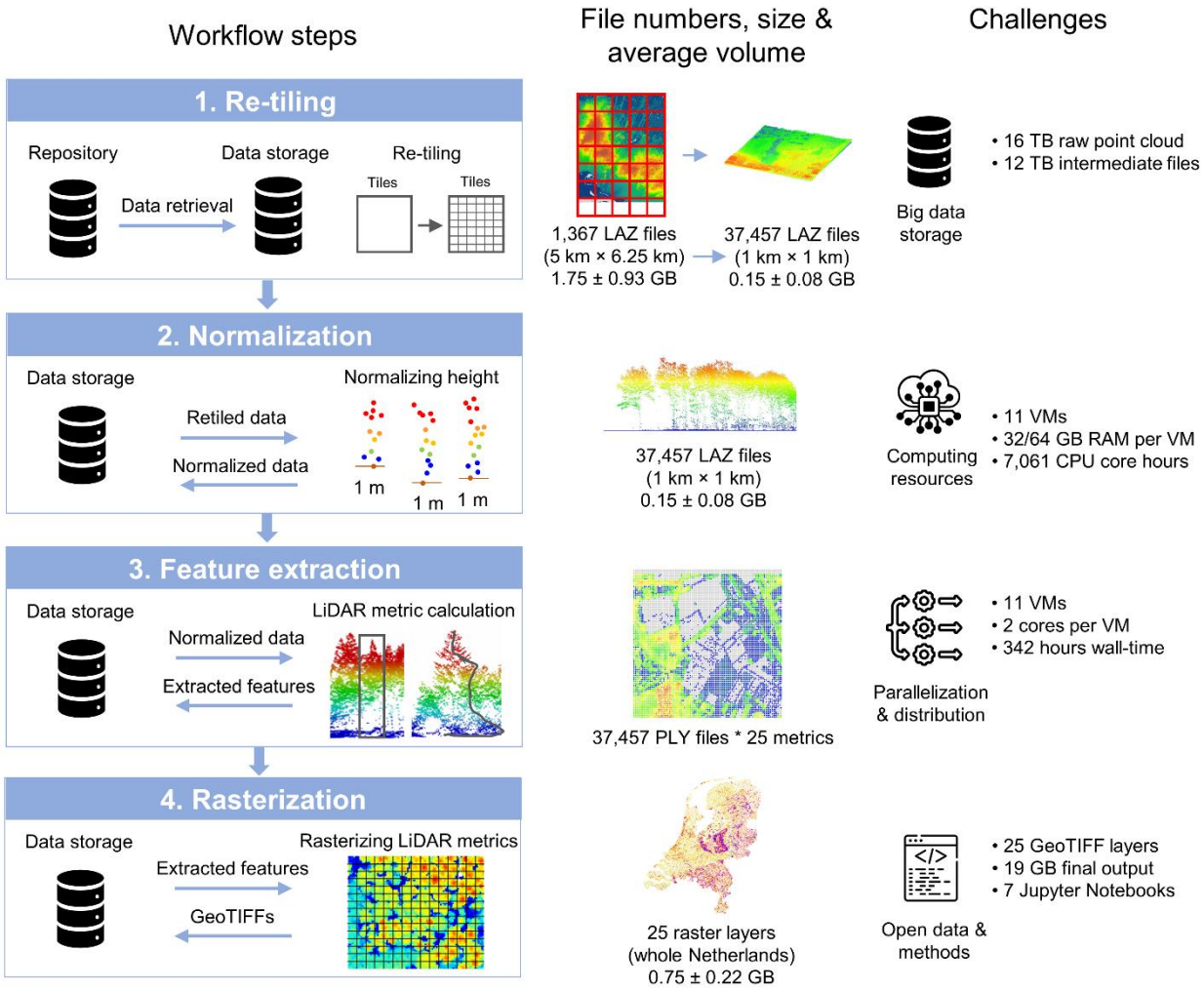


Figure 2: Example of applying the Laserfarm workflow to process a country-wide airborne laser scanning dataset. Left: The four core modules of the Laserfarm workflow. Right: Details of handling the number, sizes and volumes of files for processing the third Dutch national airborne laser scanning flight campaign (AHN3, Actueel Hoogtebestand Nederland). Figure from Kissling et al. (2024). See Kissling et al. (2022) for details of the workflow, and Kissling et al. (2023) for the derived data product.

Detailed information about the Laserfarm workflow and the underlying Laserchicken software is available from the following sources:

- Laserchicken software:
  - User manual: https://laserchicken.readthedocs.io/en/latest/
  - Source code in Python (GitHub): https://github.com/eEcoLiDAR/laserchicken
  - Tutorial as Jupyter Notebook (GitHub): https://github.com/eEcoLiDAR/laserchicken/blob/master/tutorial.ipynb
- Laserfarm workflow:

- User manual: https://laserfarm.readthedocs.io/en/latest/
- Current version: on PyPI (https://pypi.org/project/laserfarm/) or Zenodo (https://doi.org/10.5281/zenodo.3842780)
- Code produced during the development of Laserfarm (on GitHub): https://github.com/eEcoLiDAR/Laserfarm
- Jupyter Notebooks from the development phase:
  - A tutorial structured as a Jupyter Notebook (tutorial.ipynb) which illustrates the use of the Laserfarm workflow to process a subset of the Dutch AHN3 dataset (from the retrieval of an example point cloud data file in LAZ format to the export of the extracted features to a GeoTIFF file): https://github.com/eEcoLiDAR/Laserfarm/blob/master/tutorial.ipynb
  - A Jupyter Notebook (workflow.ipynb) that illustrates the processing of the full country-wide LiDAR dataset from the third Dutch national ALS flight campaign (AHN3), including setting up the login, the connection to remote storage, and the Dask cluster for pipeline calculations: https://github.com/eEcoLiDAR/Laserfarm/blob/master/workflow.ipynb
  - Ten Jupyter Notebooks illustrating each individual workflow step as used in the original processing of the AHN3 LiDAR dataset (https://github.com/eEcoLiDAR/AHN/tree/main/AHN3)

Pre-processing of LiDAR point clouds for MAMBO demonstration sites

We applied a workflow to pre-process and prepare the raw LiDAR point clouds for each of the MAMBO demonstration sites (Figure 3). This pre-processing workflow required as input the raw LiDAR point clouds from the (national) repositories and the approximate location and exact boundaries of the study areas.

