In [1]:
```
# !pip install geopandas rasterio shapely scipy scikit-learn scikit-image seaborn m
```

In [2]:
```
import sys
import os

print("Python executable:", sys.executable)
print("Python version:", sys.version)
print("Environment location:", os.path.dirname(sys.executable))
```

```
Python executable: C:\Users\colto\Documents\GitHub\saocom_project\.venv\Scripts\pyth
on.exe
Python version: 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64
bit (AMD64)]
Environment location: C:\Users\colto\Documents\GitHub\saocom_project\.venv\Scripts
```

Setup

In [3]:
```
from unittest.mock import sentinel

# ============================================================================
# IMPORTS
# ============================================================================
# Core libraries
import numpy as np
import pandas as pd
from pathlib import Path
import warnings
warnings.filterwarnings('ignore')
from pyproj import Transformer
# Geospatial libraries
import geopandas as gpd
import rasterio
from rasterio.warp import reproject, Resampling, calculate_default_transform
from rasterio.transform import from_bounds, rowcol
from rasterio.mask import mask
from rasterio import features
from shapely.geometry import Point, box, shape  # Add 'shape' here
from rasterio.features import shapes  # Add this new line
# Analysis libraries
from scipy.interpolate import griddata
from scipy.ndimage import median_filter, binary_opening, binary_closing
from scipy import stats
from sklearn.neighbors import NearestNeighbors, KernelDensity
from skimage.morphology import disk
from shapely.geometry import shape
from rasterio.features import shapes
import matplotlib.patches as mpatches
# Visualization libraries
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.patches import FancyArrowPatch
from matplotlib.colors import ListedColormap, BoundaryNorm
from matplotlib_scalebar.scalebar import ScaleBar
import seaborn as sns
```

```
# Data I/O
from dbfread import DBF

# ============================================================================
# PATHS AND DIRECTORIES
# ============================================================================
DATA_DIR = Path("data")
RESULTS_DIR = Path("results")
RESULTS_DIR.mkdir(exist_ok=True)
print(DATA_DIR)
# ============================================================================
# COORDINATE REFERENCE SYSTEM
# ============================================================================
TARGET_CRS = 'EPSG:32632'  # UTM 32N

# ============================================================================
# PROCESSING PARAMETERS
# ============================================================================
COHERENCE_THRESHOLD = 0.3
NODATA = -9999
GRID_SIZE = 10  # meters

# ============================================================================
# CORINE LAND COVER CLASSES
# ============================================================================
CORINE_CLASSES = {
    111: 'Continuous urban fabric', 112: 'Discontinuous urban fabric',
    121: 'Industrial or commercial units', 122: 'Road and rail networks and associa
    123: 'Port areas', 124: 'Airports', 131: 'Mineral extraction sites',
    132: 'Dump sites', 133: 'Construction sites', 141: 'Green urban areas',
    142: 'Sport and leisure facilities', 211: 'Non-irrigated arable land',
    212: 'Permanently irrigated land', 213: 'Rice fields', 221: 'Vineyards',
    222: 'Fruit trees and berry plantations', 223: 'Olive groves',
    231: 'Pastures', 241: 'Annual crops associated with permanent crops',
    242: 'Complex cultivation patterns', 243: 'Agriculture/natural vegetation mix',
    244: 'Agro-forestry areas', 311: 'Broad-leaved forest',
    312: 'Coniferous forest', 313: 'Mixed forest', 321: 'Natural grasslands',
    322: 'Moors and heathland', 323: 'Sclerophyllous vegetation',
    324: 'Transitional woodland-shrub', 331: 'Beaches, dunes, sands',
    332: 'Bare rocks', 333: 'Sparsely vegetated areas', 334: 'Burnt areas',
    335: 'Glaciers and perpetual snow', 411: 'Inland marshes',
    412: 'Peat bogs', 421: 'Salt marshes', 422: 'Salines',
    423: 'Intertidal flats', 511: 'Water courses', 512: 'Water bodies',
    521: 'Coastal lagoons', 522: 'Estuaries', 523: 'Sea and ocean'
}

# CORINE_COLORS = {
#     111: (230, 0, 77), 112: (255, 0, 0), 121: (204, 77, 242), 122: (204, 0, 0),
#     123: (230, 204, 204), 124: (230, 204, 230), 131: (166, 0, 204), 132: (166, 77
#     133: (255, 77, 255), 141: (255, 166, 255), 142: (255, 230, 255),
#     211: (255, 255, 168), 212: (255, 255, 0), 213: (230, 230, 0), 221: (230, 128,
#     222: (242, 166, 77), 223: (230, 166, 0), 231: (230, 230, 77), 241: (255, 230,
#     242: (255, 230, 77), 243: (230, 204, 77), 244: (242, 204, 166),
#     311: (128, 255, 0), 312: (0, 166, 0), 313: (77, 255, 0), 321: (204, 242, 77),
#     322: (166, 255, 128), # Note: Corrected color for 322 (Moors and heathland) a
```

```
#     323: (166, 230, 77),
#     324: (166, 242, 0), # Note: Corrected color for 324 (Transitional woodland-sh
#     331: (230, 230, 230), # Note: Corrected color for 331 (Beaches, dunes, sands)
#     332: (204, 204, 204), 333: (204, 255, 204), 334: (0, 0, 0), 335: (166, 230, 2
#     411: (166, 166, 255), 412: (77, 77, 255), 421: (204, 204, 255), 422: (230, 23
#     423: (166, 166, 230), 511: (0, 204, 242), 512: (128, 242, 230), 521: (0, 255,
#     522: (166, 255, 230), 523: (230, 242, 255) # Note: Corrected color for 523 (S
# }
#
# # Convert RGB (0-255) to matplotlib format (0-1)
# CORINE_COLORS_MPL = {k: (r/255, g/255, b/255) for k, (r, g, b) in CORINE_COLORS.i
# Colorblind-friendly version (paste this AFTER the original CORINE_COLORS)
CORINE_COLORS_COLORBLIND = {
    # ARTIFICIAL SURFACES (1xx) - Purples/Magentas/Grays
    111: (102, 0, 102),      # Dark purple - Continuous urban
    112: (153, 51, 153),     # Medium purple - Discontinuous urban
    121: (204, 102, 204),    # Light purple - Industrial
    122: (80, 80, 80),       # Dark gray - Roads/rail
    123: (120, 120, 120),    # Medium gray - Ports
    124: (160, 160, 160),    # Light gray - Airports
    131: (255, 0, 255),      # Bright magenta - Mineral extraction
    132: (178, 34, 34),      # Dark red-brown - Dump sites
    133: (255, 150, 180),    # Darker pink - Construction (was too light)
    141: (120, 200, 120),    # Medium green - Green urban areas (darkened)
    142: (100, 180, 100),    # Green - Sport/leisure (darkened)

    # AGRICULTURAL (2xx) - Yellows/Oranges/Browns
    211: (230, 230, 50),     # Strong yellow - Non-irrigated arable (darkened)
    212: (235, 200, 0),      # Gold yellow - Permanently irrigated (darkened)
    213: (220, 180, 0),      # Dark gold - Rice fields
    221: (255, 140, 0),      # Dark orange - Vineyards
    222: (255, 165, 79),     # Orange - Fruit trees
    223: (204, 153, 0),      # Olive-brown - Olive groves
    231: (210, 210, 80),     # Medium yellow - Pastures (MUCH darker)
    241: (200, 170, 100),    # Tan - Annual crops w/ permanent (darkened)
    242: (210, 160, 70),     # Brown - Complex cultivation (darkened)
    243: (190, 150, 80),     # Medium brown - Agriculture w/ natural
    244: (179, 143, 0),      # Dark yellow-brown - Agro-forestry

    # FORESTS & SEMI-NATURAL (3xx) - Teals/Cyans/Dark colors
    311: (0, 153, 102),      # Dark teal - Broad-leaved forest
    312: (0, 102, 76),       # Very dark teal - Coniferous forest
    313: (0, 128, 128),      # Medium teal - Mixed forest
    321: (150, 220, 150),    # Light green - Natural grasslands (darkened)
    322: (102, 204, 153),    # Mint - Moors and heathland
    323: (130, 180, 130),    # Sage - Sclerophyllous vegetation (darkened)
    324: (51, 153, 102),     # Medium green - Transitional woodland
    331: (210, 180, 140),    # Tan - Beaches/dunes/sands (MUCH darker)
    332: (140, 140, 140),    # Gray - Bare rocks (darkened)
    333: (170, 170, 120),    # Khaki - Sparsely vegetated (darkened)
    334: (40, 40, 40),       # Near black - Burnt areas
    335: (180, 210, 230),    # Light blue - Glaciers/snow (darkened)

    # WETLANDS (4xx) - Light blues/cyans
    411: (120, 170, 230),    # Sky blue - Inland marshes (darkened)
    412: (80, 140, 220),     # Medium blue - Peat bogs (darkened)
```

```
    421: (150, 190, 240),    # Light blue - Salt marshes (darkened)
    422: (140, 170, 210),    # Powder blue - Salines (darkened)
    423: (100, 160, 210),    # Cyan - Intertidal flats (darkened)

    # WATER BODIES (5xx) - Dark blues
    511: (0, 102, 204),      # Dark blue - Water courses
    512: (0, 76, 153),       # Very dark blue - Water bodies
    521: (51, 102, 153),     # Medium dark blue - Coastal lagoons
    522: (0, 51, 102),       # Navy - Estuaries
    523: (0, 25, 76)         # Very dark navy - Sea and ocean
}

# Use the colorblind-friendly version
CORINE_COLORS = CORINE_COLORS_COLORBLIND
CORINE_COLORS_MPL = {k: (r/255, g/255, b/255) for k, (r, g, b) in CORINE_COLORS.ite
# ============================================================================
# FILE DISCOVERY
# ============================================================================
# Automatically locate data files in subdirectories
saocom_files = list((DATA_DIR / "saocom_csv").glob("*.csv"))
tinitaly_files = list((DATA_DIR / "tinitaly").glob("*.tif"))
copernicus_files = list((DATA_DIR / "copernicus").glob("*.tif"))
corine_files = list((DATA_DIR / "ground_cover").glob("*.tif"))
sentinel_files = list((DATA_DIR / "sentinel_data").glob("*.tif"))
print(corine_files, saocom_files, tinitaly_files, copernicus_files, sentinel_files)
# Select first match for each dataset
saocom_path = saocom_files[0] if saocom_files else None
tinitaly_path = tinitaly_files[0] if tinitaly_files else None
copernicus_path = copernicus_files[0] if copernicus_files else None
corine_path = corine_files[0] if corine_files else None
sentinel_path = sentinel_files[0] if sentinel_files else None

# Find the corresponding .vat.dbf file for the CORINE raster
corine_dbf_path = None
if corine_path:
    corine_dbf_candidates = list((DATA_DIR / "ground_cover").glob(f"{corine_path.na
    corine_dbf_path = corine_dbf_candidates[0] if corine_dbf_candidates else None
```

```
data
[WindowsPath('data/ground_cover/land_cover_clipped.tif')] [WindowsPath('data/saocom_
csv/verona_fullGraph_weighted_Tcoh07_edited.csv')] [WindowsPath('data/tinitaly/tinit
aly_crop.tif')] [WindowsPath('data/copernicus/GLO30.tif')] [WindowsPath('data/sentin
el_data/Sentinel2Views_Clip.tif')]
```

Load Data

In [4]:

```
# ============================================================================
# LOAD SAOCOM POINT DATA
# ============================================================================
# Read CSV and standardize columns
df = pd.read_csv(saocom_path, sep=',')
df.columns = ['ID', 'SVET', 'LVET', 'LAT', 'LAT2', 'LON', 'LON2', 'HEIGHT', 'HEIGHT

# Convert to numeric and remove invalid points
for col in ['LAT', 'LON','LAT2', 'LON2',  'HEIGHT', 'COHER']:
```

```
    df[col] = pd.to_numeric(df[col], errors='coerce')
df = df.dropna(subset=['LAT', 'LON','LAT2', 'LON2', 'HEIGHT', 'COHER'])
df = df[(df['LAT2'] != 0) & (df['LON2'] != 0)]
df.rename(columns={
    'LAT': 'LAT_old',
    'LON': 'LON_old',
    'LAT2': 'LAT',
    'LON2': 'LON'
}, inplace=True)
# Apply coherence filter
df_filtered = df[df['COHER'] >= COHERENCE_THRESHOLD]

# Convert to GeoDataFrame and reproject to target CRS
geometry = [Point(lon, lat) for lon, lat in zip(df_filtered['LON'], df_filtered['LA
saocom_gdf = gpd.GeoDataFrame(df_filtered, geometry=geometry, crs='EPSG:4326')
saocom_gdf = saocom_gdf.to_crs(TARGET_CRS)
saocom_gdf['x_utm'] = saocom_gdf.geometry.x
saocom_gdf['y_utm'] = saocom_gdf.geometry.y

# ============================================================================
# LOAD REFERENCE DEMS
# ============================================================================
# TINITALY
with rasterio.open(tinitaly_path) as src:
    tinitaly_crs = src.crs
    tinitaly_res = src.res
    tinitaly_bounds = src.bounds
    tinitaly_nodata = src.nodata

# Copernicus
with rasterio.open(copernicus_path) as src:
    copernicus_crs = src.crs
    copernicus_res = src.res
    copernicus_bounds = src.bounds
    copernicus_nodata = src.nodata
# # ============================================================================
# # LOAD AND REMAP CORINE LAND COVER
# # ============================================================================
# # Load DBF lookup table
# dbf_table = DBF(corine_dbf_path, load=True)
# lookup_df = pd.DataFrame(iter(dbf_table))
#
# # Create mapping dictionaries
# value_to_code = dict(zip(lookup_df['Value'], lookup_df['CODE_18']))
# value_to_label = dict(zip(lookup_df['Value'], lookup_df['LABEL3']))
# code_to_label = dict(zip(lookup_df['CODE_18'], lookup_df['LABEL3']))
#
# # Load original CORINE raster
# with rasterio.open(corine_path) as src:
#     corine_raw = src.read(1)
#     corine_crs = src.crs
#     corine_res = src.res
#     corine_bounds = src.bounds
#     corine_nodata = src.nodata if src.nodata is not None else 255
#     corine_transform = src.transform
#     corine_profile = src.profile
```

```
#
# # Remap raster values from Value to CODE_18
# corine_remapped = np.full_like(corine_raw, 0, dtype=np.uint16)
# for value, code in value_to_code.items():
#     corine_remapped[corine_raw == value] = code
# corine_remapped[corine_raw == corine_nodata] = 0  # NoData = 0
#
# # Save remapped CORINE to temporary file
# corine_remapped_path = RESULTS_DIR / "corine_remapped.tif"
# profile_remapped = corine_profile.copy()
# profile_remapped.update(dtype='uint16', nodata=0)
# with rasterio.open(corine_remapped_path, 'w', **profile_remapped) as dst:
#     dst.write(corine_remapped, 1)
#
# # Update corine_path to use remapped version
# corine_path = corine_remapped_path
```

HORIZONTAL DATUM VERIFICATION

In [5]:

```
from sklearn.neighbors import NearestNeighbors

def remove_isolated_knn(gdf, k=5, distance_threshold=100):
    """Remove points far from k nearest neighbors."""
    coords = np.array([[p.x, p.y] for p in gdf.geometry])

    nbrs = NearestNeighbors(n_neighbors=k+1).fit(coords)
    distances, _ = nbrs.kneighbors(coords)

    # Average distance to k nearest neighbors (exclude self at index 0)
    avg_distances = distances[:, 1:].mean(axis=1)

    mask = avg_distances < distance_threshold
    return gdf[mask].reset_index(drop=True)


# ============================================================================
# HORIZONTAL DATUM VERIFICATION
# ============================================================================
# Check if datasets need reprojection to target CRS
tinitaly_needs_reproject = str(tinitaly_crs) != TARGET_CRS
copernicus_needs_reproject = str(copernicus_crs) != TARGET_CRS
# corine_needs_reproject = str(corine_crs) != TARGET_CRS

# ============================================================================
# VERTICAL DATUM VERIFICATION
# ============================================================================
# Extract vertical datum information from CRS WKT
tinitaly_wkt = tinitaly_crs.to_wkt()
copernicus_wkt = copernicus_crs.to_wkt()

# Check for vertical datum identifiers
tinitaly_vertical = 'EGM2008' in tinitaly_wkt or 'geoid' in tinitaly_wkt.lower()
copernicus_vertical = 'EGM2008' in copernicus_wkt or 'geoid' in copernicus_wkt.lowe

# Copernicus GLO-30 uses EGM2008 geoid (documented)
```

```
# TINITALY typically uses WGS84 ellipsoid (documented)

# ============================================================================
# CREATE STUDY AREA BOUNDS AND CONVEX HULL
# ============================================================================
study_bounds = saocom_gdf.total_bounds  # [xmin, ymin, xmax, ymax]
study_area_poly = box(*study_bounds)
study_area_gdf = gpd.GeoDataFrame([1], geometry=[study_area_poly], crs=TARGET_CRS)
saocom_gdf = remove_isolated_knn(saocom_gdf, k=5, distance_threshold=100)
# Create convex hull from SAOCOM points for masking
data_hull = saocom_gdf.unary_union.convex_hull
hull_gdf = gpd.GeoDataFrame(geometry=[data_hull], crs=TARGET_CRS)

# ============================================================================
# DEFINE 10M GRID PARAMETERS
# ============================================================================
xmin_grid = np.floor(study_bounds[0] / GRID_SIZE) * GRID_SIZE
ymin_grid = np.floor(study_bounds[1] / GRID_SIZE) * GRID_SIZE
xmax_grid = np.ceil(study_bounds[2] / GRID_SIZE) * GRID_SIZE
ymax_grid = np.ceil(study_bounds[3] / GRID_SIZE) * GRID_SIZE

grid_width = int((xmax_grid - xmin_grid) / GRID_SIZE)
grid_height = int((ymax_grid - ymin_grid) / GRID_SIZE)
target_transform = from_bounds(xmin_grid, ymin_grid, xmax_grid, ymax_grid, grid_wid

# ============================================================================
# STORE REFERENCE DATASET METADATA
# ============================================================================
reference_dems = {
    'tinitaly_crop': {
        'path': tinitaly_path,
        'crs': tinitaly_crs,
        'needs_reproject': tinitaly_needs_reproject,
        'vertical_datum': 'WGS84 ellipsoid'
    },
    'copernicus': {
        'path': copernicus_path,
        'crs': copernicus_crs,
        'needs_reproject': copernicus_needs_reproject,
        'vertical_datum': 'EGM2008 geoid'
    }
}
```

RESAMPLE TO 10M

In [6]:

```
# ============================================================================
# RESAMPLE TINITALY TO 10M
# ============================================================================
tinitaly_10m = np.full((grid_height, grid_width), NODATA, dtype=np.float32)

with rasterio.open(tinitaly_path) as src:
    reproject(
        source=rasterio.band(src, 1),
        destination=tinitaly_10m,
        src_transform=src.transform,
```

```
        src_crs=src.crs,
        dst_transform=target_transform,
        dst_crs=TARGET_CRS,
        resampling=Resampling.cubic,
        src_nodata=src.nodata,
        dst_nodata=NODATA
    )

# Save resampled TINITALY
tinitaly_10m_path = RESULTS_DIR / "tinitaly_10m.tif"
profile = {
    'driver': 'GTiff', 'dtype': 'float32', 'width': grid_width, 'height': grid_heig
    'count': 1, 'crs': TARGET_CRS, 'transform': target_transform,
    'nodata': NODATA, 'compress': 'lzw'
}
with rasterio.open(tinitaly_10m_path, 'w', **profile) as dst:
    dst.write(tinitaly_10m, 1)

# ============================================================================
# RESAMPLE COPERNICUS TO 10M
# ============================================================================
copernicus_10m = np.full((grid_height, grid_width), NODATA, dtype=np.float32)

with rasterio.open(copernicus_path) as src:
    reproject(
        source=rasterio.band(src, 1),
        destination=copernicus_10m,
        src_transform=src.transform,
        src_crs=src.crs,
        dst_transform=target_transform,
        dst_crs=TARGET_CRS,
        resampling=Resampling.cubic,
        src_nodata=src.nodata,
        dst_nodata=NODATA
    )

# Save resampled Copernicus
copernicus_10m_path = RESULTS_DIR / "copernicus_10m.tif"
with rasterio.open(copernicus_10m_path, 'w', **profile) as dst:
    dst.write(copernicus_10m, 1)


# ============================================================================
# UPDATE REFERENCE DATASET PATHS
# ============================================================================
reference_dems['tinitaly_crop']['resampled_path'] = tinitaly_10m_path
reference_dems['tinitaly_crop']['is_10m'] = True
reference_dems['copernicus']['resampled_path'] = copernicus_10m_path
reference_dems['copernicus']['is_10m'] = True
```

CREATE RASTERIZED MASK FROM SAOCOM CONVEX HULL

In [7]:

```
# ============================================================================
# CREATE RASTERIZED MASK FROM SAOCOM CONVEX HULL
# ============================================================================
```

```
# Rasterize the convex hull polygon to match the 10m grid
hull_mask = features.rasterize(
    shapes=[data_hull],
    out_shape=(grid_height, grid_width),
    transform=target_transform,
    fill=0,
    all_touched=True,
    dtype=np.uint8
) == 1  # Convert to boolean (True inside hull)

# ============================================================================
# MASK TINITALY
# ============================================================================
tinitaly_10m_masked = tinitaly_10m.copy()
tinitaly_10m_masked[~hull_mask] = NODATA

# Save masked TINITALY
tinitaly_masked_path = RESULTS_DIR / "tinitaly_10m_masked.tif"
with rasterio.open(tinitaly_masked_path, 'w', **profile) as dst:
    dst.write(tinitaly_10m_masked, 1)

# ============================================================================
# MASK COPERNICUS
# ============================================================================
copernicus_10m_masked = copernicus_10m.copy()
copernicus_10m_masked[~hull_mask] = NODATA

# Save masked Copernicus
copernicus_masked_path = RESULTS_DIR / "copernicus_10m_masked.tif"
with rasterio.open(copernicus_masked_path, 'w', **profile) as dst:
    dst.write(copernicus_10m_masked, 1)

# ============================================================================
# UPDATE REFERENCE DATASET PATHS TO MASKED VERSIONS
# ============================================================================
reference_dems['tinitaly_crop']['masked_path'] = tinitaly_masked_path
reference_dems['copernicus']['masked_path'] = copernicus_masked_path

# Store masked arrays in memory for quick access
tinitaly_10m = tinitaly_10m_masked
copernicus_10m = copernicus_10m_masked
# corine_10m = corine_10m_masked
```

SAMPLE REFERENCE DEMS AT SAOCOM LOCATIONS

In [8]:

```
# ============================================================================
# SAMPLE REFERENCE DEMS AT SAOCOM LOCATIONS
# ============================================================================
# Sample TINITALY at each SAOCOM point
tinitaly_heights = []
for idx, row in saocom_gdf.iterrows():
    row_idx, col_idx = rowcol(target_transform, row.geometry.x, row.geometry.y)
    if 0 <= row_idx < grid_height and 0 <= col_idx < grid_width:
        height = tinitaly_10m[row_idx, col_idx]
        tinitaly_heights.append(height if height != NODATA else np.nan)
```

```
    else:
        tinitaly_heights.append(np.nan)

saocom_gdf['tinitaly_height'] = tinitaly_heights

# Sample Copernicus at each SAOCOM point
copernicus_heights = []
for idx, row in saocom_gdf.iterrows():
    row_idx, col_idx = rowcol(target_transform, row.geometry.x, row.geometry.y)
    if 0 <= row_idx < grid_height and 0 <= col_idx < grid_width:
        height = copernicus_10m[row_idx, col_idx]
        copernicus_heights.append(height if height != NODATA else np.nan)
    else:
        copernicus_heights.append(np.nan)

saocom_gdf['copernicus_height'] = copernicus_heights

# Rename HEIGHT column for clarity
saocom_gdf['HEIGHT_RELATIVE'] = saocom_gdf['HEIGHT']

# ============================================================================
# CALIBRATE SAOCOM TO TINITALY (Method 1: Constant Offset)
# ============================================================================
# Filter to stable points: high coherence, valid reference data
stable_mask_tin = (
    (saocom_gdf['COHER'] >= 0.8) &
    (saocom_gdf['tinitaly_height'].notna()) &
    (saocom_gdf['HEIGHT_RELATIVE'].notna()) &
    (np.abs(saocom_gdf['HEIGHT_RELATIVE']) < 1000)  # Exclude extreme outliers
)

stable_points_tin = saocom_gdf[stable_mask_tin].copy()

# Calculate offset: difference between reference DEM and SAOCOM relative heights
height_diff_tin = stable_points_tin['tinitaly_height'] - stable_points_tin['HEIGHT_
offset_tinitaly = np.median(height_diff_tin)  # Median for robustness

print(f"\nTINITALY Calibration:")
print(f"  Stable points used: {len(stable_points_tin):,}")
print(f"  Constant offset: {offset_tinitaly:.3f} m")
print(f"  Offset std dev: {np.std(height_diff_tin):.3f} m")

# Apply correction to all SAOCOM points
saocom_gdf['HEIGHT_ABSOLUTE_TIN'] = saocom_gdf['HEIGHT_RELATIVE'] + offset_tinitaly

# ============================================================================
# CALIBRATE SAOCOM TO COPERNICUS (Method 1: Constant Offset)
# ============================================================================
stable_mask_cop = (
    (saocom_gdf['COHER'] >= 0.8) &
    (saocom_gdf['copernicus_height'].notna()) &
    (saocom_gdf['HEIGHT_RELATIVE'].notna()) &
    (np.abs(saocom_gdf['HEIGHT_RELATIVE']) < 1000)
)

stable_points_cop = saocom_gdf[stable_mask_cop].copy()
```

```
height_diff_cop = stable_points_cop['copernicus_height'] - stable_points_cop['HEIGH
offset_copernicus = np.median(height_diff_cop)

print(f"\nCOPERNICUS Calibration:")
print(f"  Stable points used: {len(stable_points_cop):,}")
print(f"  Constant offset: {offset_copernicus:.3f} m")
print(f"  Offset std dev: {np.std(height_diff_cop):.3f} m")

saocom_gdf['HEIGHT_ABSOLUTE_COP'] = saocom_gdf['HEIGHT_RELATIVE'] + offset_copernic

# ============================================================================
# VALIDATION: CALCULATE RMSE AT STABLE POINTS
# ============================================================================
# TINITALY validation
residuals_tin = stable_points_tin['tinitaly_height'] - (stable_points_tin['HEIGHT_R
rmse_tin = np.sqrt(np.mean(residuals_tin**2))

# Copernicus validation
residuals_cop = stable_points_cop['copernicus_height'] - (stable_points_cop['HEIGHT
rmse_cop = np.sqrt(np.mean(residuals_cop**2))

print(f"\nValidation Results:")
print(f"  TINITALY RMSE: {rmse_tin:.3f} m")
print(f"  Copernicus RMSE: {rmse_cop:.3f} m")
print(f"\nRecommendation: Use HEIGHT_ABSOLUTE_TIN (lower RMSE expected)")
```

```
TINITALY Calibration:
  Stable points used: 46,920
  Constant offset: 4.308 m
  Offset std dev: 4.918 m

COPERNICUS Calibration:
  Stable points used: 46,939
  Constant offset: 4.761 m
  Offset std dev: 4.160 m

Validation Results:
  TINITALY RMSE: 4.955 m
  Copernicus RMSE: 4.160 m

Recommendation: Use HEIGHT_ABSOLUTE_TIN (lower RMSE expected)
```

CREATE SAOCOM COVERAGE GRID

In [9]:
```
# ============================================================================
# CREATE SAOCOM COVERAGE GRID
# ============================================================================
# Initialize coverage array matching the 10m grid
saocom_coverage = np.zeros((grid_height, grid_width), dtype=bool)

# Convert SAOCOM points to grid indices
saocom_rows, saocom_cols = rowcol(target_transform,
                                  saocom_gdf.geometry.x.values,
                                  saocom_gdf.geometry.y.values)

# Mark cells with SAOCOM data
valid_indices = ((saocom_rows >= 0) & (saocom_rows < grid_height) &
                 (saocom_cols >= 0) & (saocom_cols < grid_width))
saocom_coverage[saocom_rows[valid_indices], saocom_cols[valid_indices]] = True

# ============================================================================
# CALCULATE OVERALL VOID STATISTICS
# ============================================================================
# Study area mask (inside hull, excluding nodata)
study_area_mask = hull_mask

# Void mask (study area cells without SAOCOM data)
void_mask = study_area_mask & (~saocom_coverage)

n_total_cells = np.sum(study_area_mask)
n_occupied_cells = np.sum(study_area_mask & saocom_coverage)
n_void_cells = np.sum(void_mask)
void_percentage = 100 * n_void_cells / n_total_cells if n_total_cells > 0 else 0
print(void_percentage)


# ============================================================================
# SAVE VOID MASK AS RASTER
# ============================================================================
void_mask_path = RESULTS_DIR / "saocom_void_mask.tif"
profile_void = profile.copy()
profile_void['dtype'] = 'uint8'
profile_void['nodata'] = 255
void_raster = np.full((grid_height, grid_width), 255, dtype=np.uint8)
void_raster[study_area_mask] = 0  # Data area
void_raster[void_mask] = 1  # Void cells

with rasterio.open(void_mask_path, 'w', **profile_void) as dst:
    dst.write(void_raster, 1)
```

```
87.0403491489293
```

LOAD REFERENCE DEM DATA (Already in memory from Cell 4)

In [10]:
```
# ============================================================================
# LOAD REFERENCE DEM DATA (Already in memory from Cell 4)
# ============================================================================
tinitaly_data = tinitaly_10m.copy()
copernicus_data = copernicus_10m.copy()
```

```
# ============================================================================
# CALCULATE ELEVATION DIFFERENCE (TINITALY - COPERNICUS)
# ============================================================================
elevation_diff = tinitaly_data - copernicus_data

# ============================================================================
# CREATE VALID COMPARISON MASK
# ============================================================================
valid_mask = (tinitaly_data != NODATA) & (copernicus_data != NODATA)

# Extract valid data for statistics
# valid_pixels = np.sum(valid_mask)
valid_pixels = int(np.sum(valid_mask))
valid_diffs = elevation_diff[valid_mask]
valid_tinitaly = tinitaly_data[valid_mask]
valid_copernicus = copernicus_data[valid_mask]
print(valid_diffs)
# ============================================================================
# CALCULATE REFERENCE COMPARISON STATISTICS
# ============================================================================
ref_metrics = {
    'n_pixels': int(valid_pixels),
    'mean_diff': float(np.mean(valid_diffs)),
    'median_diff': float(np.median(valid_diffs)),
    'std_diff': float(np.std(valid_diffs)),
    'rmse': float(np.sqrt(np.mean(valid_diffs**2))),
    'mae': float(np.mean(np.abs(valid_diffs))),
    'nmad': float(1.4826 * np.median(np.abs(valid_diffs - np.median(valid_diffs)))),
    'min_diff': float(np.min(valid_diffs)),
    'max_diff': float(np.max(valid_diffs)),
    'correlation': float(np.corrcoef(valid_tinitaly, valid_copernicus)[0, 1])
}

# ============================================================================
# DEFINE EQUALITY TOLERANCE USING NMAD
# ============================================================================
# Use NMAD as statistical threshold for "roughly equal"
equal_tolerance = ref_metrics['nmad']
print(equal_tolerance)
# ============================================================================
# DIRECTIONAL COMPARISON GRIDS WITH EQUALITY BUFFER
# ============================================================================
# Where TINITALY significantly > Copernicus
tinitaly_higher_mask = (valid_mask) & (elevation_diff > equal_tolerance)
tinitaly_higher_data = np.full_like(elevation_diff, np.nan)
tinitaly_higher_data[tinitaly_higher_mask] = elevation_diff[tinitaly_higher_mask]

# Where TINITALY significantly < Copernicus
tinitaly_lower_mask = (valid_mask) & (elevation_diff < -equal_tolerance)
tinitaly_lower_data = np.full_like(elevation_diff, np.nan)
tinitaly_lower_data[tinitaly_lower_mask] = elevation_diff[tinitaly_lower_mask]

# Where TINITALY ≈ Copernicus (within tolerance)
roughly_equal_mask = (valid_mask) & (np.abs(elevation_diff) <= equal_tolerance)
roughly_equal_data = np.full_like(elevation_diff, np.nan)
```

```
roughly_equal_data[roughly_equal_mask] = elevation_diff[roughly_equal_mask]

# Pixel counts and percentages
higher_pixels = int(np.sum(tinitaly_higher_mask))
lower_pixels = int(np.sum(tinitaly_lower_mask))
equal_pixels = int(np.sum(roughly_equal_mask))

pct_higher = float(100 * higher_pixels / valid_pixels) if valid_pixels > 0 else 0.0
pct_lower = float(100 * lower_pixels / valid_pixels) if valid_pixels > 0 else 0.0
pct_equal = float(100 * equal_pixels / valid_pixels) if valid_pixels > 0 else 0.0

# ============================================================================
# HEIGHT STATISTICS COMPARISON
# ============================================================================
def calculate_height_stats(data, name):
    """Calculate comprehensive height statistics"""
    valid_data = data[~np.isnan(data)]

    if len(valid_data) == 0:
        return None

    stats = {
        'Dataset': name,
        'Count': len(valid_data),
        'Min': np.min(valid_data),
        'Max': np.max(valid_data),
        'Mean': np.mean(valid_data),
        'Median': np.median(valid_data),
        'Std Dev': np.std(valid_data),
        'Range': np.max(valid_data) - np.min(valid_data),
        'Q25': np.percentile(valid_data, 25),
        'Q75': np.percentile(valid_data, 75),
        'IQR': np.percentile(valid_data, 75) - np.percentile(valid_data, 25)
    }
    return stats

# Collect statistics
stats_list = []

# SAOCOM relative heights
stats_list.append(calculate_height_stats(
    saocom_gdf['HEIGHT_RELATIVE'].values,
    'SAOCOM (Relative)'
))

# TINITALY sampled at SAOCOM points
stats_list.append(calculate_height_stats(
    saocom_gdf['tinitaly_height'].values,
    'TINITALY (at SAOCOM pts)'
))

# Copernicus sampled at SAOCOM points
stats_list.append(calculate_height_stats(
    saocom_gdf['copernicus_height'].values,
    'Copernicus (at SAOCOM pts)'
))
```

```
# TINITALY full raster (within study area)
tinitaly_valid = tinitaly_10m[tinitaly_10m != NODATA]
stats_list.append(calculate_height_stats(
    tinitaly_valid,
    'TINITALY (Full Grid)'
))

# Copernicus full raster (within study area)
copernicus_valid = copernicus_10m[copernicus_10m != NODATA]
stats_list.append(calculate_height_stats(
    copernicus_valid,
    'Copernicus (Full Grid)'
))

# Create DataFrame
stats_df = pd.DataFrame(stats_list)

# Display with formatting
print("\n" + "="*90)
print("HEIGHT STATISTICS SUMMARY (all values in meters)")
print("="*90)
print(stats_df.to_string(index=False, float_format=lambda x: f'{x:.2f}'))
print("="*90)

# Additional comparison: SAOCOM vs Reference DEMs
print("\nDIFFERENCE STATISTICS (SAOCOM Relative - Reference DEM):")
print("-"*90)

# SAOCOM - TINITALY
diff_tin = saocom_gdf['HEIGHT_RELATIVE'] - saocom_gdf['tinitaly_height']
diff_tin_valid = diff_tin.dropna()
print(f"\nSAOCOM - TINITALY:")
print(f"  Mean difference: {diff_tin_valid.mean():+.3f} m")
print(f"  Median difference: {diff_tin_valid.median():+.3f} m")
print(f"  Std deviation: {diff_tin_valid.std():.3f} m")
print(f"  RMSE: {np.sqrt((diff_tin_valid**2).mean()):.3f} m")

# SAOCOM - Copernicus
diff_cop = saocom_gdf['HEIGHT_RELATIVE'] - saocom_gdf['copernicus_height']
diff_cop_valid = diff_cop.dropna()
print(f"\nSAOCOM - Copernicus:")
print(f"  Mean difference: {diff_cop_valid.mean():+.3f} m")
print(f"  Median difference: {diff_cop_valid.median():+.3f} m")
print(f"  Std deviation: {diff_cop_valid.std():.3f} m")
print(f"  RMSE: {np.sqrt((diff_cop_valid**2).mean()):.3f} m")

# TINITALY - Copernicus (reference comparison)
ref_diff = (tinitaly_10m[valid_mask] - copernicus_10m[valid_mask])
ref_diff_valid = ref_diff[~np.isnan(ref_diff)]
print(f"\nTINITALY - Copernicus (Reference Check):")
print(f"  Mean difference: {ref_diff_valid.mean():+.3f} m")
print(f"  Median difference: {np.median(ref_diff_valid):+.3f} m")
print(f"  Std deviation: {ref_diff_valid.std():.3f} m")
print(f"  RMSE: {np.sqrt((ref_diff_valid**2).mean()):.3f} m")
```

```
[-14.70282   -11.711487   -9.1297    ...  -0.7229309  -1.0675964
  -1.7591248]
2.1790504455566406

================================================================================
======
HEIGHT STATISTICS SUMMARY (all values in meters)
================================================================================
======
                   Dataset  Count     Min     Max   Mean  Median  Std Dev   Range
Q25    Q75    IQR
         SAOCOM (Relative)  66765 -562.00 1163.70 340.09  327.00   116.87 1725.70 25
2.00 418.80 166.80
  TINITALY (at SAOCOM pts)  66690   99.32  825.80 343.79  330.04   116.54  726.48 25
5.95 421.12 165.17
Copernicus (at SAOCOM pts)  66765  100.09  826.59 345.34  331.56   116.86  726.50 25
6.84 423.97 167.13
      TINITALY (Full Grid) 493124   98.84  826.76 354.23  341.34   129.15  727.92 25
6.91 442.43 185.52
    Copernicus (Full Grid) 495376   99.60  827.02 356.89  343.88   129.64  727.42 25
8.68 446.53 187.85
================================================================================
======

DIFFERENCE STATISTICS (SAOCOM Relative - Reference DEM):
--------------------------------------------------------------------------------
------

SAOCOM - TINITALY:
  Mean difference: -3.911 m
  Median difference: -4.383 m
  Std deviation: 14.745 m
  RMSE: 15.254 m

SAOCOM - Copernicus:
  Mean difference: -5.251 m
  Median difference: -4.939 m
  Std deviation: 14.473 m
  RMSE: 15.397 m

TINITALY - Copernicus (Reference Check):
  Mean difference: -2.029 m
  Median difference: -0.585 m
  Std deviation: 4.222 m
  RMSE: 4.684 m
```

LOAD DBF LOOKUP TABLE

In [11]:

```
# ------------------------------------------------------------------------
# 3. LOAD DBF LOOKUP TABLE
# ------------------------------------------------------------------------
dbf_table = DBF(corine_dbf_path, load=True)
lookup_df = pd.DataFrame(iter(dbf_table))

# Create mapping dictionaries
```

```
value_to_code = dict(zip(lookup_df['Value'], lookup_df['CODE_18']))
value_to_label = dict(zip(lookup_df['Value'], lookup_df['LABEL3']))
code_to_label = dict(zip(lookup_df['CODE_18'], lookup_df['LABEL3']))

# ------------------------------------------------------------------------------
# 4. LOAD, MASK, AND REMAP CORINE (OPTIMIZED)
# ------------------------------------------------------------------------------
with rasterio.open(corine_path) as src:
    # Reproject hull to match CORINE CRS
    hull_corine_crs = hull_gdf.to_crs(src.crs)

    # Crop to study area using convex hull FIRST (faster processing)
    corine_raw, crop_transform = mask(src, hull_corine_crs.geometry, crop=True, fil
    corine_raw = corine_raw[0]  # Extract from (1, h, w) to (h, w)

    corine_crs = src.crs
    corine_res = src.res
    corine_nodata = src.nodata if src.nodata is not None else 255
    corine_bounds = src.bounds

# Remap values: Value → CODE_18 (on cropped array)
corine_remapped = np.full_like(corine_raw, 0, dtype=np.uint16)
for value, code in value_to_code.items():
    corine_remapped[corine_raw == value] = code
corine_remapped[corine_raw == corine_nodata] = 0  # NoData = 0

# Save intermediate remapped version
corine_remapped_path = RESULTS_DIR / "corine_remapped_cropped.tif"
profile_remapped = {
    'driver': 'GTiff', 'dtype': 'uint16',
    'width': corine_remapped.shape[1], 'height': corine_remapped.shape[0],
    'count': 1, 'crs': corine_crs, 'transform': crop_transform,
    'nodata': 0, 'compress': 'lzw'
}
with rasterio.open(corine_remapped_path, 'w', **profile_remapped) as dst:
    dst.write(corine_remapped, 1)

# ------------------------------------------------------------------------------
# 5. RESAMPLE TO 10M GRID
# ------------------------------------------------------------------------------
corine_10m = np.full((grid_height, grid_width), 0, dtype=np.uint16)

with rasterio.open(corine_remapped_path) as src:
    reproject(
        source=rasterio.band(src, 1),
        destination=corine_10m,
        src_transform=src.transform,
        src_crs=src.crs,
        dst_transform=target_transform,
        dst_crs=TARGET_CRS,
        resampling=Resampling.nearest,  # Use nearest for categorical data
        src_nodata=0,
        dst_nodata=0
    )

# Save resampled version
```

```
corine_10m_path = RESULTS_DIR / "corine_10m.tif"
profile_10m = {
    'driver': 'GTiff', 'dtype': 'uint16',
    'width': grid_width, 'height': grid_height,
    'count': 1, 'crs': TARGET_CRS, 'transform': target_transform,
    'nodata': 0, 'compress': 'lzw'
}
with rasterio.open(corine_10m_path, 'w', **profile_10m) as dst:
    dst.write(corine_10m, 1)

# ------------------------------------------------------------------------------
# 6. MASK TO HULL
# ------------------------------------------------------------------------------
corine_10m_masked = corine_10m.copy()
corine_10m_masked[~hull_mask] = 0

# Save final masked version
corine_masked_path = RESULTS_DIR / "corine_10m_masked.tif"
with rasterio.open(corine_masked_path, 'w', **profile_10m) as dst:
    dst.write(corine_10m_masked, 1)

# Update working array
corine_10m = corine_10m_masked

# ------------------------------------------------------------------------------
# SUMMARY
# ------------------------------------------------------------------------------
unique_codes = np.unique(corine_10m[corine_10m > 0])
print(f"\nCORINE Processing Complete:")
print(f"  Original CRS: {corine_crs}")
print(f"  Unique classes: {len(unique_codes)}")
print(f"  Classes present: {sorted(unique_codes)}")
print(f"  Final resolution: {GRID_SIZE}m")
print(f"  Output path: {corine_masked_path}")
print(saocom_gdf)
```

```
CORINE Processing Complete:
  Original CRS: IGNF:ETRS89LAEA
  Unique classes: 10
  Classes present: [np.uint16(112), np.uint16(221), np.uint16(223), np.uint16(231),
np.uint16(242), np.uint16(243), np.uint16(311), np.uint16(312), np.uint16(313), np.u
int16(331)]
  Final resolution: 10m
  Output path: results\corine_10m_masked.tif
          ID  SVET  LVET    LAT_old        LAT    LON_old        LON  HEIGHT  \
0          1   221   540  45.472638  45.471953  11.136250  11.130792    97.7
1          2   329   540  45.473973  45.473288  11.146259  11.140801   114.9
2          3   396   540  45.475019  45.474334  11.154119  11.148661   207.6
3          4   399   540  45.475076  45.474391  11.154547  11.149089   215.5
4          5   402   540  45.475128  45.474443  11.154941  11.149483   221.8
...      ...   ...   ...        ...        ...        ...        ...     ...
66760  66787   699  2500  45.534820  45.534135  11.171112  11.165654   465.8
66761  66788   711  2500  45.535035  45.534350  11.172733  11.167275   493.5
66762  66789   765  2500  45.535593  45.534908  11.176933  11.171475   463.9
66763  66790   927  2500  45.537737  45.537052  11.193099  11.187641   554.3
66764  66791  1006  2500  45.539339  45.538654  11.205214  11.199756   811.3

       HEIGHT_WRT_DEM  SIGMA_HEIGHT  COHER                       geometry  \
0                97.7      2.288012   0.71  POINT (666554.015 5037588.965)
1               114.9      2.207678   0.77  POINT (667332.413 5037758.078)
2               207.6      1.852991   0.90  POINT (667943.665 5037890.687)
3               215.5      1.924608   0.90  POINT (667976.949 5037897.914)
4               221.8      2.133719   0.83   POINT (668007.59 5037904.515)
...               ...           ...    ...                           ...
66760           465.8      1.976760   0.91  POINT (669092.656 5044569.999)
66761           493.5      1.936943   0.94  POINT (669218.576 5044597.302)
66762           463.9      1.876382   0.89  POINT (669544.828 5044668.159)
66763           554.3      1.962382   0.81  POINT (670800.535 5044940.634)
66764           811.3      2.671757   0.76   POINT (671741.529 5045144.47)

              x_utm         y_utm  tinitaly_height  copernicus_height  \
0      666554.014609  5.037589e+06       100.003799         100.706978
1      667332.413137  5.037758e+06       118.215897         116.914986
2      667943.665216  5.037891e+06       212.287094         213.647568
3      667976.948898  5.037898e+06       211.314804         218.554688
4      668007.589893  5.037905e+06       217.124496         226.343140
...              ...           ...              ...                ...
66760  669092.655867  5.044570e+06       467.319885         470.105194
66761  669218.576178  5.044597e+06       508.733185         505.111084
66762  669544.827517  5.044668e+06       467.017303         467.463135
66763  670800.534953  5.044941e+06       554.237671         552.381348
66764  671741.528756  5.045144e+06       818.630981         822.318176

       HEIGHT_RELATIVE  HEIGHT_ABSOLUTE_TIN  HEIGHT_ABSOLUTE_COP
0                 97.7           102.008191             102.4612
1                114.9           119.208191             119.6612
2                207.6           211.908191             212.3612
3                215.5           219.808191             220.2612
4                221.8           226.108191             226.5612
...                ...                  ...                  ...
66760            465.8           470.108191             470.5612
66761            493.5           497.808191             498.2612
```

```
66762           463.9          468.208191           468.6612
66763           554.3          558.608191           559.0612
66764           811.3          815.608191           816.0612

[66765 rows x 19 columns]
```