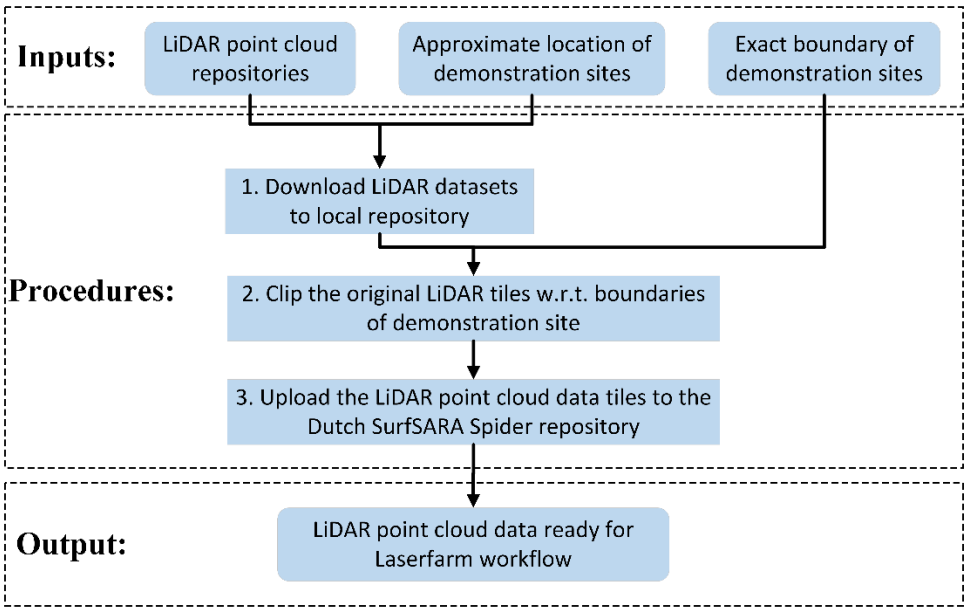


Figure 3: Workflow for pre-processing the raw LiDAR point clouds from national repositories in different European countries to obtain LiDAR point clouds clipped to the specific study areas (demonstration sites).

In a first step, we downloaded the raw LiDAR point cloud tiles from the national repositories in each country based on the approximate location of the demonstration sites (Figure 1, see access links in Appendix Table A1). In a second step, the point cloud datasets were then clipped using boundary polygons (shapefiles) of the demonstration sites (Figure 1). These shapefiles were provided by local partners who work in the study areas. In the third and final pre-processing step we uploaded the clipped LiDAR point clouds to a local data repository (Figure 1). We used the services from the Dutch IT infrastructure SURF (https://www.surf.nl/en) and created a publicly accessible repository for the clipped LiDAR point clouds and the shapefiles of the study areas (https://public.spider.surfsara.nl/project/lidarac/MAMBO/). The code for the pre-processing workflow is available on GitHub (https://github.com/Jinhu-Wang/Retile_Clip_LAZ).

In addition to downloading, clipping and storing the LiDAR point clouds for each demonstration site, we also examined the properties of each LiDAR point cloud dataset. This was done using the CloudCompare software (https://www.danielgm.net/cc/). We loaded the LAS/LAZ file into the software and ensured that the relevant properties such as intensity, classification, GPS time, number of returns, edge of flight lines, point source ID, etc. were selected. To verify the specific property, we select the dataset in the DB tree panel of CloudCompare and applied the scalar field in the properties panel of the software. We then inspected the available values in the active scalar fields. All characteristics of the LiDAR point clouds (e.g. time of acquisition, average point density, classification, number of returns and other point cloud properties) from the MAMBO demonstration sites are summarized in Appendix Table A1.

During point cloud inspection, we noticed that one of the study sites (Mols Bjerge National Park) contained outliers from errors during the LiDAR data acquisition. We removed those outliers using 'Sparse Outlier Removal', a density-based filtering method (Rusu et al. 2008). This method detects sparse outliers by first computing the mean μ and standard deviation σ of the k nearest neighbour distances. It then trims points that fall outside $\mu \pm \alpha\sigma$. The value of α depends on the size of the neighbourhood k. We used the SOR tool in CloudCompare to perform the outlier cleaning. We first loaded the point cloud data, then selected it in the DB Tree panel of CloudCompare, and then applied the "SOR" button tool. The two parameters α and k were set to 1 and 10, respectively.

Implementing the Laserfarm workflow for MAMBO demonstration sites

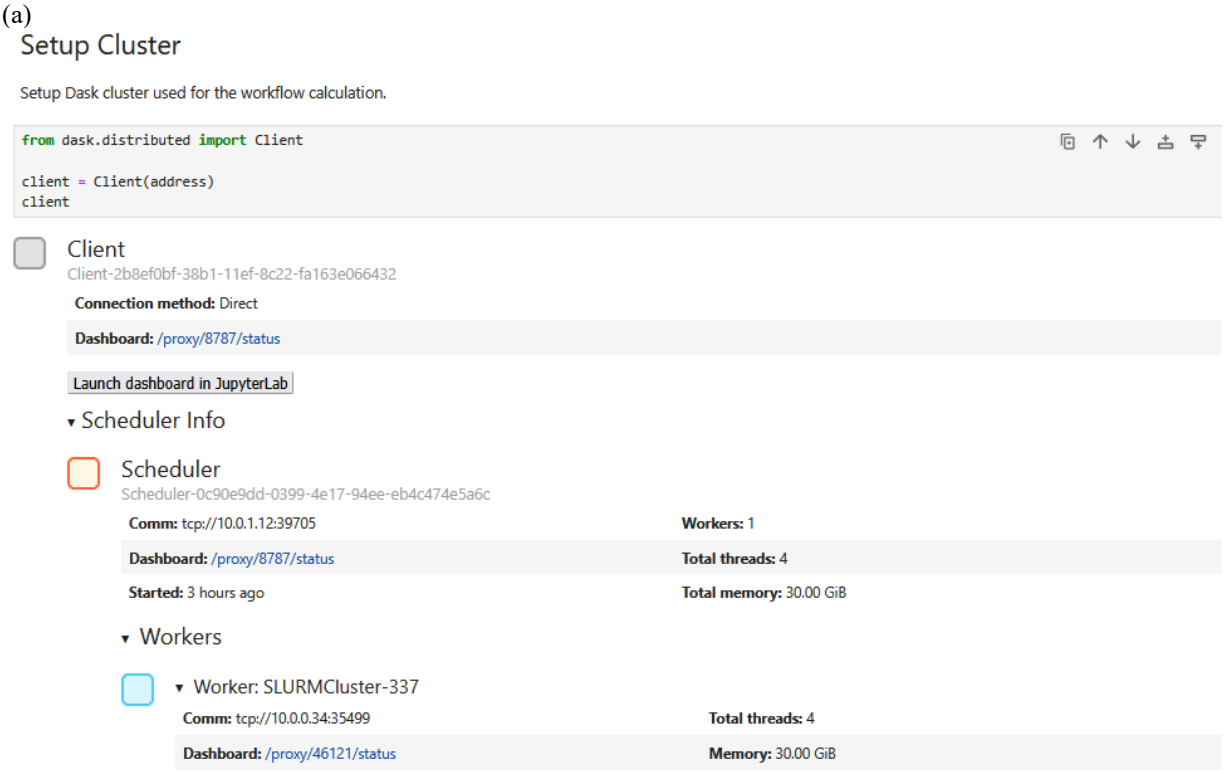
Setting up the high-performance computing environment

To implement the Laserfarm Jupyter Notebooks for processing the LiDAR point clouds from the MAMBO demonstration sites, we used the IT services of the Dutch national facility for information and communication technology SURF (https://www.surf.nl/en/ict-facilities). SURF provides access to a national IT infrastructure for the Dutch academic community, including the high-performance data processing platform 'Spider' on which the computations were performed (https://www.surf.nl/en/services/high-performance-data-processing). Spider is an in house compute cluster of SURF and allows users to run highly parallel jobs on distributed resources, using scalable processing of many terabytes of data and utilizing many hundreds of cores simultaneously (https://servicedesk.surf.nl/wiki/display/WIKI/Spider). We conducted several execution steps to set-up the Laserfarm Jupyter Notebooks in the Spider HPC environment. This included to make a directory, to build and activate the Jupyter-Dask environment, to install the Laserfarm software, to prepare the configuration files, and to start the Jupyter server (see details

of execution steps in Appendix Table A2). We also used the free software Cyberduck (https://cyberduck.io/) to have a graphical user interface to the folders and files in our HPC environment. This made the previewing and downloading of files easier.

In each Laserfarm Jupyter Notebook, a connection to the HPC environment can be established with the 'Client' function (Figure 4a). Users can configure the computing environment, e.g. in terms of number of workers and cores for each worker. The compute cluster must also be specified in the MacroPipeline function of each Jupyter Notebook (Figure 4b). This is done by making the connection to the address of the cluster ('macro.setup_cluster' in Figure 4b).

(a)



(b)

```
macro = MacroPipeline()

# add pipeline list to macro-pipeline object and set the corresponding labels
macro.tasks = [Retiler(file).config(retiling_input) for file in laz_files]
macro.set_labels([os.path.splitext(file)[0] for file in laz_files])

#EDIT
macro.setup_cluster(cluster=address)

# run!
macro.run()

# save outcome results and check that no error occurred before continuing
macro.print_outcome(to_file='retile.out')

failed = macro.get_failed_pipelines()
if failed:
    with open(filename, 'w') as f:
        json.dump([pip.label + '.LAZ' for pip in failed], f)
    raise RuntimeError('Some of the pipelines have failed')
print('finished')
```

Figure 4: Example of setting-up the compute cluster in a Laserfarm Jupyter Notebook. (a) Use of the client function to establish a connection to the computing environment. (b) Code in the Jupyter Notebook to connect to the cluster.

In addition to setting up the cluster, each Laserfarm Jupyter Notebook needs to specify the input/ output files and the root directory (Figure 5a). The connection to the data storage also needs to be checked (Figure 5b).

(a)

Set Run-Specific Input

Fill in the username/password for the SURF dCache. Choose whether you want to i) run all input files, ii) run the only input files listed in `filename`, or iii) run the input that was updated since the last workflow run.

```
### SPECIFY ROOTS
root = '/project/lidarac/Data/Wessel'
cc = 'FR'
area = 'Bagnas'
address = 'tcp://10.0.0.34:43683'
input = 'unprocessed'
output = 'retiled'
spatial_resolution = 10

output = output + '_' + str(spatial_resolution) + 'm'

filename = cc + '_' + output + '_failed.json'

### WHAT TO RUN
run = 'all' # 'all', 'updated', 'from_file'
# if run is 'from_file', set name of file with input file names
assert run in ['all', 'updated', 'from_file']

# define path
remote_path_root = pathlib.Path(root)
remote_path_root = remote_path_root / cc / area

# dCache path to raw LAZ files
remote_path_input = remote_path_root / input

# dCache path where to copy retiled LAZ files
#EDIT
remote_path_output = remote_path_root / output
```

(b)

Check Connection to Remote Storage

```
laz_files = [f for f in listdir(remote_path_input) if isfile(join(remote_path_input, f))
             if f.lower().endswith('.laz')]
print('Found: {} LAZ files'.format(len(laz_files)))
if run == 'updated':
    laz_files = [f for f in laz_files]
elif run == 'from_file':
    with open(filename, 'r') as f:
        laz_files_read = json.load(f)
    # check whether all files are available on dCache
    assert all([f in laz_files for f in laz_files_read]), f'Some of the files in {filename} are not in remote dir'
    laz_files = laz_files_read
print('Retrieve and retile: {} LAZ files'.format(len(laz_files)))
```

Figure 5: Example of setting-up and checking the file connection in a Laserfarm Jupyter Notebook. (a) Specifying the input/output. (b) Checking the connection to the data storage.