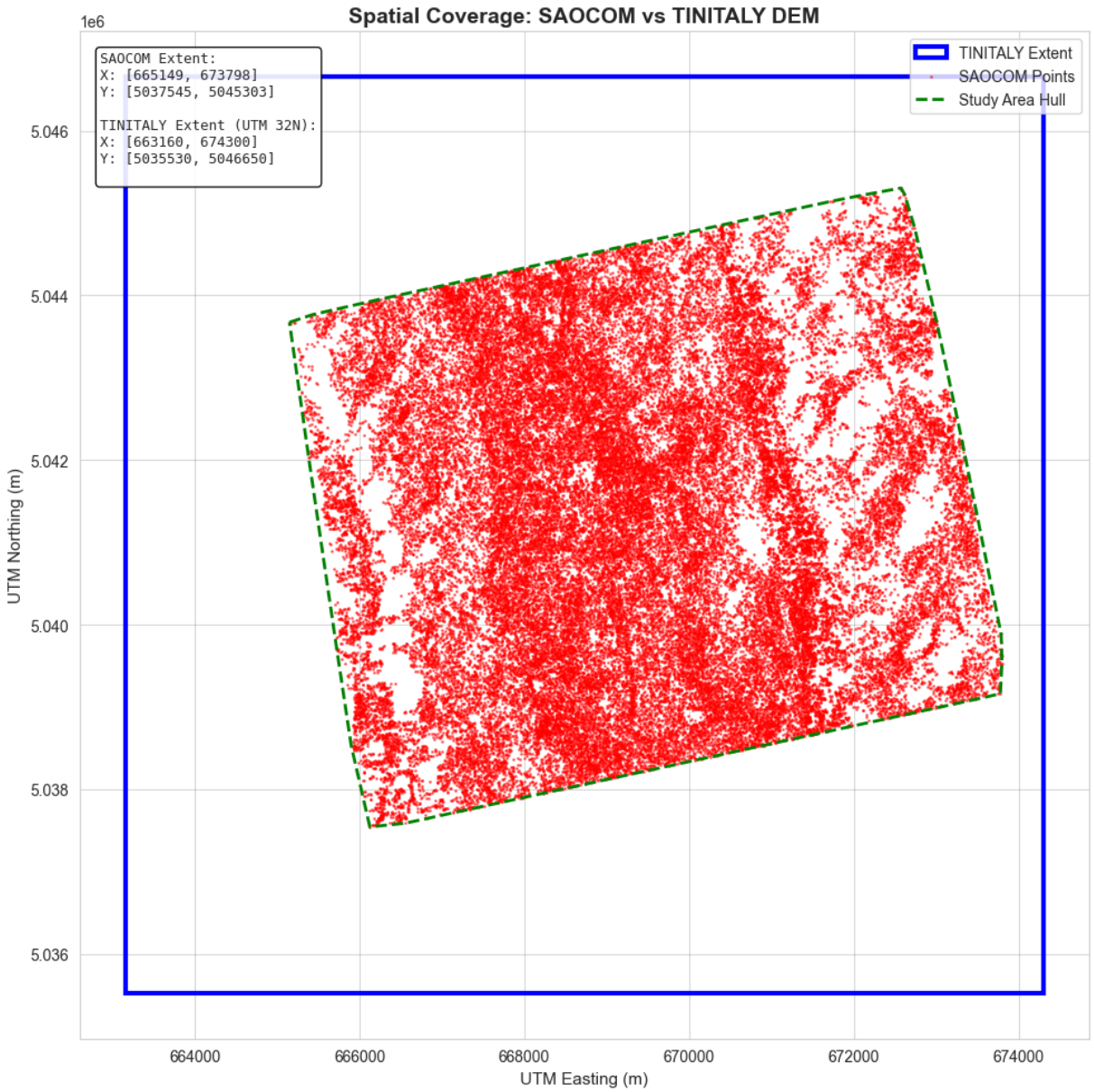
SIMPLE SPATIAL OVERLAP VISUALIZATION

In [12]:

```
# ============================================================================
# SIMPLE SPATIAL OVERLAP VISUALIZATION
# ============================================================================
import matplotlib.pyplot as plt
import rasterio
from rasterio.plot import show

fig, ax = plt.subplots(1, 1, figsize=(12, 10), facecolor='white')
ax.set_facecolor('white')

# 1. Plot original TINITALY DEM
with rasterio.open(tinitaly_path) as src:
    # Reproject to target CRS for comparison
    from rasterio.warp import transform_bounds

    # Get DEM bounds in target CRS
    dem_bounds_target = transform_bounds(src.crs, TARGET_CRS, *src.bounds)

    print(f"TINITALY bounds (original CRS): {src.bounds}")
    print(f"TINITALY bounds (UTM 32N): {dem_bounds_target}")
    print(f"TINITALY CRS: {src.crs}")

    # Read a downsampled version for quick plotting
    dem_data = src.read(1, out_shape=(
        int(src.height / 10),
        int(src.width / 10)
    ))

    # Plot DEM extent as a rectangle
    from matplotlib.patches import Rectangle
    dem_rect = Rectangle(
        (dem_bounds_target[0], dem_bounds_target[1]),
        dem_bounds_target[2] - dem_bounds_target[0],
        dem_bounds_target[3] - dem_bounds_target[1],
        linewidth=3, edgecolor='blue', facecolor='none',
        label='TINITALY Extent'
    )
    ax.add_patch(dem_rect)

# 2. Plot SAOCOM points
saocom_gdf.plot(ax=ax, markersize=1, color='red', alpha=0.5, label='SAOCOM Points')

# 3. Plot study area (convex hull)
hull_gdf.boundary.plot(ax=ax, color='green', linewidth=2, linestyle='--', label='St

# Labels and formatting
ax.set_xlabel('UTM Easting (m)', fontsize=11)
```

```
ax.set_ylabel('UTM Northing (m)', fontsize=11)
ax.set_title('Spatial Coverage: SAOCOM vs TINITALY DEM', fontweight='bold', fontsiz
ax.legend(loc='best', fontsize=10)
ax.grid(True, alpha=0.3, color='gray')

# Add text box with extents
info_text = f"""SAOCOM Extent:
X: [{saocom_gdf.geometry.x.min():.0f}, {saocom_gdf.geometry.x.max():.0f}]
Y: [{saocom_gdf.geometry.y.min():.0f}, {saocom_gdf.geometry.y.max():.0f}]

TINITALY Extent (UTM 32N):
X: [{dem_bounds_target[0]:.0f}, {dem_bounds_target[2]:.0f}]
Y: [{dem_bounds_target[1]:.0f}, {dem_bounds_target[3]:.0f}]
"""
ax.text(0.02, 0.98, info_text, transform=ax.transAxes,
        fontsize=9, verticalalignment='top', fontfamily='monospace',
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, edgecolor='black'

plt.tight_layout()
plt.show()

# ============================================================================
# CHECK IF EXTENTS OVERLAP
# ============================================================================
print("\n=== OVERLAP CHECK ===")
overlap_x = not (saocom_gdf.geometry.x.max() < dem_bounds_target[0] or
                 saocom_gdf.geometry.x.min() > dem_bounds_target[2])
overlap_y = not (saocom_gdf.geometry.y.max() < dem_bounds_target[1] or
                 saocom_gdf.geometry.y.min() > dem_bounds_target[3])

print(f"X-axis overlap: {overlap_x}")
print(f"Y-axis overlap: {overlap_y}")
print(f"Full overlap: {overlap_x and overlap_y}")

if not (overlap_x and overlap_y):
    print("\n⚠️ NO OVERLAP DETECTED - SAOCOM data is outside TINITALY coverage!")
    print("You need a different TINITALY tile that covers this area.")
```

```
TINITALY bounds (original CRS): BoundingBox(left=663160.0, bottom=5035530.0, right=6
74300.0, top=5046650.0)
TINITALY bounds (UTM 32N): (663160.0, 5035530.0, 674300.0, 5046650.0)
TINITALY CRS: EPSG:32632
```




```
=== OVERLAP CHECK ===
X-axis overlap: True
Y-axis overlap: True
Full overlap: True
```

COMPREHENSIVE REFERENCE DEM COMPARISON VISUALIZATION

In [13]:
```
# ================================================================================
# COMPREHENSIVE REFERENCE DEM COMPARISON VISUALIZATION
# ================================================================================
fig, axes = plt.subplots(4, 2, figsize=(20, 28), facecolor='white')

extent = [xmin_grid, xmax_grid, ymin_grid, ymax_grid]

# Plot 1: TINITALY elevation
ax = axes[0, 0]
ax.set_facecolor('white')
tinitaly_display = np.ma.masked_where(tinitaly_data == NODATA, tinitaly_data)
```

```
cmap1 = plt.cm.terrain.copy()
cmap1.set_bad(color='white', alpha=0)
im1 = ax.imshow(tinitaly_display, cmap=cmap1, origin='upper', extent=extent)
hull_gdf.boundary.plot(ax=ax, color='darkred', linewidth=2.5, label='Study Area')
ax.set_title('TINITALY Elevation', fontweight='bold', fontsize=12, color='black')
ax.set_xlabel('UTM Easting (m)', color='black')
ax.set_ylabel('UTM Northing (m)', color='black')
ax.grid(True, color='black', alpha=0.3, linewidth=0.5)
ax.tick_params(colors='black')
cbar1 = plt.colorbar(im1, ax=ax, label='Elevation (m)', shrink=0.8)
cbar1.ax.yaxis.label.set_color('black')
cbar1.ax.tick_params(colors='black')
stats1 = f"""Min: {np.nanmin(tinitaly_display):.1f}m
Max: {np.nanmax(tinitaly_display):.1f}m
Mean: {np.nanmean(tinitaly_display):.1f}m
Std: {np.nanstd(tinitaly_display):.1f}m"""
ax.text(0.02, 0.98, stats1, transform=ax.transAxes, fontsize=9, verticalalignment='
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, edgecolor='black'

# Plot 2: Copernicus elevation
ax = axes[0, 1]
ax.set_facecolor('white')
copernicus_display = np.ma.masked_where(copernicus_data == NODATA, copernicus_data)
cmap2 = plt.cm.terrain.copy()
cmap2.set_bad(color='white', alpha=0)
im2 = ax.imshow(copernicus_display, cmap=cmap2, origin='upper', extent=extent)
hull_gdf.boundary.plot(ax=ax, color='darkred', linewidth=2.5, label='Study Area')
ax.set_title('Copernicus Elevation', fontweight='bold', fontsize=12, color='black')
ax.set_xlabel('UTM Easting (m)', color='black')
ax.set_ylabel('UTM Northing (m)', color='black')
ax.grid(True, color='black', alpha=0.3, linewidth=0.5)
ax.tick_params(colors='black')
cbar2 = plt.colorbar(im2, ax=ax, label='Elevation (m)', shrink=0.8)
cbar2.ax.yaxis.label.set_color('black')
cbar2.ax.tick_params(colors='black')
stats2 = f"""Min: {np.nanmin(copernicus_display):.1f}m
Max: {np.nanmax(copernicus_display):.1f}m
Mean: {np.nanmean(copernicus_display):.1f}m
Std: {np.nanstd(copernicus_display):.1f}m"""
ax.text(0.02, 0.98, stats2, transform=ax.transAxes, fontsize=9, verticalalignment='
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, edgecolor='black'

# Plot 3: Difference map (full)
ax = axes[1, 0]
ax.set_facecolor('white')
diff_display = np.ma.masked_where(~valid_mask, elevation_diff)
diff_limit = np.percentile(np.abs(valid_diffs), 95)
cmap3 = plt.cm.coolwarm.copy()
cmap3.set_bad(color='white', alpha=0)
im3 = ax.imshow(diff_display, cmap=cmap3, origin='upper', extent=extent,
                vmin=-diff_limit, vmax=diff_limit)
hull_gdf.boundary.plot(ax=ax, color='darkred', linewidth=2.5, label='Study Area')
ax.set_title(f'Elevation Difference\n(TINITALY - Copernicus)', fontweight='bold', f
ax.set_xlabel('UTM Easting (m)', color='black')
ax.set_ylabel('UTM Northing (m)', color='black')
ax.grid(True, color='black', alpha=0.3, linewidth=0.5)
```

```
ax.tick_params(colors='black')
cbar3 = plt.colorbar(im3, ax=ax, label='Difference (m)', shrink=0.8)
cbar3.ax.yaxis.label.set_color('black')
cbar3.ax.tick_params(colors='black')
stats3 = f"""Pixels: {valid_pixels:,}
Mean: {ref_metrics['mean_diff']:+.2f}m
RMSE: {ref_metrics['rmse']:.2f}m
NMAD: {ref_metrics['nmad']:.2f}m
MAE: {ref_metrics['mae']:.2f}m
Std: {ref_metrics['std_diff']:.2f}m
Corr: {ref_metrics['correlation']:.3f}
Range: [{ref_metrics['min_diff']:.1f}, {ref_metrics['max_diff']:.1f}]m"""
ax.text(0.02, 0.98, stats3, transform=ax.transAxes, fontsize=8, verticalalignment='
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, edgecolor='black'

# Plot 4: Statistics summary
ax = axes[1, 1]
ax.set_facecolor('white')
le68 = np.percentile(np.abs(valid_diffs), 68.27)
le90 = np.percentile(np.abs(valid_diffs), 90)
le95 = np.percentile(np.abs(valid_diffs), 95)
stats_text = f"""REFERENCE DEM COMPARISON

Valid Pixels: {valid_pixels:,}

CRITICAL METRICS:
Mean Error (Bias): {ref_metrics['mean_diff']:+.2f} m
RMSE: {ref_metrics['rmse']:.2f} m
NMAD (robust): {ref_metrics['nmad']:.2f} m
Std Deviation: {ref_metrics['std_diff']:.2f} m

SECONDARY METRICS:
MAE: {ref_metrics['mae']:.2f} m
Correlation: {ref_metrics['correlation']:.4f}
LE68: {le68:.2f} m
LE90: {le90:.2f} m
LE95: {le95:.2f} m
Median: {ref_metrics['median_diff']:+.2f} m

DIRECTIONAL BREAKDOWN:
(Tolerance: ±{equal_tolerance:.2f} m)
TINITALY Higher: {higher_pixels:,} ({pct_higher:.1f}%)
Copernicus Higher: {lower_pixels:,} ({pct_lower:.1f}%)
Roughly Equal: {equal_pixels:,} ({pct_equal:.1f}%)

Range: {ref_metrics['min_diff']:+.1f} to {ref_metrics['max_diff']:+.1f} m
"""
ax.text(0.05, 0.5, stats_text, transform=ax.transAxes,
        fontfamily='monospace', fontsize=9, verticalalignment='center', color='blac
ax.axis('off')
ax.set_title('Summary Statistics', fontweight='bold', fontsize=12, color='black')

# Plot 5: Where TINITALY > Copernicus
ax = axes[2, 0]
ax.set_facecolor('white')
cmap5 = plt.cm.YlOrRd.copy()
```

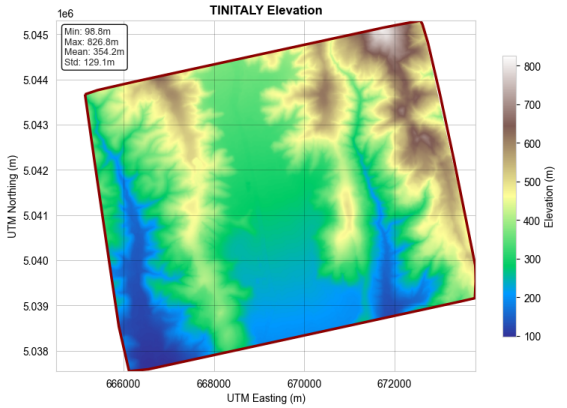
```
cmap5.set_bad(color='white', alpha=0)
im5 = ax.imshow(tinitaly_higher_data, cmap=cmap5, origin='upper', extent=extent,
                vmin=0, vmax=np.nanmax(tinitaly_higher_data))
hull_gdf.boundary.plot(ax=ax, color='darkred', linewidth=2.5, label='Study Area')
ax.set_title(f'TINITALY > Copernicus', fontweight='bold', fontsize=12, color='black
ax.set_xlabel('UTM Easting (m)', color='black')
ax.set_ylabel('UTM Northing (m)', color='black')
ax.grid(True, color='black', alpha=0.3, linewidth=0.5)
ax.tick_params(colors='black')
cbar5 = plt.colorbar(im5, ax=ax, label='Difference (m)', shrink=0.8)
cbar5.ax.yaxis.label.set_color('black')
cbar5.ax.tick_params(colors='black')
higher_vals = tinitaly_higher_data[~np.isnan(tinitaly_higher_data)]
stats5 = f"""Pixels: {higher_pixels:,} ({pct_higher:.1f}%)
Mean: {np.mean(higher_vals):.2f}m
Std: {np.std(higher_vals):.2f}m
RMSE: {np.sqrt(np.mean(higher_vals**2)):.2f}m
Max: {np.max(higher_vals):.2f}m"""
ax.text(0.02, 0.98, stats5, transform=ax.transAxes, fontsize=8, verticalalignment='
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, edgecolor='black'

# Plot 6: Where TINITALY < Copernicus
ax = axes[2, 1]
ax.set_facecolor('white')
cmap6 = plt.cm.Blues_r.copy()
cmap6.set_bad(color='white', alpha=0)
im6 = ax.imshow(tinitaly_lower_data, cmap=cmap6, origin='upper', extent=extent,
                vmin=np.nanmin(tinitaly_lower_data), vmax=0)
hull_gdf.boundary.plot(ax=ax, color='darkred', linewidth=2.5, label='Study Area')
ax.set_title(f'Copernicus > TINITALY', fontweight='bold', fontsize=12, color='black
ax.set_xlabel('UTM Easting (m)', color='black')
ax.set_ylabel('UTM Northing (m)', color='black')
ax.grid(True, color='black', alpha=0.3, linewidth=0.5)
ax.tick_params(colors='black')
cbar6 = plt.colorbar(im6, ax=ax, label='Difference (m)', shrink=0.8)
cbar6.ax.yaxis.label.set_color('black')
cbar6.ax.tick_params(colors='black')
lower_vals = tinitaly_lower_data[~np.isnan(tinitaly_lower_data)]
stats6 = f"""Pixels: {lower_pixels:,} ({pct_lower:.1f}%)
Mean: {np.mean(lower_vals):.2f}m
Std: {np.std(lower_vals):.2f}m
RMSE: {np.sqrt(np.mean(lower_vals**2)):.2f}m
Min: {np.min(lower_vals):.2f}m"""
ax.text(0.02, 0.98, stats6, transform=ax.transAxes, fontsize=8, verticalalignment='
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, edgecolor='black'

# Plot 7: Where roughly equal
ax = axes[3, 0]
ax.set_facecolor('white')
cmap7 = plt.cm.Greens.copy()
cmap7.set_bad(color='white', alpha=0)
im7 = ax.imshow(roughly_equal_data, cmap=cmap7, origin='upper', extent=extent,
                vmin=-equal_tolerance, vmax=equal_tolerance)
hull_gdf.boundary.plot(ax=ax, color='darkred', linewidth=2.5, label='Study Area')
ax.set_title(f'Roughly Equal (±{equal_tolerance:.2f}m)', fontweight='bold', fontsiz
ax.set_xlabel('UTM Easting (m)', color='black')
```

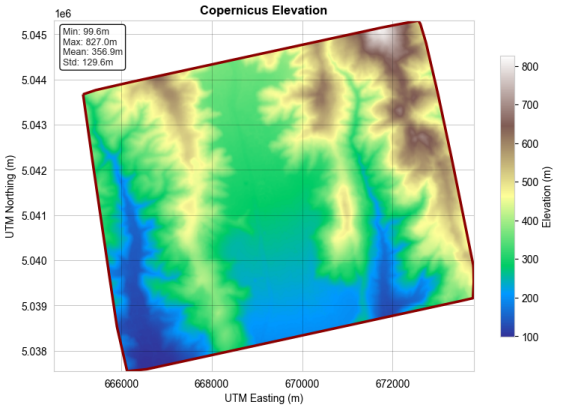
```
ax.set_ylabel('UTM Northing (m)', color='black')
ax.grid(True, color='black', alpha=0.3, linewidth=0.5)
ax.tick_params(colors='black')
cbar7 = plt.colorbar(im7, ax=ax, label='Difference (m)', shrink=0.8)
cbar7.ax.yaxis.label.set_color('black')
cbar7.ax.tick_params(colors='black')
equal_vals = roughly_equal_data[~np.isnan(roughly_equal_data)]
stats7 = f"""Pixels: {equal_pixels:,} ({pct_equal:.1f}%)
Mean: {np.mean(equal_vals):.2f}m
Std: {np.std(equal_vals):.2f}m
RMSE: {np.sqrt(np.mean(equal_vals**2)):.2f}m
MAE: {np.mean(np.abs(equal_vals)):.2f}m"""
ax.text(0.02, 0.98, stats7, transform=ax.transAxes, fontsize=8, verticalalignment='
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, edgecolor='black'

# # Plot 8: Histogram of differences
# ax = axes[3, 1]
# ax.set_facecolor('white')
# ax.hist(valid_diffs, bins=100, alpha=0.7, color='steelblue', edgecolor='black')
# ax.set_xlim(valid_diffs.min() * 1.05, valid_diffs.max() * 1.05)
# ax.axvline(0, color='red', linestyle='--', linewidth=2, label='Zero')
# ax.axvline(ref_metrics['mean_diff'], color='green', linestyle='-', linewidth=2,
#            label=f'Mean: {ref_metrics["mean_diff"]:+.2f}m')
# ax.axvline(equal_tolerance, color='orange', linestyle='--', linewidth=1.5,
#            label=f'±{equal_tolerance:.2f}m')
# ax.axvline(-equal_tolerance, color='orange', linestyle='--', linewidth=1.5)
# ax.set_xlabel('Elevation Difference (m)', color='black')
# ax.set_ylabel('Frequency', color='black')
# ax.set_title('Difference Distribution', fontweight='bold', fontsize=12, color='bl
# ax.tick_params(colors='black')
# ax.legend(loc='upper right', fontsize=9)
# ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
# for spine in ax.spines.values():
#     spine.set_edgecolor('black')
# ax.set_yscale('log')
# plt.tight_layout()
# plt.show()
# Plot 8: Histogram of differences
ax = axes[3, 1]
ax.set_facecolor('white')

# Create histogram
n, bins, patches = ax.hist(valid_diffs, bins=50, alpha=0.75,
                           color='steelblue', edgecolor='black')

# Add reference lines
ax.axvline(0, color='red', linestyle='--', linewidth=2, label='Zero')
ax.axvline(ref_metrics['mean_diff'], color='green', linestyle='-', linewidth=2,
           label=f'Mean: {ref_metrics["mean_diff"]:+.2f}m')
ax.axvline(equal_tolerance, color='orange', linestyle='--', linewidth=1.5,
           label=f'±NMAD: {equal_tolerance:.2f}m')
ax.axvline(-equal_tolerance, color='orange', linestyle='--', linewidth=1.5)

# --- ADD STATISTICS BOX ---
# Calculate the required statistics directly from the data
stats_text = (f"Mean = {valid_diffs.mean():+.2f} m\n"
```

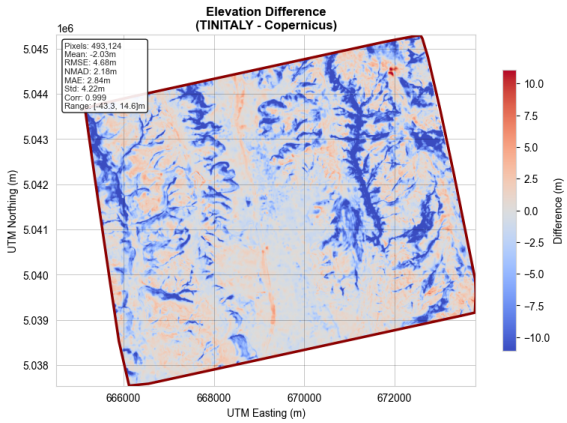
```
                f"Std Dev = {valid_diffs.std():.2f} m\n"
                f"Min = {valid_diffs.min():.2f} m\n"
                f"Max = {valid_diffs.max():.2f} m\n"
                f"RMSE = {ref_metrics['rmse']:.2f} m\n"
                f"NMAD = {ref_metrics['nmad']:.2f} m")

# Place text box in the top-right corner
ax.text(0.97, 0.97, stats_text, transform=ax.transAxes, fontsize=10,
        verticalalignment='top', horizontalalignment='right',
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.8, edgecolor='black'

# --- SET AXIS LIMITS TO DATA EXTENT ---
x_min = float(valid_diffs.min())
x_max = float(valid_diffs.max())
x_padding = (x_max - x_min) * 0.05 # Increased padding slightly
ax.set_xlim(x_min - x_padding, x_max + x_padding)

# --- LABELS AND STYLING ---
ax.set_xlabel('Elevation Difference (m)', color='black', fontsize=11)
ax.set_ylabel('Frequency', color='black', fontsize=11)
ax.set_title('Difference Distribution', fontweight='bold', fontsize=12, color='blac
ax.tick_params(colors='black')
ax.set_yscale('log')
ax.legend(loc='upper left', fontsize=9)
ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
for spine in ax.spines.values():
    spine.set_edgecolor('black')

# ============================================================================
# 1. PREPARE DIFFERENCE DATA
# ============================================================================
# Calculate differences using calibrated SAOCOM heights
saocom_gdf['diff_tinitaly'] = saocom_gdf['HEIGHT_ABSOLUTE_TIN'] - saocom_gdf['tinit
saocom_gdf['diff_copernicus'] = saocom_gdf['HEIGHT_ABSOLUTE_COP'] - saocom_gdf['cop

# Create coherence bins if not already present
if 'coherence_bin' not in saocom_gdf.columns:
    coherence_bins = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
    coherence_labels = [f"{coherence_bins[i]:.1f}-{coherence_bins[i+1]:.1f}"
                        for i in range(len(coherence_bins)-1)]
    saocom_gdf['coherence_bin'] = pd.cut(saocom_gdf['COHER'],
                                         bins=coherence_bins,
                                         labels=coherence_labels,
                                         include_lowest=True)

# ============================================================================
# 2. BASIC VIOLIN PLOTS - ALL REFERENCE COMPARISONS
# ============================================================================
fig, ax = plt.subplots(1, 1, figsize=(12, 8), facecolor='white')

# Prepare data
plot_data = pd.DataFrame({
    'SAOCOM - TINITALY': saocom_gdf['diff_tinitaly'],
    'SAOCOM - Copernicus': saocom_gdf['diff_copernicus']
})
```

```
# Create violin plot
parts = ax.violinplot([plot_data['SAOCOM - TINITALY'].dropna(),
                       plot_data['SAOCOM - Copernicus'].dropna()],
                      positions=[1, 2],
                      showmeans=True,
                      showmedians=True,
                      showextrema=True)

padding = (x_max - x_min) * 0.05
plt.xlim(x_min - padding, x_max + padding)
# Color the violins
colors = ['#4472C4', '#ED7D31']
for pc, color in zip(parts['bodies'], colors):
    pc.set_facecolor(color)
    pc.set_alpha(0.7)
    pc.set_edgecolor('black')

# Style other elements
for partname in ('cbars', 'cmins', 'cmaxes', 'cmedians', 'cmeans'):
    if partname in parts:
        parts[partname].set_edgecolor('black')
        parts[partname].set_linewidth(1.5)

ax.axhline(y=0, color='red', linestyle='--', linewidth=1.5, alpha=0.7, label='Zero'
ax.set_xticks([1, 2])
ax.set_xticklabels(['SAOCOM -\nTINITALY', 'SAOCOM -\nCopernicus'], fontsize=11)
ax.set_ylabel('Elevation Difference (m)', fontsize=12)
ax.set_title('Distribution of Elevation Differences', fontweight='bold', fontsize=1
ax.grid(axis='y', linestyle='--', alpha=0.3)
ax.legend()

# Add statistics
for i, col in enumerate(['SAOCOM - TINITALY', 'SAOCOM - Copernicus'], 1):
    data = plot_data[col].dropna()
    stats_text = f"n={len(data):,}\nμ={data.mean():.2f}m\nσ={data.std():.2f}m"
    ax.text(i, data.max()*0.9, stats_text, ha='center', fontsize=9,
            bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.tight_layout()
plt.show()

# ============================================================================
# 3. VIOLIN PLOTS BY COHERENCE BINS
# ============================================================================
fig, axes = plt.subplots(1, 2, figsize=(16, 7), facecolor='white')

for idx, (col, title) in enumerate([('diff_tinitaly', 'SAOCOM - TINITALY'),
                                    ('diff_copernicus', 'SAOCOM - Copernicus')]):
    ax = axes[idx]

    # Filter data with valid coherence bins and differences
    plot_df = saocom_gdf[saocom_gdf['coherence_bin'].notna() &
                         saocom_gdf[col].notna()].copy()

    if len(plot_df) > 0:
        sns.violinplot(x='coherence_bin', y=col, data=plot_df,
```

```
                        inner='quartile', palette='viridis', ax=ax)
        y_min = plot_df[col].min()
        y_max = plot_df[col].max()
        padding = (y_max - y_min) * 0.05
        ax.set_ylim(y_min - padding, y_max + padding)
        ax.axhline(y=0, color='red', linestyle='--', linewidth=1.5, alpha=0.7)
        ax.set_xlabel('Coherence Bin', fontsize=11)
        ax.set_ylabel('Difference (m)', fontsize=11)
        ax.set_title(f'{title} by Coherence', fontweight='bold', fontsize=12)
        ax.grid(axis='y', linestyle='--', alpha=0.3)

        # Add sample counts
        bin_counts = plot_df['coherence_bin'].value_counts().sort_index()
        labels = [f"{label}\n(n={bin_counts.get(label, 0):,})"
                  for label in coherence_labels]
        ax.set_xticklabels(labels, rotation=45, ha='right', fontsize=9)

plt.tight_layout()
plt.show()

# ============================================================================
# 4. SUMMARY STATISTICS TABLE
# ============================================================================
summary_stats = []

for comparison in ['diff_tinitaly', 'diff_copernicus']:
    data = saocom_gdf[comparison].dropna()
    name = 'SAOCOM - TINITALY' if 'tinitaly' in comparison else 'SAOCOM - Copernicu

    summary_stats.append({
        'Comparison': name,
        'N Points': f"{len(data):,}",
        'Mean': f"{data.mean():+.2f} m",
        'Median': f"{data.median():+.2f} m",
        'Std Dev': f"{data.std():.2f} m",
        'RMSE': f"{np.sqrt((data**2).mean()):.2f} m",
        'Range': f"[{data.min():.1f}, {data.max():.1f}] m"
    })

summary_df = pd.DataFrame(summary_stats)
print("\n" + "="*80)
print("VIOLIN PLOT SUMMARY STATISTICS")
print("="*80)
print(summary_df.to_string(index=False))
print("="*80)
```

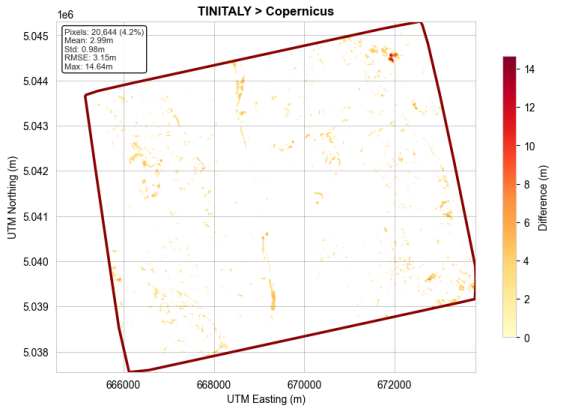


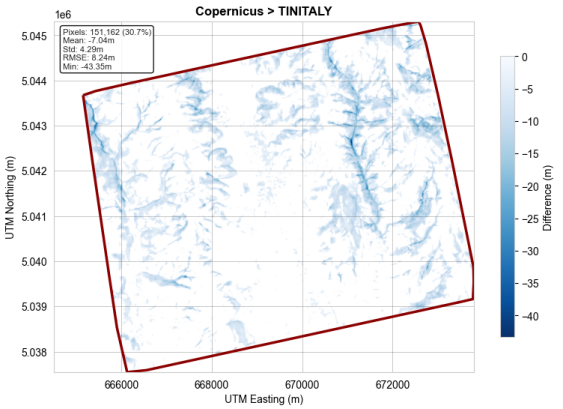


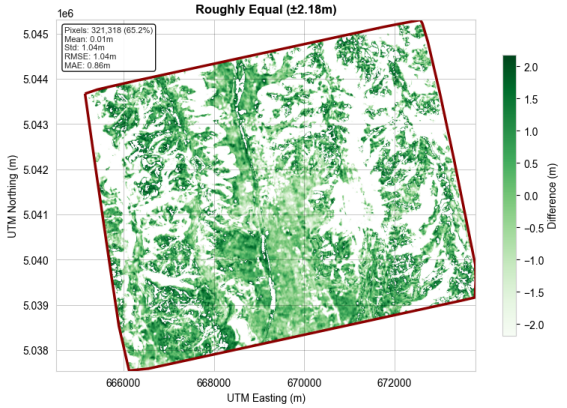


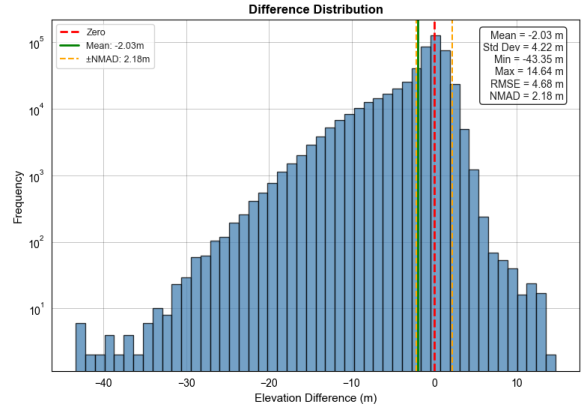
Summary Statistics

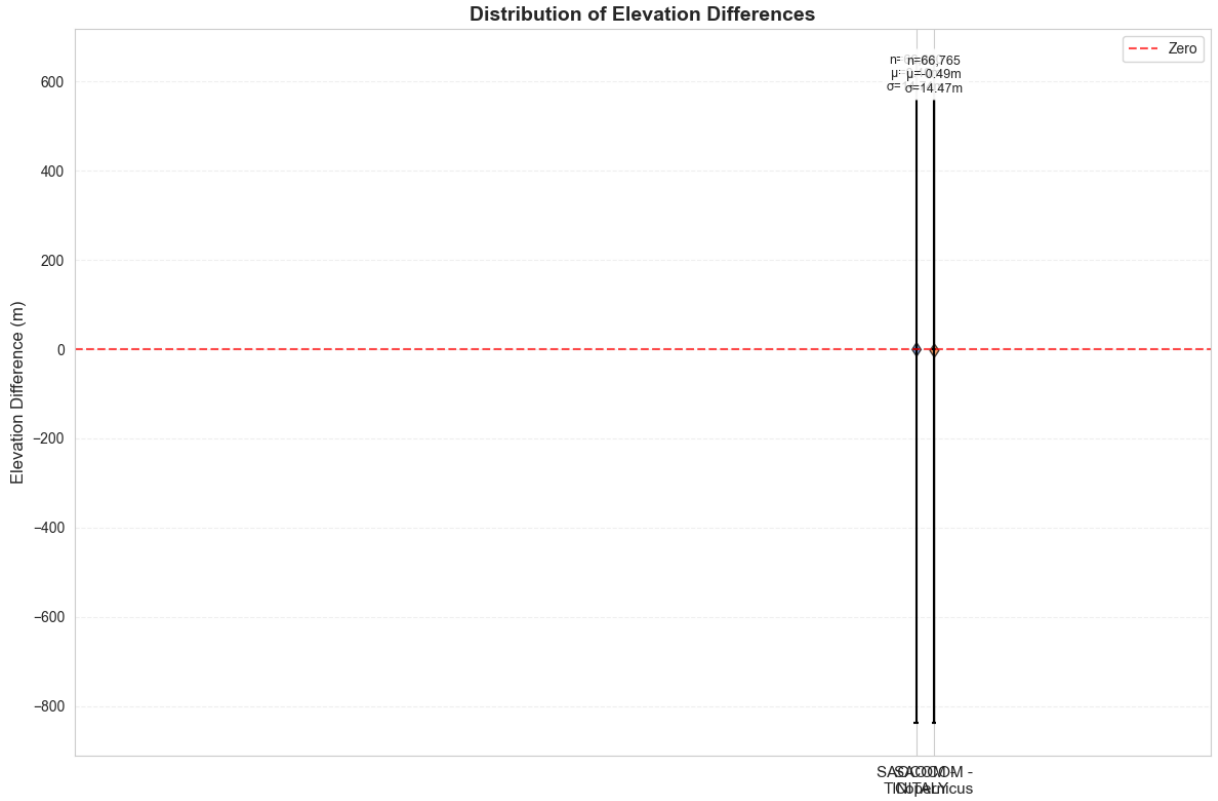
```
REFERENCE DEM COMPARISON

Valid Pixels: 493,124

CRITICAL METRICS:
Mean Error (Bias): -2.03 m
RMSE: 4.68 m
NMAD (robust): 2.18 m
Std Deviation: 4.22 m

SECONDARY METRICS:
MAE: 2.84 m
Correlation: 0.9995
LE68: 2.46 m
LE90: 8.10 m
LE95: 11.08 m
Median: -0.58 m

DIRECTIONAL BREAKDOWN:
(Tolerance: ±2.18 m)
TINITALY Higher: 20,644 (4.2%)
Copernicus Higher: 151,162 (30.7%)
Roughly Equal: 321,318 (65.2%)

Range: -43.3 to +14.6 m
```

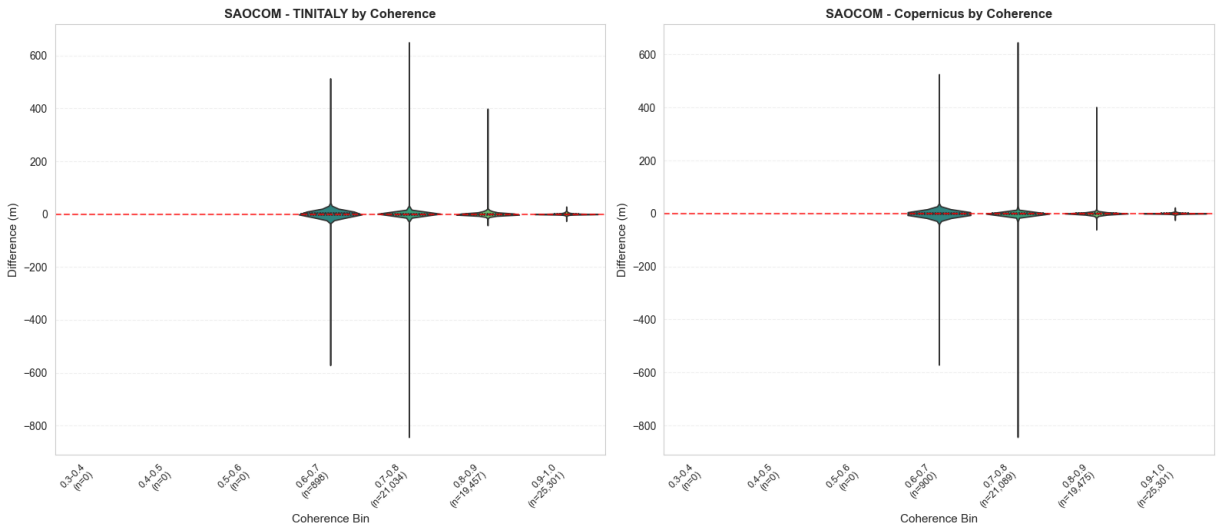













```
================================================================================
VIOLIN PLOT SUMMARY STATISTICS
================================================================================
        Comparison N Points    Mean  Median Std Dev    RMSE             Range
  SAOCOM - TINITALY   66,690 +0.40 m -0.08 m 14.74 m 14.75 m [-837.1, 643.6] m
SAOCOM - Copernicus   66,765 -0.49 m -0.18 m 14.47 m 14.48 m [-837.5, 639.7] m
================================================================================
```

SAOCOM HEIGHT RESIDUAL OUTLIER DETECTION AND VISUALIZATION

In [14]:
```
import numpy as np
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
```

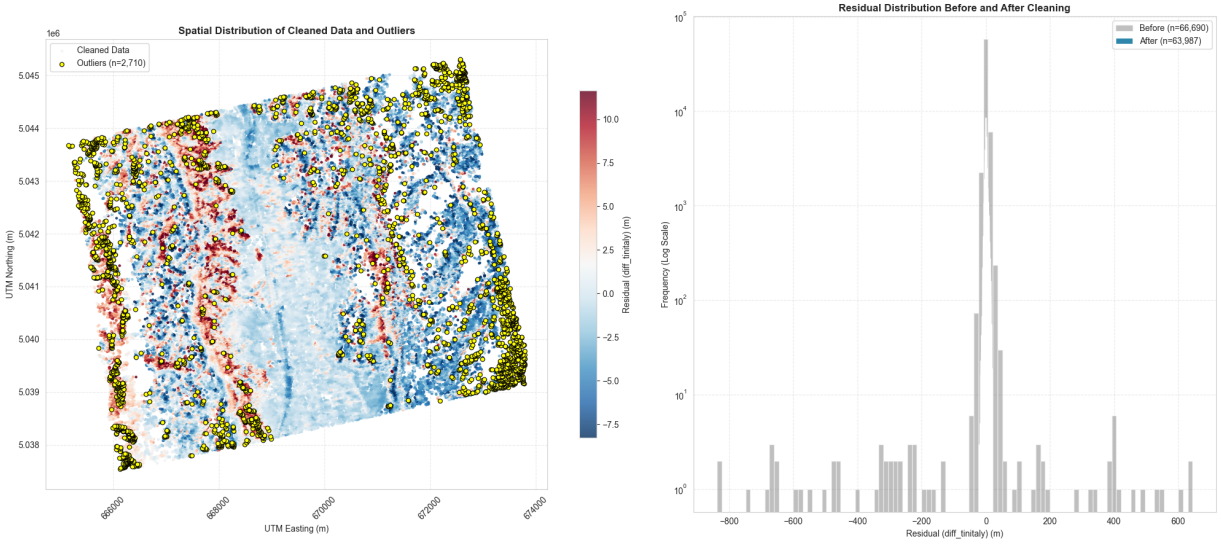
```python
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler

def score_outliers_isolation_forest(
    gdf: gpd.GeoDataFrame,
    residual_col: str,
    **kwargs
) -> gpd.GeoDataFrame:
    """
    Assigns an anomaly score to each point using the Isolation Forest algorithm.
    This function is unchanged and remains the core of the detection process.

    Args:
        gdf (gpd.GeoDataFrame): The input GeoDataFrame.
        residual_col (str): The name of the column containing the height residuals.
        **kwargs: Additional keyword arguments to pass to the IsolationForest model

    Returns:
        gpd.GeoDataFrame: The original GeoDataFrame with a new 'outlier_score' colu
    """
    if gdf.empty or residual_col not in gdf.columns:
        print("Warning: Input GeoDataFrame is empty or residual column not found.")
        gdf['outlier_score'] = np.nan
        return gdf

    points_3d = np.c_[
        gdf.geometry.x,
        gdf.geometry.y,
        gdf[residual_col].fillna(0)
    ]
    scaler = StandardScaler()
    points_3d_scaled = scaler.fit_transform(points_3d)

    model_params = {'n_estimators': 100, 'contamination': 'auto', 'random_state': 4
    model_params.update(kwargs)

    print("Fitting Isolation Forest model...")
    model = IsolationForest(**model_params)
    model.fit(points_3d_scaled)

    gdf_scored = gdf.copy()
    gdf_scored['outlier_score'] = model.decision_function(points_3d_scaled)

    print("Scoring complete. Added 'outlier_score' column.")
    return gdf_scored

def filter_by_score_iqr(
    gdf_scored: gpd.GeoDataFrame,
    iqr_multiplier: float = 1
) -> tuple[gpd.GeoDataFrame, gpd.GeoDataFrame]:
    """
    Filters a scored GeoDataFrame using a statistically-driven threshold.

    This function calculates the outlier threshold by applying the IQR method
    to the 'outlier_score' column itself. This removes the need for an
    arbitrary percentage cutoff.
```

```
    Args:
        gdf_scored (gpd.GeoDataFrame): GeoDataFrame with an 'outlier_score' column.
        iqr_multiplier (float): The multiplier for the IQR to define the cutoff.
                                A standard value is 1.5.

    Returns:
        tuple: A tuple containing the cleaned GeoDataFrame and the outlier GeoDataF
    """
    if 'outlier_score' not in gdf_scored.columns:
        raise ValueError("Input GeoDataFrame must have an 'outlier_score' column.")

    scores = gdf_scored['outlier_score']
    q1 = scores.quantile(0.25)
    q3 = scores.quantile(0.75)
    iqr = q3 - q1

    # The threshold is statistically determined from the distribution of scores
    score_threshold = q1 - (iqr_multiplier * iqr)
    print(score_threshold)
    is_outlier_mask = scores < score_threshold

    outliers = gdf_scored[is_outlier_mask].copy()
    gdf_cleaned = gdf_scored[~is_outlier_mask].copy()

    total_points, n_outliers = len(gdf_scored), len(outliers)
    pct_outliers = (n_outliers / total_points) * 100 if total_points > 0 else 0

    print("\n" + "=" * 60)
    print(f"Statistically-Driven Filtering (IQR on Anomaly Scores)")
    print("-" * 60)
    print(f"Score Q1: {q1:.4f}, Q3: {q3:.4f}, IQR: {iqr:.4f}")
    print(f"IQR Multiplier: {iqr_multiplier}")
    print(f"Calculated Score Threshold: < {score_threshold:.4f}")
    print("-" * 60)
    print(f"Total Points: {total_points:,}")
    print(f"Outliers Removed: {n_outliers:,} ({pct_outliers:.2f}%)")
    print(f"Remaining Points: {len(gdf_cleaned):,}")
    print("=" * 60)

    return gdf_cleaned, outliers

def visualize_outlier_results(
    gdf_original: gpd.GeoDataFrame,
    gdf_cleaned: gpd.GeoDataFrame,
    outliers: gpd.GeoDataFrame,
    residual_col: str
):
    """
    Creates a two-panel plot to visualize the outlier removal results.
    (This function is unchanged).
    """
    fig, axes = plt.subplots(1, 2, figsize=(20, 9), facecolor='white', gridspec_kw=

    # --- Panel 1: Spatial Distribution ---
    ax1 = axes[0]
```

```
    vmin, vmax = np.nanpercentile(gdf_cleaned[residual_col], [2, 98])

    scatter = ax1.scatter(
        gdf_cleaned.geometry.x, gdf_cleaned.geometry.y,
        c=gdf_cleaned[residual_col], cmap='RdBu_r', s=5,
        vmin=vmin, vmax=vmax, alpha=0.8, label='Cleaned Data'
    )
    plt.colorbar(scatter, ax=ax1, label=f'Residual ({residual_col}) (m)', shrink=0.

    if not outliers.empty:
        outliers.plot(ax=ax1, markersize=25, color='yellow',
                      edgecolors='black', linewidth=0.8,
                      label=f'Outliers (n={len(outliers):,})', zorder=5)

    ax1.set_title('Spatial Distribution of Cleaned Data and Outliers', fontweight='
    ax1.set_xlabel('UTM Easting (m)')
    ax1.set_ylabel('UTM Northing (m)')
    ax1.legend()
    ax1.grid(True, linestyle='--', alpha=0.4)
    ax1.tick_params(axis='x', rotation=45)
    ax1.set_aspect('equal', adjustable='box')

    # --- Panel 2: Residual Distribution Histogram ---
    ax2 = axes[1]
    original_residuals = gdf_original[residual_col].dropna()
    cleaned_residuals = gdf_cleaned[residual_col].dropna()

    ax2.hist(original_residuals, bins=100, alpha=0.5, label=f'Before (n={len(origin
    ax2.hist(cleaned_residuals, bins=50, alpha=1, label=f'After (n={len(cleaned_res

    ax2.set_title('Residual Distribution Before and After Cleaning', fontweight='bo
    ax2.set_xlabel(f'Residual ({residual_col}) (m)')
    ax2.set_ylabel('Frequency (Log Scale)')
    ax2.set_yscale('log')
    ax2.legend()
    ax2.grid(True, linestyle='--', alpha=0.4)

    plt.tight_layout()
    plt.show()

# # ==============================================================================
# # EXAMPLE USAGE
# # ==============================================================================
# if __name__ == '__main__':
#     # 1. Create a sample GeoDataFrame for demonstration
#     print("Creating sample GeoDataFrame for demonstration...")
#     np.random.seed(42)
#     clean_points = np.random.rand(1000, 2) * 1000
#     clean_residuals = np.random.normal(0, 2, 1000)
#     outlier_points = np.array([[500, 500], [100, 900], [900, 100], [10, 10]])
#     outlier_residuals = np.array([80, -65, 95, 5])
#
#     all_coords = np.vstack([clean_points, outlier_points])
#     all_residuals = np.concatenate([clean_residuals, outlier_residuals])
#     sample_gdf = gpd.GeoDataFrame(
#         {'diff_tinitaly': all_residuals},
```

```
#         geometry=[Point(x, y) for x, y in all_coords]
#     )

# --- Step 1: Score all the points ---
print(len(saocom_gdf))
saocom_gdf_scored = score_outliers_isolation_forest(saocom_gdf, 'diff_tinitaly')

# --- Step 2: Filter using the STATISTICAL method ---
# No more guessing a percentage! The function calculates the cutoff itself.
saocom_gdf_cleaned, saocom_outliers = filter_by_score_iqr(saocom_gdf_scored)

# --- Step 3: Visualize the results ---
visualize_outlier_results(saocom_gdf, saocom_gdf_cleaned, saocom_outliers, 'diff_ti
saocom_gdf = saocom_gdf_cleaned
```

```
66765
Fitting Isolation Forest model...
Scoring complete. Added 'outlier_score' column.
-0.07015801537863658

============================================================
Statistically-Driven Filtering (IQR on Anomaly Scores)
------------------------------------------------------------
Score Q1: 0.0003, Q3: 0.0707, IQR: 0.0704
IQR Multiplier: 1
Calculated Score Threshold: < -0.0702
------------------------------------------------------------
Total Points: 66,765
Outliers Removed: 2,710 (4.06%)
Remaining Points: 64,055
============================================================
```



1. SPATIAL SAMPLE CORINE LAND COVER AT SAOCOM POINTS

In [15]:
```
# ============================================================================
# PRE-REQUISITE: CALCULATE HEIGHT RESIDUALS
# (This step was missing/unexecuted from the previous cell sequence)
# ============================================================================
# Residual = Calibrated SAOCOM Height - Reference DEM Height
```

```
saocom_gdf['diff_tinitaly'] = saocom_gdf['HEIGHT_ABSOLUTE_TIN'] - saocom_gdf['tinit
# saocom_gdf['diff_copernicus'] = saocom_gdf['HEIGHT_ABSOLUTE_COP'] - saocom_gdf['c

# ============================================================================
# 1. SPATIAL SAMPLE CORINE LAND COVER AT SAOCOM POINTS
# ============================================================================
# The corine_10m raster is already loaded, reprojected, and masked (Cell 202).

corine_codes = []
# Assuming the global variables target_transform, grid_height, grid_width, corine_1
for idx, row in saocom_gdf.iterrows():
    # Use the same transform and grid dimensions as defined in Cell 200
    row_idx, col_idx = rowcol(target_transform, row.geometry.x, row.geometry.y)

    if 0 <= row_idx < grid_height and 0 <= col_idx < grid_width:
        code = corine_10m[row_idx, col_idx]
        # Skip NoData/Masked value (255)
        corine_codes.append(code if code != 255 else 0)
    else:
        corine_codes.append(0) # 0 represents NoData/outside study area

saocom_gdf['corine_code'] = corine_codes

# Filter out points outside the valid CORINE area (where code is 0)
saocom_lc_analysis = saocom_gdf[saocom_gdf['corine_code'] != 0].copy()

# ============================================================================
# 2. CALCULATE ROBUST STATISTICS BY LAND COVER CLASS
# ============================================================================

def nmad(data):
    """Normalized Median Absolute Deviation (robust measure of spread)"""
    return 1.4826 * np.median(np.abs(data - np.median(data)))

# Calculate stats for the recommended residual: SAOCOM (calibrated to TINITALY) - T
lc_height_stats = saocom_lc_analysis.groupby('corine_code')['diff_tinitaly'].agg([
    'count',
    'median',
    'mean',
    'std',
    nmad
]).reset_index()

# Rename columns for clarity
lc_height_stats.rename(columns={
    'count': 'N_Points',
    'median': 'Median_Diff_m',
    'mean': 'Mean_Diff_m',
    'std': 'Std_Dev_m',
    'nmad': 'NMAD_m'
}, inplace=True)

# Add land cover label for interpretability
lc_height_stats['LC_Label'] = lc_height_stats['corine_code'].map(CORINE_CLASSES)

# Reorder and filter for classes with enough samples (N > 50)
```

```
MIN_SAMPLES = 50
lc_height_stats_filtered = lc_height_stats[lc_height_stats['N_Points'] >= MIN_SAMPL
lc_height_stats_filtered = lc_height_stats_filtered.sort_values('LC_Label', ascendi

# ============================================================================
# 3. DISPLAY RESULTS
# ============================================================================

print("\n" + "="*100)
print(f"HEIGHT RESIDUAL STATISTICS by CORINE Land Cover (N > {MIN_SAMPLES})")
print("(Residual = Calibrated SAOCOM Height - TINITALY Reference DEM)")
print("="*100)

display_cols = ['corine_code', 'LC_Label', 'N_Points', 'Median_Diff_m', 'NMAD_m', '

# Print the filtered, robustly sorted table
print(lc_height_stats_filtered[display_cols].to_string(
    index=False,
    float_format=lambda x: f'{x:+.2f}' if 'Diff' in lc_height_stats_filtered.column
    formatters={'N_Points': '{:,}'.format,
                'Median_Diff_m': '{:+.2f} m'.format,
                'NMAD_m': '{:.2f} m'.format,
                'Mean_Diff_m': '{:+.2f} m'.format,
                'Std_Dev_m': '{:.2f} m'.format}
))

print("="*100)

# Store the filtered results for later use in plotting/reporting
lc_height_stats_final = lc_height_stats_filtered.copy()

# ============================================================================
# 1. SETUP AND DEFINITIONS (from previous cells)
# ============================================================================
# Global variables assumed defined:
# corine_10m (masked CLC raster), study_area_mask (valid area inside hull),
# saocom_coverage (boolean array: True where SAOCOM data exists),
# void_mask (study_area_mask & ~saocom_coverage), GRID_SIZE (10m)
# CORINE_CLASSES (LC code lookup)

# Recalculate global void stats for context
n_total_cells = np.sum(study_area_mask)
n_void_cells = np.sum(void_mask)
void_percentage_global = 100 * n_void_cells / n_total_cells if n_total_cells > 0 el

# Get unique, valid CORINE codes within the study area
unique_lc_codes = np.unique(corine_10m[study_area_mask])
unique_lc_codes = unique_lc_codes[unique_lc_codes != 0] # Filter out 0/NoData

# ============================================================================
# 2. VOID ANALYSIS BY LAND COVER CLASS
# ============================================================================
void_stats_by_lc = []
cell_area_km2 = (GRID_SIZE / 1000.0) ** 2 # 0.0001 km^2

for lc_code in unique_lc_codes:
```

```
    # Mask for this land cover class within the study area
    lc_mask = study_area_mask & (corine_10m == lc_code)

    # Total cells of this land cover
    total_lc_cells = np.sum(lc_mask)

    if total_lc_cells == 0:
        continue

    # Void cells within this land cover
    void_lc_cells = np.sum(lc_mask & void_mask)

    # METRIC 1: What % of this land cover is void? (Key Metric for coverage perform
    pct_of_lc_that_is_void = 100 * void_lc_cells / total_lc_cells

    # METRIC 2: What % of total voids is this land cover? (Key Metric for contribut
    pct_of_total_voids = 100 * void_lc_cells / n_void_cells if n_void_cells > 0 els

    void_stats_by_lc.append({
        'corine_code': lc_code,
        'label': CORINE_CLASSES.get(lc_code, f'Unknown_{lc_code}'),
        'total_cells': total_lc_cells,
        'void_cells': void_lc_cells,
        'Area_km2': total_lc_cells * cell_area_km2,
        'Pct_LC_is_Void': pct_of_lc_that_is_void,
        'Pct_of_Total_Voids': pct_of_total_voids,
    })

# Create DataFrame
void_stats_df = pd.DataFrame(void_stats_by_lc)

# ============================================================================
# 3. DISPLAY RESULTS
# ============================================================================
# Filter for significant land cover classes (e.g., > 1 km2 area)
MIN_AREA_KM2 = 1.0
void_stats_filtered = void_stats_df[void_stats_df['Area_km2'] >= MIN_AREA_KM2].copy

# Sort by the primary metric: Percentage of the LC class that is void
void_stats_filtered = void_stats_filtered.sort_values('Pct_LC_is_Void', ascending=F

print("\n" + "="*120)
print(f"VOID ANALYSIS by CORINE Land Cover (Area > {MIN_AREA_KM2:.1f} km²)")
print(f"Overall Void Percentage (Study Area): {void_percentage_global:.2f}%")
print("="*120)

display_cols = ['corine_code', 'label', 'Area_km2', 'void_cells', 'Pct_LC_is_Void',

# Print the filtered table
print(void_stats_filtered[display_cols].to_string(
    index=False,
    float_format=lambda x: f'{x:.2f}',
    formatters={'Area_km2': '{:.1f} km²'.format,
                'void_cells': '{:,}'.format,
                'Pct_LC_is_Void': '{:.2f} %'.format,
                'Pct_of_Total_Voids': '{:.2f} %'.format}
```

```
))

print("="*120)

# Store the filtered results for later use in plotting/reporting
lc_void_stats_final = void_stats_filtered.copy()
```

```
================================================================================
================
HEIGHT RESIDUAL STATISTICS by CORINE Land Cover (N > 50)
(Residual = Calibrated SAOCOM Height - TINITALY Reference DEM)
================================================================================
================
 corine_code                          LC_Label N_Points Median_Diff_m NMAD_m Mean_D
iff_m Std_Dev_m
         243 Agriculture/natural vegetation mix    9,796       -0.98 m    NaN
-0.70 m    4.68 m
         331              Beaches, dunes, sands    1,804       -1.03 m 1.34 m
-1.13 m    1.67 m
         311                Broad-leaved forest   12,631       +1.98 m    NaN
+2.30 m    5.85 m
         242       Complex cultivation patterns    2,807       -0.89 m 1.35 m
-0.62 m    2.38 m
         312                  Coniferous forest      978       +2.79 m 6.22 m
+2.70 m    6.53 m
         112         Discontinuous urban fabric    5,107       +0.55 m 1.59 m
+0.83 m    1.99 m
         313                       Mixed forest      263       +2.09 m 5.40 m
+2.05 m    5.38 m
         223                       Olive groves    1,552       -3.41 m 2.72 m
-2.91 m    3.61 m
         231                           Pastures    5,872       -0.82 m    NaN
-0.49 m    4.51 m
         221                          Vineyards   23,142       -0.12 m 1.92 m
+0.29 m    3.34 m
================================================================================
================

================================================================================
====================================
VOID ANALYSIS by CORINE Land Cover (Area > 1.0 km²)
Overall Void Percentage (Study Area): 87.04%
================================================================================
====================================
 corine_code                                label Area_km2 void_cells Pct_LC_is_Void P
ct_of_Total_Voids
         311                Broad-leaved forest 14.6 km²    132,547        90.84 %
30.74 %
         243 Agriculture/natural vegetation mix  9.6 km²     85,772        89.33 %
19.89 %
         223                       Olive groves  1.3 km²     11,300        87.29 %
2.62 %
         231                           Pastures  3.9 km²     33,421        85.25 %
7.75 %
         221                          Vineyards 14.2 km²    119,813        84.16 %
27.79 %
         242       Complex cultivation patterns  1.5 km²     12,342        81.95 %
2.86 %
         112         Discontinuous urban fabric  2.2 km²     17,299        77.93 %
4.01 %
================================================================================
====================================
```

1. PREPARE DATA FOR PLOTTING

In [16]:
```
# ============================================================================
# 1. PREPARE DATA FOR PLOTTING
# ============================================================================
# Use the full saocom_lc_analysis DataFrame which contains both the residuals
# ('diff_tinitaly') and the sampled land cover codes ('corine_code').

# Define major land cover groups for better visualization (CLC Level 1)
def get_clc_level1(code):
    """Maps CLC Level 3 code to Level 1 category"""
    if 100 <= code < 200: return '1. Artificial Surfaces'
    if 200 <= code < 300: return '2. Agricultural Areas'
    if 300 <= code < 400: return '3. Forest & Semi-Natural Areas'
    if 400 <= code < 500: return '4. Wetlands'
    if 500 <= code < 600: return '5. Water Bodies'
    return 'Other'

# Add Level 1 categories to the analysis DataFrame
saocom_lc_analysis['LC_Level_1'] = saocom_lc_analysis['corine_code'].apply(get_clc_
saocom_lc_analysis['LC_Label'] = saocom_lc_analysis['corine_code'].map(CORINE_CLASS

# Filter for the most common Level 3 classes (using the N_Points filter from Step 1
common_codes = lc_height_stats_final['corine_code'].unique()
plot_df_L3 = saocom_lc_analysis[saocom_lc_analysis['corine_code'].isin(common_codes

# Filter extreme outliers for better plot scaling (e.g., 99th percentile)
q_low = plot_df_L3['diff_tinitaly'].quantile(0.005)
q_high = plot_df_L3['diff_tinitaly'].quantile(0.995)
plot_df_L3_filtered = plot_df_L3[(plot_df_L3['diff_tinitaly'] >= q_low) &
                                 (plot_df_L3['diff_tinitaly'] <= q_high)]

# Sort the categories by the NMAD metric (best to worst performance)
nmad_order = lc_height_stats_final.sort_values('LC_Label', ascending=False)['LC_Lab
plot_df_L3_filtered['LC_Label'] = pd.Categorical(
    plot_df_L3_filtered['LC_Label'],
    categories=nmad_order,
    ordered=True
)

q_low = plot_df_L3['diff_copernicus'].quantile(0.005)
q_high = plot_df_L3['diff_copernicus'].quantile(0.995)
plot_df_cop_filtered = plot_df_L3[(plot_df_L3['diff_copernicus'] >= q_low) &
                                 (plot_df_L3['diff_copernicus'] <= q_high)]

# Sort the categories by the NMAD metric (best to worst performance)
nmad_order = lc_height_stats_final.sort_values('LC_Label', ascending=False)['LC_Lab
plot_df_cop_filtered['LC_Label'] = pd.Categorical(
    plot_df_cop_filtered['LC_Label'],
    categories=nmad_order,
    ordered=True
)
```

SENTINEL-2 RGB PREPARATION

In [17]:
```
# ============================================================================
# SENTINEL-2 RGB PREPARATION
# ============================================================================

# File discovery
sentinel_files = list((DATA_DIR / "sentinel_data").glob("*.tif"))
if not sentinel_files:
    raise FileNotFoundError("No Sentinel files found in sentinel_data directory")

# Load Sentinel bands (assuming separate R, G, B files or multi-band)
with rasterio.open(sentinel_files[0]) as src:
    sentinel_count = src.count

    if sentinel_count >= 3:
        # Multi-band file - read RGB bands
        sentinel_r = src.read(1)  # Band 1 (Red)
        sentinel_g = src.read(2)  # Band 2 (Green)
        sentinel_b = src.read(3)  # Band 3 (Blue)
        sentinel_transform_orig = src.transform
        sentinel_crs = src.crs
    else:
        # Single band files - need to find R, G, B separately
        r_file = next((f for f in sentinel_files if 'B04' in f.name or 'red' in f.n
        g_file = next((f for f in sentinel_files if 'B03' in f.name or 'green' in f
        b_file = next((f for f in sentinel_files if 'B02' in f.name or 'blue' in f.

        if not all([r_file, g_file, b_file]):
            raise FileNotFoundError("Could not find RGB bands in Sentinel files")

        with rasterio.open(r_file) as r_src:
            sentinel_r = r_src.read(1)
            sentinel_transform_orig = r_src.transform
            sentinel_crs = r_src.crs
        with rasterio.open(g_file) as g_src:
            sentinel_g = g_src.read(1)
        with rasterio.open(b_file) as b_src:
            sentinel_b = b_src.read(1)

# Resample each band to 10m grid
sentinel_r_10m = np.zeros((grid_height, grid_width), dtype=np.float32)
sentinel_g_10m = np.zeros((grid_height, grid_width), dtype=np.float32)
sentinel_b_10m = np.zeros((grid_height, grid_width), dtype=np.float32)

for band_src, band_dst in [(sentinel_r, sentinel_r_10m),
                           (sentinel_g, sentinel_g_10m),
                           (sentinel_b, sentinel_b_10m)]:
    reproject(
        source=band_src,
        destination=band_dst,
        src_transform=sentinel_transform_orig,
        src_crs=sentinel_crs,
        dst_transform=target_transform,
```

```
        dst_crs=TARGET_CRS,
        resampling=Resampling.bilinear
    )

# Stack into RGB array
sentinel_rgb = np.stack([sentinel_r_10m, sentinel_g_10m, sentinel_b_10m], axis=2)

# Mask to study area
sentinel_rgb[~hull_mask] = 0

# Normalize to 0-1 for display (using 2-98 percentile stretch for contrast)
sentinel_rgb_norm = np.zeros_like(sentinel_rgb, dtype=np.float32)
for i in range(3):
    band = sentinel_rgb[:, :, i]
    valid_pixels = band[hull_mask]

    if len(valid_pixels) > 0:
        p2, p98 = np.percentile(valid_pixels[valid_pixels > 0], [2, 98])
        band_norm = np.clip((band - p2) / (p98 - p2), 0, 1)
        sentinel_rgb_norm[:, :, i] = band_norm

print(f"\nSentinel-2 RGB Prepared:")
print(f"  Shape: {sentinel_rgb_norm.shape}")
print(f"  Resolution: {GRID_SIZE}m")
print(f"  Value range: [{sentinel_rgb_norm.min():.3f}, {sentinel_rgb_norm.max():.3f
```

```
Sentinel-2 RGB Prepared:
  Shape: (777, 929, 3)
  Resolution: 10m
  Value range: [0.000, 1.000]
```

2. GENERATE VIOLIN PLOT (Level 3 - Detailed Performance)

In [18]:

```
# ================================================================================
# 1. GENERATE STATS WITHIN THE PLOT SCRIPT
# ================================================================================
print("Generating statistics just-in-time for the plot...")

def calculate_nmad(series):
    """Calculates the Normalized Median Absolute Deviation (NMAD)."""
    return (series - series.median()).abs().median() * 1.4826

# Create a new, temporary stats DataFrame by grouping the plotting data
stats_for_plot = plot_df_L3_filtered.groupby('LC_Label')['diff_tinitaly'].agg(
    Median_Diff_m='median',
    NMAD_m=calculate_nmad,
    Std_Dev_m='std',
    Min_Diff_m='min',
    Max_Diff_m='max'
).reset_index()

# Determine the plotting order based on the NMAD we just calculated
nmad_order = stats_for_plot.sort_values('NMAD_m')['LC_Label'].tolist()

# Define an anchor point for the text annotations
```

```
q_high = plot_df_L3_filtered['diff_tinitaly'].quantile(0.99) + 5


# ==============================================================================
# 2. GENERATE VIOLIN PLOT (Level 3 - Detailed Performance)
# ==============================================================================
plt.figure(figsize=(16, 9), facecolor='white')

# Use violin plot, ordering it with the 'nmad_order' list we just created
sns.violinplot(
    x='diff_tinitaly',
    y='LC_Label',
    data=plot_df_L3_filtered,
    order=nmad_order,  # Use the calculated order here
    inner='quartile',
    palette='Spectral_r',
    orient='h',
    linewidth=1.0,
    cut=0
)

# Add a vertical line at zero error
plt.axvline(0, color='red', linestyle='--', linewidth=1.5, alpha=0.7)

# Set labels and title
plt.title('Distribution of SAOCOM Height Residuals by Land Cover (CLC Level 3)', fo
plt.xlabel('Height Residual (Calibrated SAOCOM - TINITALY DEM) [m]', fontsize=14)
plt.ylabel('CORINE Land Cover Class (Ordered by NMAD)', fontsize=14)
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# --- Add Annotations using the stats we just generated ---
for i, label in enumerate(nmad_order):
    # Query the 'stats_for_plot' DataFrame we created above
    stats = stats_for_plot.query("LC_Label == @label").iloc[0]

    stats_text = (
        f"Median: {stats['Median_Diff_m']:+.2f} m\n"
        f"NMAD: {stats['NMAD_m']:.2f} m\n"
        f"Std Dev: {stats['Std_Dev_m']:.2f} m\n"
        f"Min/Max: [{stats['Min_Diff_m']:.1f}, {stats['Max_Diff_m']:.1f}] m"
    )

    plt.text(q_high, i, stats_text,
             verticalalignment='center',
             horizontalalignment='left',
             fontsize=10,
             bbox=dict(boxstyle='round,pad=0.4', facecolor='white', alpha=0.8)
            )

plt.tight_layout(rect=[0, 0, 0.8, 1])
plt.show()
print("Violin Plot for Level 3 Land Cover classes generated successfully.")
# Add this line to save the figure
plt.savefig(RESULTS_DIR / 'saocom_tinitaly_residuals_by_landcover.png', dpi=300, bb
```

```
plt.show()

# ============================================================================
# COPERNICUS STATISTICS AND VIOLIN PLOT (Level 3)
# ============================================================================

print("Generating statistics for Copernicus plot...")

def calculate_nmad(series):
    """Calculates the Normalized Median Absolute Deviation (NMAD)."""
    return (series - series.median()).abs().median() * 1.4826

# Create a new stats DataFrame by grouping the filtered Copernicus plotting data
stats_for_cop_plot = plot_df_cop_filtered.groupby('LC_Label')['diff_copernicus'].ag
    Median_Diff_m='median',
    NMAD_m=calculate_nmad,
    Std_Dev_m='std',
    Min_Diff_m='min',
    Max_Diff_m='max'
).reset_index()

# Determine the plotting order based on the NMAD we just calculated
nmad_order_cop = stats_for_cop_plot.sort_values('NMAD_m')['LC_Label'].tolist()

# Define an anchor point for the text annotations
q_high_cop = plot_df_cop_filtered['diff_copernicus'].quantile(0.995) + 5


# ============================================================================
# 2. GENERATE COPERNICUS VIOLIN PLOT (Level 3)
# ============================================================================
plt.figure(figsize=(16, 9), facecolor='white')

# Generate the violin plot, ordering it with the 'nmad_order_cop' list
sns.violinplot(
    x='diff_copernicus',
    y='LC_Label',
    data=plot_df_cop_filtered,
    order=nmad_order_cop,  # Use the calculated order here
    inner='quartile',
    palette='Spectral_r',
    orient='h',
    linewidth=1.0,
    cut=0
)

# Add a vertical line at zero error
plt.axvline(0, color='red', linestyle='--', linewidth=1.5, alpha=0.7)

# Set labels and title
plt.title('Distribution of SAOCOM Height Residuals by Land Cover (vs. Copernicus)',
plt.xlabel('Height Residual (Calibrated SAOCOM - Copernicus DEM) [m]', fontsize=14)
plt.ylabel('CORINE Land Cover Class (Ordered by NMAD)', fontsize=14)
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.xticks(fontsize=12)
```

```
plt.yticks(fontsize=12)

# --- Add Annotations using the stats we just generated ---
for i, label in enumerate(nmad_order_cop):
    # Query the 'stats_for_cop_plot' DataFrame
    stats = stats_for_cop_plot.query("LC_Label == @label").iloc[0]

    stats_text = (
        f"Median: {stats['Median_Diff_m']:+.2f} m\n"
        f"NMAD: {stats['NMAD_m']:.2f} m\n"
        f"Std Dev: {stats['Std_Dev_m']:.2f} m\n"
        f"Min/Max: [{stats['Min_Diff_m']:.1f}, {stats['Max_Diff_m']:.1f}] m"
    )

    plt.text(q_high_cop, i, stats_text,
             verticalalignment='center',
             horizontalalignment='left',
             fontsize=10,
             bbox=dict(boxstyle='round,pad=0.4', facecolor='white', alpha=0.8)
            )

plt.tight_layout(rect=[0, 0, 0.8, 1])
plt.show()
print("Violin Plot for Copernicus comparison generated successfully.")
# Add this line to save the figure
plt.savefig(RESULTS_DIR / 'saocom_copernicus_residuals_by_landcover.png', dpi=300,

plt.show()
# ============================================================================
# 3. GENERATE BOX PLOT (Level 1 - Broad Comparison)
# ============================================================================
plt.figure(figsize=(10, 6), facecolor='white')

# Use box plot for a cleaner Level 1 aggregation
sns.boxplot(
    x='LC_Level_1',
    y='diff_tinitaly',
    data=plot_df_L3_filtered,
    palette='Set2',
    linewidth=1.0,
    showfliers=False # Do not show outliers already filtered
)

# Add a horizontal line at zero error
plt.axhline(0, color='red', linestyle='--', linewidth=1.5, alpha=0.7)

# Set labels and title
plt.title(
    'SAOCOM Height Residuals by Land Cover (CLC Level 1)',
    fontweight='bold',
    fontsize=14
)
plt.xlabel('CORINE Land Cover Category (Level 1)', fontsize=11)
plt.ylabel('Height Residual (m)', fontsize=11)
plt.xticks(rotation=15, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.5)
```

```
plt.tight_layout()
plt.show()
print("Box Plot for Level 1 Land Cover categories generated successfully.")

# ============================================================================
# CORINE LAND COVER VISUALIZATION
# ============================================================================
fig, ax = plt.subplots(figsize=(14, 10), facecolor='white')
ax.set_facecolor('white')

# Get extent
extent = [xmin_grid, xmax_grid, ymin_grid, ymax_grid]

# Mask NoData/zero values for transparency
corine_display = np.ma.masked_where((corine_10m == 0) | (corine_10m == 255), corine

# Get unique classes in study area
unique_codes = np.unique(corine_display.compressed())

# Create colormap for present classes only
colors_list = [CORINE_COLORS_MPL.get(code, (0.5, 0.5, 0.5)) for code in unique_code
cmap = ListedColormap(colors_list)
norm = BoundaryNorm(boundaries=np.append(unique_codes, unique_codes[-1]+1) - 0.5,
                    ncolors=len(unique_codes))

# Plot CORINE
im = ax.imshow(corine_display, cmap=cmap, norm=norm, origin='upper', extent=extent)

# Add study area boundary
hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2, label='Study Area')

# Labels and title
ax.set_xlabel('UTM Easting (m)', fontsize=14, color='black')
ax.set_ylabel('UTM Northing (m)', fontsize=14, color='black')
ax.set_title('CORINE Land Cover 2018', fontweight='bold', fontsize=18, color='black
ax.tick_params(colors='black', labelsize=12)
ax.grid(True, alpha=0.3, linewidth=0.5, color='black')

# Create legend with only present classes
legend_elements = [plt.Rectangle((0,0),1,1, facecolor=CORINE_COLORS_MPL[code],
                                 edgecolor='black', linewidth=0.5,
                                 label=f"{code}: {CORINE_CLASSES.get(code, 'Unknown
                   for code in sorted(unique_codes)]

ax.legend(handles=legend_elements, loc='center left', bbox_to_anchor=(1, 0.5),
          fontsize=13, frameon=True, fancybox=False, edgecolor='black')

# Add scale bar
scalebar = ScaleBar(1, location='lower right', box_alpha=0.8, color='black')
ax.add_artist(scalebar)

# Add statistics box
total_area_km2 = np.sum(study_area_mask) * 0.0001
stats_text = f"Study Area: {total_area_km2:.2f} km²\nClasses: {len(unique_codes)}"
ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=12,
        verticalalignment='top', bbox=dict(boxstyle='round', facecolor='white',
```

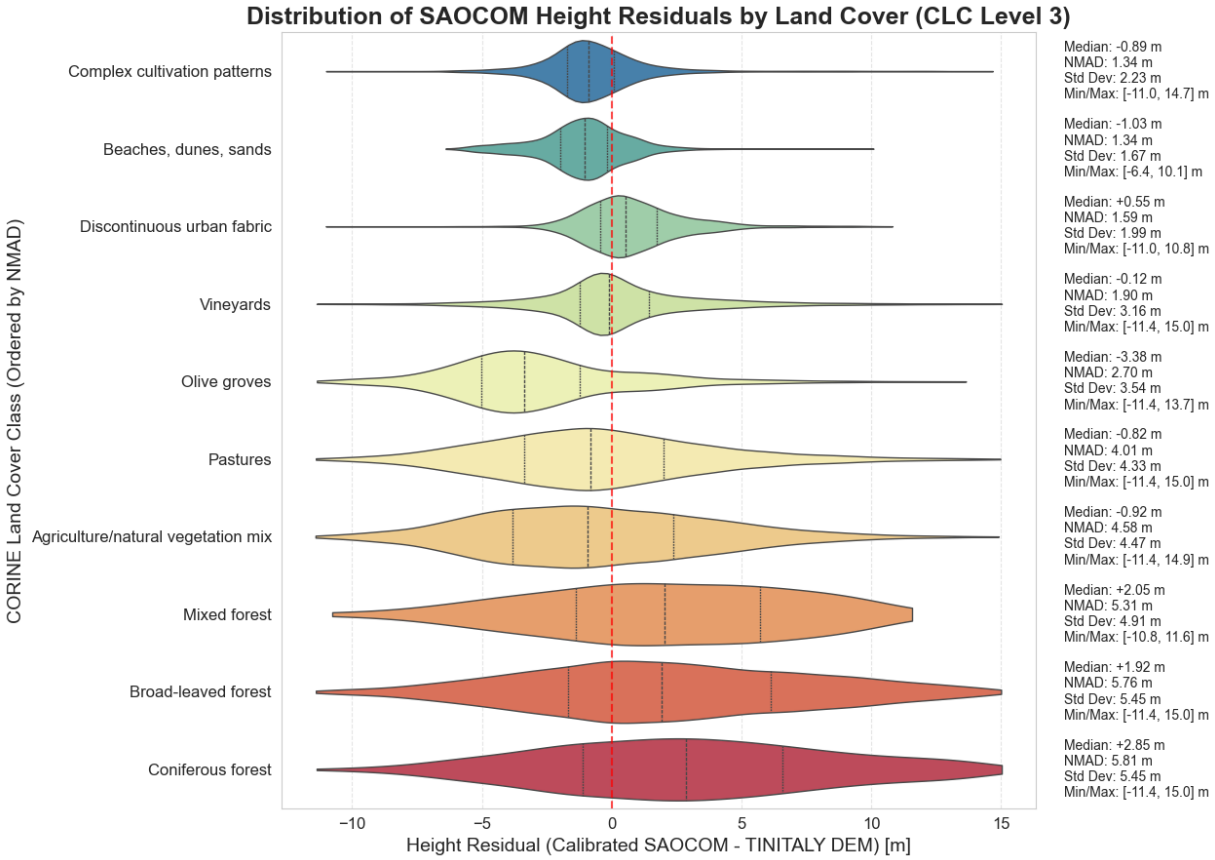
```
        alpha=0.9, edgecolor='black'))

plt.tight_layout()
plt.show()

print(f"\nCORINE Land Cover Map:")
print(f"  Total classes present: {len(unique_codes)}")
print(f"  Study area: {total_area_km2:.2f} km²")
## ============================================================================
# SAOCOM HEIGHT RESIDUALS - INTERPOLATED HEAT MAPS
# ============================================================================
from scipy.interpolate import griddata
from scipy.ndimage import gaussian_filter

fig, axes = plt.subplots(1, 2, figsize=(22, 10), facecolor='white')

extent = [xmin_grid, xmax_grid, ymin_grid, ymax_grid]

# Filter valid points
valid_tin = saocom_gdf[saocom_gdf['diff_tinitaly'].notna()].copy()
valid_cop = saocom_gdf[saocom_gdf['diff_copernicus'].notna()].copy()

# Calculate symmetric color limits (95th percentile)
tin_limit = np.percentile(np.abs(valid_tin['diff_tinitaly']), 95)
cop_limit = np.percentile(np.abs(valid_cop['diff_copernicus']), 95)
common_limit = max(tin_limit, cop_limit)

# Create interpolation grid (matching the 10m grid)
xi = np.linspace(xmin_grid, xmax_grid, grid_width)
yi = np.linspace(ymax_grid, ymin_grid, grid_height)
xi_grid, yi_grid = np.meshgrid(xi, yi)

# ============================================================================
# Plot 1: SAOCOM - TINITALY Heat Map
# ============================================================================
ax = axes[0]
ax.set_facecolor('white')

# Background: Sentinel RGB (very faint)
ax.imshow(sentinel_rgb_norm, extent=extent, origin='upper', alpha=0.2)

# Extract coordinates and values
x_tin = valid_tin.geometry.x.values
y_tin = valid_tin.geometry.y.values
z_tin = valid_tin['diff_tinitaly'].values

# Interpolate to grid using cubic method
print("Interpolating TINITALY differences...")
zi_tin = griddata((x_tin, y_tin), z_tin, (xi_grid, yi_grid),
                  method='cubic', fill_value=np.nan)

# Apply Gaussian smoothing for smoother heat map
zi_tin_smooth = gaussian_filter(np.nan_to_num(zi_tin, nan=0), sigma=2)
zi_tin_smooth[~hull_mask] = np.nan  # Mask to study area

# Plot heat map
```

```
im1 = ax.imshow(zi_tin_smooth, extent=extent, origin='upper',
                cmap='RdBu_r', alpha=0.8,
                vmin=-common_limit, vmax=common_limit,
                interpolation='bilinear')

# Overlay original points (small, for reference)
ax.scatter(x_tin, y_tin, c=z_tin, cmap='RdBu_r',
           s=0.5, alpha=0.3, edgecolors='none',
           vmin=-common_limit, vmax=common_limit)

# Study area boundary
hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--')

# Colorbar
cbar1 = plt.colorbar(im1, ax=ax, label='Height Difference (m)',
                     shrink=0.8, pad=0.02)
cbar1.ax.tick_params(labelsize=10, colors='black')
cbar1.ax.yaxis.label.set_color('black')

ax.set_xlabel('UTM Easting (m)', fontsize=12, color='black')
ax.set_ylabel('UTM Northing (m)', fontsize=12, color='black')
ax.set_title('SAOCOM - TINITALY\nInterpolated Height Residual Heat Map',
             fontweight='bold', fontsize=14, color='black')
ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
ax.tick_params(colors='black')

# Statistics box
stats_text = f"""Points: {len(valid_tin):,}
Mean: {valid_tin['diff_tinitaly'].mean():+.2f} m
Median: {valid_tin['diff_tinitaly'].median():+.2f} m
RMSE: {np.sqrt((valid_tin['diff_tinitaly']**2).mean()):.2f} m
NMAD: {1.4826 * np.median(np.abs(valid_tin['diff_tinitaly'] - valid_tin['diff_tinit
Std: {valid_tin['diff_tinitaly'].std():.2f} m

Interpolation: Cubic + Gaussian
Color Scale: ±{common_limit:.1f} m
Red = SAOCOM Higher
Blue = TINITALY Higher"""

ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=9,
        verticalalignment='top', fontfamily='monospace',
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9,
                  edgecolor='black'))

# ============================================================================
# Plot 2: SAOCOM - Copernicus Heat Map
# ============================================================================
ax = axes[1]
ax.set_facecolor('white')

# Background: Sentinel RGB (very faint)
ax.imshow(sentinel_rgb_norm, extent=extent, origin='upper', alpha=0.2)

# Extract coordinates and values
x_cop = valid_cop.geometry.x.values
y_cop = valid_cop.geometry.y.values
```

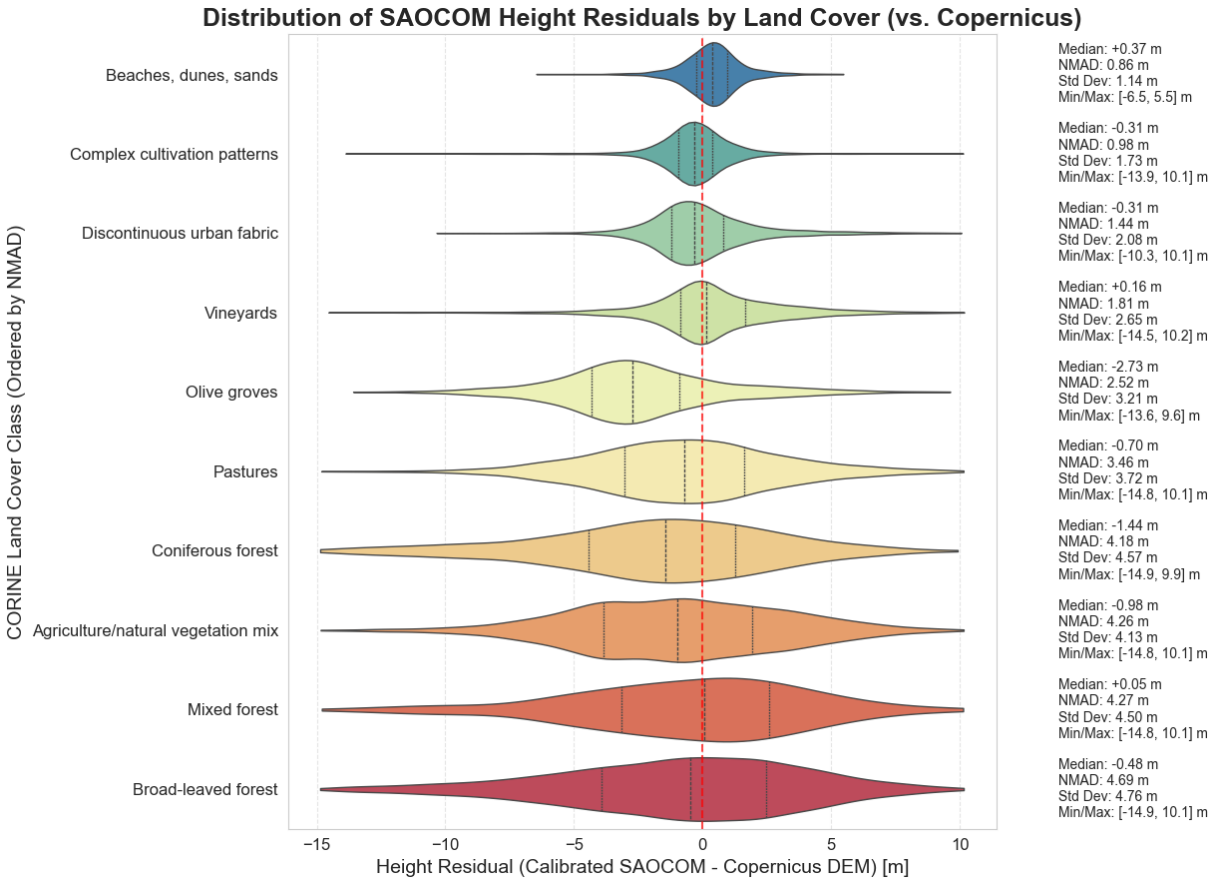
```
z_cop = valid_cop['diff_copernicus'].values

# Interpolate to grid using cubic method
print("Interpolating Copernicus differences...")
zi_cop = griddata((x_cop, y_cop), z_cop, (xi_grid, yi_grid),
                  method='cubic', fill_value=np.nan)

# Apply Gaussian smoothing
zi_cop_smooth = gaussian_filter(np.nan_to_num(zi_cop, nan=0), sigma=2)
zi_cop_smooth[~hull_mask] = np.nan  # Mask to study area

# Plot heat map
im2 = ax.imshow(zi_cop_smooth, extent=extent, origin='upper',
                cmap='RdBu_r', alpha=0.8,
                vmin=-common_limit, vmax=common_limit,
                interpolation='bilinear')

# Overlay original points (small, for reference)
ax.scatter(x_cop, y_cop, c=z_cop, cmap='RdBu_r',
           s=0.5, alpha=0.3, edgecolors='none',
           vmin=-common_limit, vmax=common_limit)

# Study area boundary
hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--')

# Colorbar
cbar2 = plt.colorbar(im2, ax=ax, label='Height Difference (m)',
                     shrink=0.8, pad=0.02)
cbar2.ax.tick_params(labelsize=10, colors='black')
cbar2.ax.yaxis.label.set_color('black')

ax.set_xlabel('UTM Easting (m)', fontsize=12, color='black')
ax.set_ylabel('UTM Northing (m)', fontsize=12, color='black')
ax.set_title('SAOCOM - Copernicus\nInterpolated Height Residual Heat Map',
             fontweight='bold', fontsize=14, color='black')
ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
ax.tick_params(colors='black')

# Statistics box
stats_text = f"""Points: {len(valid_cop):,}
Mean: {valid_cop['diff_copernicus'].mean():+.2f} m
Median: {valid_cop['diff_copernicus'].median():+.2f} m
RMSE: {np.sqrt((valid_cop['diff_copernicus']**2).mean()):.2f} m
NMAD: {1.4826 * np.median(np.abs(valid_cop['diff_copernicus'] - valid_cop['diff_cop
Std: {valid_cop['diff_copernicus'].std():.2f} m

Interpolation: Cubic + Gaussian
Color Scale: ±{common_limit:.1f} m
Red = SAOCOM Higher
Blue = Copernicus Higher"""

ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=9,
        verticalalignment='top', fontfamily='monospace',
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9,
                  edgecolor='black'))
```

```
plt.tight_layout()
plt.savefig(RESULTS_DIR / 'saocom_residual_heatmaps_interpolated.png',
            dpi=300, bbox_inches='tight', facecolor='white')
plt.show()

print("\nSaved: saocom_residual_heatmaps_interpolated.png")
print(f"Interpolation method: Cubic spline + Gaussian smoothing (sigma=2)")
print(f"Color scale range: ±{common_limit:.2f} m")
```