Overview of Jupyter Notebooks

The Laserfarm workflow includes four modules (Figure 2). For each module, we used 1–2 Jupyter Notebooks, resulting in a total of six Jupyter Notebooks to implement the four Laserfarm modules for the MAMBO demonstration site (Table 1). For both the feature extraction and rasterization module, two Jupyter Notebooks were needed because the metric calculation required different LiDAR point clouds as input, one with vegetation points only (most metrics) and another one with all points (only one metric). The vegetation points were defined using the standard point classes that were available from the classification of the LiDAR point clouds of each demonstration site (Appendix Table A1).

Table 1: Details of Jupyter Notebooks for implementing the Laserfarm workflow for the MAMBO demonstration sites.

Jupyter Notebooks	Description
Re-tiling	
1_Retiling.ipynb	Re-tiles the original files from LiDAR repositories into smaller chunks for further efficient, scalable and distributed processing. Requires defining a grid-like structure for re-tiling and a spatial resolution (tile mesh size) for the final GeoTIFF files (here 10 m).
Normalization	
2_Normalization.ipynb	Normalizes the point cloud heights (z-values) relative to the terrain surface by calculating the normalized height for each individual point as the height relative to the lowest point within a grid cell. Requires defining a spatial resolution of the grid cell size for normalization (here 1 m).
Feature extraction	
3_Feature_extraction_veg.ipynb	Calculates LiDAR metrics ('features') with vegetation points, e.g. related to vegetation height, density, and vertical variability. Requires defining the spatial resolution (tile mesh size) for the metric calculation (here 10 m), the list of features, and the class(es) of points which are vegetation.
4_Feature_extraction_all.ipynb	Calculates LiDAR metrics ('features') of openness which use all points (not only vegetation points), namely the pulse penetration ratio (i.e. the ratio of the number of ground points to the total number of points within a grid cell). Requires defining the spatial resolution (tile mesh size) for the metric calculation (here 10 m) and the list of features (here only the pulse penetration ratio).
Rasterization	
5_Geotiff_export_veg.ipynb	Rasterizes the extracted features of vegetation (e.g. related to vegetation height, density, and vertical variability) and exports them as raster layers (here GeoTIFF format).
6_Geotiff_export_all.ipynb	Rasterizes the extracted features of openness (here pulse penetration ratio) and exports them as raster layers (here GeoTIFF format).

Re-tiling Jupyter Notebook

The first Jupyter Notebook in the Laserfarm workflow covers the re-tiling (Table 1). This facilitates computational speed by dividing the large tiles from the raw LiDAR point clouds into manageable smaller-sized chunks for subsequent processing. The resulting tile size must not exceed the memory limits of the workers that are available in the compute cluster. The re-tiling requires to define a regular grid, i.e. the minimum and maximum of the X and Y coordinates (min_x, max_x, min_y, max_y) of the bounding box around the region of interest and the number of tiles (n_tiles_side) along the side of the bounding box (Figure 6). The parameter value for n_tiles_side must be an integer that fits with its multiplication into the side length of the bounding box (e.g. max_x − min_x). For the MAMBO demonstration sites, the number of tiles per side ranged between 1–20 (Appendix Table A3), resulting in a mesh size (spatial resolution) of approximately 100 m for the resulting tiles. The specifications of the re-tiling grid are accessed through a text file in the Jupyter Notebook ('grids.txt' in Figure 6).

Re-tiling

The raw point cloud files are downloaded and retiled to a regular grid.

```
with open('grids.txt', 'r') as file:
    exec(file.read())

current_grid = grid_dict[area]
print(current_grid)
```

```
{'min_x': 739243.0, 'max_x': 744243.0, 'min_y': 6244168.0, 'max_y': 6249168.0, 'n_tiles_side': 20}
```

```
# set path where output will be written
retiling_input = {
    'setup_local_fs': {
        'input_folder': remote_path_input.as_posix(),
        'output_folder': remote_path_output.as_posix()
    },
    'set_grid': current_grid,
    'split_and_redistribute': {},
    'validate': {}
}
```

Figure 6: Code for re-tiling in the Laserfarm Jupyter Notebook. This requires the user to define the details of the re-tiling grid via a text file ('grids.txt') which includes the minimum and maximum of the X and Y coordinates (min_x, max_x, min_y, max_y) and the number of tiles (n_tiles_side).

Normalization Jupyter Notebook

The second Jupyter Notebook in the Laserfarm workflow covers the normalization (Table 1). This calculates the normalized height of each point by subtracting the height (z-value) of the lowest point within a grid cell from each individual point in the cell. The only parameter value that needs to be specified by the user is the spatial resolution for 'normalize' (Figure 7). We chose normalize = 1, i.e. normalization within a 1 m × 1 m grid cell. Additionally, artificially high points can be removed using the 'threshold' flag in the 'apply_filter' function.