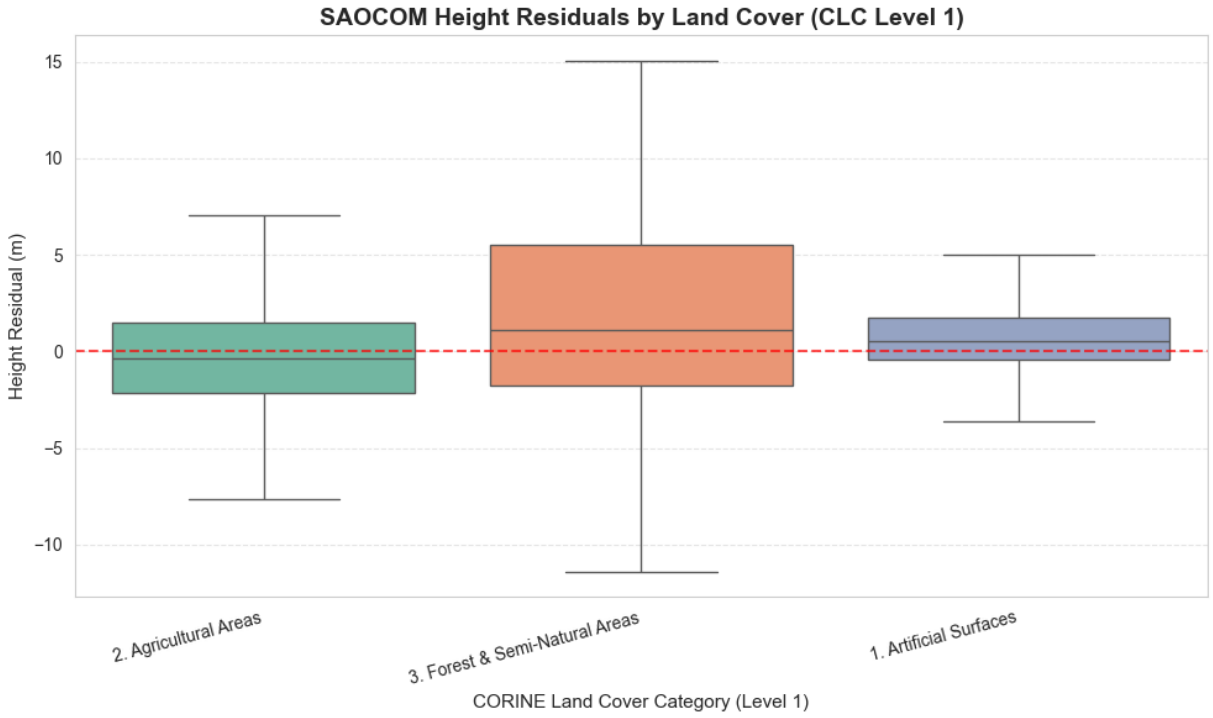
```
Generating statistics just-in-time for the plot...
```



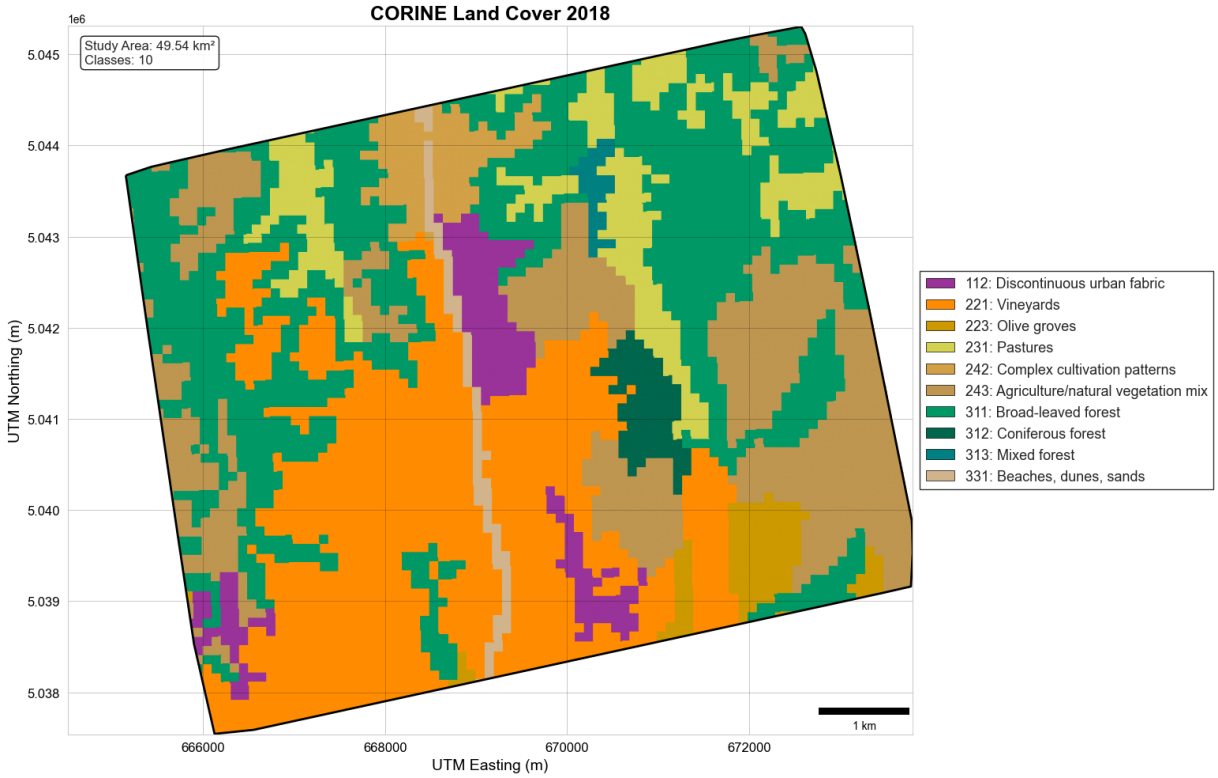

```
Violin Plot for Level 3 Land Cover classes generated successfully.
<Figure size 640x480 with 0 Axes>
Generating statistics for Copernicus plot...
```



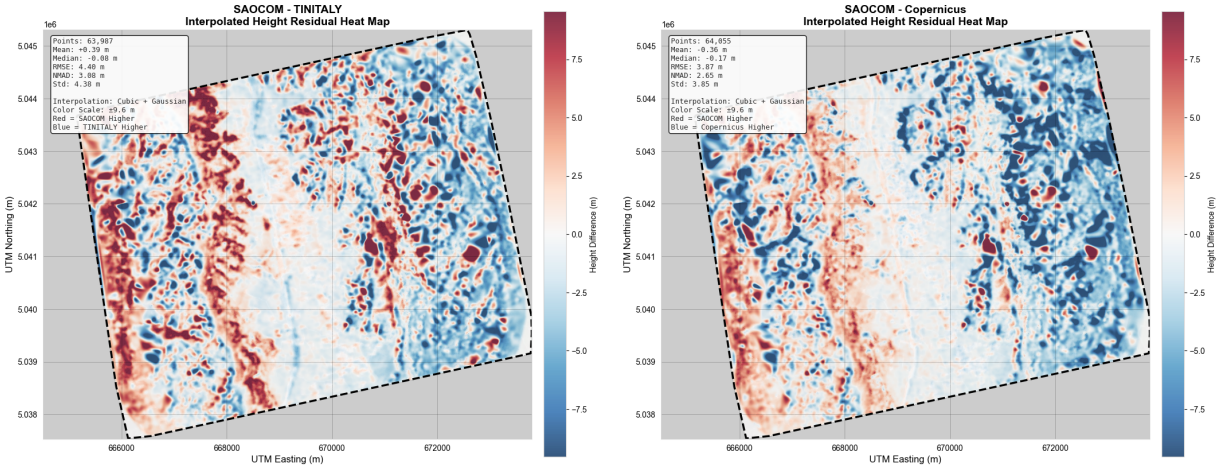

```
Violin Plot for Copernicus comparison generated successfully.
<Figure size 640x480 with 0 Axes>
```




```
Box Plot for Level 1 Land Cover categories generated successfully.
```




```
CORINE Land Cover Map:
  Total classes present: 10
  Study area: 49.54 km²
Interpolating TINITALY differences...
Interpolating Copernicus differences...
```




```
Saved: saocom_residual_heatmaps_interpolated.png
Interpolation method: Cubic spline + Gaussian smoothing (sigma=2)
Color scale range: ±9.55 m
```

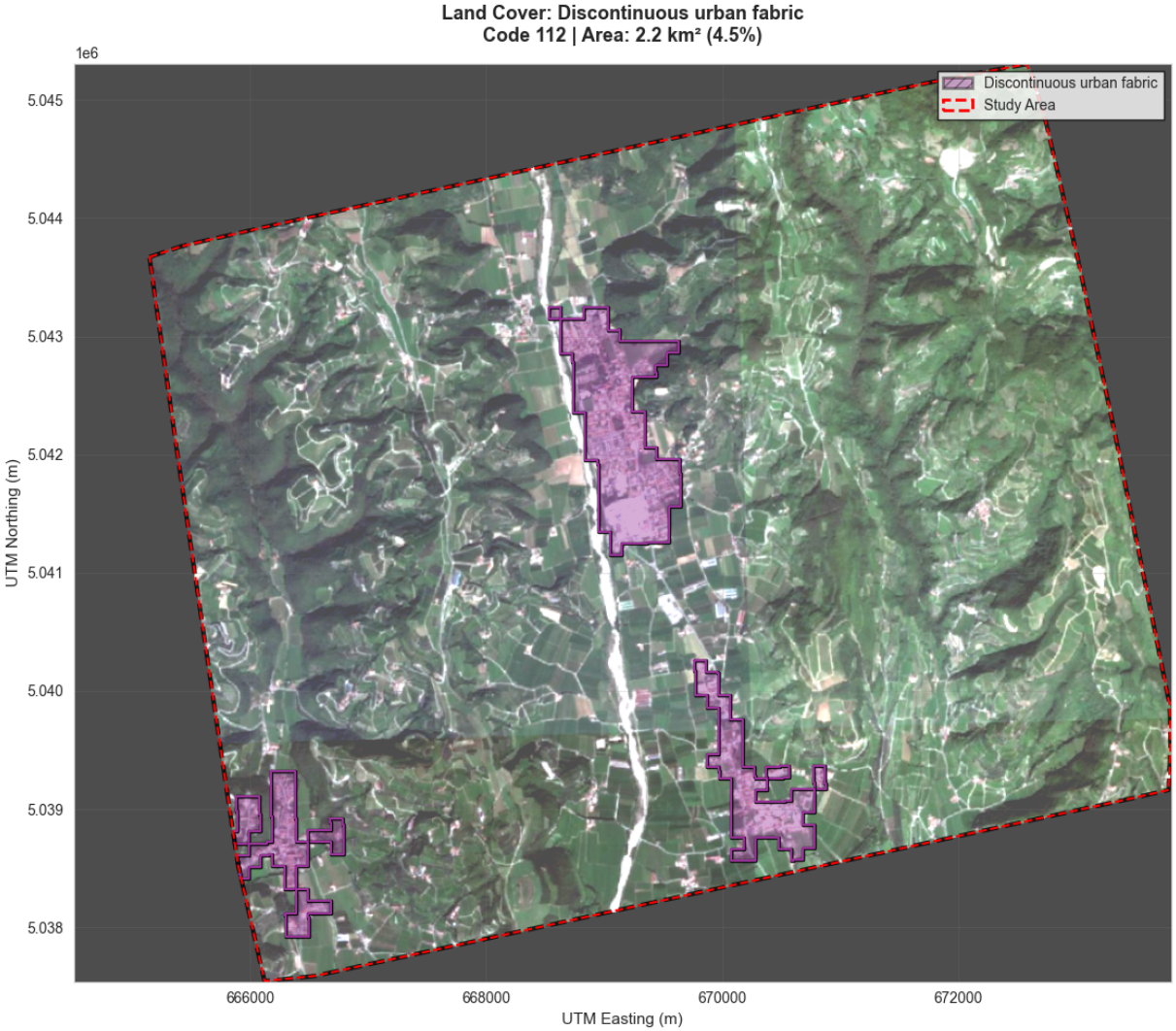
Class Overlays Basic

INDIVIDUAL CLASS OVERLAY MAPS (COLORBLIND-FRIENDLY)

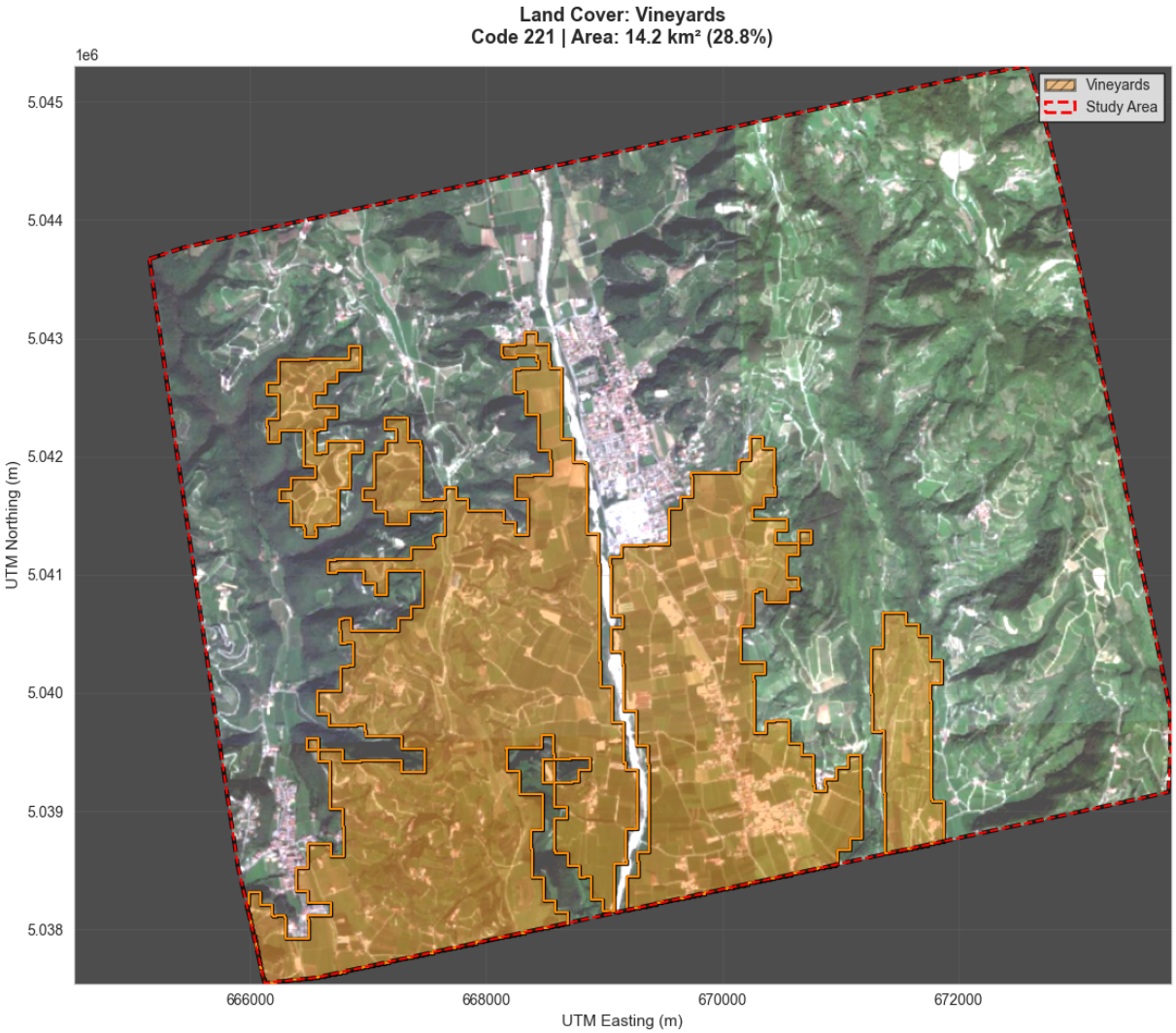
In [19]:
```
# ============================================================================
# INDIVIDUAL CLASS OVERLAY MAPS (COLORBLIND-FRIENDLY)
# ============================================================================
from matplotlib.patches import Patch
```

```
# Get unique classes present in data
unique_classes = np.unique(corine_10m[corine_10m > 0])

# Create one map per class
for lc_code in sorted(unique_classes):
    fig, ax = plt.subplots(1, 1, figsize=(14, 10), facecolor='white')
    ax.set_facecolor('white')

    # Display Sentinel RGB as background
    ax.imshow(sentinel_rgb_norm, extent=[xmin_grid, xmax_grid, ymin_grid, ymax_grid
              origin='upper', alpha=0.7)  # Slight transparency to help overlay sho

    # Get color for this land cover class
    fill_color = tuple(c/255 for c in CORINE_COLORS.get(lc_code, (128, 128, 128)))

    # Create mask for this land cover class
    lc_mask = (corine_10m == lc_code)

    # Vectorize to get boundaries
    mask_shapes = shapes(lc_mask.astype(np.uint8), mask=lc_mask, transform=target_t

    # Convert to polygons and plot
    polys = [shape(geom) for geom, val in mask_shapes if val == 1]

    if polys:
        for poly in polys:
            if poly.is_valid:
                x, y = poly.exterior.xy

                # Fill with class-specific color + hatching for visibility
                ax.fill(x, y, color=fill_color, alpha=0.4,
                        edgecolor='none', hatch='///', linewidth=0)

                # Bold black outline for definition
                ax.plot(x, y, color='black', linewidth=2.5, alpha=0.9)

                # Colored inner outline
                ax.plot(x, y, color=fill_color, linewidth=1.5, alpha=1.0)

    # Add study area boundary
    hull_gdf.boundary.plot(ax=ax, color='black', linewidth=3, linestyle='--', alpha
    hull_gdf.boundary.plot(ax=ax, color='red', linewidth=1.5, linestyle='--', alpha

    # Calculate statistics
    lc_count = np.sum(lc_mask)
    area_km2 = lc_count * (GRID_SIZE**2) / 1e6
    pct_area = 100 * lc_count / np.sum(corine_10m > 0)

    # Title with statistics
    class_name = CORINE_CLASSES.get(lc_code, f'Class {lc_code}')
    ax.set_title(f'Land Cover: {class_name}\n'
                 f'Code {lc_code} | Area: {area_km2:.1f} km² ({pct_area:.1f}%)',
                 fontweight='bold', fontsize=13, pad=15)

    ax.set_xlabel('UTM Easting (m)', fontsize=11)
```

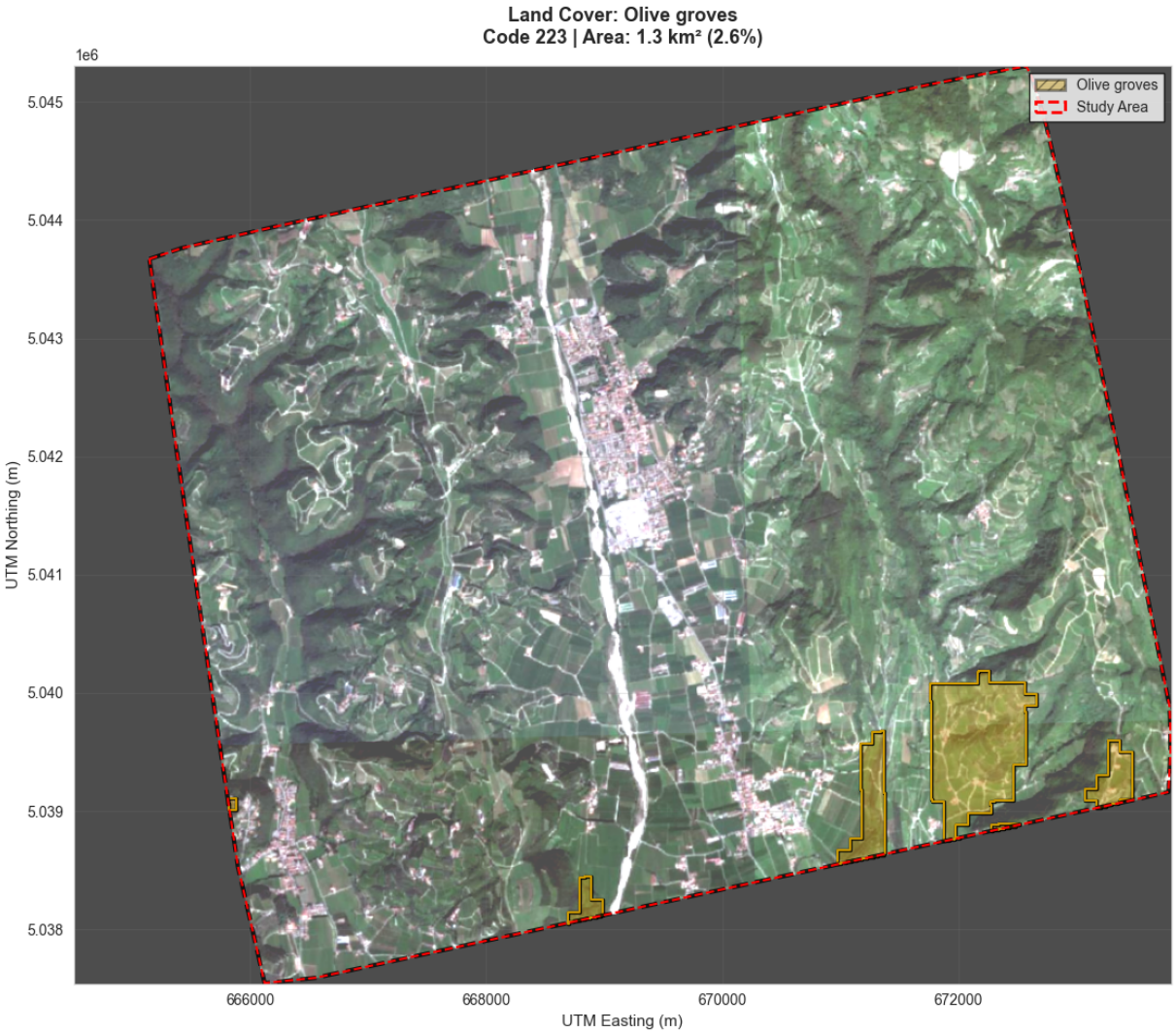
```
    ax.set_ylabel('UTM Northing (m)', fontsize=11)
    ax.grid(True, alpha=0.3, color='gray', linewidth=0.5)

    # Legend with hatching
    legend_elements = [
        Patch(facecolor=fill_color, edgecolor='black', linewidth=2,
              alpha=0.4, hatch='///', label=class_name),
        Patch(facecolor='none', edgecolor='red', linestyle='--',
              linewidth=2, label='Study Area')
    ]
    ax.legend(handles=legend_elements, loc='upper right', fontsize=10,
              frameon=True, fancybox=False, edgecolor='black')

    plt.tight_layout()

    # Save
    safe_name = class_name.replace(' ', '_').replace(',', '').replace('/', '_')
    filename = f'landcover_{lc_code}_{safe_name}.png'
    plt.savefig(RESULTS_DIR / filename, dpi=300, bbox_inches='tight', facecolor='wh
    plt.show()
    plt.close()

    print(f"Saved: {filename}")

print(f"\nGenerated {len(unique_classes)} individual land cover overlay maps")
```



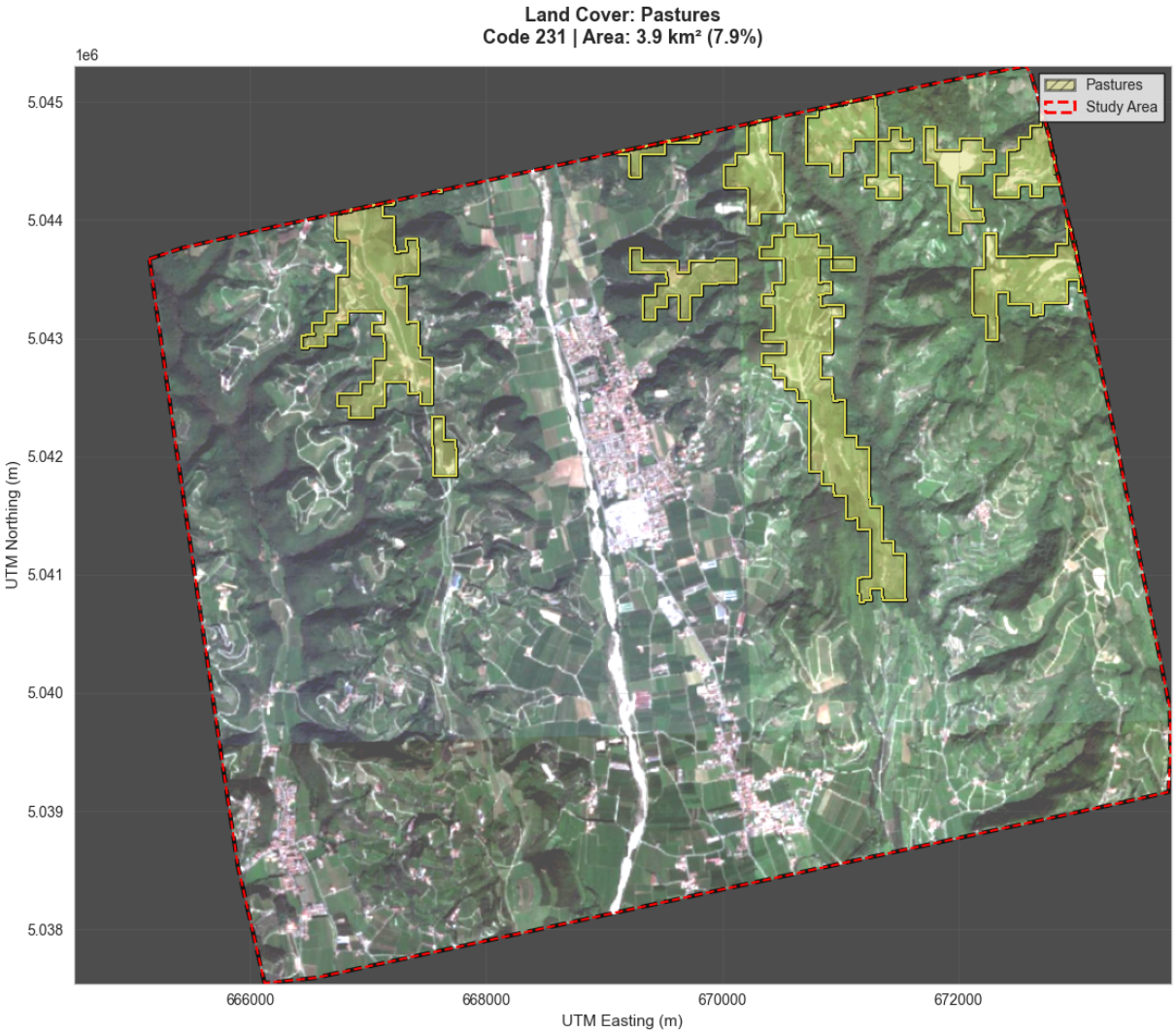

```
Saved: landcover_112_Discontinuous_urban_fabric.png
```



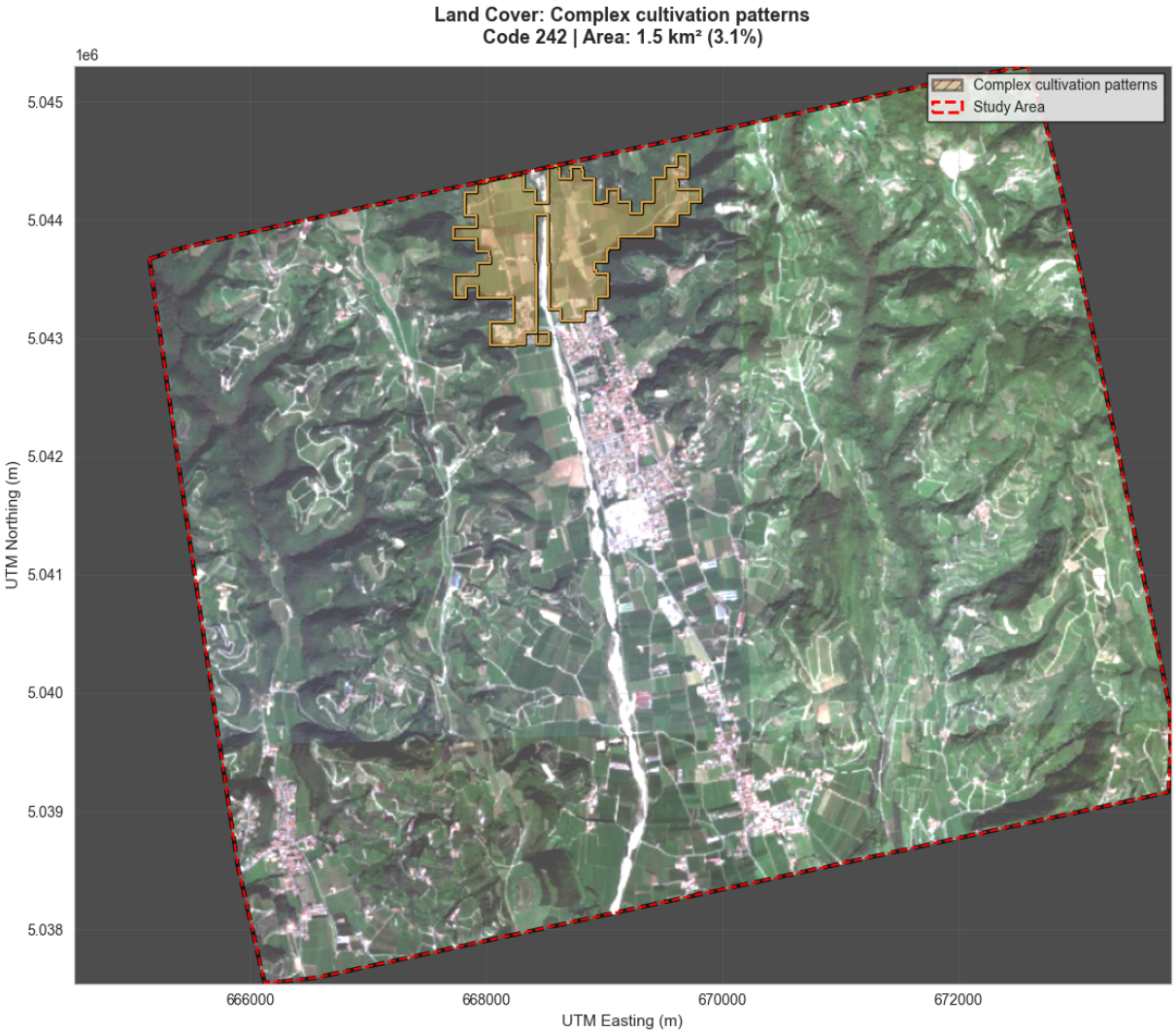

```
Saved: landcover_221_Vineyards.png
```



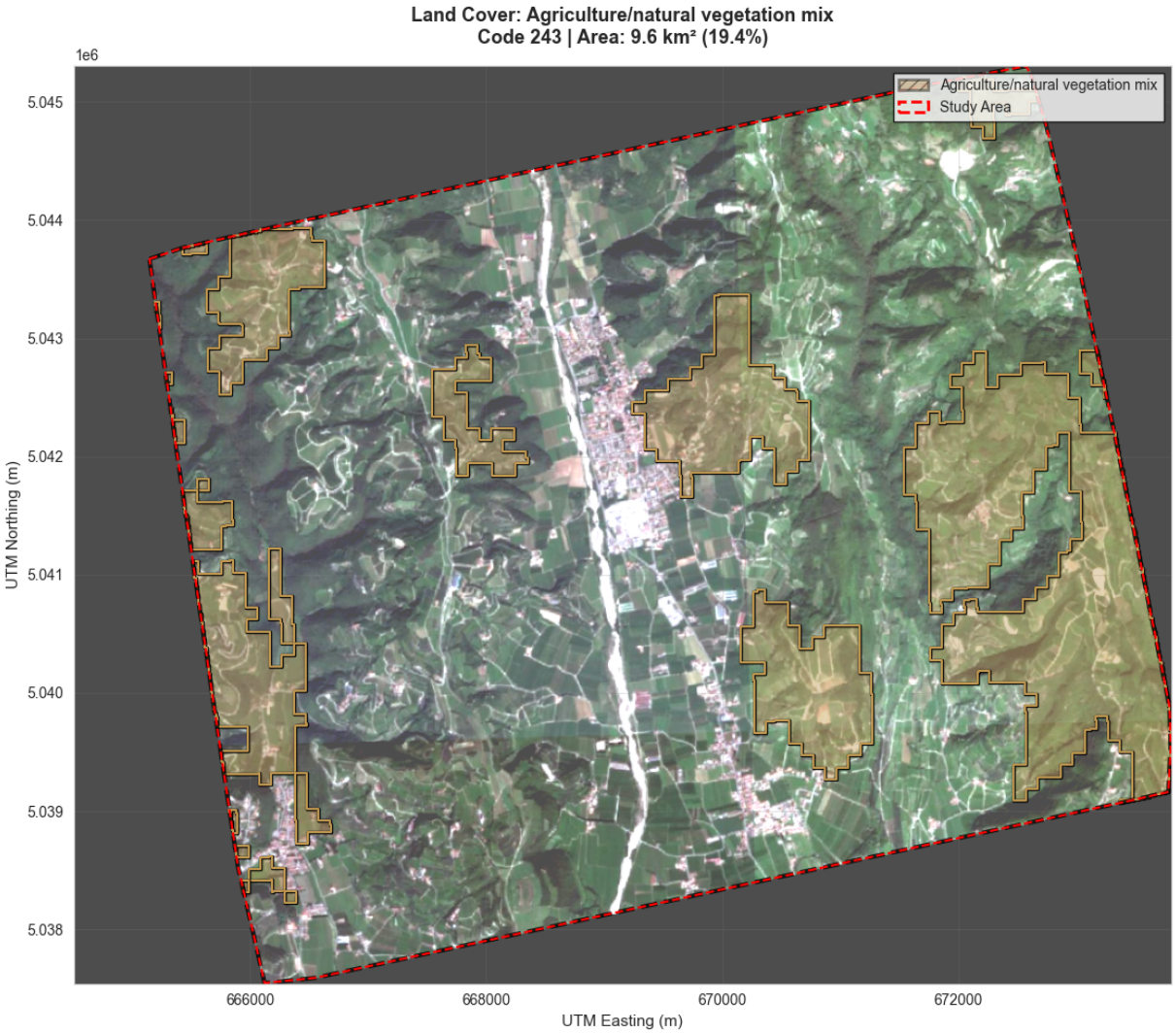

```
Saved: landcover_223_Olive_groves.png
```



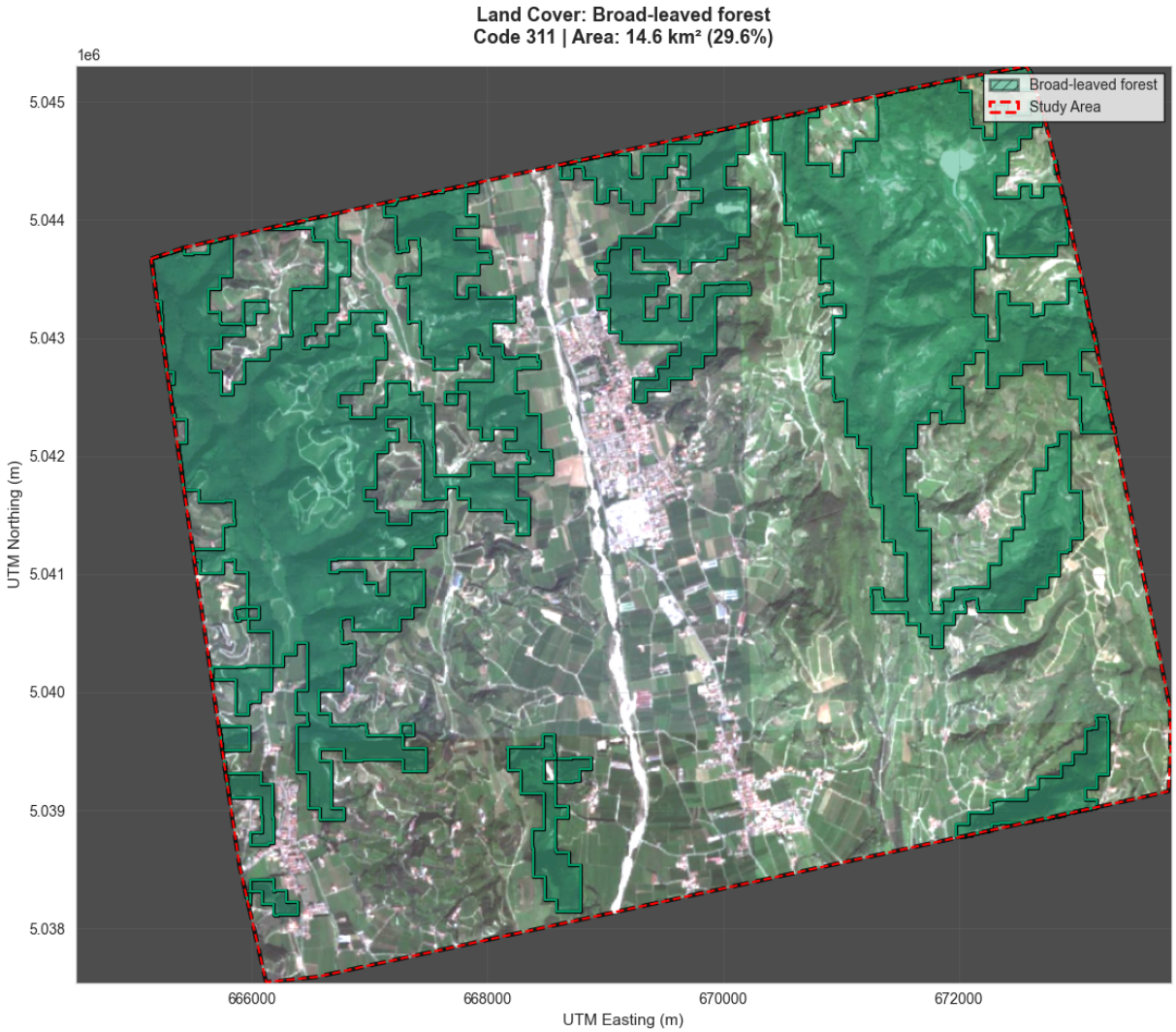

```
Saved: landcover_231_Pastures.png
```



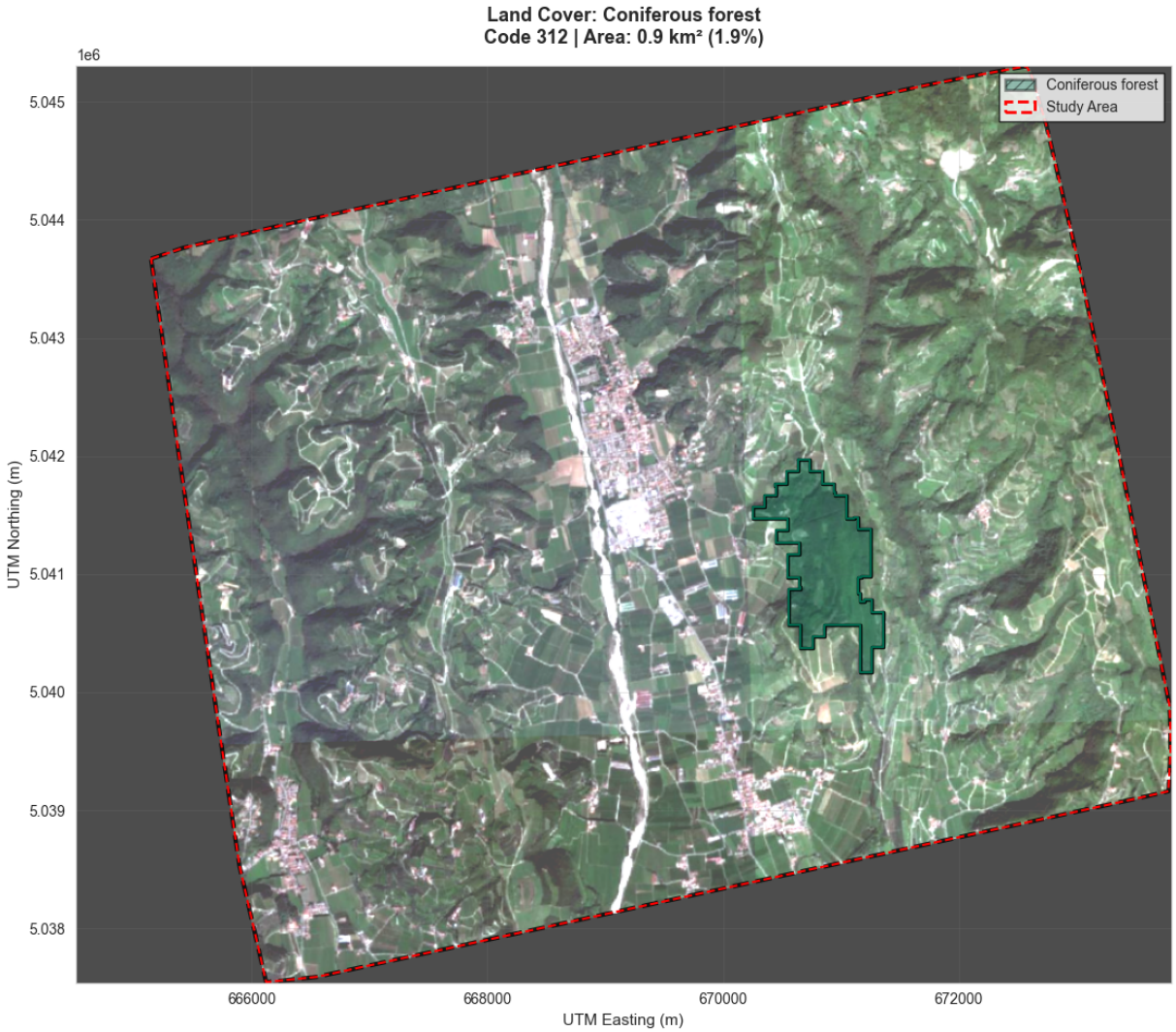
Saved: landcover_242_Complex_cultivation_patterns.png



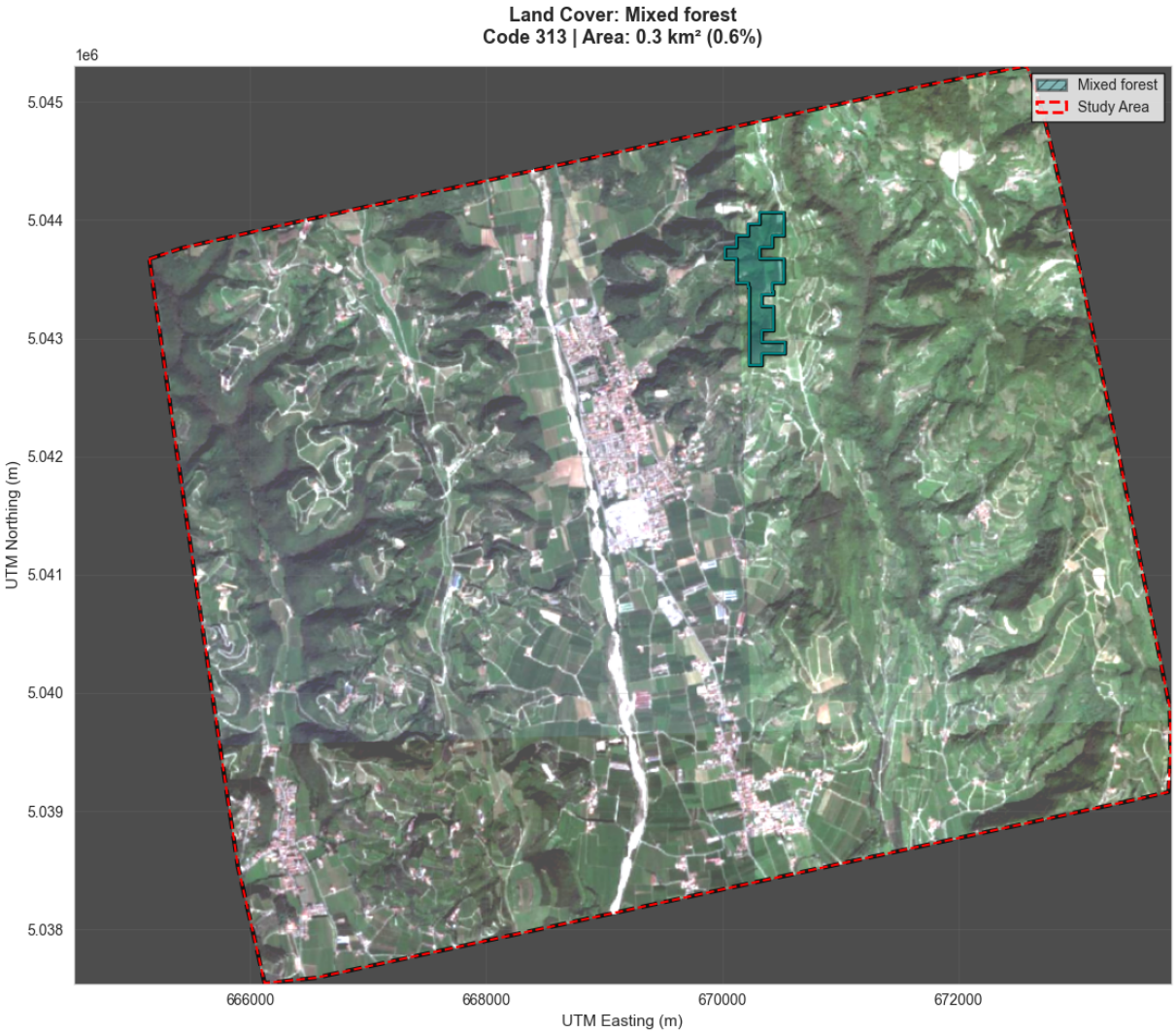
Saved: landcover_243_Agriculture_natural_vegetation_mix.png