Normalization

Generate the normalized height for each point.

```
# setup input dictionary to configure the normalization pipeline
normalization_input = {
    'setup_local_fs': {'input_folder': remote_path_input.as_posix(),
                       'output_folder': remote_path_output.as_posix()},
    'load': {'attributes': 'all'},
    #Filter out artifically high points - give overflow error when writing
    'apply_filter': {'filter_type':'select_below',
                     'attribute': 'z',
                     'threshold': 10000}, # remove non-physically heigh points
    #filter point cloud using polygons
    #'apply_filter': {'filter_type':'select_polygon',
    #                 'polygon_string': str(shapefile),
    #                 'read_from_file': True
    #                 },
    'normalize': normalize,
    'clear_cache' : {},
}

# write input dictionary to JSON file
with open('normalized.json', 'w') as f:
    json.dump(normalization_input, f)
```

Figure 7: Code for normalization in the Laserfarm Jupyter Notebook. This requires specifying the spatial resolution for normalization under 'normalize' (here: normalize = 1).

Feature extraction Jupyter Notebooks

The third and fourth Jupyter Notebooks in the Laserfarm workflow cover the LiDAR metrics calculation (Table 1). We calculated a total of 35 metrics (= features) which are available from the

Laserchicken software (https://laserchicken.readthedocs.io/en/latest/#features). A total of 28 metrics reflect various aspects of vegetation height, vegetation cover and vegetation vertical variability (Appendix Table A4). Seven additional features include point density, three eigenvalues and three normal vectors (Appendix Table A4). The feature calculation uses the normalized height from the Normalization Jupyter Notebook and requires specifying each LiDAR metric as a 'feature' (Figure 8). It is also important to define the point classes of interest for the feature extraction using the 'apply_filter' function (Figure 8). This specifies which points are used as vegetation points. For all demonstration sites except the one in the Netherlands, the available ALS point clouds provide a classification for vegetation points, i.e. the ASPRS standard point classes for low vegetation (3), medium vegetation (4), and high vegetation (5) (Appendix Table A1). This means that the 'value' in Figure 8 was defined with filter_values = [3,4,5]. For the ALS point cloud from Netherlands, vegetation points are specified using the class 'unclassified' (= 1). This corresponds to with filter_values = [1]. The class 'unclassified' can introduce some biases in the calculation of vegetation metrics if a grid cell contains not only vegetation points but also points from other objects such as cars, fences, poles, and boats in the class 'unclassified' (Kissling et al. 2023). However, this is negligible in the MAMBO demonstration site Oostvaardersplassen.

Feature Extraction

We extract features for the vegetation points.

```
# target mesh size

# select features
z = 'normalized_height' # most of the features are based on the normalized height
features = ['point_density', 'sigma_z']
features += [f"perc_{percentile}_{z}" for percentile in (25, 50, 75, 95)]
features += [f"{feature}_{z}" for feature in ("max",
                                              "median",
                                              "mean",
                                              "var",
                                              "std",
                                              "skew",
                                              "kurto",
                                              "coeff_var",
                                              "entropy",
                                              "density_absolute_mean")]
features += [f"band_ratio_{z}_1", f"band_ratio_1_{z}_2", f"band_ratio_2_{z}_3", f"band_ratio_3_{z}"]
features += [f"band_ratio_3_{z}_4",f"band_ratio_4_{z}_5",f"band_ratio_5_{z}_6",f"band_ratio_6_{z}_10" ]
features += [f"band_ratio_{z}_5",f"band_ratio_{z}_3", f"band_ratio_5_{z}_20", f"band_ratio_20_{z}"]
features += [f"eigenv_{i}" for i in range(1, 4)]
features += [f"normal_vector_{i}" for i in range(1, 4)]
#define custom feature extractors
custom_feature_extractors = [{'extractor_name': 'BandRatioFeatureExtractor',
                              'data_key': z,
                              'lower_limit': low_lim,
                              'upper_limit': up_lim} for low_lim, up_lim in ((None, 3),(None, 5), (3, 4), (4, 5), (5, 6), (6, 10), (5, 20), (20, None))]



# setup input dictionary to configure the feature extraction pipeline
feature_extraction_input_non_ground = {
    'setup_local_fs': {'input_folder': remote_path_input.as_posix(),
                       'output_folder': remote_path_output.as_posix()},
    'load': {'attributes': ['raw_classification', 'normalized_height']},
    'add_custom_features': {
        'custom_feature_list': custom_feature_extractors,
    },
    'apply_filter': {
        #unclassified (1), ground (2), buildings (6), water (9), wire conductor (14), artificial objects (26), never classified (0)
        #For Spain: unclassified (1), ground (2), low vegetation (3), medium vegetation (4), high vegetation (5), buildings (6),
        # noise (7), water (9), overlap (12), bridges (17).
        'filter_type': 'select_equal',
        'attribute': 'raw_classification',
        'value': filter_values,
```

Figure 8: Code for calculating vegetation-related features in the Laserfarm Jupyter Notebook. (a) Defining features. (b) Defining point classes of interest for the feature extraction by specifying the available standard point classes in filter_values. Here, filter_values included low, medium and high vegetation points (classes 3,4,5), except for the Dutch demonstration site where filter_values included the class unclassified (class 1).

Some of these Laserfarm features (e.g. density_absolute_mean_normalized_height, entropy_normalized_height, point_density, some eigenvalues and normal vectors) can result in zero values if no vegetation points are available in a grid cell, e.g. above water. Other metrics (i.e. all vegetation height features and all band ratios) will result in NA values in such areas.

The pulse penetration ratio is a measure of openness and is calculated as the ratio of the number of ground points to the total number of points within a grid cell (Appendix Table A4). It is the only metric that uses all points, and not only vegetation points. It requires a separate Jupyter Notebook (Table 1) which is like the other feature calculation code but without filtering for vegetation points (Figure 9). Above water, the pulse penetration ratio can result in zero values which wrongly indicate a high vegetation cover (i.e. low openness). This is caused by a lack of ground points above water surfaces which results in zero values. It is recommended to apply a water mask (e.g. shapefiles from water body mapping) if available for a specific study area.

Figure 9: Code for calculating the pulse penetration ratio in the Laserfarm Jupyter Notebook. The notebook is similar to the other feature calculations (see Figure 8) but does not include the 'apply_filter' function because it uses all points rather than only vegetation points.

Rasterization Jupyter Notebooks

The two final Jupyter Notebooks in the Laserfarm workflow allow to rasterize the extracted features and to export them as raster layers in GeoTIFF format (Table 1). The only input required from the user is to specify the coordinate reference system (Figure 10). This spatial reference system for each country must be defined using the standard machine-readable code from the European Petroleum Survey Group (EPSG) for coordinate systems worldwide (https://epsg.io/). The EPSG specifications are loaded with a text file ('epsgs.txt' in Figure 10). The country-specific EPSG codes for the MAMBO demonstration sites are provided in Appendix Table A3.

GeoTIFF Export

Export the rasterized features from the target grid to GeoTIFF files.

```
with open('epsgs.txt', 'r') as file:
    exec(file.read())

current_epsg = epsg_dict[cc]
print(current_epsg)

# setup input dictionary to configure the geotiff export pipeline
geotiff_export_input_nonground = {
    'parse_point_cloud': {},
    'data_split': {'xSub': 1, 'ySub': 1},
    'create_subregion_geotiffs': {'output_handle': output_handle, 'EPSG': current_epsg}, #important flag
}

# write input dictionary to JSON file
with open(filename, 'w') as f:
    json.dump(geotiff_export_input_nonground, f)
```

Figure 10: Code for rasterizing the extracted features in the Laserfarm Jupyter Notebook. This requires providing the EPSG code for a specific coordinate reference system via a text file ('epsgs.txt').

References

Høye, T.T., August, T., Balzan, M.V., Biesmeijer, K., Bonnet, P., Breeze, T.D., Dominik, C., Gerard, F., Joly, A., Kalkman, V., Kissling, W.D., Metodiev, T., Moeslund, J., Potts, S., Roy, D.B., Schweiger, O., Senapathi, D., Settele, J., Stoev, P., & Stowell, D. (2023). Modern Approaches to the Monitoring of Biodiversity (MAMBO). *Research Ideas and Outcomes, 9*, e116951.

Kissling, W.D., Shi, Y., Koma, Z., Meijer, C., Ku, O., Nattino, F., Seijmonsbergen, A.C., & Grootes, M.W. (2022). Laserfarm – A high-throughput workflow for generating geospatial data products of ecosystem structure from airborne laser scanning point clouds. *Ecological Informatics, 72*, 101836.

Kissling, W.D., Shi, Y., Koma, Z., Meijer, C., Ku, O., Nattino, F., Seijmonsbergen, A.C., & Grootes, M.W. (2023). Country-wide data of ecosystem structure from the third Dutch airborne laser scanning survey. *Data in Brief, 46*, 108798.

Kissling, W.D., Shi, Y., Wang, J., Walicka, A., George, C., Moeslund, J.E., & Gerard, F. (2024). Towards consistently measuring and monitoring habitat condition with airborne laser scanning and unmanned aerial vehicles. *Ecological Indicators, 169*, 112970.

Meijer, C., Grootes, M.W., Koma, Z., Dzigan, Y., Gonçalves, R., Andela, B., van den Oord, G., Ranguelova, E., Renaud, N., & Kissling, W.D. (2020). Laserchicken—A tool for distributed feature calculation from massive LiDAR point cloud datasets. *SoftwareX, 12*, 100626.

Rusu, R.B., Marton, Z.C., Blodow, N., Dolha, M., & Beetz, M. (2008). Towards 3D Point cloud based object maps for household environments. *Robotics and Autonomous Systems, 56*, 927-941.

Appendix Table A1: Characteristics of LiDAR point clouds from seven MAMBO demonstration sites. Raw LiDAR point clouds were accessed from national airborne laser scanning datasets in five European countries and clipped to the boundaries of the demonstration sites using polygon shapefiles of the study areas.

	Mols Bjerge National Park	**Reserve Naturelle Nationale du Bagnas**	**Oostvaarders-plassen**	**Salisbury Plain**	**Knepp Estate**	**Monks Wood**	**Comino**
Country	Denmark	France	Netherlands	United Kingdom	United Kingdom	United Kingdom	Malta
Site abbreviation	MolsBjerge	Bagnas	Oostvaardersplassen	SalisburyPlain	Knepp	MonksWood	Comino
Approximate location (x,y)	10.478393 E 56.289050 N	3.514360 E 43.314332 N	5.418842 E 52.456870 N	1.866189 W 51.220814 N	0.3758 W 50.969859 N	0.2386 W 52.400 N	36.0113 E 14.3362 N
Link to public raw data of LiDAR point clouds	https://dataforsyningen.dk/data/3931	https://geoservices.ign.fr/lidarhd#telechargement	https://www.arcgis.com/home/webscene/viewer.html?layers=77da2e9eeea8427aab2ac83b79097b1a	https://environment.data.gov.uk/DefraDataDownload/?Mode=survey	https://environment.data.gov.uk/DefraDataDownload/?Mode=survey	https://environment.data.gov.uk/DefraDataDownload/?Mode=survey	NA
Time of acquisition	2014–2015	2021–2022	2020 (AHN4)	2021	2019	2017, 2019	2018
Approximate areal coverage	1.29 km^2	7.5 km^2	54 km^2	7.95 km^2	5.55 km^2	0.08 km^2	3.5 km^2
Indices of tiles	PUNKTSKY_623_59_TIF_UTM32-ETRS89 (6231_598, 6232_598, 6232_597, 6232_596, 6233_597, 6233_598)	LIDARHD_1-0_LAZ_MQ-0742_6247-2021, LIDARHD_1-0_LAZ_MQ-0740_6247-2021, LIDARHD_1-0_LAZ_MQ-0744_6247-2021, LIDARHD_1-0_LAZ_MQ-0742_6245-2021, LIDARHD_1-0_LAZ_MQ-0740_6245-2021, LIDARHD_1-0_LAZ_MQ-0740_6249-2021	26AN2, 26AZ2, 26BN1, 26BZ1, 20DZ2, 26BN2, 26BZ2	SU0040_P_11780	TQ1020_P_10749	TL2075_P_10756	438_3985, 438_3986, 439_3984, 439_3985, 439_3986, 440_3984, 440_3985, 440_3986, 441_3985, 441_3986