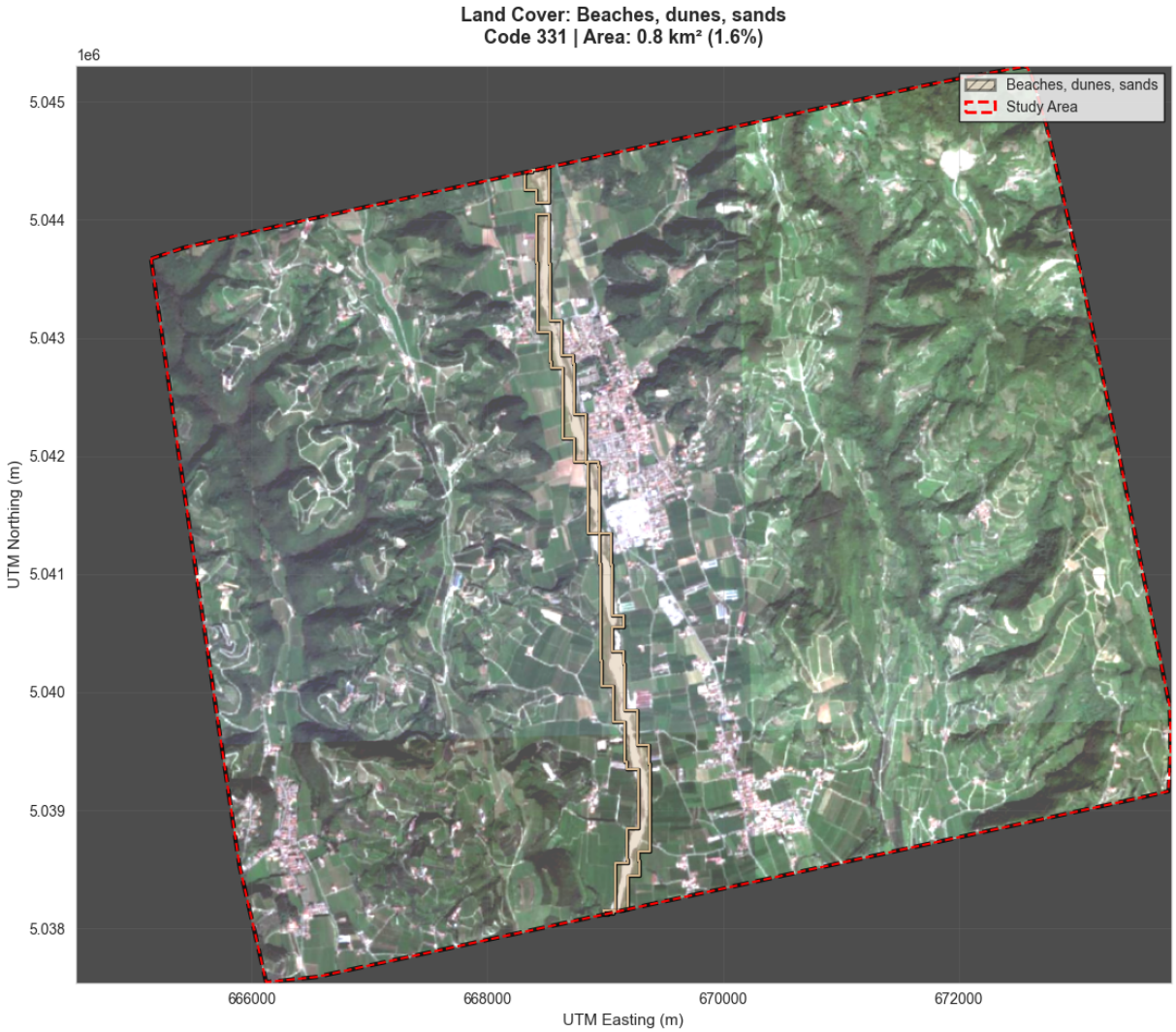

```
Saved: landcover_311_Broad-leaved_forest.png
```




```
Saved: landcover_312_Coniferous_forest.png
```




```
Saved: landcover_313_Mixed_forest.png
```




```
Saved: landcover_331_Beaches_dunes_sands.png

Generated 10 individual land cover overlay maps
```

SAOCOM VS TINITALY COMPARISON

In [20]:
```
# ============================================================================
# SAOCOM VS TINITALY COMPARISON
# ============================================================================
# Filter for valid comparisons with elevation range check
valid_elevation_range = (50, 850)

saocom_tinitaly_mask = (
    (saocom_gdf['HEIGHT_ABSOLUTE_TIN'].notna()) &
    (saocom_gdf['tinitaly_height'].notna()) &
    (saocom_gdf['HEIGHT_ABSOLUTE_TIN'] >= valid_elevation_range[0]) &
    (saocom_gdf['HEIGHT_ABSOLUTE_TIN'] <= valid_elevation_range[1]) &
    (saocom_gdf['COHER'] >= 0.5)
)
saocom_tinitaly_valid = saocom_gdf[saocom_tinitaly_mask]

saocom_tinitaly_diff = (saocom_tinitaly_valid['HEIGHT_ABSOLUTE_TIN'] -
                        saocom_tinitaly_valid['tinitaly_height']).values
```

```
saocom_tinitaly_metrics = {
    'n_points': int(len(saocom_tinitaly_diff)),
    'mean_diff': float(np.mean(saocom_tinitaly_diff)),
    'median_diff': float(np.median(saocom_tinitaly_diff)),
    'std_diff': float(np.std(saocom_tinitaly_diff)),
    'rmse': float(np.sqrt(np.mean(saocom_tinitaly_diff**2))),
    'mae': float(np.mean(np.abs(saocom_tinitaly_diff))),
    'nmad': float(1.4826 * np.median(np.abs(saocom_tinitaly_diff - np.median(saocom
    'min_diff': float(np.min(saocom_tinitaly_diff)),
    'max_diff': float(np.max(saocom_tinitaly_diff)),
    'correlation': float(np.corrcoef(saocom_tinitaly_valid['HEIGHT_ABSOLUTE_TIN'].v
                                     saocom_tinitaly_valid['tinitaly_height'].value
}

saocom_tinitaly_tolerance = float(saocom_tinitaly_metrics['nmad'])
saocom_tinitaly_higher_mask = saocom_tinitaly_diff > saocom_tinitaly_tolerance
saocom_tinitaly_lower_mask = saocom_tinitaly_diff < -saocom_tinitaly_tolerance
saocom_tinitaly_equal_mask = np.abs(saocom_tinitaly_diff) <= saocom_tinitaly_tolera

saocom_tinitaly_higher_count = np.sum(saocom_tinitaly_higher_mask)
saocom_tinitaly_lower_count = np.sum(saocom_tinitaly_lower_mask)
saocom_tinitaly_equal_count = np.sum(saocom_tinitaly_equal_mask)

saocom_tinitaly_pct_higher = 100 * saocom_tinitaly_higher_count / len(saocom_tinita
saocom_tinitaly_pct_lower = 100 * saocom_tinitaly_lower_count / len(saocom_tinitaly
saocom_tinitaly_pct_equal = 100 * saocom_tinitaly_equal_count / len(saocom_tinitaly

# ============================================================================
# SAOCOM VS COPERNICUS COMPARISON
# ============================================================================
saocom_copernicus_mask = (
    (saocom_gdf['HEIGHT_ABSOLUTE_COP'].notna()) &
    (saocom_gdf['copernicus_height'].notna()) &
    (saocom_gdf['HEIGHT_ABSOLUTE_COP'] >= valid_elevation_range[0]) &
    (saocom_gdf['HEIGHT_ABSOLUTE_COP'] <= valid_elevation_range[1]) &
    (saocom_gdf['COHER'] >= 0.5)
)
saocom_copernicus_valid = saocom_gdf[saocom_copernicus_mask]

saocom_copernicus_diff = (saocom_copernicus_valid['HEIGHT_ABSOLUTE_COP'] -
                          saocom_copernicus_valid['copernicus_height']).values

saocom_copernicus_metrics = {
    'n_points': int(len(saocom_copernicus_diff)),
    'mean_diff': float(np.mean(saocom_copernicus_diff)),
    'median_diff': float(np.median(saocom_copernicus_diff)),
    'std_diff': float(np.std(saocom_copernicus_diff)),
    'rmse': float(np.sqrt(np.mean(saocom_copernicus_diff**2))),
    'mae': float(np.mean(np.abs(saocom_copernicus_diff))),
    'nmad': float(1.4826 * np.median(np.abs(saocom_copernicus_diff - np.median(saoc
    'min_diff': float(np.min(saocom_copernicus_diff)),
    'max_diff': float(np.max(saocom_copernicus_diff)),
    'correlation': float(np.corrcoef(saocom_copernicus_valid['HEIGHT_ABSOLUTE_COP']
                                     saocom_copernicus_valid['copernicus_height'].v
}
```

```
saocom_copernicus_tolerance = float(saocom_copernicus_metrics['nmad'])
saocom_copernicus_higher_mask = saocom_copernicus_diff > saocom_copernicus_toleranc
saocom_copernicus_lower_mask = saocom_copernicus_diff < -saocom_copernicus_toleranc
saocom_copernicus_equal_mask = np.abs(saocom_copernicus_diff) <= saocom_copernicus_

saocom_copernicus_higher_count = int(np.sum(saocom_copernicus_higher_mask))
saocom_copernicus_lower_count = int(np.sum(saocom_copernicus_lower_mask))
saocom_copernicus_equal_count = int(np.sum(saocom_copernicus_equal_mask))

saocom_copernicus_pct_higher = float(100 * saocom_copernicus_higher_count / len(sao
saocom_copernicus_pct_lower = float(100 * saocom_copernicus_lower_count / len(saoco
saocom_copernicus_pct_equal = float(100 * saocom_copernicus_equal_count / len(saoco
```

Calculate slope from TINITALY DEM

In [21]:

```
# # REPLACE the old outlier detection section with:
# # (The Isolation Forest code becomes the primary outlier detection)
# #
# # Update variable names for consistency:
# normal_points = saocom_gdf_cleaned  # Previously defined differently
# outlier_points = saocom_outliers    # Previously defined differently
#
# # Remove/comment out the old IQR-based outlier detection that created:
# # - outliers_iqr
# # - outliers_zscore
# # - outliers_nmad
# # - is_outlier (based on voting)
#
# # Update all downstream references:
# n_outliers = len(saocom_outliers)  # Use this consistently
# pct_outliers = (n_outliers / len(saocom_gdf_scored)) * 100
# # ============================================================================
# # TOPOGRAPHIC ANALYSIS OF OUTLIERS
# # ============================================================================
# from scipy.ndimage import sobel
#
# def calculate_slope(dem, grid_size=10):
#     """Calculate slope in degrees"""
#     dx = sobel(dem, axis=1) / (8 * grid_size)
#     dy = sobel(dem, axis=0) / (8 * grid_size)
#     slope = np.degrees(np.arctan(np.sqrt(dx**2 + dy**2)))
#     slope[dem == NODATA] = np.nan
#     return slope
#
# # Calculate slope
# slope_grid = calculate_slope(tinitaly_10m, GRID_SIZE)
#
# # Sample terrain at point locations
# def sample_terrain(gdf, slope_grid, dem):
#     """Sample slope and elevation at point locations"""
#     slopes, elevs = [], []
#     for _, row in gdf.iterrows():
#         r, c = rowcol(target_transform, row.geometry.x, row.geometry.y)
#         if 0 <= r < grid_height and 0 <= c < grid_width:
```

```
#             slopes.append(slope_grid[r, c] if not np.isnan(slope_grid[r, c]) else
#             elevs.append(dem[r, c] if dem[r, c] != NODATA else np.nan)
#         else:
#             slopes.append(np.nan)
#             elevs.append(np.nan)
#     return slopes, elevs
#
# # Sample outliers
# outlier_slopes, outlier_elevs = sample_terrain(saocom_outliers, slope_grid, tinit
# saocom_outliers['slope'] = outlier_slopes
# saocom_outliers['elevation'] = outlier_elevs
#
# # Sample normal points
# normal_sample = saocom_gdf.sample(n=min(5000, len(saocom_gdf)), random_state=42)
# normal_slopes, normal_elevs = sample_terrain(normal_sample, slope_grid, tinitaly_
# normal_sample['slope'] = normal_slopes
# normal_sample['elevation'] = normal_elevs
#
# # Visualization
# fig, axes = plt.subplots(2, 2, figsize=(18, 16), facecolor='white')
#
# # Slope map
# ax = axes[0, 0]
# im = ax.imshow(np.ma.masked_invalid(slope_grid), cmap='terrain', origin='upper',
#                extent=[xmin_grid, xmax_grid, ymin_grid, ymax_grid], vmin=0, vmax=
# saocom_outliers.plot(ax=ax, markersize=15, color='red', edgecolors='black', linew
# plt.colorbar(im, ax=ax, label='Slope (°)')
# ax.set_title('Terrain Slope with Outliers', fontweight='bold')
# ax.set_xlabel('UTM Easting (m)')
# ax.set_ylabel('UTM Northing (m)')
#
# # Residual vs Slope
# ax = axes[0, 1]
# ax.scatter(normal_sample['slope'], np.abs(normal_sample['diff_tinitaly']),
#            s=5, alpha=0.3, c='steelblue', label='Normal')
# ax.scatter(saocom_outliers['slope'], np.abs(saocom_outliers['diff_tinitaly']),
#            s=30, alpha=0.7, c='red', edgecolors='black', linewidth=0.5, label='Ou
# ax.set_xlabel('Slope (°)')
# ax.set_ylabel('|Residual| (m)')
# ax.set_title('Residual vs Slope', fontweight='bold')
# ax.legend()
# ax.grid(True, alpha=0.3)
#
# # Residual vs Elevation
# ax = axes[1, 0]
# ax.scatter(normal_sample['elevation'], np.abs(normal_sample['diff_tinitaly']),
#            s=5, alpha=0.3, c='steelblue', label='Normal')
# ax.scatter(saocom_outliers['elevation'], np.abs(saocom_outliers['diff_tinitaly']),
#            s=30, alpha=0.7, c='red', edgecolors='black', linewidth=0.5, label='Ou
# ax.set_xlabel('Elevation (m)')
# ax.set_ylabel('|Residual| (m)')
# ax.set_title('Residual vs Elevation', fontweight='bold')
# ax.legend()
# ax.grid(True, alpha=0.3)
#
# # Statistics
```

```
# ax = axes[1, 1]
# ax.axis('off')
#
# outlier_stats = saocom_outliers[['slope', 'elevation']].describe()
# normal_stats = normal_sample[['slope', 'elevation']].describe()
#
# stats_text = f"""TOPOGRAPHIC ANALYSIS
#
# OUTLIERS (n={len(saocom_outliers):,}):
# Slope: {outlier_stats.loc['mean', 'slope']:.1f}° ± {outlier_stats.loc['std', 'slo
# Elevation: {outlier_stats.loc['mean', 'elevation']:.1f} ± {outlier_stats.loc['std
#
# NORMAL (n={len(normal_sample):,}):
# Slope: {normal_stats.loc['mean', 'slope']:.1f}° ± {normal_stats.loc['std', 'slope
# Elevation: {normal_stats.loc['mean', 'elevation']:.1f} ± {normal_stats.loc['std',
#
# DIFFERENCE:
# Slope: {outlier_stats.loc['mean', 'slope'] - normal_stats.loc['mean', 'slope']:+.
# Elevation: {outlier_stats.loc['mean', 'elevation'] - normal_stats.loc['mean', 'el
#
# ax.text(0.1, 0.5, stats_text, transform=ax.transAxes, fontsize=10,
#         verticalalignment='center', fontfamily='monospace',
#         bbox=dict(boxstyle='round', facecolor='white', edgecolor='black'))
#
# plt.tight_layout()
# plt.savefig(RESULTS_DIR / 'outliers_vs_topography.png', dpi=300, bbox_inches='tig
# plt.show()
```

3d model

In [22]:
```
import plotly.graph_objects as go
from scipy.interpolate import griddata

# Use the cleaned dataset and outliers from Isolation Forest
normal_sample_3d = saocom_gdf.sample(n=min(5000, len(saocom_gdf)), random_state=42)
outlier_sample_3d = saocom_outliers.sample(n=min(500, len(saocom_outliers)), random

# Create SAOCOM interpolated surface
saocom_valid = saocom_gdf[saocom_gdf['HEIGHT_ABSOLUTE_TIN'].notna()].copy()
x_saocom = saocom_valid.geometry.x.values
y_saocom = saocom_valid.geometry.y.values
z_saocom = saocom_valid['HEIGHT_ABSOLUTE_TIN'].values

# Create grid for surface
xi = np.linspace(xmin_grid, xmax_grid, 100)
yi = np.linspace(ymin_grid, ymax_grid, 100)
xi_grid, yi_grid = np.meshgrid(xi, yi)

# Interpolate SAOCOM surface
zi_saocom = griddata((x_saocom, y_saocom), z_saocom, (xi_grid, yi_grid),
                     method='linear', fill_value=np.nan)

# Downsample TINITALY surface
tinitaly_downsampled = tinitaly_10m[::10, ::10]
```

```
x_tin = np.linspace(xmin_grid, xmax_grid, tinitaly_downsampled.shape[1])
y_tin = np.linspace(ymax_grid, ymin_grid, tinitaly_downsampled.shape[0])
x_tin_grid, y_tin_grid = np.meshgrid(x_tin, y_tin)
tinitaly_downsampled = np.where(tinitaly_downsampled == NODATA, np.nan, tinitaly_do

# Create figure
fig = go.Figure()

# TINITALY Surface
fig.add_trace(go.Surface(
    x=x_tin_grid, y=y_tin_grid, z=tinitaly_downsampled,
    colorscale='Earth', name='TINITALY DEM', showscale=True,
    colorbar=dict(x=1.15, title='Elevation (m)'),
    visible=True, opacity=0.8,
    hovertemplate='X: %{x:.0f}<br>Y: %{y:.0f}<br>TINITALY: %{z:.1f}m<extra></extra>
))

# SAOCOM Interpolated Surface
fig.add_trace(go.Surface(
    x=xi_grid, y=yi_grid, z=zi_saocom,
    colorscale='Viridis', name='SAOCOM Surface',
    showscale=False, visible=False, opacity=0.7,
    hovertemplate='X: %{x:.0f}<br>Y: %{y:.0f}<br>SAOCOM: %{z:.1f}m<extra></extra>'
))

# Normal SAOCOM Points
fig.add_trace(go.Scatter3d(
    x=normal_sample_3d.geometry.x, y=normal_sample_3d.geometry.y,
    z=normal_sample_3d['HEIGHT_ABSOLUTE_TIN'],
    mode='markers', name='SAOCOM Points',
    marker=dict(size=2, color=normal_sample_3d['diff_tinitaly'],
                colorscale='RdBu_r', cmin=-10, cmax=10,
                colorbar=dict(x=1.0, title='Residual (m)', len=0.5, y=0.25),
                showscale=True, line=dict(width=0)),
    text=[f"Residual: {r:+.2f}m<br>Height: {h:.1f}m<br>Coherence: {c:.2f}"
          for r, h, c in zip(normal_sample_3d['diff_tinitaly'],
                             normal_sample_3d['HEIGHT_ABSOLUTE_TIN'],
                             normal_sample_3d['COHER'])],
    hovertemplate='%{text}<extra></extra>', visible=True
))

# Outlier Points
if len(outlier_sample_3d) > 0:
    fig.add_trace(go.Scatter3d(
        x=outlier_sample_3d.geometry.x, y=outlier_sample_3d.geometry.y,
        z=outlier_sample_3d['HEIGHT_ABSOLUTE_TIN'],
        mode='markers', name='Outliers',
        marker=dict(size=6, color='red', symbol='diamond',
                    line=dict(color='black', width=1)),
        text=[f"<b>OUTLIER</b><br>Residual: {r:+.2f}m<br>Height: {h:.1f}m<br>Cohere
              for r, h, c in zip(outlier_sample_3d['diff_tinitaly'],
                                 outlier_sample_3d['HEIGHT_ABSOLUTE_TIN'],
                                 outlier_sample_3d['COHER'])],
        hovertemplate='%{text}<extra></extra>', visible=True
    ))
```

```
# Layout
fig.update_layout(
    updatemenus=[
        dict(type="buttons", direction="down", x=0.02, xanchor="left",
             y=0.98, yanchor="top",
             buttons=[
                 dict(label="All", method="update",
                      args=[{"visible": [True, True, True, True]}]),
                 dict(label="Surfaces Only", method="update",
                      args=[{"visible": [True, True, False, False]}]),
                 dict(label="Points Only", method="update",
                      args=[{"visible": [False, False, True, True]}]),
                 dict(label="TINITALY + Points", method="update",
                      args=[{"visible": [True, False, True, True]}]),
                 dict(label="SAOCOM Surface + Outliers", method="update",
                      args=[{"visible": [False, True, False, True]}]),
             ]),
        dict(type="buttons", direction="right", x=0.02, xanchor="left",
             y=0.02, yanchor="bottom",
             buttons=[
                 dict(label="Toggle TINITALY", method="restyle",
                      args=["visible", "toggle"], args2=[{"visible": [True]}, [0]])
                 dict(label="Toggle SAOCOM Surface", method="restyle",
                      args=["visible", "toggle"], args2=[{"visible": [True]}, [1]])
                 dict(label="Toggle Points", method="restyle",
                      args=["visible", "toggle"], args2=[{"visible": [True]}, [2]])
                 dict(label="Toggle Outliers", method="restyle",
                      args=["visible", "toggle"], args2=[{"visible": [True]}, [3]])
             ])
    ],
    scene=dict(xaxis_title='UTM Easting (m)', yaxis_title='UTM Northing (m)',
               zaxis_title='Elevation (m)', aspectmode='manual',
               aspectratio=dict(x=1, y=1, z=0.3),
               camera=dict(eye=dict(x=1.5, y=1.5, z=1.2))),
    title=dict(text='3D SAOCOM vs TINITALY Analysis<br><sub>Top: Presets | Bottom:
               x=0.5, xanchor='center'),
    width=1400, height=900, showlegend=True,
    legend=dict(x=0.02, y=0.5),
    paper_bgcolor='white', plot_bgcolor='white'
)

fig.write_html(RESULTS_DIR / 'saocom_3d_interactive.html')
print(f"Saved: {RESULTS_DIR / 'saocom_3d_interactive.html'}")
fig.show()
```

```
Saved: results\saocom_3d_interactive.html
```

SAOCOM RESIDUAL OUTLIERS - KERNEL DENSITY HEAT MAP (ArcGIS-style)

In [23]:

```
# # ============================================================================
# # SAOCOM RESIDUAL OUTLIERS - KERNEL DENSITY HEAT MAP (ArcGIS-style)
# # ============================================================================
# from scipy.stats import gaussian_kde
# from matplotlib.colors import LinearSegmentedColormap
```

```
# import numpy as np
# import matplotlib.pyplot as plt
#
# # ============================================================================
# # 1. PREPARE OUTLIER DATA
# # ============================================================================
# # Separate by sign for better visualization
# positive_outliers = outlier_points[outlier_points['diff_tinitaly'] > 0].copy()
# negative_outliers = outlier_points[outlier_points['diff_tinitaly'] < 0].copy()
#
# print(f"Positive outliers (SAOCOM higher): {len(positive_outliers)}")
# print(f"Negative outliers (TINITALY higher): {len(negative_outliers)}")
#
# # ============================================================================
# # 2. CREATE KERNEL DENSITY ESTIMATION
# # ============================================================================
#
# def create_kde_heatmap(points_gdf, grid_res=50):
#     """Create kernel density heat map from point data"""
#     if len(points_gdf) < 2:
#         return None, None, None
#
#     # Extract coordinates
#     x = points_gdf.geometry.x.values
#     y = points_gdf.geometry.y.values
#
#     # Create grid
#     xi = np.linspace(xmin_grid, xmax_grid, int((xmax_grid - xmin_grid) / grid_res
#     yi = np.linspace(ymax_grid, ymin_grid, int((ymax_grid - ymin_grid) / grid_res
#     xi_grid, yi_grid = np.meshgrid(xi, yi)
#
#     # Perform kernel density estimation
#     positions = np.vstack([xi_grid.ravel(), yi_grid.ravel()])
#     values = np.vstack([x, y])
#     kernel = gaussian_kde(values)
#     density = np.reshape(kernel(positions).T, xi_grid.shape)
#
#     # Apply hull mask
#     from scipy.ndimage import zoom
#     mask_factor_y = density.shape[0] / hull_mask.shape[0]
#     mask_factor_x = density.shape[1] / hull_mask.shape[1]
#     hull_mask_resampled = zoom(hull_mask.astype(float),
#                                (mask_factor_y, mask_factor_x), order=0) > 0.5
#     density[~hull_mask_resampled] = np.nan
#
#     return density, xi, yi
#
# # Generate density maps
# print("\nGenerating kernel density heat maps...")
#
# all_outlier_density, all_xi, all_yi = create_kde_heatmap(outlier_points, grid_res
# positive_density, pos_xi, pos_yi = create_kde_heatmap(positive_outliers, grid_res
# negative_density, neg_xi, neg_yi = create_kde_heatmap(negative_outliers, grid_res
#
# # ============================================================================
# # 3. CREATE ARCGIS-STYLE HEAT MAP COLORMAP
```

```
# # ============================================================================
# # ArcGIS heat map style: transparent -> blue -> cyan -> yellow -> red
# heat_colors = [
#     (0.0, (1.0, 1.0, 1.0, 0.0)),    # Transparent white
#     (0.2, (0.0, 0.0, 1.0, 0.3)),    # Blue (low density)
#     (0.4, (0.0, 1.0, 1.0, 0.5)),    # Cyan
#     (0.6, (0.0, 1.0, 0.0, 0.7)),    # Green
#     (0.8, (1.0, 1.0, 0.0, 0.85)),   # Yellow
#     (1.0, (1.0, 0.0, 0.0, 1.0))     # Red (high density)
# ]
# arcgis_cmap = LinearSegmentedColormap.from_list('arcgis_heat',
#                                                 [c for _, c in heat_colors])
#
# # ============================================================================
# # 4. VISUALIZATION - COMBINED AND SEPARATE HEAT MAPS
# # ============================================================================
# fig, axes = plt.subplots(2, 2, figsize=(22, 20), facecolor='white')
#
# extent_grid = [xmin_grid, xmax_grid, ymin_grid, ymax_grid]
#
# # Plot 1: ALL OUTLIERS - Combined Heat Map
# ax = axes[0, 0]
# ax.set_facecolor('white')
#
# # Background
# ax.imshow(sentinel_rgb_norm, extent=extent_grid, origin='upper', alpha=0.3)
#
# # Heat map
# if all_outlier_density is not None:
#     # Normalize density for better visualization
#     density_normalized = all_outlier_density / np.nanmax(all_outlier_density)
#
#     im1 = ax.imshow(
#         density_normalized,
#         extent=[all_xi.min(), all_xi.max(), all_yi.min(), all_yi.max()],
#         origin='upper',
#         cmap=arcgis_cmap,
#         alpha=0.85,
#         interpolation='bilinear'
#     )
#
#     cbar1 = plt.colorbar(im1, ax=ax, label='Outlier Density', shrink=0.7)
#     cbar1.ax.tick_params(labelsize=10)
#
# # Overlay actual outlier points (small)
# outlier_points.plot(ax=ax, markersize=2, color='white', alpha=0.3,
#                     edgecolors='black', linewidth=0.3)
#
# # Study area boundary
# hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--')
#
# ax.set_xlabel('UTM Easting (m)', fontsize=12, color='black')
# ax.set_ylabel('UTM Northing (m)', fontsize=12, color='black')
# ax.set_title('All Residual Outliers - Kernel Density Heat Map\n(ArcGIS Style)',
#              fontweight='bold', fontsize=14, color='black')
# ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
```

```
# ax.tick_params(colors='black')
#
# stats_text = f"""Total Outliers: {n_outliers:,}
# Positive: {len(positive_outliers):,}
# Negative: {len(negative_outliers):,}
#
# Residual Range:
# [{outlier_points['diff_tinitaly'].min():+.1f},
#  {outlier_points['diff_tinitaly'].max():+.1f}] m
#
# Heat colors show
# concentration of outliers"""
#
# ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=10,
#         verticalalignment='top', fontfamily='monospace',
#         bbox=dict(boxstyle='round', facecolor='white', alpha=0.95,
#                   edgecolor='black', linewidth=1.5))
#
# # Plot 2: POSITIVE OUTLIERS (SAOCOM Higher)
# ax = axes[0, 1]
# ax.set_facecolor('white')
#
# ax.imshow(sentinel_rgb_norm, extent=extent_grid, origin='upper', alpha=0.3)
#
# if positive_density is not None and len(positive_outliers) >= 2:
#     density_normalized = positive_density / np.nanmax(positive_density)
#
#     im2 = ax.imshow(
#         density_normalized,
#         extent=[pos_xi.min(), pos_xi.max(), pos_yi.min(), pos_yi.max()],
#         origin='upper',
#         cmap=arcgis_cmap,
#         alpha=0.85,
#         interpolation='bilinear'
#     )
#
#     cbar2 = plt.colorbar(im2, ax=ax, label='Outlier Density', shrink=0.7)
#     cbar2.ax.tick_params(labelsize=10)
#
#     # Overlay points
#     positive_outliers.plot(ax=ax, markersize=3, color='white', alpha=0.4,
#                            edgecolors='red', linewidth=0.5)
#
# hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--')
#
# ax.set_xlabel('UTM Easting (m)', fontsize=12, color='black')
# ax.set_ylabel('UTM Northing (m)', fontsize=12, color='black')
# ax.set_title('Positive Outliers - Kernel Density\n(SAOCOM > TINITALY)',
#              fontweight='bold', fontsize=14, color='black')
# ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
# ax.tick_params(colors='black')
#
# if len(positive_outliers) > 0:
#     stats_text = f"""Count: {len(positive_outliers):,}
# Mean: {positive_outliers['diff_tinitaly'].mean():+.2f} m
# Max: {positive_outliers['diff_tinitaly'].max():+.2f} m
```

```
#
# SAOCOM reports
# HIGHER than TINITALY"""
#
#     ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=10,
#             verticalalignment='top', fontfamily='monospace',
#             bbox=dict(boxstyle='round', facecolor='white', alpha=0.95,
#                       edgecolor='black', linewidth=1.5))
#
# # Plot 3: NEGATIVE OUTLIERS (TINITALY Higher)
# ax = axes[1, 0]
# ax.set_facecolor('white')
#
# ax.imshow(sentinel_rgb_norm, extent=extent_grid, origin='upper', alpha=0.3)
#
# if negative_density is not None and len(negative_outliers) >= 2:
#     density_normalized = negative_density / np.nanmax(negative_density)
#
#     im3 = ax.imshow(
#         density_normalized,
#         extent=[neg_xi.min(), neg_xi.max(), neg_yi.min(), neg_yi.max()],
#         origin='upper',
#         cmap=arcgis_cmap,
#         alpha=0.85,
#         interpolation='bilinear'
#     )
#
#     cbar3 = plt.colorbar(im3, ax=ax, label='Outlier Density', shrink=0.7)
#     cbar3.ax.tick_params(labelsize=10)
#
#     # Overlay points
#     negative_outliers.plot(ax=ax, markersize=3, color='white', alpha=0.4,
#                            edgecolors='blue', linewidth=0.5)
#
# hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--')
#
# ax.set_xlabel('UTM Easting (m)', fontsize=12, color='black')
# ax.set_ylabel('UTM Northing (m)', fontsize=12, color='black')
# ax.set_title('Negative Outliers - Kernel Density\n(TINITALY > SAOCOM)',
#              fontweight='bold', fontsize=14, color='black')
# ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
# ax.tick_params(colors='black')
#
# if len(negative_outliers) > 0:
#     stats_text = f"""Count: {len(negative_outliers):,}
# Mean: {negative_outliers['diff_tinitaly'].mean():+.2f} m
# Min: {negative_outliers['diff_tinitaly'].min():+.2f} m
#
# TINITALY reports
# HIGHER than SAOCOM"""
#
#     ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=10,
#             verticalalignment='top', fontfamily='monospace',
#             bbox=dict(boxstyle='round', facecolor='white', alpha=0.95,
#                       edgecolor='black', linewidth=1.5))
#
```

```
# # Plot 4: OUTLIERS BY RESIDUAL MAGNITUDE (Points colored by value)
# ax = axes[1, 1]
# ax.set_facecolor('white')
#
# ax.imshow(sentinel_rgb_norm, extent=extent_grid, origin='upper', alpha=0.3)
#
# # Scatter plot colored by residual magnitude
# scatter = ax.scatter(
#     outlier_points.geometry.x,
#     outlier_points.geometry.y,
#     c=outlier_points['diff_tinitaly'],
#     cmap='RdBu_r',
#     s=30,
#     alpha=0.8,
#     edgecolors='black',
#     linewidth=0.5,
#     vmin=outlier_points['diff_tinitaly'].min(),
#     vmax=outlier_points['diff_tinitaly'].max()
# )
#
# cbar4 = plt.colorbar(scatter, ax=ax, label='Residual (m)', shrink=0.7)
# cbar4.ax.tick_params(labelsize=10)
#
# # Mark extreme outliers
# max_positive = outlier_points.loc[outlier_points['diff_tinitaly'].idxmax()]
# max_negative = outlier_points.loc[outlier_points['diff_tinitaly'].idxmin()]
#
# ax.scatter(max_positive.geometry.x, max_positive.geometry.y,
#            s=300, marker='*', color='yellow', edgecolors='black',
#            linewidth=2.5, zorder=10, label=f'Max: {max_positive["diff_tinitaly"]:
# ax.scatter(max_negative.geometry.x, max_negative.geometry.y,
#            s=100, marker='v', color='cyan', edgecolors='black',
#            linewidth=2.5, zorder=10, label=f'Min: {max_negative["diff_tinitaly"]:
#
# hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--')
#
# ax.set_xlabel('UTM Easting (m)', fontsize=12, color='black')
# ax.set_ylabel('UTM Northing (m)', fontsize=12, color='black')
# ax.set_title('Outliers by Residual Magnitude\n(Point Size = Fixed, Color = Residu
#              fontweight='bold', fontsize=14, color='black')
# ax.legend(loc='upper right', fontsize=10)
# ax.grid(True, alpha=0.3, color='black', linewidth=0.5)
# ax.tick_params(colors='black')
#
# stats_text = f"""Outliers: {n_outliers:,}
#
# Red = SAOCOM Higher
# Blue = TINITALY Higher
#
# Extreme Values:
# Max: {outlier_points['diff_tinitaly'].max():+.2f} m (★)
# Min: {outlier_points['diff_tinitaly'].min():+.2f} m (▼)"""
#
# ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=10,
#         verticalalignment='top', fontfamily='monospace',
#         bbox=dict(boxstyle='round', facecolor='white', alpha=0.95,
```

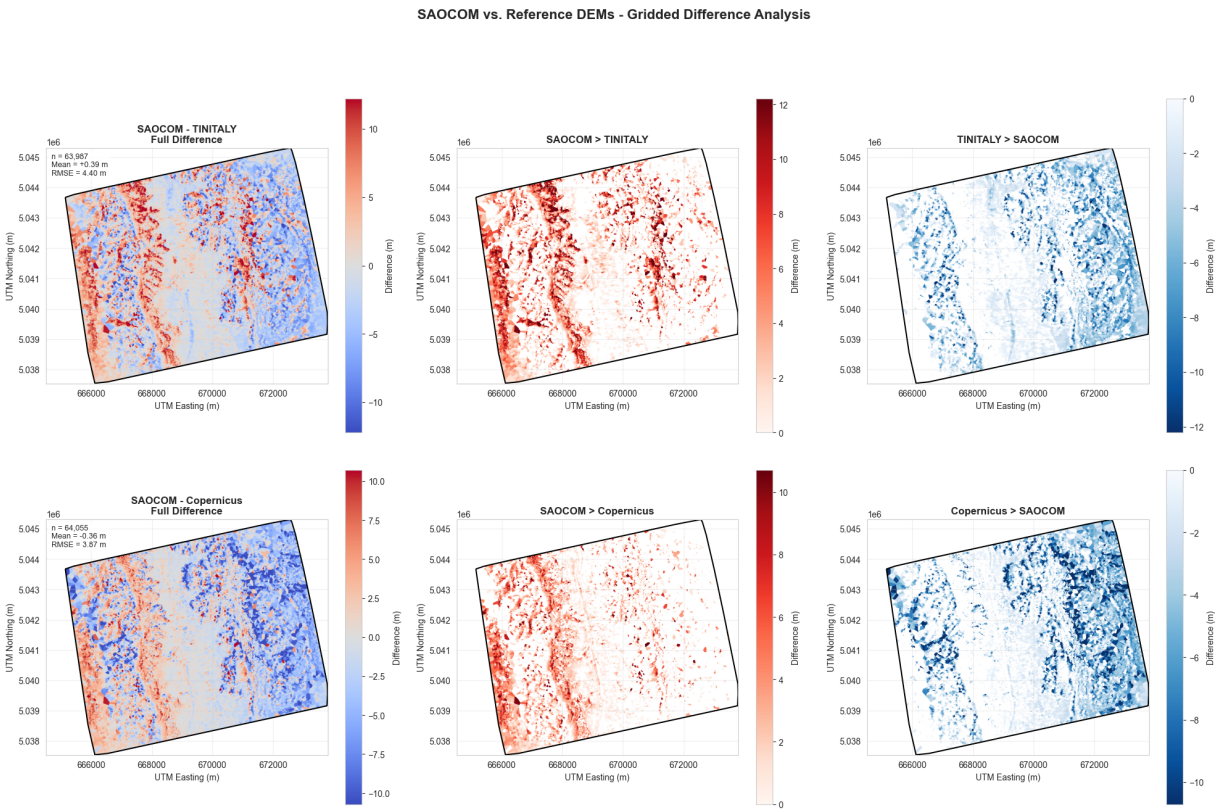
```
#                  edgecolor='black', linewidth=1.5))
#
# plt.tight_layout()
# plt.savefig(RESULTS_DIR / 'saocom_outlier_heatmap_kde.png',
#             dpi=300, bbox_inches='tight', facecolor='white')
# plt.show()
#
# print("\nSaved: saocom_outlier_heatmap_kde.png")
# print(f"Total outliers visualized: {n_outliers:,}")
# print(f"  Positive (SAOCOM > TINITALY): {len(positive_outliers):,}")
# print(f"  Negative (TINITALY > SAOCOM): {len(negative_outliers):,}")
```

In [24]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import griddata

# Note: This script assumes the following variables are pre-defined from previous c
# saocom_gdf, COHERENCE_THRESHOLD, hull_mask, hull_gdf, grid_height, grid_width,
# target_transform, xmin_grid, xmax_grid, ymin_grid, ymax_grid,
# saocom_tinitaly_metrics, saocom_copernicus_metrics, RESULTS_DIR

def create_difference_grid(gdf, height_col, ref_col, grid_shape, transform, hull_ma
    """Grids point data differences onto a raster grid using nearest neighbor inter
    query_str = f"`{height_col}`.notna() & `{ref_col}`.notna() & COHER >= @COHERENC
    valid_points = gdf.query(query_str).copy()
    valid_points['diff'] = valid_points[height_col] - valid_points[ref_col]

    if valid_points.empty:
        return np.full(grid_shape, np.nan), valid_points

    # Create grid coordinates for interpolation
    grid_height, grid_width = grid_shape
    x_coords = np.linspace(transform.c, transform.c + transform.a * grid_width, gri
    y_coords = np.linspace(transform.f, transform.f + transform.e * grid_height, gr
    xi_grid, yi_grid = np.meshgrid(x_coords, y_coords)

    # Interpolate and mask the grid
    diff_grid = griddata(
        (valid_points.geometry.x, valid_points.geometry.y),
        valid_points['diff'],
        (xi_grid, yi_grid),
        method='nearest'
    )
    diff_grid[~hull_mask] = np.nan
    return diff_grid, valid_points

def plot_panel(ax, data, title, cmap, vmin=None, vmax=None, stats_text=None):
    """Helper function to plot a single map panel."""
    ax.set_facecolor('white')
    cmap.set_bad(color='white', alpha=0)
    extent = [xmin_grid, xmax_grid, ymin_grid, ymax_grid]
    im = ax.imshow(data, cmap=cmap, origin='upper', extent=extent, vmin=vmin, vmax=
    hull_gdf.boundary.plot(ax=ax, color='black', linewidth=1.5)
    ax.set_title(title, fontweight='bold', fontsize=12)
    ax.set_xlabel('UTM Easting (m)')
```

```
    ax.set_ylabel('UTM Northing (m)')
    ax.grid(True, alpha=0.3)
    plt.colorbar(im, ax=ax, label='Difference (m)', shrink=0.8)
    if stats_text:
        ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=9,
                verticalalignment='top', bbox=dict(boxstyle='round', facecolor='whi

# --- Main Execution ---
print("Gridding SAOCOM differences (using cleaned data)...")
diff_grid_tin, points_tin = create_difference_grid(saocom_gdf, 'HEIGHT_ABSOLUTE_TIN
diff_grid_cop, points_cop = create_difference_grid(saocom_gdf, 'HEIGHT_ABSOLUTE_COP

# --- Visualization ---
fig, axes = plt.subplots(2, 3, figsize=(20, 14), facecolor='white')
fig.suptitle('SAOCOM vs. Reference DEMs - Gridded Difference Analysis', fontsize=16

# Plotting parameters
v_tin = np.nanpercentile(np.abs(points_tin['diff']), 98)
v_cop = np.nanpercentile(np.abs(points_cop['diff']), 98)
stats_tin = f"n = {len(points_tin):,}\nMean = {points_tin['diff'].mean():+.2f} m\nR
stats_cop = f"n = {len(points_cop):,}\nMean = {points_cop['diff'].mean():+.2f} m\nR

# Row 1: SAOCOM vs TINITALY
plot_panel(axes[0, 0], diff_grid_tin, 'SAOCOM - TINITALY\nFull Difference', plt.cm.
plot_panel(axes[0, 1], np.where(diff_grid_tin > 0, diff_grid_tin, np.nan), 'SAOCOM
plot_panel(axes[0, 2], np.where(diff_grid_tin < 0, diff_grid_tin, np.nan), 'TINITAL

# Row 2: SAOCOM vs Copernicus
plot_panel(axes[1, 0], diff_grid_cop, 'SAOCOM - Copernicus\nFull Difference', plt.c
plot_panel(axes[1, 1], np.where(diff_grid_cop > 0, diff_grid_cop, np.nan), 'SAOCOM
plot_panel(axes[1, 2], np.where(diff_grid_cop < 0, diff_grid_cop, np.nan), 'Coperni

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.savefig(RESULTS_DIR / 'saocom_gridded_comparison.png', dpi=300, facecolor='whit
plt.show()
```

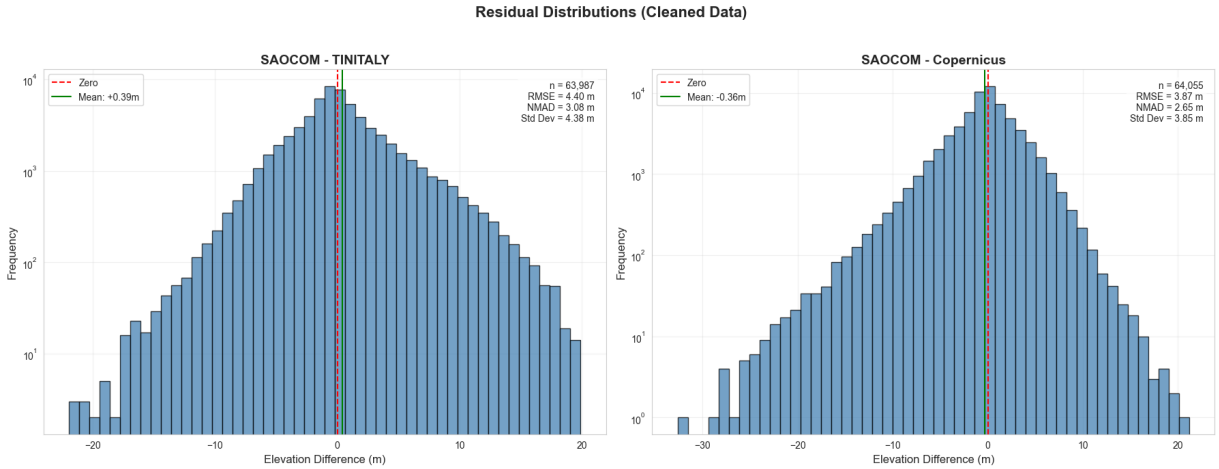
```
Gridding SAOCOM differences (using cleaned data)...
```



Edited Histograms

In [25]:
```
def plot_distribution(ax, diff_series, title, metrics):
    """Helper function to plot a single residual distribution histogram."""
    ax.set_facecolor('white')
    ax.hist(diff_series, bins=50, alpha=0.75, color='steelblue', edgecolor='black')
    ax.axvline(0, color='red', linestyle='--', label='Zero')
    ax.axvline(metrics['mean_diff'], color='green', linestyle='-', label=f"Mean: {m

    stats_text = (f"n = {metrics['n_points']:,}\n"
                  f"RMSE = {metrics['rmse']:.2f} m\n"
                  f"NMAD = {metrics['nmad']:.2f} m\n"
                  f"Std Dev = {metrics['std_diff']:.2f} m")

    ax.text(0.98, 0.97, stats_text, transform=ax.transAxes, fontsize=10,
            verticalalignment='top', horizontalalignment='right',
            bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

    ax.set_title(title, fontweight='bold', fontsize=14)
    ax.set_xlabel('Elevation Difference (m)', fontsize=12)
    ax.set_ylabel('Frequency', fontsize=12)
    ax.set_yscale('log')
    ax.legend()
    ax.grid(True, alpha=0.3)

# --- Visualization ---
fig, axes = plt.subplots(1, 2, figsize=(18, 7), facecolor='white')
fig.suptitle('Residual Distributions (Cleaned Data)', fontsize=16, fontweight='bold

# Use the 'diff' column from the points DataFrames created in the previous cell
```

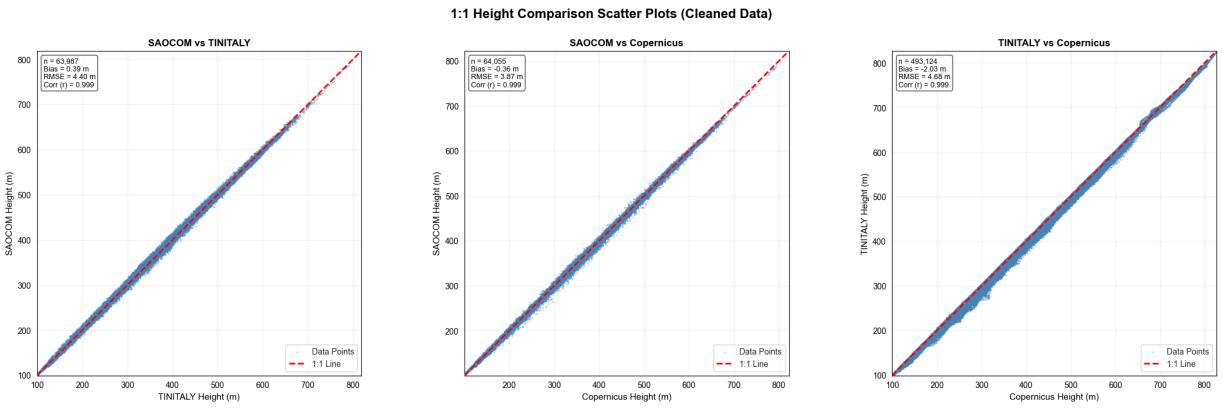
```
plot_distribution(axes[0], points_tin['diff'], 'SAOCOM - TINITALY', saocom_tinitaly
plot_distribution(axes[1], points_cop['diff'], 'SAOCOM - Copernicus', saocom_copern

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig(RESULTS_DIR / 'saocom_residual_distributions.png', dpi=300, facecolor='
plt.show()
```



In [26]:

```
import matplotlib.pyplot as plt
import numpy as np

def plot_scatter_comparison(ax, x_data, y_data, x_label, y_label, title, stats):
    """
    Creates a simple 1:1 scatter plot to compare two sets of height data.
    """
    ax.set_facecolor('white') # Set background to white

    # Plot the individual data points
    ax.scatter(x_data, y_data, s=1, alpha=0.3, c='steelblue', label='Data Points')

    # Determine plot limits and draw the 1:1 line
    lims = [
        np.min([x_data.min(), y_data.min()]),
        np.max([x_data.max(), y_data.max()]),
    ]
    ax.plot(lims, lims, 'r--', linewidth=2, label='1:1 Line', zorder=10)
    ax.set_xlim(lims)
    ax.set_ylim(lims)

    # Add statistics box
    stats_text = (f"n = {stats.get('n_points', stats.get('n_pixels')):,}\n"
                  f"Bias = {stats['mean_diff']:.2f} m\n"
                  f"RMSE = {stats['rmse']:.2f} m\n"
                  f"Corr (r) = {stats['correlation']:.3f}")

    ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=9,
            verticalalignment='top', color='black', # Text color to black
            bbox=dict(boxstyle='round', facecolor='white', alpha=0.7, edgecolor='bl

    # Style the plot
    ax.set_title(title, fontweight='bold', fontsize=12, color='black')
    ax.set_xlabel(x_label, fontsize=11, color='black')
```

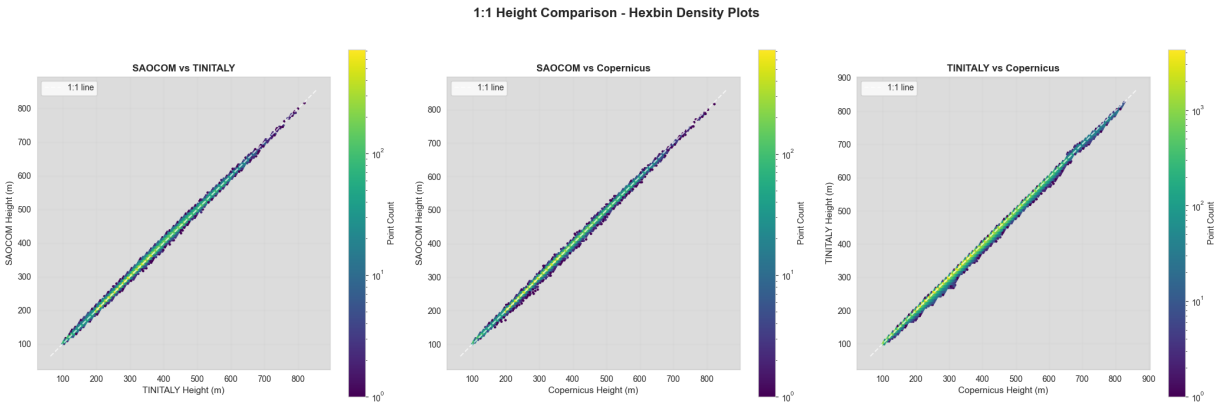
```
    ax.set_ylabel(y_label, fontsize=11, color='black')
    ax.grid(True, alpha=0.3)
    ax.legend(loc='lower right')
    ax.set_aspect('equal', 'box')

    # Style ticks and spines
    ax.tick_params(colors='black')
    for spine in ax.spines.values():
        spine.set_edgecolor('black')


# --- Visualization ---
fig, axes = plt.subplots(1, 3, figsize=(22, 7), facecolor='white')
fig.suptitle('1:1 Height Comparison Scatter Plots (Cleaned Data)', fontsize=16, fon

# Note: Assumes variables from previous cells are available
# (points_tin, points_cop, valid_copernicus, valid_tinitaly, and all metrics dicts)

plot_scatter_comparison(axes[0], points_tin['tinitaly_height'], points_tin['HEIGHT_
                        'TINITALY Height (m)', 'SAOCOM Height (m)', 'SAOCOM vs TINI

plot_scatter_comparison(axes[1], points_cop['copernicus_height'], points_cop['HEIGH
                        'Copernicus Height (m)', 'SAOCOM Height (m)', 'SAOCOM vs Co

plot_scatter_comparison(axes[2], valid_copernicus, valid_tinitaly,
                        'Copernicus Height (m)', 'TINITALY Height (m)', 'TINITALY v

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.savefig(RESULTS_DIR / 'saocom_scatter_comparisons.png', dpi=300, facecolor='whi
plt.show()
```



Density Plots Color

In [27]:
```
import matplotlib.pyplot as plt
import matplotlib.colors as colors

# --- Visualization ---
fig, axes = plt.subplots(1, 3, figsize=(21, 7), facecolor='white')
fig.suptitle('1:1 Height Comparison - Hexbin Density Plots', fontsize=16, fontweigh

def plot_hexbin(ax, x_data, y_data, x_label, y_label, title):
    """Helper function to create a hexbin plot."""
```

```
    ax.set_facecolor('gainsboro')
    # Create the hexbin plot
    hb = ax.hexbin(x_data, y_data, gridsize=150, cmap='viridis',
                   norm=colors.LogNorm(), mincnt=1) # Log scale is essential

    # Add a color bar
    fig.colorbar(hb, ax=ax, label='Point Count')

    # Add 1:1 line
    lims = [min(ax.get_xlim()[0], ax.get_ylim()[0]), max(ax.get_xlim()[1], ax.get_y
    ax.plot(lims, lims, 'w--', linewidth=1.5, label='1:1 line', alpha=0.7)

    ax.set_title(title, fontweight='bold', fontsize=12)
    ax.set_xlabel(x_label, fontsize=11)
    ax.set_ylabel(y_label, fontsize=11)
    ax.grid(True, alpha=0.2)
    ax.legend()
    ax.set_aspect('equal', 'box')

# Plot 1: SAOCOM vs TINITALY
plot_hexbin(axes[0], points_tin['tinitaly_height'], points_tin['HEIGHT_ABSOLUTE_TIN
            'TINITALY Height (m)', 'SAOCOM Height (m)', 'SAOCOM vs TINITALY')

# Plot 2: SAOCOM vs Copernicus
plot_hexbin(axes[1], points_cop['copernicus_height'], points_cop['HEIGHT_ABSOLUTE_C
            'Copernicus Height (m)', 'SAOCOM Height (m)', 'SAOCOM vs Copernicus')

# Plot 3: TINITALY vs Copernicus
plot_hexbin(axes[2], valid_copernicus, valid_tinitaly,
            'Copernicus Height (m)', 'TINITALY Height (m)', 'TINITALY vs Copernicus

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig(RESULTS_DIR / 'saocom_hexbin_comparisons.png', dpi=300, facecolor='whit
plt.show()
```



SAOCOM VS REFERENCE DEMs - GRIDDED COMPARISON ANALYSIS

In [29]:
```
# import numpy as np
# import pandas as pd
# import matplotlib.pyplot as plt
# from scipy.interpolate import griddata
```

```
# from scipy import stats
#
# # ==============================================================================
# # 1. GRIDDED DEM COMPARISON
# # ==============================================================================
#
# def process_dem_comparison(gdf, height_col, ref_col, grid_shape, transform, hull_
#     """Grids SAOCOM point differences and classifies them against a tolerance."""
#     # Filter for valid, coherent points and calculate difference
#     valid_mask = gdf[height_col].notna() & gdf[ref_col].notna() & (gdf['COHER'] >
#     points = gdf[valid_mask].copy()
#     points['diff'] = points[height_col] - points[ref_col]
#
#     # Interpolate difference onto a grid
#     grid_height, grid_width = grid_shape
#     xi, yi = np.meshgrid(
#         np.linspace(transform.c, transform.c + transform.a * grid_width, grid_wid
#         np.linspace(transform.f, transform.f + transform.e * grid_height, grid_he
#     )
#     diff_grid = griddata(
#         (points.geometry.x, points.geometry.y), points['diff'],
#         (xi, yi), method='nearest'
#     )
#     diff_grid[~hull_mask] = np.nan # Apply hull mask
#
#     # Classify differences based on NMAD tolerance
#     tolerance = metrics['nmad']
#     with np.errstate(invalid='ignore'): # Ignore warnings from comparing with NaN
#         higher_mask = diff_grid > tolerance
#         lower_mask = diff_grid < -tolerance
#
#     # Generate statistics
#     n_valid = np.count_nonzero(~np.isnan(diff_grid))
#     stats_dict = {
#         'n_total': n_valid,
#         'n_higher': np.sum(higher_mask), 'pct_higher': 100 * np.sum(higher_mask) /
#         'n_lower': np.sum(lower_mask), 'pct_lower': 100 * np.sum(lower_mask) / n_
#     }
#     return diff_grid, higher_mask, lower_mask, stats_dict, points
#
# def plot_gridded_comparison(axes, data, title, cmap, vmin=None, vmax=None, stats_
#     """Helper function to plot a single difference map."""
#     ax = axes
#     ax.set_facecolor('white')
#     cmap.set_bad(color='white', alpha=0)
#     im = ax.imshow(data, cmap=cmap, origin='upper', extent=[xmin_grid, xmax_grid,
#     hull_gdf.boundary.plot(ax=ax, color='black', linewidth=1.5)
#     plt.colorbar(im, ax=ax, label='Difference (m)', shrink=0.8)
#     ax.set_title(title, fontweight='bold', fontsize=12)
#     ax.set_xlabel('UTM Easting (m)')
#     ax.set_ylabel('UTM Northing (m)')
#     ax.grid(True, alpha=0.3)
#     if stats_text:
#         ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=9, va='t
#                 bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, ec='bla
#
```

```
# # --- Main Plotting Logic for Gridded Comparison ---
# fig, axes = plt.subplots(2, 3, figsize=(20, 14), facecolor='white')
#
# # Define datasets to compare
# comparisons = [
#     {'name': 'TINITALY', 'h_col': 'HEIGHT_ABSOLUTE_TIN', 'r_col': 'tinitaly_heigh
#     {'name': 'Copernicus', 'h_col': 'HEIGHT_ABSOLUTE_COP', 'r_col': 'copernicus_h
# ]
#
# for i, p in enumerate(comparisons):
#     diff_grid, higher, lower, stats, points = process_dem_comparison(
#         saocom_gdf, p['h_col'], p['r_col'], (grid_height, grid_width), target_tra
#     )
#
#     # Plot 1: Full Difference
#     diff_limit = np.percentile(np.abs(points['diff']), 95)
#     stats1 = f"Points: {stats['n_total']:,}\nMean: {p['metrics']['mean_diff']:+.2
#     plot_gridded_comparison(axes[i, 0], diff_grid, f"SAOCOM - {p['name']}\nFull D
#
#     # Plot 2: SAOCOM Higher
#     higher_grid = np.where(higher, diff_grid, np.nan)
#     stats2 = f"Points: {stats['n_higher']:,}\nMean: {np.nanmean(higher_grid):.2f}
#     plot_gridded_comparison(axes[i, 1], higher_grid, f"SAOCOM > {p['name']}\n({st
#
#     # Plot 3: SAOCOM Lower
#     lower_grid = np.where(lower, diff_grid, np.nan)
#     stats3 = f"Points: {stats['n_lower']:,}\nMean: {np.nanmean(lower_grid):.2f}m\
#     plot_gridded_comparison(axes[i, 2], lower_grid, f"{p['name']} > SAOCOM\n({sta
#
# plt.tight_layout()
# plt.savefig(RESULTS_DIR / 'saocom_comparison_directional.png', dpi=300, bbox_inch
# plt.show()
#
#
# # ============================================================================
# # 2. RESIDUAL DISTRIBUTION HISTOGRAMS
# # ============================================================================
#
# def plot_residual_histogram(ax, diff_series, metrics, tolerance, title):
#     """Helper function to plot a single residual distribution histogram."""
#     ax.hist(diff_series, bins=50, alpha=0.7, color='steelblue', edgecolor='black'
#     ax.axvline(0, color='red', ls='--', lw=2, label='Zero')
#     ax.axvline(metrics['mean_diff'], color='green', ls='-', lw=2, label=f"Mean: {
#     ax.axvline(tolerance, color='orange', ls='--', lw=1.5, label=f"±NMAD: {tolera
#     ax.axvline(-tolerance, color='orange', ls='--', lw=1.5)
#
#     stats_text = f"n = {metrics['n_points']:,}\nRMSE = {metrics['rmse']:.2f}m\nNM
#     ax.text(0.98, 0.97, stats_text, transform=ax.transAxes, fontsize=9, va='top',
#             bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, ec='black')
#
#     ax.set(xlabel='Elevation Difference (m)', ylabel='Frequency (Log Scale)',
#            title=title, yscale='log')
#     ax.legend(loc='upper right', fontsize=10)
#     ax.grid(True, alpha=0.3)
#
# fig, axes = plt.subplots(1, 2, figsize=(18, 7), facecolor='white')
```

```
# plot_residual_histogram(axes[0], saocom_tinitaly_diff, saocom_tinitaly_metrics, s
# plot_residual_histogram(axes[1], saocom_copernicus_diff, saocom_copernicus_metric
# plt.tight_layout()
# plt.savefig(RESULTS_DIR / 'saocom_residual_distributions.png', dpi=300, bbox_inch
# plt.show()
#
#
# # ============================================================================
# # 3. ENSEMBLE OUTLIER DETECTION
# # ============================================================================
#
# # Filter for valid heights and run detection methods
# valid_saocom = saocom_gdf[saocom_gdf['HEIGHT_ABSOLUTE_TIN'].notna()].copy()
# heights = valid_saocom['HEIGHT_ABSOLUTE_TIN']
#
# # Method 1: IQR (3x multiplier for extreme outliers)
# q1, q3 = heights.quantile(0.25), heights.quantile(0.75)
# iqr = q3 - q1
# outliers_iqr = (heights < (q1 - 3 * iqr)) | (heights > (q3 + 3 * iqr))
#
# # Method 2: Z-score
# outliers_zscore = np.abs(stats.zscore(heights)) > 3
#
# # Method 3: Modified Z-score (NMAD-based)
# median_abs_dev = np.median(np.abs(heights - heights.median()))
# mod_z_scores = 0.6745 * (heights - heights.median()) / median_abs_dev
# outliers_nmad = np.abs(mod_z_scores) > 3.5
#
# # Combine methods: outlier if flagged by at least 2
# is_outlier = np.sum([outliers_iqr, outliers_zscore, outliers_nmad], axis=0) >= 2
# valid_saocom['is_outlier'] = is_outlier
# normal_points = valid_saocom[~is_outlier]
# outlier_points = valid_saocom[is_outlier]
#
# print(f"Ensemble outlier detection complete. Found {len(outlier_points)} outliers
#
#
# # ============================================================================
# # 4. OUTLIER VISUALIZATION & ANALYSIS
# # ============================================================================
#
# fig = plt.figure(figsize=(18, 16), facecolor='white')
# gs = fig.add_gridspec(2, 2)
#
# # Plot 1: Spatial Distribution
# ax1 = fig.add_subplot(gs[0, 0])
# normal_points.plot(ax=ax1, markersize=0.5, color='#2E86AB', alpha=0.4, label=f'No
# if not outlier_points.empty:
#     outlier_points.plot(ax=ax1, markersize=8, color='#E63946', alpha=0.9, ec='bla
# hull_gdf.boundary.plot(ax=ax1, color='black', lw=2, ls='--')
# ax1.set(title='SAOCOM Height Outliers - Spatial Distribution', xlabel='UTM Eastin
# ax1.legend(); ax1.grid(True, alpha=0.3)
#
# # Plot 2: Height Distribution
# ax2 = fig.add_subplot(gs[0, 1])
# ax2.hist(normal_points['HEIGHT_ABSOLUTE_TIN'], bins=50, alpha=0.6, color='#2E86AB
```

```
# if not outlier_points.empty:
#     ax2.hist(outlier_points['HEIGHT_ABSOLUTE_TIN'], bins=20, alpha=0.8, color='#E
# ax2.axvline(q1 - 3 * iqr, color='orange', ls='--', label=f'IQR Bound')
# ax2.axvline(q3 + 3 * iqr, color='orange', ls='--')
# ax2.set(title='Height Distribution with Outlier Thresholds', xlabel='Height (m)',
# ax2.legend(); ax2.grid(True, alpha=0.3)
#
# # Plot 3: Box Plot
# ax3 = fig.add_subplot(gs[1, 0])
# box_data = [normal_points['HEIGHT_ABSOLUTE_TIN'].dropna()]
# if not outlier_points.empty:
#     box_data.append(outlier_points['HEIGHT_ABSOLUTE_TIN'].dropna())
# bp = ax3.boxplot(box_data, patch_artist=True, showmeans=True, meanline=True,
#                  labels=['Normal', 'Outliers'] if not outlier_points.empty else [
# colors = ['#2E86AB', '#E63946']
# for patch, color in zip(bp['boxes'], colors):
#     patch.set_facecolor(color)
# ax3.set(title='Height Distribution Comparison', ylabel='Height (m)')
# ax3.grid(True, axis='y', alpha=0.3)
#
# # Plot 4: Summary Text
# ax4 = fig.add_subplot(gs[1, 1])
# ax4.axis('off')
# summary_text = f"""OUTLIER ANALYSIS SUMMARY
# ----------------------------------
# Flagged by IQR Method: {np.sum(outliers_iqr):,}
# Flagged by Z-score: {np.sum(outliers_zscore):,}
# Flagged by Mod. Z-score: {np.sum(outliers_nmad):,}
# ----------------------------------
# Final Outliers (≥2 methods): {len(outlier_points):,} ({len(outlier_points)/len(va
# Normal Points: {len(normal_points):,}
# ----------------------------------
# Normal Mean Height: {normal_points['HEIGHT_ABSOLUTE_TIN'].mean():.2f} m
# Outlier Mean Height: {outlier_points['HEIGHT_ABSOLUTE_TIN'].mean():.2f} m
# """
# ax4.text(0.05, 0.5, summary_text, va='center', fontfamily='monospace',
#          bbox=dict(boxstyle='round', facecolor='white', ec='black'))
#
# plt.tight_layout()
# plt.savefig(RESULTS_DIR / 'saocom_height_outliers_ensemble.png', dpi=300, bbox_in
# plt.show()
#
# # ============================================================================
# # 5. SCATTER PLOT COMPARISONS
# # ============================================================================
#
# fig, axes = plt.subplots(2, 2, figsize=(18, 16), facecolor='white')
# axes[1, 1].axis('off') # Hide unused subplot
#
# # Define scatter plot configurations
# scatter_plots = [
#     {'ax': axes[0, 0], 'x': saocom_tinitaly_valid['tinitaly_height'], 'y': saocom_
#      'title': 'SAOCOM vs TINITALY', 'xlabel': 'TINITALY Height (m)', 'metrics': s
#     {'ax': axes[0, 1], 'x': saocom_copernicus_valid['copernicus_height'], 'y': sa
#      'title': 'SAOCOM vs Copernicus', 'xlabel': 'Copernicus Height (m)', 'metrics
#     {'ax': axes[1, 0], 'x': valid_copernicus, 'y': valid_tinitaly,
```

```
#      'title': 'TINITALY vs Copernicus', 'xlabel': 'Copernicus Height (m)', 'ylabe
# ]
#
# for p in scatter_plots:
#     ax = p['ax']
#     ax.scatter(p['x'], p['y'], s=1, alpha=0.3, c='steelblue', ec='none')
#     min_val, max_val = min(p['x'].min(), p['y'].min()), max(p['x'].max(), p['y'].
#     ax.plot([min_val, max_val], [min_val, max_val], 'r--', lw=2, label='1:1 line'
#
#     stats_text = (f"n = {p['metrics'].get('n_points', p['metrics'].get('n_pixels'
#                   f"Bias = {p['metrics']['mean_diff']:.2f}m\nRMSE = {p['metrics']
#                   f"NMAD = {p['metrics']['nmad']:.2f}m\nCorr (r) = {p['metrics'][
#
#     ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=9, va='top',
#             bbox=dict(boxstyle='round', facecolor='white', alpha=0.9, ec='black')
#
#     ax.set(title=p['title'], xlabel=p['xlabel'], ylabel=p.get('ylabel', 'SAOCOM H
#     ax.legend(); ax.grid(True, alpha=0.3)
#
# plt.tight_layout()
# plt.show()
```

VOID ZONES vs LAND COVER ANALYSIS

In [30]:

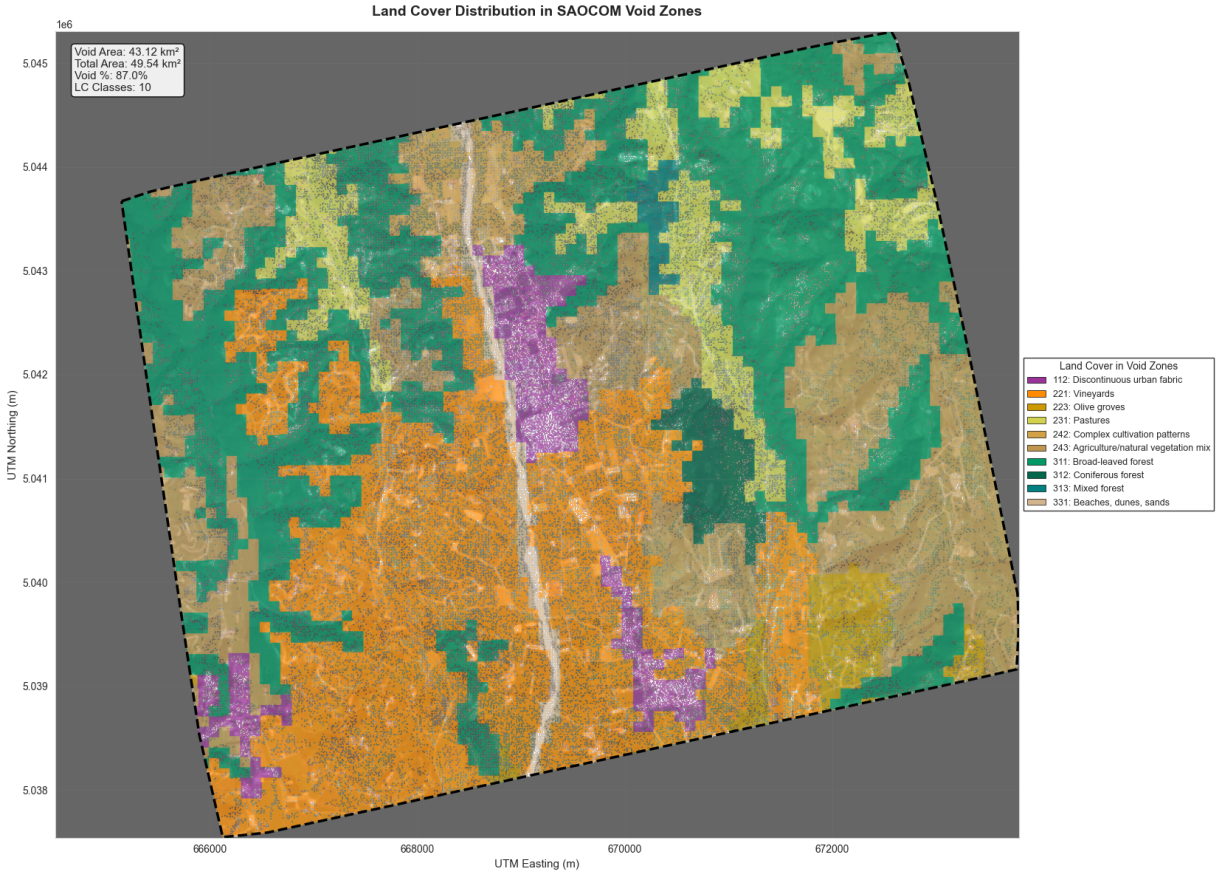
```
# ============================================================================
# VOID ZONES vs LAND COVER ANALYSIS
# ============================================================================

# ============================================================================
# 1. CALCULATE VOID STATISTICS BY LAND COVER
# ============================================================================
void_lc_stats = []

for lc_code in np.unique(corine_10m[corine_10m > 0]):
    lc_mask = (corine_10m == lc_code) & study_area_mask

    total_lc_cells = np.sum(lc_mask)
    void_lc_cells = np.sum(lc_mask & void_mask)

    if total_lc_cells == 0:
        continue

    pct_lc_is_void = 100 * void_lc_cells / total_lc_cells
    pct_of_total_voids = 100 * void_lc_cells / np.sum(void_mask)

    void_lc_stats.append({
        'LC_Code': lc_code,
        'LC_Name': CORINE_CLASSES.get(lc_code, f'Unknown_{lc_code}'),
        'Total_Cells': total_lc_cells,
        'Void_Cells': void_lc_cells,
        'Area_km2': total_lc_cells * (GRID_SIZE**2) / 1e6,
        'Void_Area_km2': void_lc_cells * (GRID_SIZE**2) / 1e6,
        'Pct_LC_is_Void': pct_lc_is_void,
        'Pct_of_Total_Voids': pct_of_total_voids
    })
```

```
void_lc_df = pd.DataFrame(void_lc_stats).sort_values('Pct_LC_is_Void', ascending=Fa

# Display table
print(f"\n{'='*120}")
print("VOID ZONES BY LAND COVER CLASS")
print(f"{'='*120}")
print(void_lc_df[['LC_Code', 'LC_Name', 'Area_km2', 'Void_Area_km2', 'Pct_LC_is_Voi
print(f"{'='*120}\n")

# ============================================================================
# 2. MAP: VOID ZONES WITH LAND COVER OVERLAY
# ============================================================================
fig, ax = plt.subplots(1, 1, figsize=(16, 12), facecolor='white')
ax.set_facecolor('white')

extent = [xmin_grid, xmax_grid, ymin_grid, ymax_grid]

# Sentinel background
ax.imshow(sentinel_rgb_norm, extent=extent, origin='upper', alpha=0.6)

# Land cover in void zones only
lc_in_voids = corine_10m.copy()
lc_in_voids[~void_mask] = 0
lc_display = np.ma.masked_where(lc_in_voids == 0, lc_in_voids)

void_codes = np.unique(lc_display.compressed())

if len(void_codes) > 0:
    colors_list = [CORINE_COLORS_MPL.get(code, (0.5, 0.5, 0.5)) for code in void_co
    cmap = ListedColormap(colors_list)
    norm = BoundaryNorm(boundaries=np.append(void_codes, void_codes[-1]+1) - 0.5,
                        ncolors=len(void_codes))

    im = ax.imshow(lc_display, cmap=cmap, norm=norm, origin='upper',
                   extent=extent, alpha=0.7)

# Study area boundary
hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--')

# Statistics box
void_area = np.sum(void_mask) * (GRID_SIZE**2) / 1e6
total_area = np.sum(study_area_mask) * (GRID_SIZE**2) / 1e6
pct_void = 100 * np.sum(void_mask) / np.sum(study_area_mask)

stats_text = f"""Void Area: {void_area:.2f} km²
Total Area: {total_area:.2f} km²
Void %: {pct_void:.1f}%
LC Classes: {len(void_codes)}"""

ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=11,
        verticalalignment='top', bbox=dict(boxstyle='round', facecolor='white',
        alpha=0.9, edgecolor='black'))

# Legend
if len(void_codes) > 0:
```

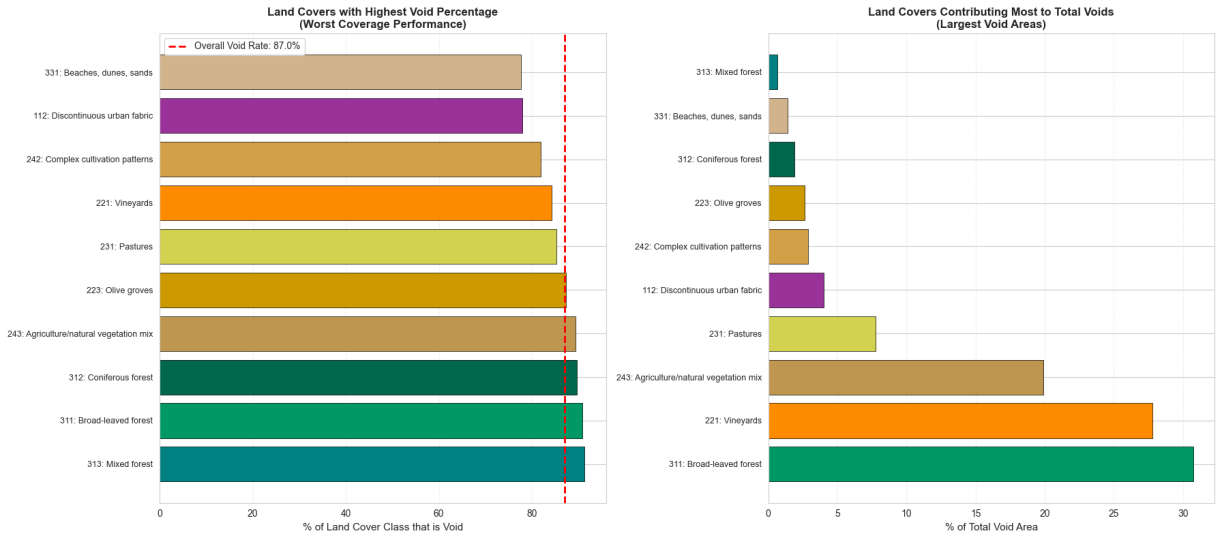
```
    legend_elements = [mpatches.Rectangle((0,0),1,1,
                                          facecolor=CORINE_COLORS_MPL[code],
                                          edgecolor='black', linewidth=0.5,
                                          label=f"{code}: {CORINE_CLASSES.get(code,
                       for code in sorted(void_codes)]

    ax.legend(handles=legend_elements, loc='center left', bbox_to_anchor=(1, 0.5),
              fontsize=9, frameon=True, fancybox=False, edgecolor='black',
              title='Land Cover in Void Zones')

ax.set_title('Land Cover Distribution in SAOCOM Void Zones',
             fontweight='bold', fontsize=14, pad=15)
ax.set_xlabel('UTM Easting (m)', fontsize=11)
ax.set_ylabel('UTM Northing (m)', fontsize=11)
ax.grid(True, alpha=0.3, color='gray', linewidth=0.5)

plt.tight_layout()
plt.savefig(RESULTS_DIR / 'voids_by_landcover_map.png', dpi=300, bbox_inches='tight
plt.show()

print("Saved: voids_by_landcover_map.png")

# ============================================================================
# 3. BAR CHART: WHICH LAND COVERS HAVE THE MOST VOIDS
# ============================================================================
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8), facecolor='white')

# Chart 1: % of each land cover that is void
top_pct = void_lc_df.nlargest(15, 'Pct_LC_is_Void')
bars1 = ax1.barh(range(len(top_pct)), top_pct['Pct_LC_is_Void'])

for i, (_, row) in enumerate(top_pct.iterrows()):
    bars1[i].set_color(CORINE_COLORS_MPL.get(row['LC_Code'], (0.5, 0.5, 0.5)))
    bars1[i].set_edgecolor('black')
    bars1[i].set_linewidth(0.5)

ax1.set_yticks(range(len(top_pct)))
ax1.set_yticklabels([f"{row['LC_Code']}: {row['LC_Name'][:35]}"
                     for _, row in top_pct.iterrows()], fontsize=9)
ax1.set_xlabel('% of Land Cover Class that is Void', fontsize=11)
ax1.set_title('Land Covers with Highest Void Percentage\n(Worst Coverage Performanc
              fontweight='bold', fontsize=12)
ax1.grid(axis='x', alpha=0.3, linestyle='--')
ax1.axvline(pct_void, color='red', linestyle='--', linewidth=2,
            label=f'Overall Void Rate: {pct_void:.1f}%')
ax1.legend()

# Chart 2: Contribution to total voids
top_contrib = void_lc_df.nlargest(15, 'Pct_of_Total_Voids')
bars2 = ax2.barh(range(len(top_contrib)), top_contrib['Pct_of_Total_Voids'])

for i, (_, row) in enumerate(top_contrib.iterrows()):
    bars2[i].set_color(CORINE_COLORS_MPL.get(row['LC_Code'], (0.5, 0.5, 0.5)))
    bars2[i].set_edgecolor('black')
    bars2[i].set_linewidth(0.5)
```

```
ax2.set_yticks(range(len(top_contrib)))
ax2.set_yticklabels([f"{row['LC_Code']}: {row['LC_Name'][:35]}"
                     for _, row in top_contrib.iterrows()], fontsize=9)
ax2.set_xlabel('% of Total Void Area', fontsize=11)
ax2.set_title('Land Covers Contributing Most to Total Voids\n(Largest Void Areas)',
              fontweight='bold', fontsize=12)
ax2.grid(axis='x', alpha=0.3, linestyle='--')

plt.tight_layout()
plt.savefig(RESULTS_DIR / 'voids_by_landcover_charts.png', dpi=300, bbox_inches='ti
plt.show()

print("Saved: voids_by_landcover_charts.png")
print("\nVoid analysis complete!")
```

```
================================================================================
==================================
VOID ZONES BY LAND COVER CLASS
================================================================================
==================================
 LC_Code                            LC_Name  Area_km2  Void_Area_km2  Pct_LC_is_Void
Pct_of_Total_Voids
     313                       Mixed forest    0.3097         0.2828       91.314175
0.655879
     311                Broad-leaved forest   14.5910        13.2547       90.841615
30.740740
     312                  Coniferous forest    0.9201         0.8249       89.653299
1.913135
     243 Agriculture/natural vegetation mix    9.6012         8.5772       89.334667
19.892527
     223                       Olive groves    1.2945         1.1300       87.292391
2.620733
     231                           Pastures    3.9204         3.3421       85.248954
7.751109
     221                          Vineyards   14.2358        11.9813       84.163166
27.787428
     242       Complex cultivation patterns    1.5060         1.2342       81.952191
2.862398
     112         Discontinuous urban fabric    2.2198         1.7299       77.930444
4.012041
     331              Beaches, dunes, sands    0.7752         0.6019       77.644479
1.395946
================================================================================
==================================
```




```
Saved: voids_by_landcover_map.png
```




```
Saved: voids_by_landcover_charts.png

Void analysis complete!
```

VOID ZONES vs LAND COVER - "SWISS CHEESE" VISUALIZATION

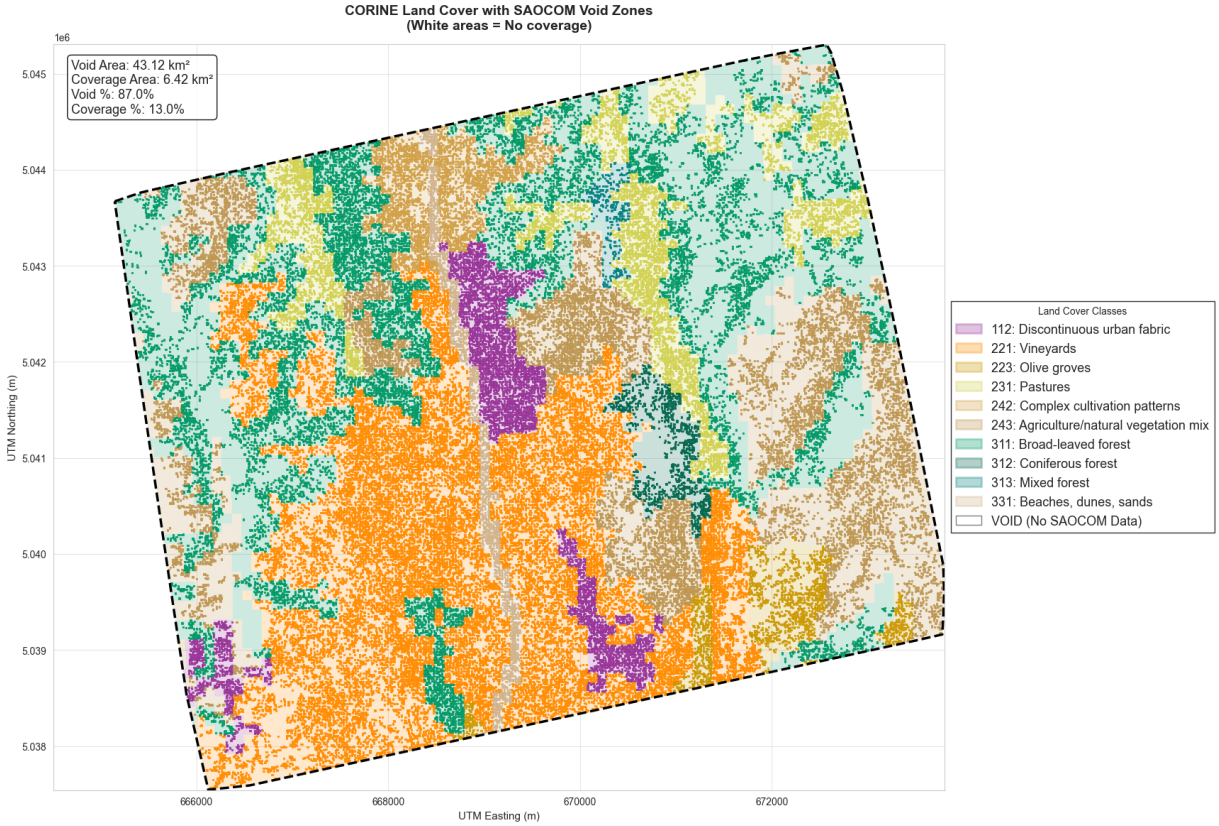
In [31]:
```
# ================================================================================
# VOID ZONES vs LAND COVER - "SWISS CHEESE" VISUALIZATION
# ================================================================================

# ================================================================================
# LAND COVER WITH VOIDS - OUTLINED POLYGONS VERSION
```

```
# ============================================================================



fig, ax = plt.subplots(1, 1, figsize=(16, 12), facecolor='white')
ax.set_facecolor('white')

extent = [xmin_grid, xmax_grid, ymin_grid, ymax_grid]

# Get unique land cover classes
display_codes = np.unique(corine_10m[(corine_10m > 0) & ~void_mask])

print("Vectorizing land cover polygons (this may take a moment)...")
im = ax.imshow(corine_display, cmap=cmap, norm=norm, origin='upper', extent=extent,

# Process each land cover class
for lc_code in sorted(display_codes):
    # Create mask: this land cover AND has SAOCOM coverage (not void)
    lc_with_coverage = (corine_10m == lc_code) & (~void_mask)

    # Vectorize to get boundaries
    mask_shapes = shapes(lc_with_coverage.astype(np.uint8),
                         mask=lc_with_coverage,
                         transform=target_transform)

    # Convert to shapely polygons
    polys = [shape(geom) for geom, val in mask_shapes if val == 1]

    # Get color for this land cover
    fill_color = CORINE_COLORS_MPL.get(lc_code, (0.5, 0.5, 0.5))

    # Draw each polygon
    for poly in polys:
        if poly.is_valid:
            x, y = poly.exterior.xy

            # Very faint fill
            ax.fill(x, y, color=fill_color, alpha=0.15, edgecolor='none', zorder=1)

            # Colored outline
            ax.plot(x, y, color=fill_color, linewidth=1.5, alpha=0.9, zorder=2)

# Study area boundary (on top)
hull_gdf.boundary.plot(ax=ax, color='black', linewidth=2.5, linestyle='--', zorder=

# Statistics
void_area = np.sum(void_mask) * (GRID_SIZE**2) / 1e6
total_area = np.sum(study_area_mask) * (GRID_SIZE**2) / 1e6
pct_void = 100 * np.sum(void_mask) / np.sum(study_area_mask)

stats_text = f"""Void Area: {void_area:.2f} km²
Coverage Area: {total_area - void_area:.2f} km²
Void %: {pct_void:.1f}%
Coverage %: {100 - pct_void:.1f}%"""

ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=13,
```

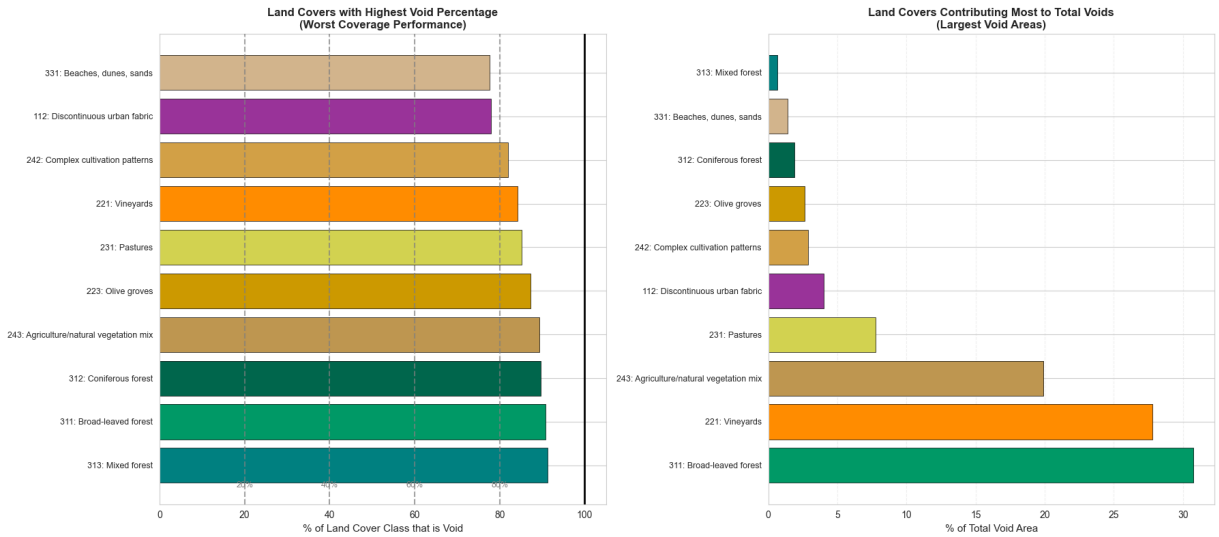
```
        verticalalignment='top', bbox=dict(boxstyle='round', facecolor='white',
        alpha=0.9, edgecolor='black'), zorder=4)
# Legend
legend_elements = []
for code in sorted(display_codes):
    color = CORINE_COLORS_MPL.get(code, (0.5, 0.5, 0.5))
    legend_elements.append(mpatches.Rectangle((0,0),1,1,
                                              facecolor=color,
                                              edgecolor=color,
                                              alpha=0.3,
                                              linewidth=2,
                                              label=f"{code}: {CORINE_CLASSES.get(co

legend_elements.append(mpatches.Rectangle((0,0),1,1,
                                          facecolor='white',
                                          edgecolor='gray',
                                          linewidth=1,
                                          label='VOID (No SAOCOM Data)'))

ax.legend(handles=legend_elements, loc='center left', bbox_to_anchor=(1, 0.5),
         fontsize=13, frameon=True, fancybox=False, edgecolor='black',
         title='Land Cover Classes')

ax.set_title('CORINE Land Cover with SAOCOM Void Zones\n(White areas = No coverage)
             fontweight='bold', fontsize=14, pad=15)
ax.set_xlabel('UTM Easting (m)', fontsize=11)
ax.set_ylabel('UTM Northing (m)', fontsize=11)
ax.set_xlim(extent[0], extent[1])
ax.set_ylim(extent[2], extent[3])
ax.grid(True, alpha=0.3, color='gray', linewidth=0.5)

plt.tight_layout()
plt.savefig(RESULTS_DIR / 'landcover_with_voids_outlined.png', dpi=300, bbox_inches
plt.show()

print("Saved: landcover_with_voids_outlined.png")