	Mols Bjerge National Park	**Reserve Naturelle Nationale du Bagnas**	**Oostvaarders-plassen**	**Salisbury plain**	**Knepp estate**	**Monks Wood**	**Comino**
Link to public location of clipped LiDAR point clouds	https://public.spider.surfsara.nl/project/lidarac/MAMBO/Data/DK/	https://public.spider.surfsara.nl/project/lidarac/MAMBO/Data/FR/	https://public.spider.surfsara.nl/project/lidarac/MAMBO/Data/NL/	https://public.spider.surfsara.nl/project/lidarac/MAMBO/Data/UK/	https://public.spider.surfsara.nl/project/lidarac/MAMBO/Data/UK/	https://public.spider.surfsara.nl/project/lidarac/MAMBO/Data/UK/	NA
Approximate average point density	10 pts/m^2	10 pts/m^2	15 pts/m^2	7 pts/m^2	7 pts/m^2	7 pts/m^2	5 pts/m^2 (downsampled)
Local coordinate system	GEOGCS [Projection-""ERTS89"", Height-""DVR90""]	GEOGCS [Projection-""WGS84"", Height - ""RGF93""]	GEOGCS [Projection-""WGS84"", Height - ""RD_New""]	GEOGCS [Projection-""WGS84""]	GEOGCS [Projection-""WGS84""]	GEOGCS [Projection-""WGS84""]	GEOGCS [Projection-""WGS84""]
Point data record format	6	1	3	3	3	3	3
Classification available?	yes	yes	yes	yes	yes	yes	Yes
ASPRS standard point classes in study area	ground (2); low vegetation (3); medium vegetation (4); high vegetation (5); building (6); low point - noise (7)	unclassified (1); ground (2); low vegetation (3); medium vegetation (4); high vegetation (5); water (9); powerline (64)	unclassified (1); terrain (2); buildings (6); water (9)	unclassified (1); ground (2); low vegetation (3); medium vegetation (4); high vegetation (5); buildings (6); low points - noise (7)	unclassified (1); ground (2); low vegetation (3); medium vegetation (4); high vegetation (5); buildings (6); low points - noise (7)	unclassified (1); ground (2); low vegetation (3); medium vegetation (4); high vegetation (5); buildings (6)	unclassified (1); ground (2); low vegetation (3); medium vegetation (4); high vegetation (5); buildings (6); water (9)
Number of returns	5	5	6	4	5	5	5
Other point cloud properties	Intensity, return number, scan angle rank, edge of flight lines, point source ID, GPS time	Intensity, return number, scan angle rank, edge of flight lines, point source ID, GPS time	Intensity, return number, scan angle rank, edge of flight lines, point source ID, GPS time	Intensity, return number, scan angle rank, edge of flight lines, point source ID, GPS time	Intensity, return number, scan angle rank, edge of flight lines, point source ID, GPS time	Intensity, return number, scan angle rank, edge of flight lines, point source ID, GPS time	Intensity, return number, scan angle rank, edge of flight lines, point source ID, GPS time

Appendix Table A2: An example of execution steps for setting up a Laserfarm Jupyter Notebook in a high-performance computing (HPC) environment. The example shows the details of command-line interface and terminal executions that are necessary to set up the Jupyter Notebook in the high-performance data processing platform 'Spider' of the Dutch IT infrastructure SURF (https://www.surf.nl/en/services/high-performance-data-processing). Spider is an in house compute cluster of SURF and allows users to run highly parallel jobs on distributed resources, using scalable processing of many terabytes of data and utilizing many hundreds of cores simultaneously (https://servicedesk.surf.nl/wiki/display/WIKI/Spider). For other HPC environments, the code for execution steps might look different, and it is best to consult the documentation or helpdesk of a specific HPC environment.

Execution step	Command-line interface and terminal executions
Login to your HPC environment	*ssh <username>@hpc-environment.nl*
Make a directory for all files of interest	*mkdir PointClouds*
Build the Jupyter-Dask environment	*git clone http://github.com/RS-DAT/JupyterDaskOnSLURM.git*
Go to the directory	*cd PointClouds/JupyterDaskOnSLURM*
Create a new environment	*conda env create -f environment.yaml*
Activate the Jupyter-Dask environment	*conda activate jupyter_dask*
Install Laserfarm	*conda install pdal python-pdal gdal -c conda-forge; pip install Laserfarm*
Check and edit configuration files	*cat environment.yaml; vim config_spider.yml*
Copy the configuration file to home directory	*cp -r config/dask/config_spider.yml ~/.config/dask/*
Go back to workspace	*cd /PointClouds/JupyterDaskOnSLURM*
Submit a job to start Jupyter server	*sbatch scripts/jupyter_dask_spider.bsh*
Check the job	*squeue -u <username>*
Copy the ssh command printed in the slurm-<JOB_ID>.out and paste it into a new terminal on your local machine.	*ssh -i /path/to/private/ssh/key -N -L 8889:NODE:8888 <user>@hpc-environment.nl*
Access Jupyter session from your browser at localhost:8889	
Copy Laserfarm Jupyter Notebooks to your own workspace (an example can be downloaded from: https://github.com/eEcoLiDAR/AHN/tree/main/AHN3)	-

Appendix Table A3: User-defined parameter values for the Jupyter Notebooks implementing the Laserfarm workflow for the seven MAMBO demonstration sites.