# ============================================================================
# 3. BAR CHART WITH REFERENCE LINES
# ============================================================================
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8), facecolor='white')

# Chart 1: % of each land cover that is void
top_pct = void_lc_df.nlargest(15, 'Pct_LC_is_Void')
bars1 = ax1.barh(range(len(top_pct)), top_pct['Pct_LC_is_Void'])

for i, (_, row) in enumerate(top_pct.iterrows()):
    bars1[i].set_color(CORINE_COLORS_MPL.get(row['LC_Code'], (0.5, 0.5, 0.5)))
    bars1[i].set_edgecolor('black')
    bars1[i].set_linewidth(0.5)

ax1.set_yticks(range(len(top_pct)))
ax1.set_yticklabels([f"{row['LC_Code']}: {row['LC_Name'][:35]}"
                     for _, row in top_pct.iterrows()], fontsize=9)
ax1.set_xlabel('% of Land Cover Class that is Void', fontsize=11)
ax1.set_title('Land Covers with Highest Void Percentage\n(Worst Coverage Performanc
```

```
             fontweight='bold', fontsize=12)
ax1.grid(axis='x', alpha=0.3, linestyle='--')

# Add reference lines
ax1.axvline(100, color='black', linestyle='-', linewidth=2, label='100% (Total Void
for pct in [20, 40, 60, 80]:
    ax1.axvline(pct, color='gray', linestyle='--', linewidth=1.5, alpha=0.7, zorder
    ax1.text(pct, -0.5, f'{pct}%', ha='center', fontsize=9, color='gray')

ax1.set_xlim(0, 105)
# ax1.legend(loc='lower right')

# Chart 2: Contribution to total voids
top_contrib = void_lc_df.nlargest(15, 'Pct_of_Total_Voids')
bars2 = ax2.barh(range(len(top_contrib)), top_contrib['Pct_of_Total_Voids'])

for i, (_, row) in enumerate(top_contrib.iterrows()):
    bars2[i].set_color(CORINE_COLORS_MPL.get(row['LC_Code'], (0.5, 0.5, 0.5)))
    bars2[i].set_edgecolor('black')
    bars2[i].set_linewidth(0.5)

ax2.set_yticks(range(len(top_contrib)))
ax2.set_yticklabels([f"{row['LC_Code']}: {row['LC_Name'][:35]}"
                     for _, row in top_contrib.iterrows()], fontsize=9)
ax2.set_xlabel('% of Total Void Area', fontsize=11)
ax2.set_title('Land Covers Contributing Most to Total Voids\n(Largest Void Areas)',
             fontweight='bold', fontsize=12)
ax2.grid(axis='x', alpha=0.3, linestyle='--')

plt.tight_layout()
plt.savefig(RESULTS_DIR / 'voids_by_landcover_charts.png', dpi=300, bbox_inches='ti
plt.show()

print("Saved: voids_by_landcover_charts.png")
print("\nVoid analysis complete!")
```

```
Vectorizing land cover polygons (this may take a moment)...
```




```
Saved: landcover_with_voids_outlined.png
```




```
Saved: voids_by_landcover_charts.png

Void analysis complete!
```

INDIVIDUAL LAND COVER MAPS WITH VOID VISUALIZATION

In [32]:

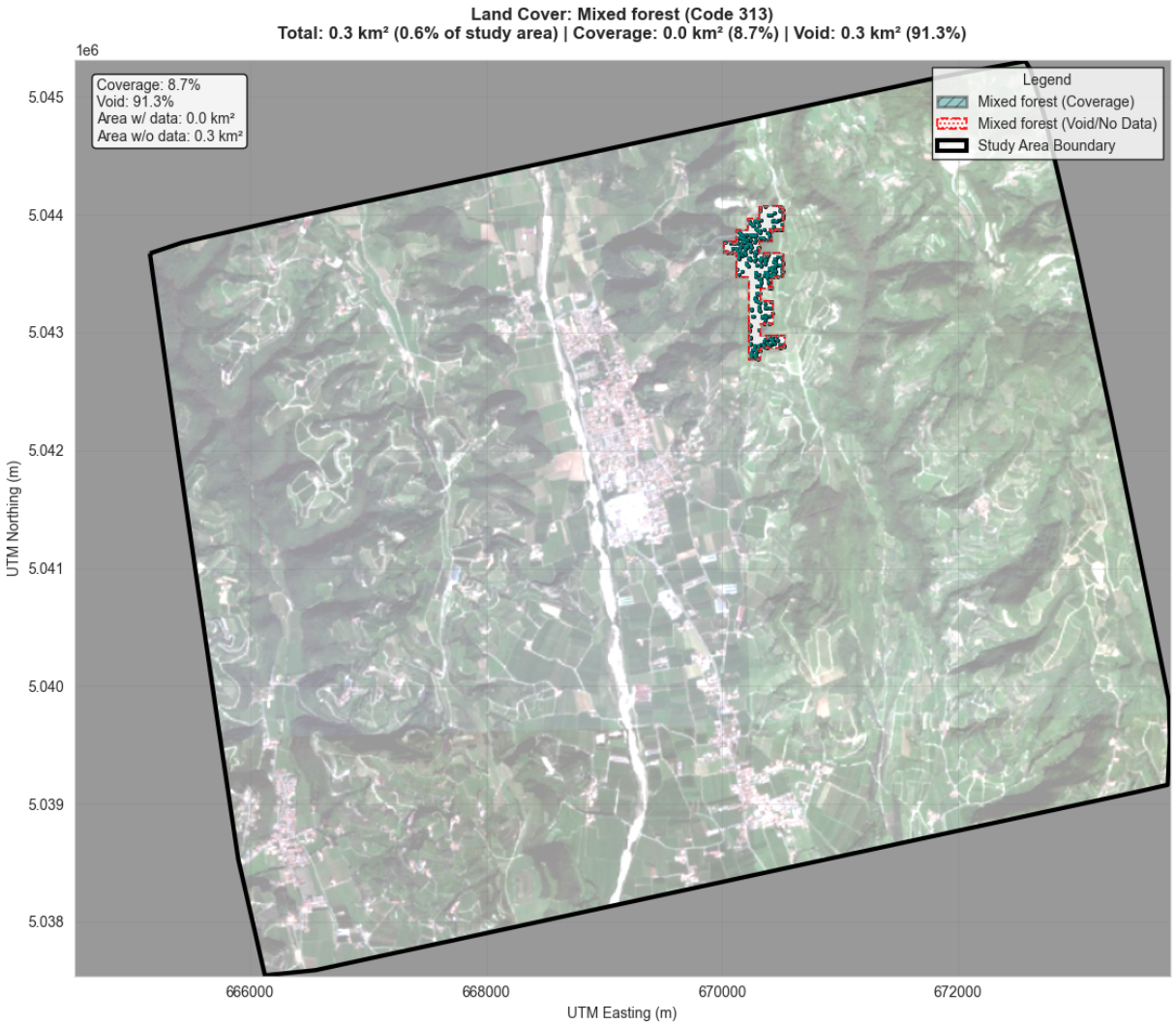
```
# ============================================================================
# 1. PRE-CALCULATE ALL STATISTICS (DERIVED FROM "SWISS CHEESE" DATA)
# ============================================================================
import pandas as pd
import numpy as np

print("Calculating statistics for all land cover classes...")
```

```
# Get unique classes present in the data
unique_classes = np.unique(corine_10m[corine_10m > 0])

summary_data = []
for lc_code in sorted(unique_classes):
    # Create masks from the main "swiss cheese" arrays
    lc_mask = (corine_10m == lc_code)
    lc_total_pixels = np.sum(lc_mask)

    if lc_total_pixels > 0:
        lc_void_pixels = np.sum(lc_mask & void_mask)
        lc_coverage_pixels = lc_total_pixels - lc_void_pixels

        # Calculate statistics
        total_area_km2 = lc_total_pixels * (GRID_SIZE**2) / 1e6
        coverage_area_km2 = lc_coverage_pixels * (GRID_SIZE**2) / 1e6
        void_area_km2 = lc_void_pixels * (GRID_SIZE**2) / 1e6

        pct_coverage = 100 * lc_coverage_pixels / lc_total_pixels
        pct_void = 100 * lc_void_pixels / lc_total_pixels
        pct_of_study_area = 100 * lc_total_pixels / np.sum(corine_10m > 0)

        summary_data.append({
            'LC_Code': lc_code,
            'LC_Name': CORINE_CLASSES.get(lc_code, f'Class {lc_code}'),
            'Total_km2': total_area_km2,
            'Coverage_km2': coverage_area_km2,
            'Void_km2': void_area_km2,
            'Pct_Coverage': pct_coverage,
            'Pct_Void': pct_void,
            'Pct_of_Study_Area': pct_of_study_area
        })

# Create the master DataFrame with all stats
stats_df = pd.DataFrame(summary_data)
stats_df = stats_df.sort_values(by='Pct_Void', ascending=False) # Sort by worst cov

print("✓ Statistics DataFrame created.")
print(stats_df.to_string(index=False, float_format="%.2f"))

print(f"\nGenerating {len(stats_df)} land cover maps using pre-calculated stats..."

# Loop through the DataFrame, one row per land cover class
for index, row in stats_df.iterrows():
    lc_code = row['LC_Code']
    class_name = row['LC_Name']

    fig, ax = plt.subplots(1, 1, figsize=(14, 10), facecolor='white')
    ax.set_facecolor('white')

    # Display Sentinel RGB as background
    ax.imshow(sentinel_rgb_norm, extent=[xmin_grid, xmax_grid, ymin_grid, ymax_grid
              origin='upper', alpha=0.4)

    fill_color = tuple(c/255 for c in CORINE_COLORS.get(lc_code, (128, 128, 128)))
```

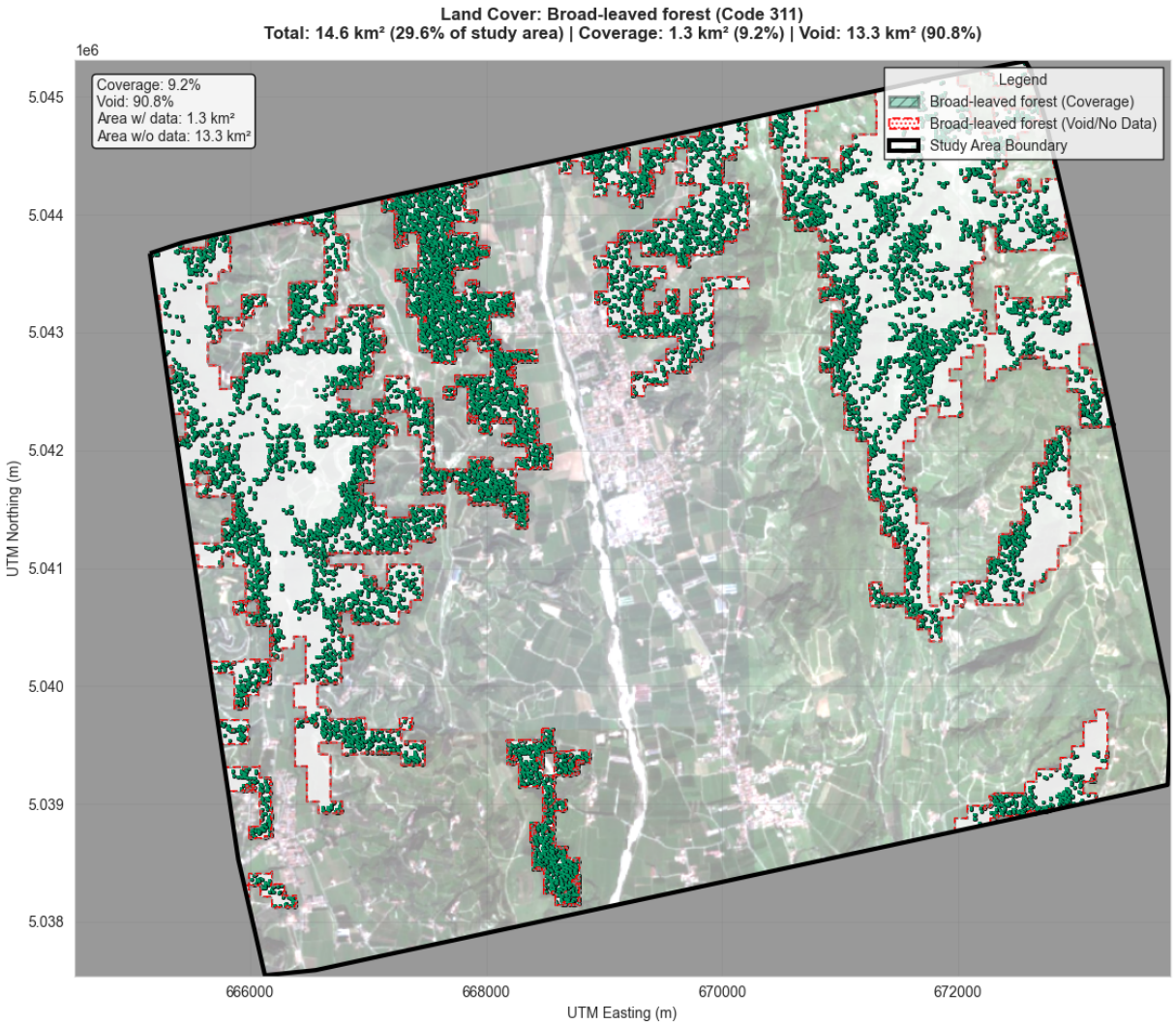
```
    # --- Visualization (Masking and Vectorizing is still required here) ---
    lc_mask = (corine_10m == lc_code)

    # 1. Draw COVERAGE areas (with data)
    lc_with_coverage = lc_mask & (~void_mask)
    if np.any(lc_with_coverage):
        coverage_shapes = shapes(lc_with_coverage.astype(np.uint8), mask=lc_with_co
        for geom, val in coverage_shapes:
            if val == 1:
                poly = shape(geom)
                if poly.is_valid:
                    x, y = poly.exterior.xy
                    ax.fill(x, y, color=fill_color, alpha=0.35, edgecolor='none', h
                    ax.plot(x, y, color='black', linewidth=2.0)
                    ax.plot(x, y, color=fill_color, linewidth=1.2)

    # 2. Draw VOID areas (no data)
    lc_void = lc_mask & void_mask
    if np.any(lc_void):
        void_shapes = shapes(lc_void.astype(np.uint8), mask=lc_void, transform=targ
        for geom, val in void_shapes:
            if val == 1:
                poly = shape(geom)
                if poly.is_valid:
                    x, y = poly.exterior.xy
                    ax.fill(x, y, color='white', alpha=0.8, edgecolor='none', hatch
                    ax.plot(x, y, color='red', linewidth=1.5, linestyle='--')
                    ax.plot(x, y, color='gray', linewidth=0.8, linestyle='--')

    hull_gdf.boundary.plot(ax=ax, color='black', linewidth=3, linestyle='-')

    # --- Use Pre-Calculated Stats for Title and Text ---
    ax.set_title(
        f"Land Cover: {class_name} (Code {lc_code})\n"
        f"Total: {row['Total_km2']:.1f} km² ({row['Pct_of_Study_Area']:.1f}% of stu
        f"Coverage: {row['Coverage_km2']:.1f} km² ({row['Pct_Coverage']:.1f}%) | "
        f"Void: {row['Void_km2']:.1f} km² ({row['Pct_Void']:.1f}%)",
        fontweight='bold', fontsize=12, pad=15
    )

    stats_text = (
        f"Coverage: {row['Pct_Coverage']:.1f}%\n"
        f"Void: {row['Pct_Void']:.1f}%\n"
        f"Area w/ data: {row['Coverage_km2']:.1f} km²\n"
        f"Area w/o data: {row['Void_km2']:.1f} km²"
    )
    ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=10,
            verticalalignment='top', bbox=dict(boxstyle='round', facecolor='white',

    # --- Legend ---
    legend_elements = [
        Patch(facecolor=fill_color, edgecolor='black', linewidth=2, alpha=0.35, hat
        Patch(facecolor='white', edgecolor='red', linewidth=1.5, alpha=0.8, hatch='
        Patch(facecolor='none', edgecolor='black', linewidth=3, label='Study Area B
    ]
    ax.legend(handles=legend_elements, loc='upper right', fontsize=10, frameon=True
```

```
    ax.set_xlabel('UTM Easting (m)')
    ax.set_ylabel('UTM Northing (m)')
    ax.grid(True, alpha=0.3, color='gray', linewidth=0.5)
    plt.tight_layout()

    # --- Save File ---
    safe_name = class_name.replace(' ', '_').replace(',', '').replace('/', '_')
    filename = f'landcover_{lc_code}_{safe_name}_coverage_void.png'
    plt.savefig(RESULTS_DIR / filename, dpi=300, bbox_inches='tight', facecolor='wh
    plt.show()
    plt.close()

    print(f"Saved: {filename}")

print(f"\n✓ Generated {len(stats_df)} land cover maps.")
print(f"All maps saved to: {RESULTS_DIR}")
```

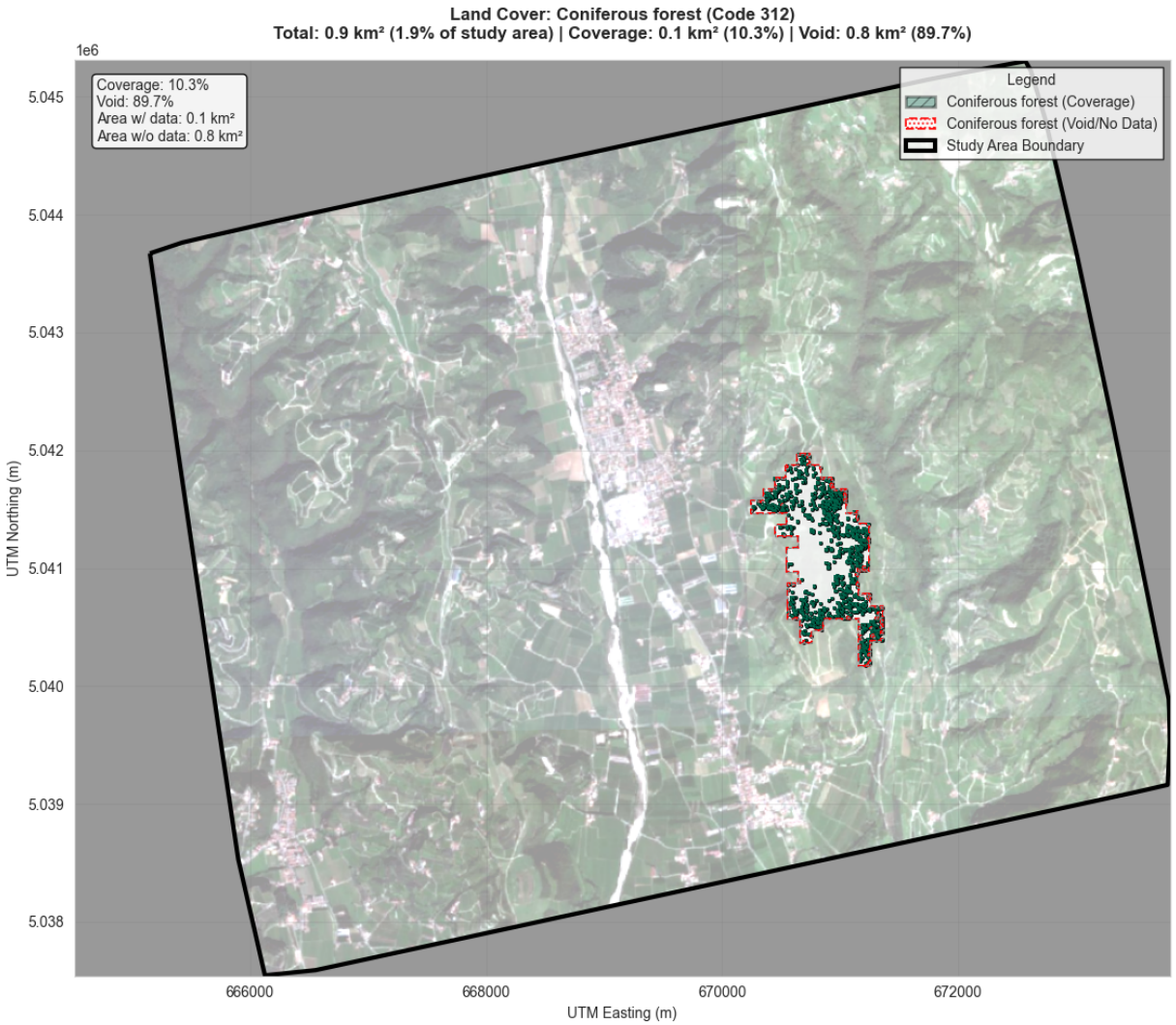
```
Calculating statistics for all land cover classes...
✓ Statistics DataFrame created.
 LC_Code                          LC_Name  Total_km2  Coverage_km2  Void_km2  Pct_
Coverage  Pct_Void  Pct_of_Study_Area
     313                     Mixed forest       0.31          0.03      0.28
8.69     91.31               0.63
     311                Broad-leaved forest      14.59          1.34     13.25
9.16     90.84              29.55
     312                  Coniferous forest       0.92          0.10      0.82
10.35     89.65               1.86
     243 Agriculture/natural vegetation mix       9.60          1.02      8.58
10.67     89.33              19.45
     223                      Olive groves       1.29          0.16      1.13
12.71     87.29               2.62
     231                           Pastures       3.92          0.58      3.34
14.75     85.25               7.94
     221                          Vineyards      14.24          2.25     11.98
15.84     84.16              28.83
     242       Complex cultivation patterns       1.51          0.27      1.23
18.05     81.95               3.05
     112         Discontinuous urban fabric       2.22          0.49      1.73
22.07     77.93               4.50
     331              Beaches, dunes, sands       0.78          0.17      0.60
22.36     77.64               1.57

Generating 10 land cover maps using pre-calculated stats...
```



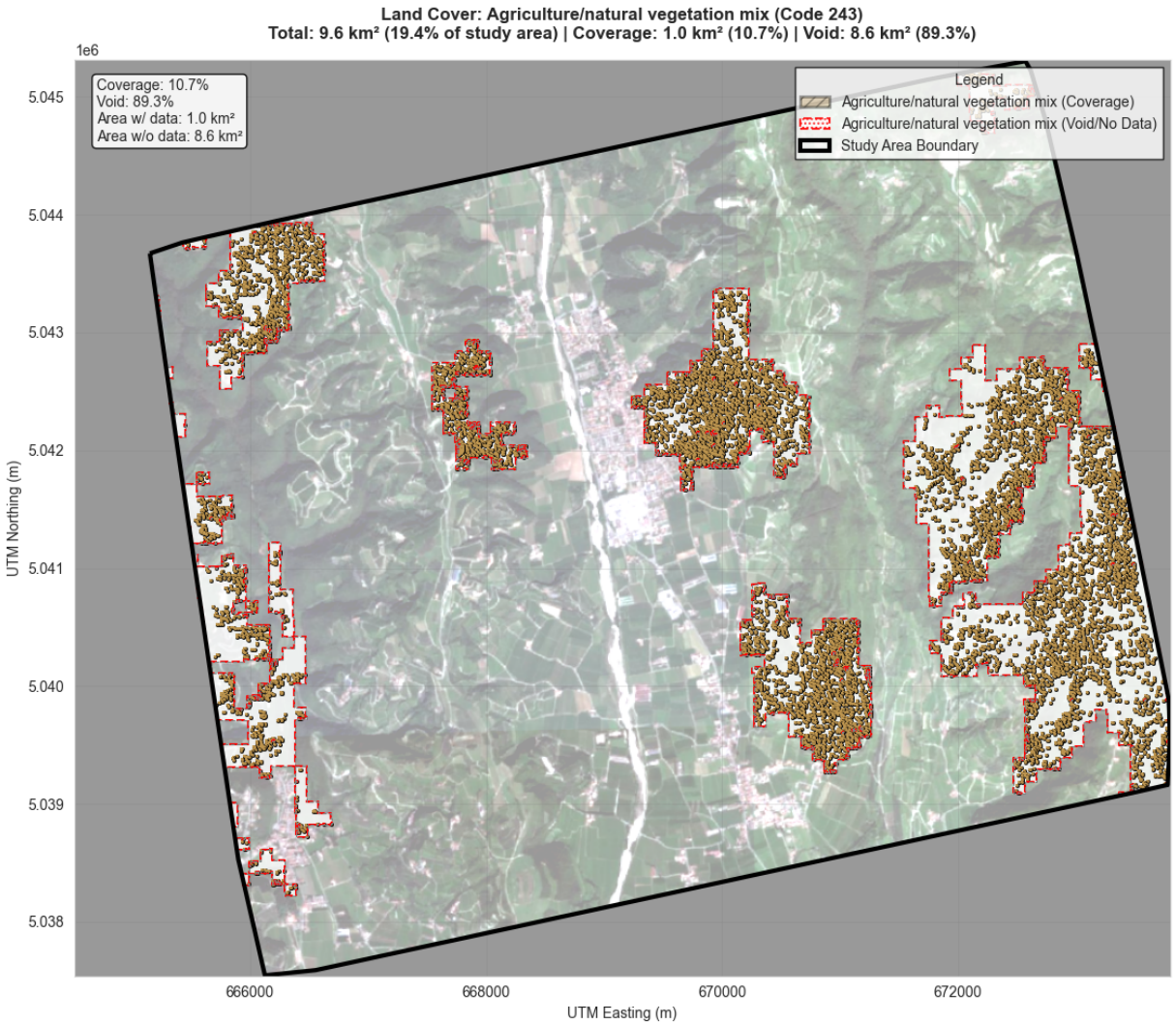
Saved: landcover_313_Mixed_forest_coverage_void.png



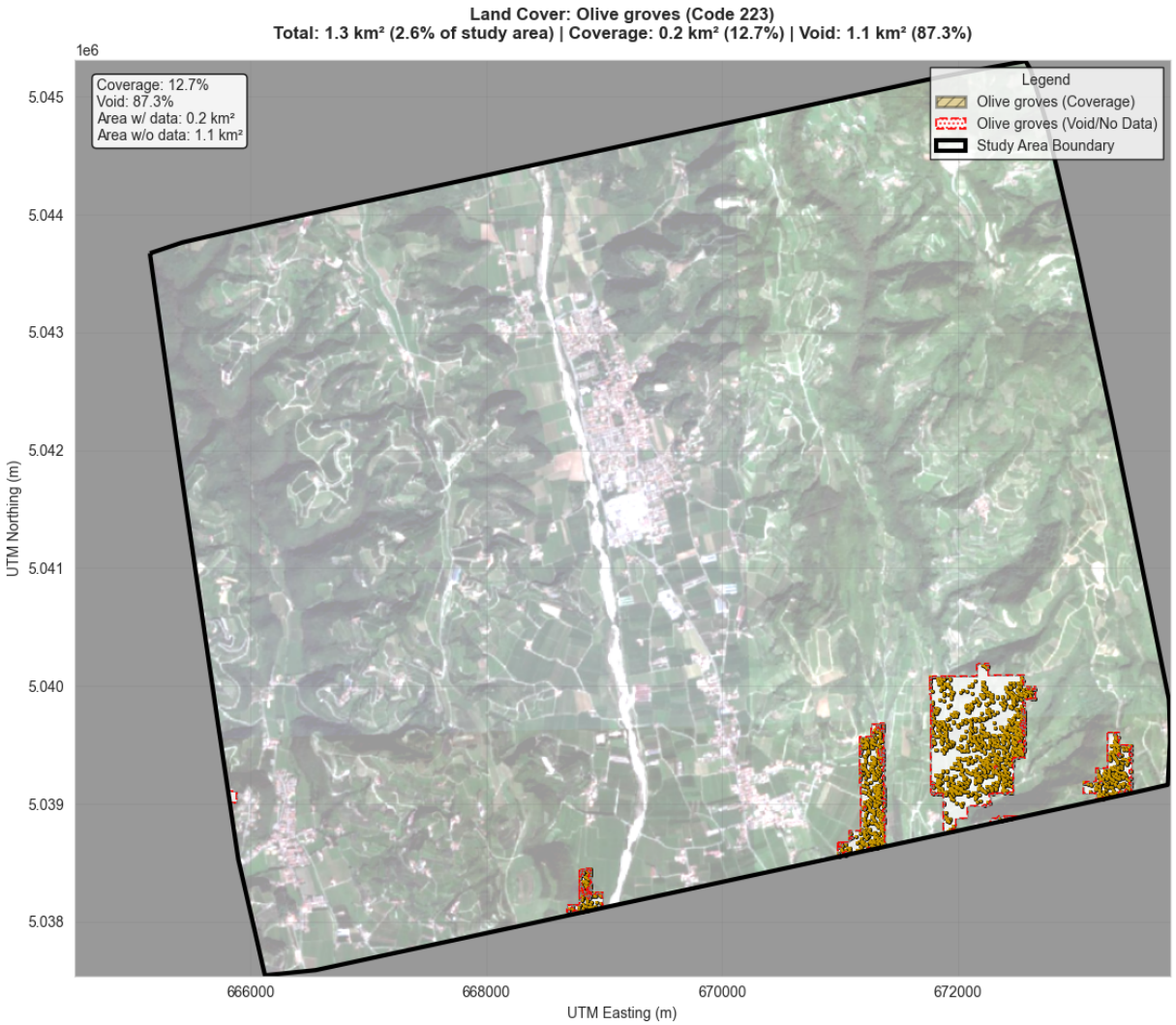
Saved: landcover_311_Broad-leaved_forest_coverage_void.png



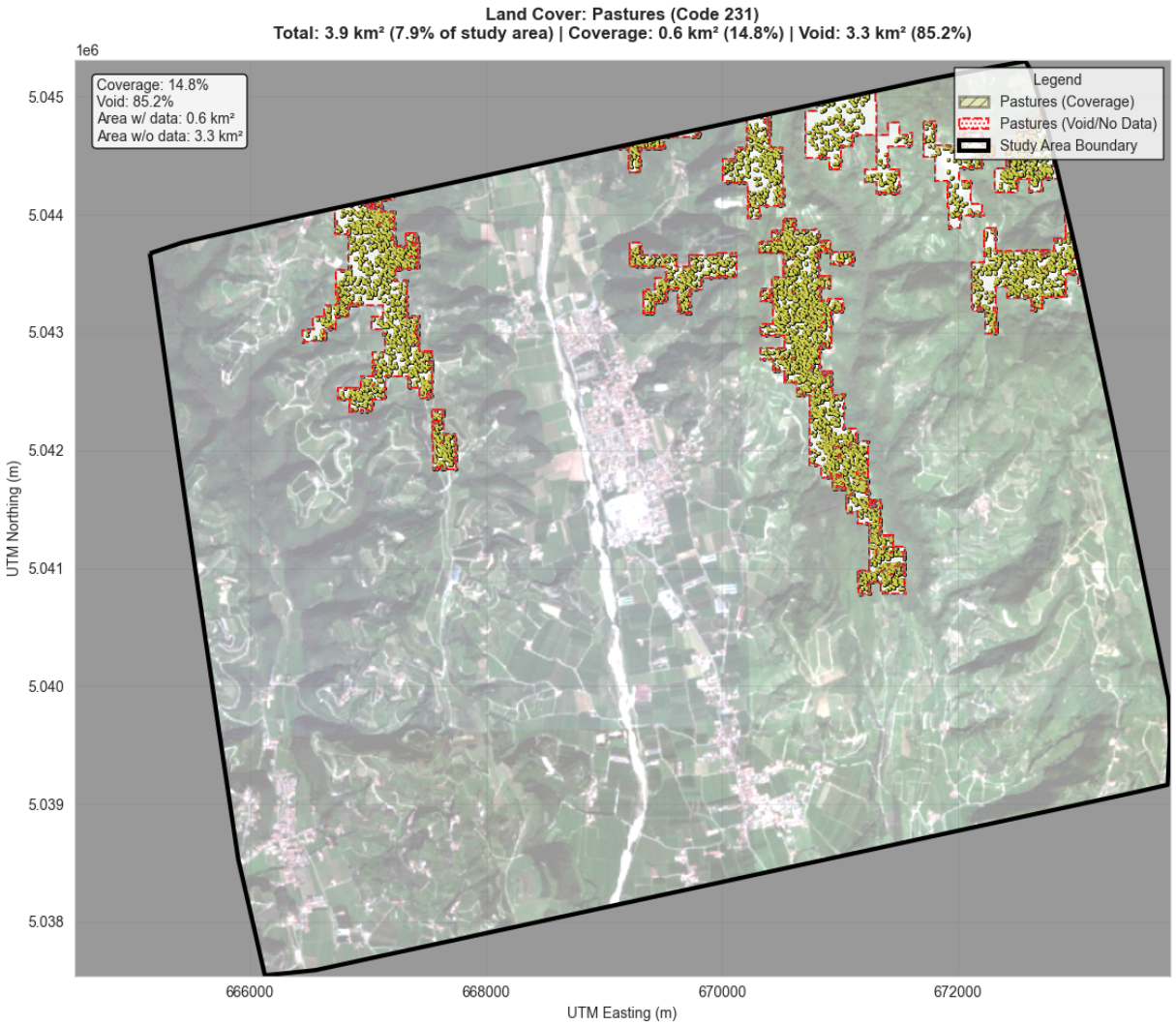
Saved: landcover_312_Coniferous_forest_coverage_void.png



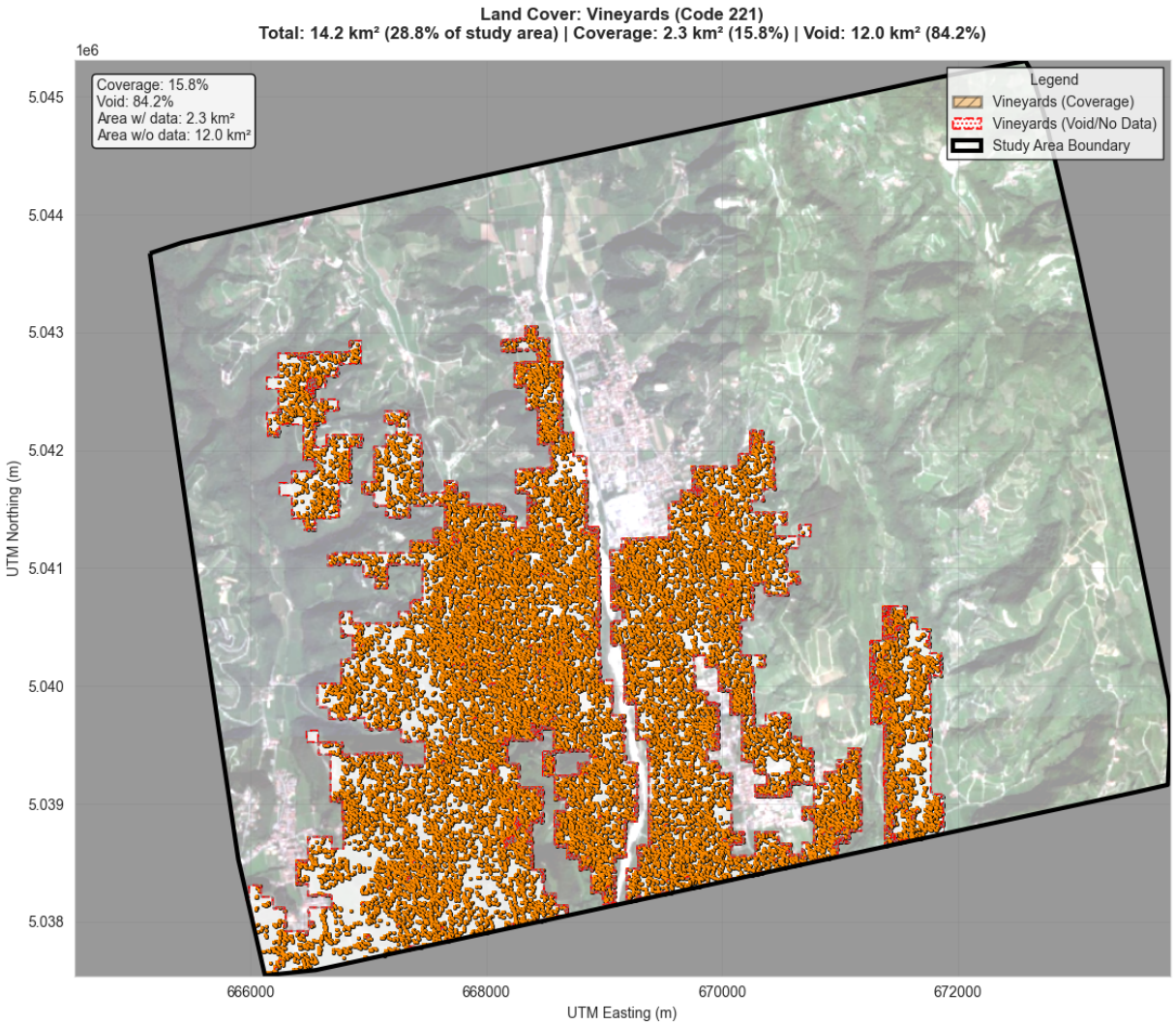
Saved: landcover_243_Agriculture_natural_vegetation_mix_coverage_void.png



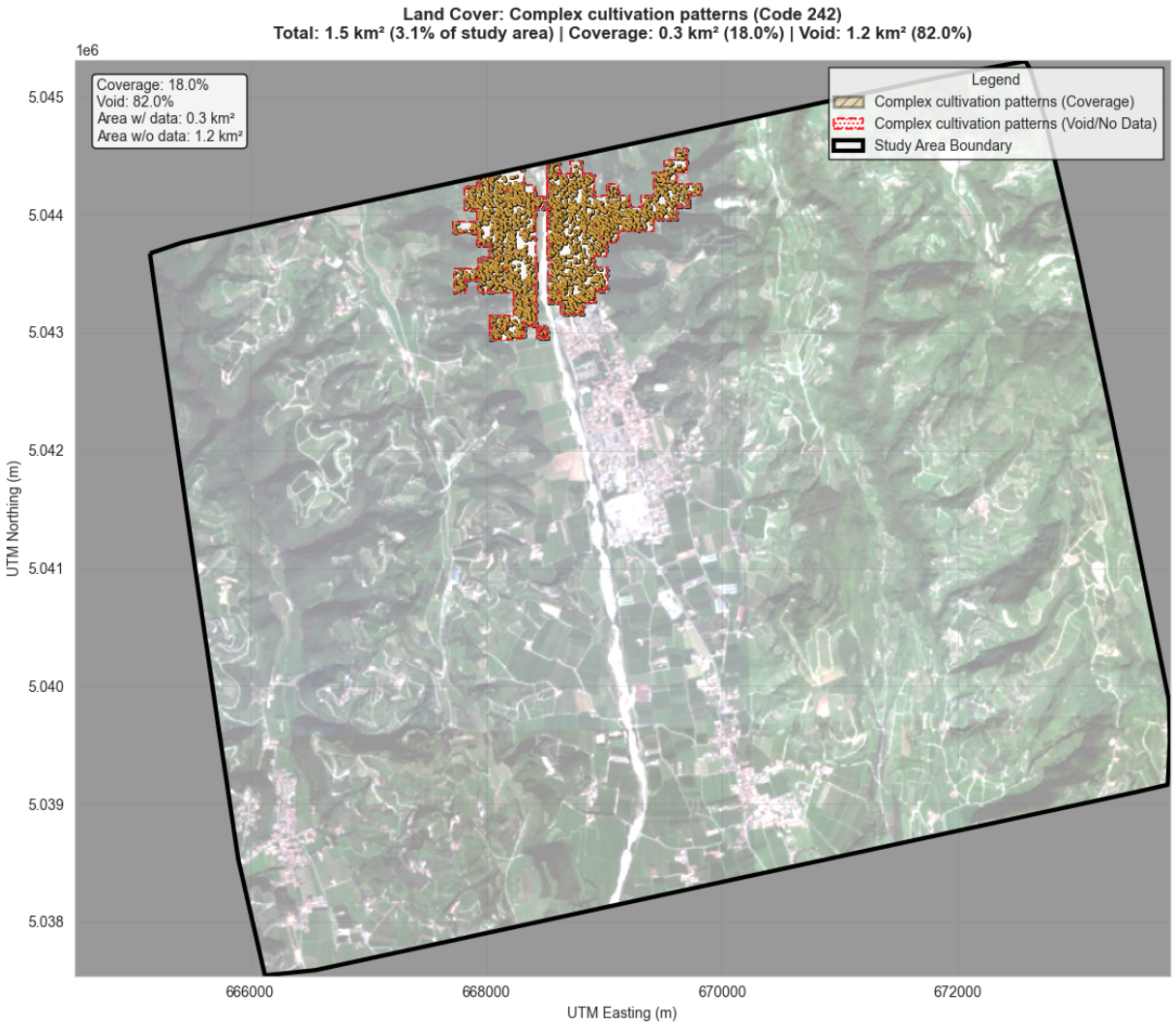
Saved: landcover_223_Olive_groves_coverage_void.png



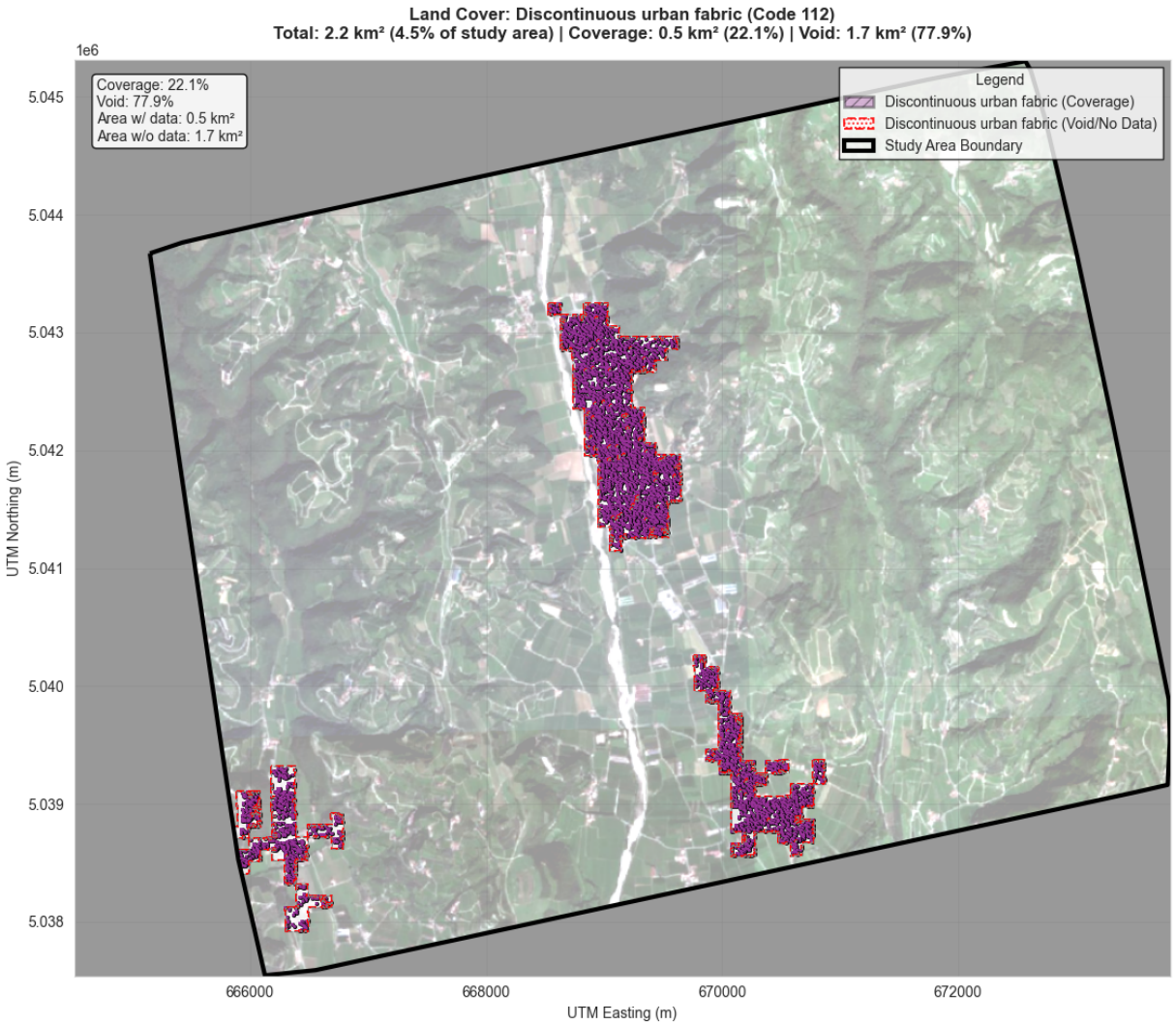
Saved: landcover_231_Pastures_coverage_void.png



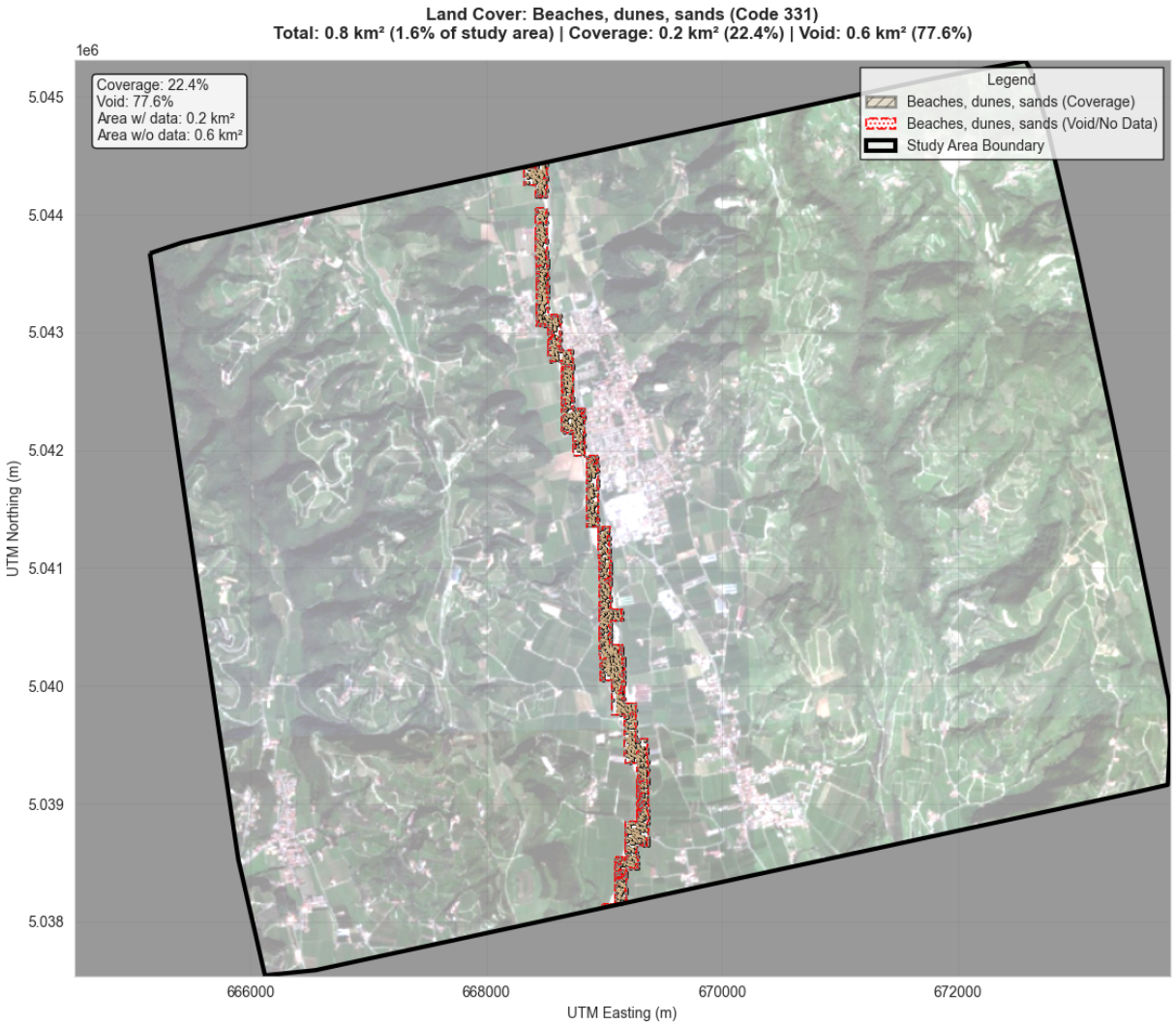
Saved: landcover_221_Vineyards_coverage_void.png



Saved: landcover_242_Complex_cultivation_patterns_coverage_void.png



Saved: landcover_112_Discontinuous_urban_fabric_coverage_void.png