	Mols Bjerge National Park	**Reserve Naturelle Nationale du Bagnas**	**Oostvaarders-plassen**	**Salisbury plain**	**Knepp estate**	**Monks Wood**	**Comino**
Re-tiling							
min_x	596605.00	739243.00	144232.50	401539.00	512626.00	519397.00	438423.00
max_x	598945.00	744243.00	159232.50	403689.00	515626.00	520497.00	441823.00
min_y	6231377.00	6244168.00	488499.00	146344.00	118720.00	279015.00	3983780.00
max_y	6233717.00	6249168.00	503499.00	148494.00	121720.00	280115.00	3987180.00
n_tiles_side	2	20	15	1	3	1	1
Normalization							
normalize	1	1	1	1	1	1	1
Feature extraction							
features	See Appendix Table A4	See Appendix Table A4	See Appendix Table A4	See Appendix Table A4	See Appendix Table A4	See Appendix Table A4	See Appendix Table A4
apply_filter 'value' of ASPRS standard point classes for vegetation points	3,4,5	3,4,5	1	3,4,5	3,4,5	3,4,5	3,4,5
Rasterization							
current_epsg	25832	2154	28992	27700	27700	27700	32633

Appendix Table A4: LiDAR metrics ('features') extracted with the Laserfarm workflow for the MAMBO demonstration sites.

Laserfarm feature	LiDAR metric name	Description
Vegetation height		
perc_25_normalized_height	25th percentile of vegetation height	25th percentile of normalized z within a 10 m × 10 m grid cell
perc_50_normalized_height	50th percentile of vegetation height	50th percentile of normalized z within a 10 m × 10 m grid cell
perc_75_normalized_height	75th percentile of vegetation height	75th percentile of normalized z within a 10 m × 10 m grid cell
perc_95_normalized_height	95th percentile of vegetation height	95th percentile of normalized z within a 10 m × 10 m grid cell
max_normalized_height	Maximum vegetation height	Maximum of normalized z within a 10 m × 10 m grid cell
median_normalized_height	Median of vegetation height	Median of normalized z within a 10 m × 10 m grid cell
mean_normalized_height	Mean of vegetation height	Mean of normalized z within a grid cell
Vegetation cover		
pulse_penetration_ratio	Pulse penetration ratio	Ratio of number of ground points (N_{ground}) to the total number of points (N_{total}) within a 10 m × 10 m grid cell
density_absolute_mean_normalized_height	Density of upper vegetation layer	Number of returns above mean height within a 10 m × 10 m grid cell
band_ratio_normalized_height_1	Density of vegetation points below 1 m	Ratio of number of vegetation points (< 1 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_1_normalized_height_2	Density of vegetation points between 1–2 m	Ratio of number of vegetation points (between 1–2 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_2_normalized_height_3	Density of vegetation points between 2–3 m	Ratio of number of vegetation points (between 2–3 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_normalized_height_3	Density of vegetation points below 3 m	Ratio of number of vegetation points (< 3 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_3_normalized_height	Density of vegetation points above 3 m	Ratio of number of vegetation points (> 3 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_3_normalized_height_4	Density of vegetation points between 3–4 m	Ratio of number of vegetation points (between 3–4 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_4_normalized_height_5	Density of vegetation points between 4–5 m	Ratio of number of vegetation points (between 4–5 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_normalized_height_5	Density of vegetation points below 5 m	Ratio of number of vegetation points (< 5 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_5_normalized_height_6	Density of vegetation points between 5–6 m	Ratio of number of vegetation points (between 5–6 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_5_normalized_height_20	Density of vegetation points between 5–20 m	Ratio of number of vegetation points (between 5–20 m) to the total number of vegetation points within a 10 m × 10 m grid cell
band_ratio_6_normalized_height_10	Density of vegetation points between 6–10 m	Ratio of number of vegetation points (between 6–10 m) to the total number of vegetation points within a 10 m × 10 m grid cell

Laserfarm feature	LiDAR metric name	Description
band_ratio_20_normalized_height	Density of vegetation points above 20 m	Ratio of number of vegetation points (> 20 m) to the total number of vegetation points within a 10 m × 10 m grid cell
Vegetation vertical variability		
coeff_var_normalized_height	Coefficient of variation of vegetation height	Coefficient of variation of normalized z within a 10 m × 10 m grid cell
entropy_normalized_height	Shannon index	The negative sum of the proportion of points within 0.5 m height layers multiplied with the logarithm of the proportion of points within 0.5 m height layers within a 10 m × 10 m grid cell
kurto_normalized_height	Kurtosis of vegetation height	Kurtosis of normalized z within a 10 m × 10 m grid cell
sigma_z	Roughness of vegetation	Standard deviation of the residuals of a fitted plane within a 10 m × 10 m grid cell
skew_normalized_height	Skewness of vegetation height	Skewness of normalized z within a 10 m × 10 m grid cell
std_normalized_height	Standard deviation of vegetation height	Standard deviation of normalized z within a 10 m × 10 m grid cell
var_normalized_height	Variance of vegetation height	Variance of normalized z within a 10 m × 10 m grid cell
Other features		
point_density	Point density	Point density within a 10 m × 10 m grid cell
eigenv_1	First eigenvalue	First principal direction of variation within neighbourhoods around target points
eigenv_2	Second eigenvalue	Second principal direction of variation within neighbourhoods around target points
eigenv_3	Third eigenvalue	Third principal direction of variation within neighbourhoods around target points
normal_vector_1	First normal vector	First principal direction of the surface calculated from eigenvectors in three-dimensional space
normal_vector_2	Second normal vector	Second principal direction of the surface calculated from eigenvectors in three-dimensional space
normal_vector_3	Third normal vector	Third principal direction of the surface calculated from eigenvectors in three-dimensional space