```
Saved: landcover_331_Beaches_dunes_sands_coverage_void.png

✓ Generated 10 land cover maps.
All maps saved to: results
```

Land cover histograms

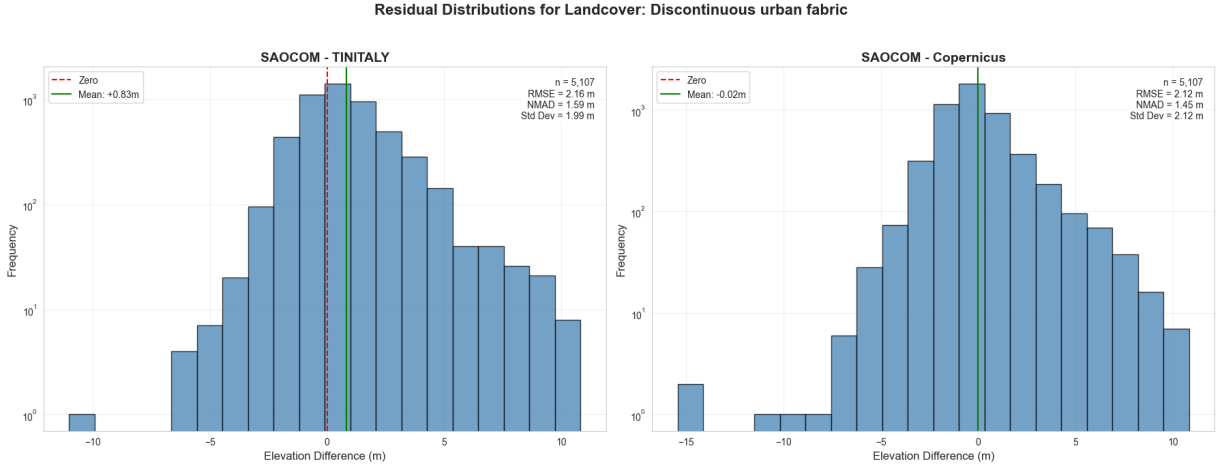
In [33]:

```
import numpy as np
import pandas as pd
import geopandas as gpd
import rasterio
from rasterio.transform import rowcol
from pathlib import Path
import matplotlib.pyplot as plt
from scipy import stats
import warnings
warnings.filterwarnings('ignore')


# ============================================================================
# Re-define necessary variables and functions from your notebook
# ============================================================================
RESULTS_DIR = Path("results")
NODATA = -9999
```

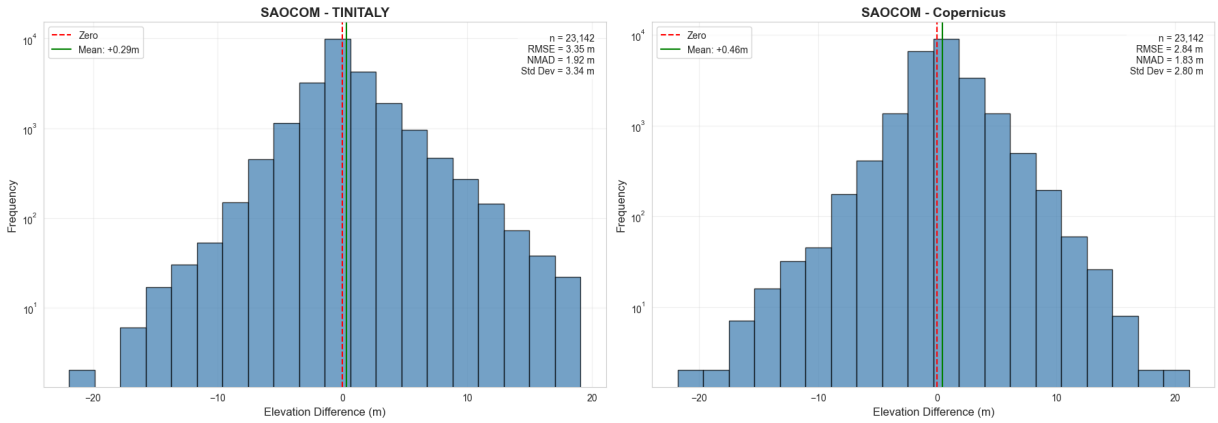
```
CORINE_CLASSES = {
    111: 'Continuous urban fabric', 112: 'Discontinuous urban fabric',
    121: 'Industrial or commercial units', 122: 'Road and rail networks and associa
    123: 'Port areas', 124: 'Airports', 131: 'Mineral extraction sites',
    132: 'Dump sites', 133: 'Construction sites', 141: 'Green urban areas',
    142: 'Sport and leisure facilities', 211: 'Non-irrigated arable land',
    212: 'Permanently irrigated land', 213: 'Rice fields', 221: 'Vineyards',
    222: 'Fruit trees and berry plantations', 223: 'Olive groves',
    231: 'Pastures', 241: 'Annual crops associated with permanent crops',
    242: 'Complex cultivation patterns', 243: 'Agriculture/natural vegetation mix',
    244: 'Agro-forestry areas', 311: 'Broad-leaved forest',
    312: 'Coniferous forest', 313: 'Mixed forest', 321: 'Natural grasslands',
    322: 'Moors and heathland', 323: 'Sclerophyllous vegetation',
    324: 'Transitional woodland-shrub', 331: 'Beaches, dunes, sands',
    332: 'Bare rocks', 333: 'Sparsely vegetated areas', 334: 'Burnt areas',
    335: 'Glaciers and perpetual snow', 411: 'Inland marshes',
    412: 'Peat bogs', 421: 'Salt marshes', 422: 'Salines',
    423: 'Intertidal flats', 511: 'Water courses', 512: 'Water bodies',
    521: 'Coastal lagoons', 522: 'Estuaries', 523: 'Sea and ocean'
}

# Assuming 'corine_10m_masked.tif' is in the RESULTS_DIR from previous cells
corine_masked_path = RESULTS_DIR / "corine_10m_masked.tif"
with rasterio.open(corine_masked_path) as src:
    corine_10m = src.read(1)
    target_transform = src.transform
    grid_height, grid_width = src.height, src.width

# Helper function from your notebook
def plot_distribution(ax, diff_series, title, metrics):
    """Helper function to plot a single residual distribution histogram."""
    ax.set_facecolor('white')
    ax.hist(diff_series, bins=20, alpha=0.75, color='steelblue', edgecolor='black')
    ax.axvline(0, color='red', linestyle='--', label='Zero')
    ax.axvline(metrics['mean_diff'], color='green', linestyle='-', label=f"Mean: {m

    stats_text = (f"n = {metrics['n_points']:,}\n"
                  f"RMSE = {metrics['rmse']:.2f} m\n"
                  f"NMAD = {metrics['nmad']:.2f} m\n"
                  f"Std Dev = {metrics['std_diff']:.2f} m")

    ax.text(0.98, 0.97, stats_text, transform=ax.transAxes, fontsize=10,
            verticalalignment='top', horizontalalignment='right',
            bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

    ax.set_title(title, fontweight='bold', fontsize=14)
    ax.set_xlabel('Elevation Difference (m)', fontsize=12)
    ax.set_ylabel('Frequency', fontsize=12)
    ax.set_yscale('log')
    ax.legend()
    ax.grid(True, alpha=0.3)


# ============================================================================
# 1. Sample CORINE Land Cover at SAOCOM point locations
# ============================================================================
```

```
landcover_values = []
for idx, row in saocom_gdf.iterrows():
    row_idx, col_idx = rowcol(target_transform, row.geometry.x, row.geometry.y)
    if 0 <= row_idx < grid_height and 0 <= col_idx < grid_width:
        lc_val = corine_10m[row_idx, col_idx]
        landcover_values.append(lc_val if lc_val != 0 else np.nan)
    else:
        landcover_values.append(np.nan)

saocom_gdf['landcover'] = landcover_values


# ============================================================================
# 2. Create points dataframes with landcover information
# ============================================================================

# SAOCOM - TINITALY
points_tin = saocom_gdf[['HEIGHT_ABSOLUTE_TIN', 'tinitaly_height', 'landcover']].co
points_tin.dropna(inplace=True)
points_tin['diff'] = points_tin['HEIGHT_ABSOLUTE_TIN'] - points_tin['tinitaly_heigh

# SAOCOM - Copernicus
points_cop = saocom_gdf[['HEIGHT_ABSOLUTE_COP', 'copernicus_height', 'landcover']].
points_cop.dropna(inplace=True)
points_cop['diff'] = points_cop['HEIGHT_ABSOLUTE_COP'] - points_cop['copernicus_hei

# ============================================================================
# 3. Group by landcover and calculate metrics
# ============================================================================

landcover_groups_tin = points_tin.groupby('landcover')
landcover_groups_cop = points_cop.groupby('landcover')

metrics_by_landcover = {}

unique_landcovers = sorted(points_tin['landcover'].unique())

for lc_code in unique_landcovers:
    if np.isnan(lc_code):
        continue
    lc_name = CORINE_CLASSES.get(lc_code, f'Unknown ({lc_code})')
    metrics_by_landcover[lc_name] = {}

    # TINITALY metrics
    if lc_code in landcover_groups_tin.groups:
        group_tin = landcover_groups_tin.get_group(lc_code)
        diff_tin = group_tin['diff']
        metrics_by_landcover[lc_name]['tin'] = {
            'n_points': len(diff_tin),
            'rmse': np.sqrt(np.mean(diff_tin**2)),
            'nmad': stats.median_abs_deviation(diff_tin, scale='normal'),
            'mean_diff': np.mean(diff_tin),
            'std_diff': np.std(diff_tin),
            'diff_series': diff_tin
        }
```

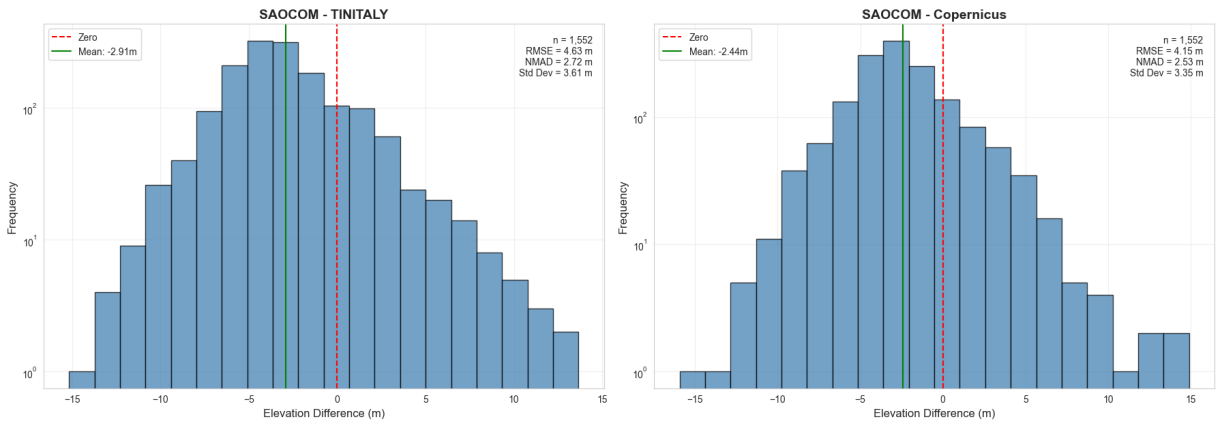
```
    # Copernicus metrics
    if lc_code in landcover_groups_cop.groups:
        group_cop = landcover_groups_cop.get_group(lc_code)
        diff_cop = group_cop['diff']
        metrics_by_landcover[lc_name]['cop'] = {
            'n_points': len(diff_cop),
            'rmse': np.sqrt(np.mean(diff_cop**2)),
            'nmad': stats.median_abs_deviation(diff_cop, scale='normal'),
            'mean_diff': np.mean(diff_cop),
            'std_diff': np.std(diff_cop),
            'diff_series': diff_cop
        }

# ============================================================================
# 4. Generate and save histograms for each landcover
# ============================================================================

for lc_name, metrics in metrics_by_landcover.items():
    fig, axes = plt.subplots(1, 2, figsize=(18, 7), facecolor='white')
    fig.suptitle(f'Residual Distributions for Landcover: {lc_name}', fontsize=16, f

    if 'tin' in metrics:
        plot_distribution(axes[0], metrics['tin']['diff_series'], 'SAOCOM - TINITAL
    else:
        axes[0].set_title('SAOCOM - TINITALY\n(No Data)', fontweight='bold', fontsi
        axes[0].set_facecolor('lightgray')


    if 'cop' in metrics:
        plot_distribution(axes[1], metrics['cop']['diff_series'], 'SAOCOM - Coperni
    else:
        axes[1].set_title('SAOCOM - Copernicus\n(No Data)', fontweight='bold', font
        axes[1].set_facecolor('lightgray')

    plt.tight_layout(rect=[0, 0, 1, 0.95])

    # Sanitize filename
    safe_lc_name = lc_name.replace('/', '_').replace(' ', '_').lower()
    plt.savefig(RESULTS_DIR / f'saocom_residuals_{safe_lc_name}.png', dpi=300, face
    plt.show()
```



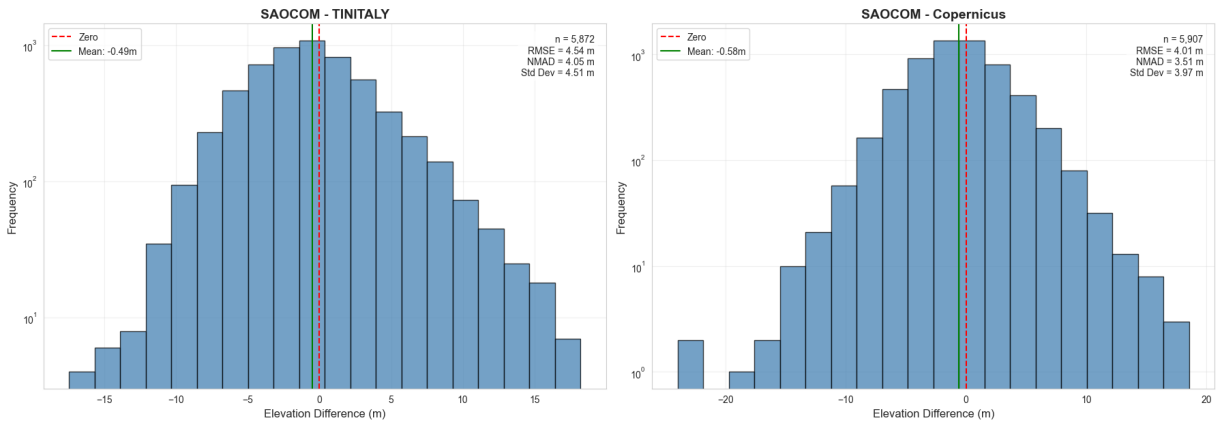
Residual Distributions for Landcover: Vineyards



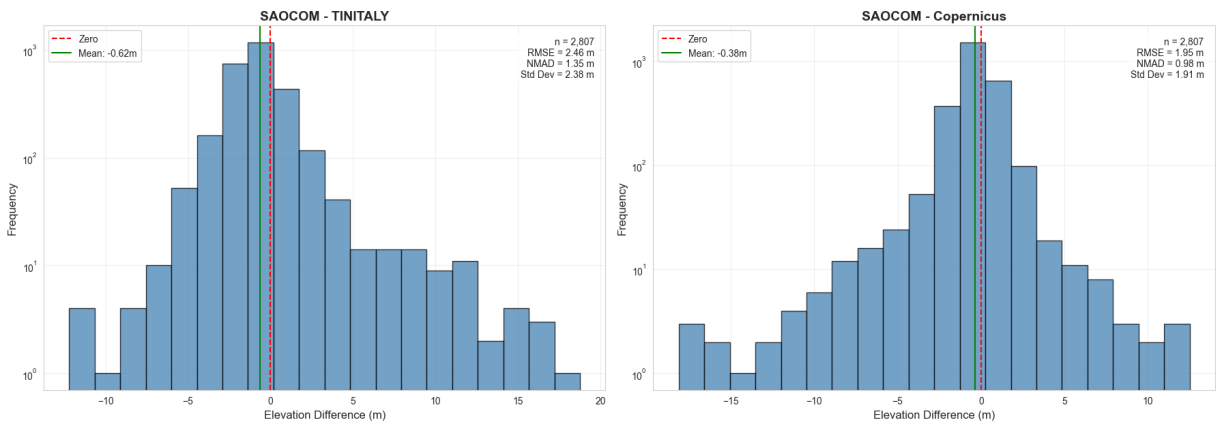
Residual Distributions for Landcover: Olive groves



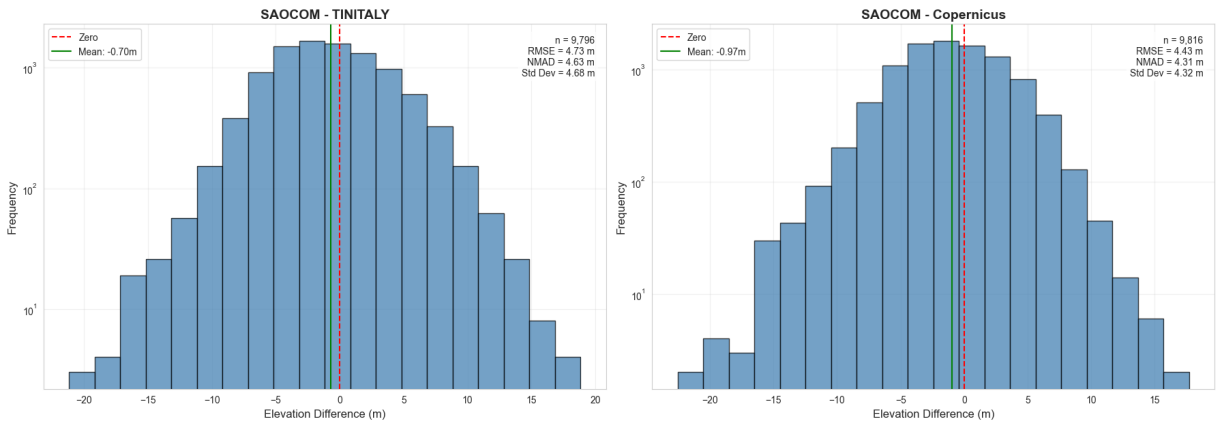
Residual Distributions for Landcover: Pastures



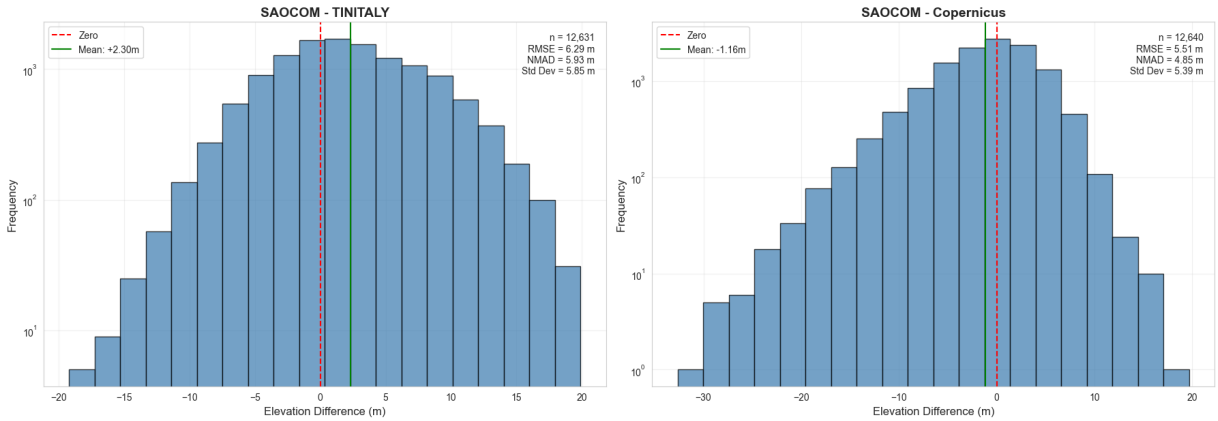
Residual Distributions for Landcover: Complex cultivation patterns



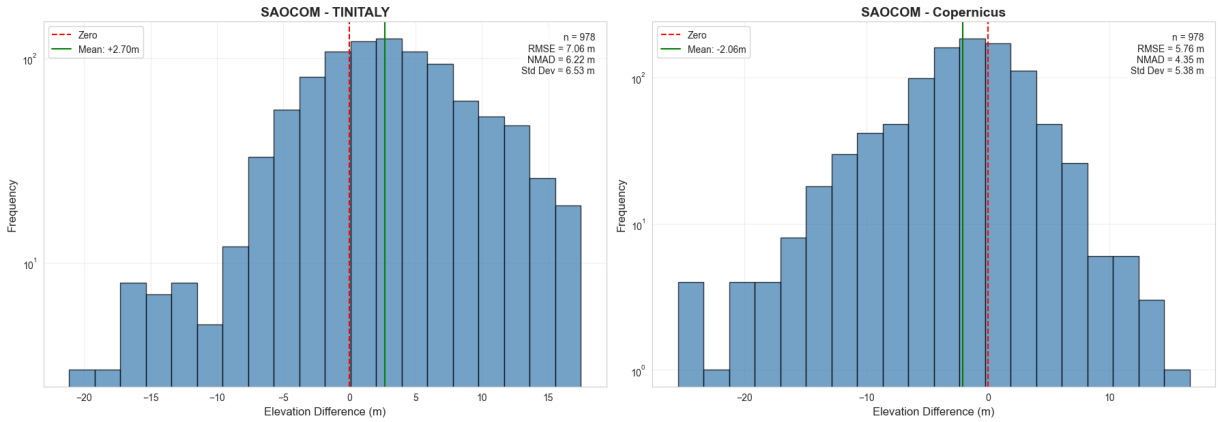
Residual Distributions for Landcover: Agriculture/natural vegetation mix



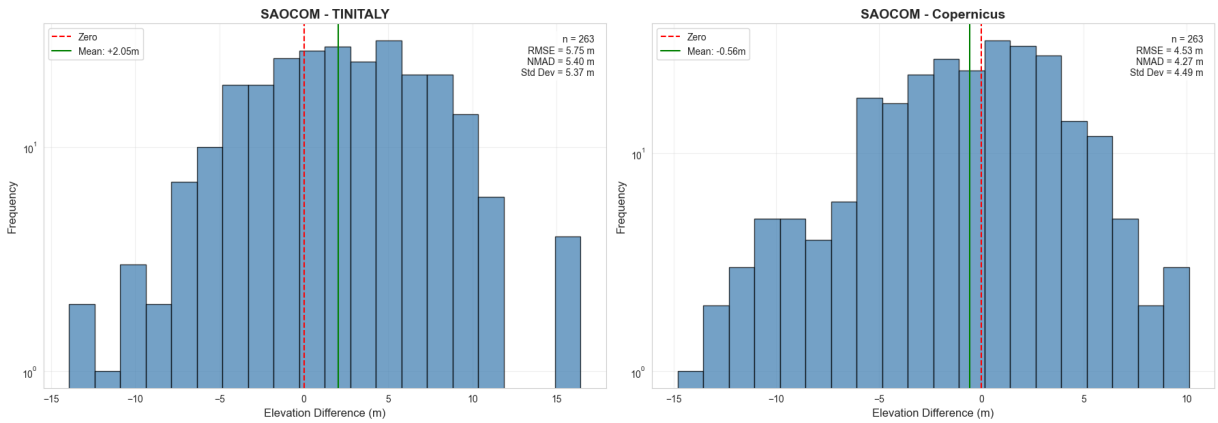
Residual Distributions for Landcover: Broad-leaved forest

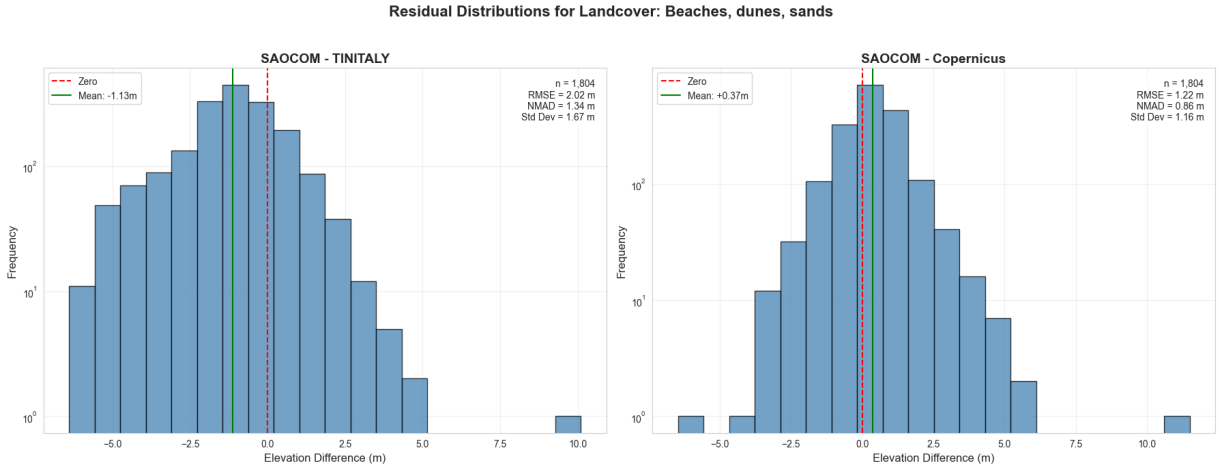


Residual Distributions for Landcover: Coniferous forest



Residual Distributions for Landcover: Mixed forest



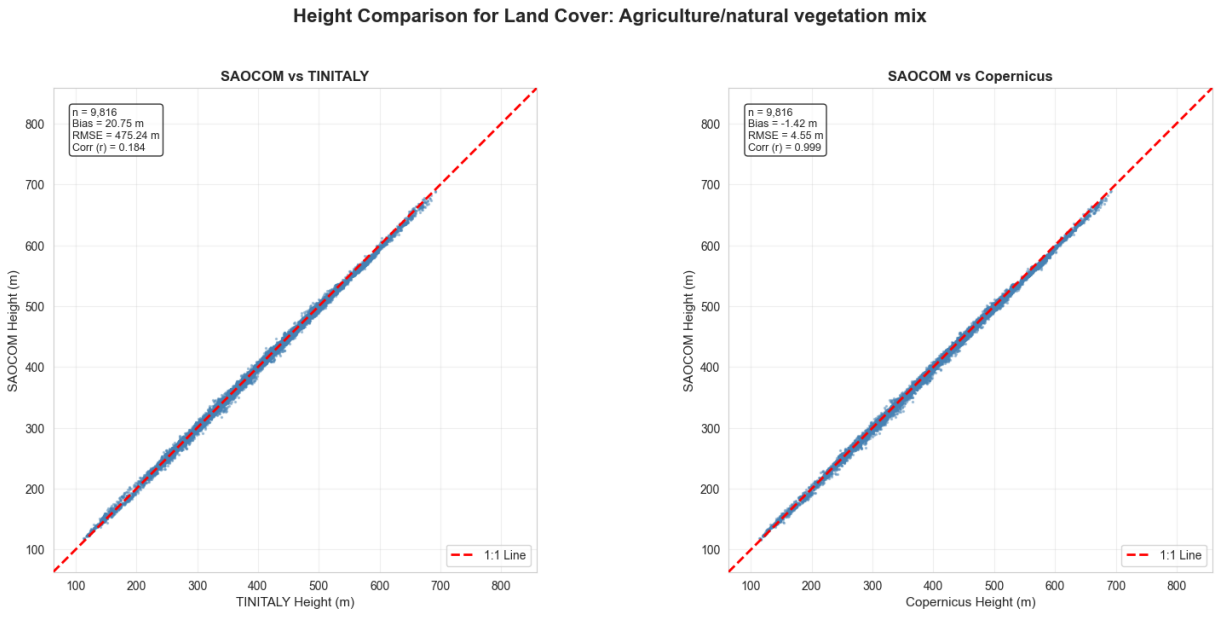


In [52]:
```
import pandas as pd
import geopandas as gpd
import numpy as np
import rasterio
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# --- Assuming 'saocom_gdf_lc', 'RESULTS_DIR', etc., are pre-loaded ---

# ==============================================================================
# 1. DEFINE A ROBUST SCATTER PLOTTING FUNCTION
# ==============================================================================
def plot_scatter_comparison(ax, x_data, y_data, x_label, y_label, title, global_lim
    """
    Creates a 1:1 scatter plot with consistent axes and statistics.
    """
    ax.set_facecolor('white')

    # --- Clean data and calculate stats ---
    valid_mask = ~np.isnan(x_data) & ~np.isnan(y_data)
    x_val, y_val = x_data[valid_mask], y_data[valid_mask]

    if len(x_val) > 1:
        n_points = len(x_val)
        bias = np.mean(y_val - x_val)
        rmse = np.sqrt(mean_squared_error(x_val, y_val))
        correlation = np.corrcoef(x_val, y_val)[0, 1]
        stats_text = (f"n = {n_points:,}\n"
                      f"Bias = {bias:.2f} m\n"
                      f"RMSE = {rmse:.2f} m\n"
                      f"Corr (r) = {correlation:.3f}")
    else:
        stats_text = "Not enough data"

    # --- Plotting ---
    ax.scatter(x_val, y_val, s=2, alpha=0.4, c='steelblue')
    ax.plot(global_lims, global_lims, 'r--', linewidth=2, label='1:1 Line', zorder=
    ax.set_xlim(global_lims)
    ax.set_ylim(global_lims)

    # --- Annotation and Styling ---
```

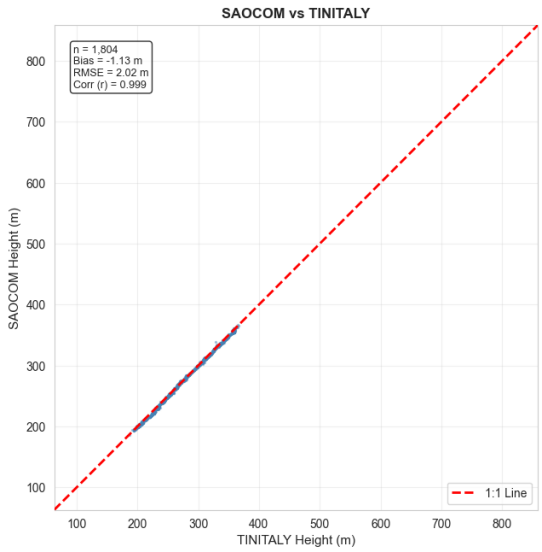
```
    ax.text(0.04, 0.96, stats_text, transform=ax.transAxes, fontsize=9,
            verticalalignment='top', bbox=dict(boxstyle='round', facecolor='white',
    ax.set_title(title, fontweight='bold', fontsize=12)
    ax.set_xlabel(x_label, fontsize=11)
    ax.set_ylabel(y_label, fontsize=11)
    ax.grid(True, alpha=0.3)
    ax.legend(loc='lower right')
    ax.set_aspect('equal', 'box')

# ============================================================================
# 2. CALCULATE GLOBAL AXIS LIMITS FOR CONSISTENCY
# ============================================================================
# Combine all height data to find the absolute min and max
all_heights = pd.concat([
    saocom_gdf_lc['HEIGHT_ABSOLUTE_TIN'],
    saocom_gdf_lc['tinitaly_height'],
    saocom_gdf_lc['copernicus_height']
]).dropna()
all_heights = np.array([i for i in all_heights if i != -9999])
print(sorted(all_heights)[0])
min_lim = all_heights.min()
max_lim = all_heights.max()
print(min_lim, max_lim)
buffer = (max_lim - min_lim) * 0.05  # 5% buffer

# Define the global limits to be used in all plots
global_axis_lims = [min_lim - buffer, max_lim + buffer]

# ============================================================================
# 3. LOOP THROUGH EACH LAND COVER TYPE AND CREATE PLOTS
# ============================================================================
# Get a sorted list of unique land cover types
unique_land_covers = sorted(saocom_gdf_lc['land_cover'].unique())

for lc_name in unique_land_covers:
    # Create a new figure for each land cover type
    fig, axes = plt.subplots(1, 2, figsize=(16, 7.5), facecolor='white')
    fig.suptitle(f'Height Comparison for Land Cover: {lc_name}', fontsize=16, fontw

    # Filter the data for the current land cover
    subset_gdf = saocom_gdf_lc[saocom_gdf_lc['land_cover'] == lc_name]

    # --- Plot 1: SAOCOM vs TINITALY ---
    if not subset_gdf['tinitaly_height'].dropna().empty:
        plot_scatter_comparison(axes[0],
                                x_data=subset_gdf['tinitaly_height'],
                                y_data=subset_gdf['HEIGHT_ABSOLUTE_TIN'],
                                x_label='TINITALY Height (m)',
                                y_label='SAOCOM Height (m)',
                                title='SAOCOM vs TINITALY',
                                global_lims=global_axis_lims)
    else:
        axes[0].set_title('SAOCOM vs TINITALY\n(No Data)', fontweight='bold')
        axes[0].set_facecolor('lightgray')
        axes[0].set_aspect('equal', 'box')
```

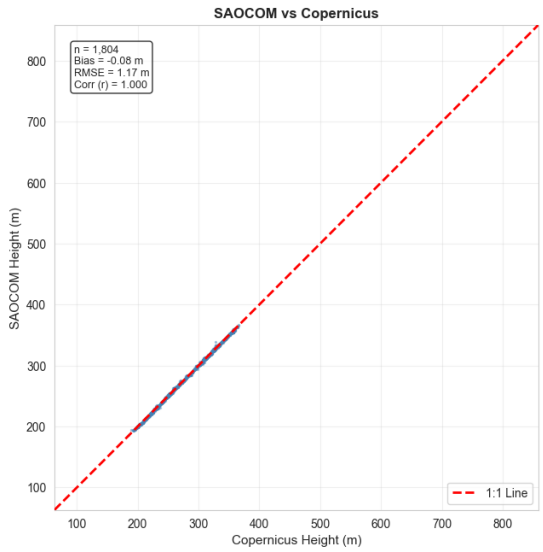
```
    # --- Plot 2: SAOCOM vs Copernicus ---
    if not subset_gdf['copernicus_height'].dropna().empty:
        plot_scatter_comparison(axes[1],
                                x_data=subset_gdf['copernicus_height'],
                                y_data=subset_gdf['HEIGHT_ABSOLUTE_TIN'],
                                x_label='Copernicus Height (m)',
                                y_label='SAOCOM Height (m)',
                                title='SAOCOM vs Copernicus',
                                global_lims=global_axis_lims)
    else:
        axes[1].set_title('SAOCOM vs Copernicus\n(No Data)', fontweight='bold')
        axes[1].set_facecolor('lightgray')
        axes[1].set_aspect('equal', 'box')


    plt.tight_layout(rect=[0, 0.03, 1, 0.94])

    # Sanitize filename and save the figure
    safe_lc_name = lc_name.replace('/', '_').replace(' ', '_').lower()
    plt.savefig(RESULTS_DIR / f'saocom_scatter_comparison_{safe_lc_name}.png', dpi=
    plt.show()
```

```
99.32097625732422
99.32097625732422 822.3181762695312
```

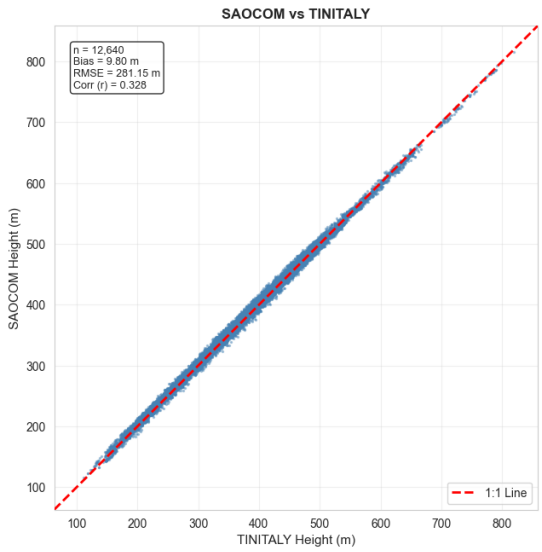


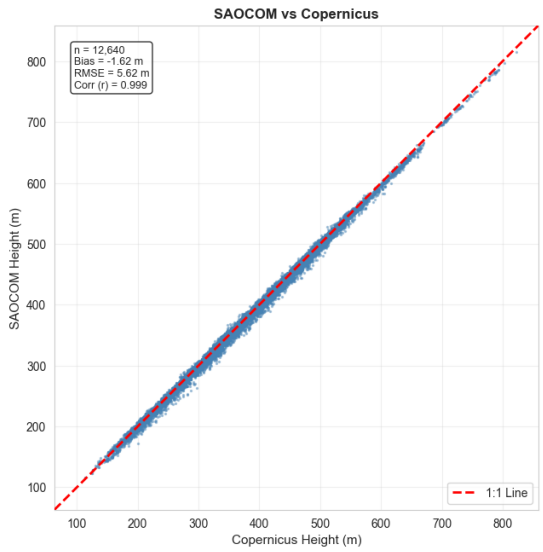
Height Comparison for Land Cover: Beaches, dunes, sands



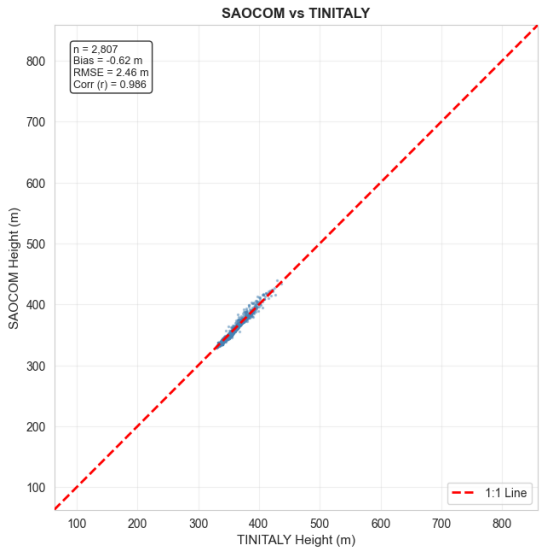


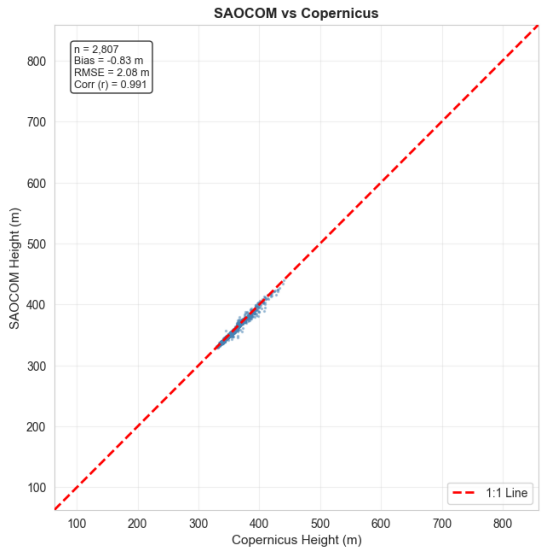
Height Comparison for Land Cover: Broad-leaved forest



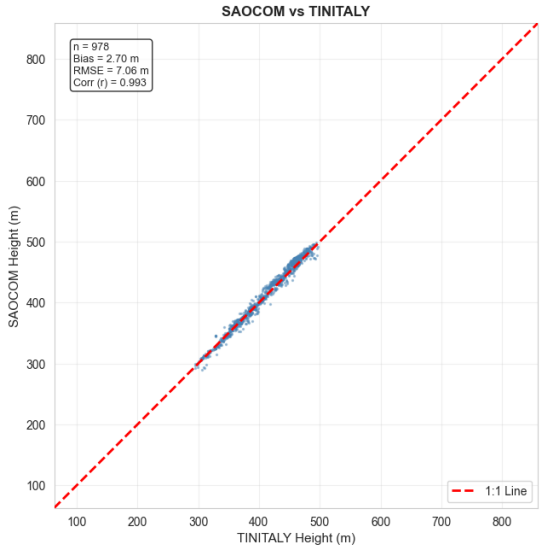


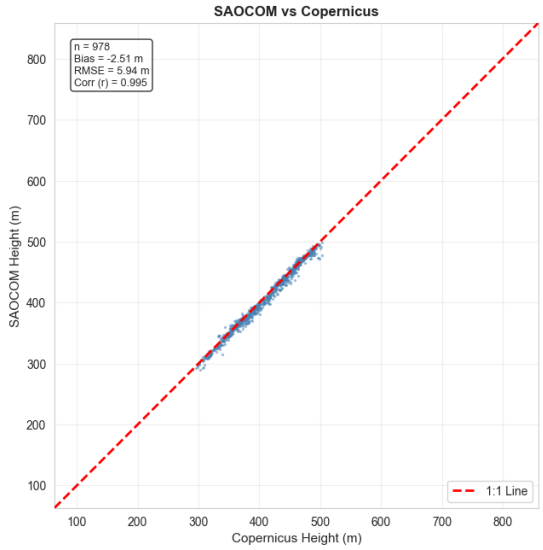
Height Comparison for Land Cover: Complex cultivation patterns



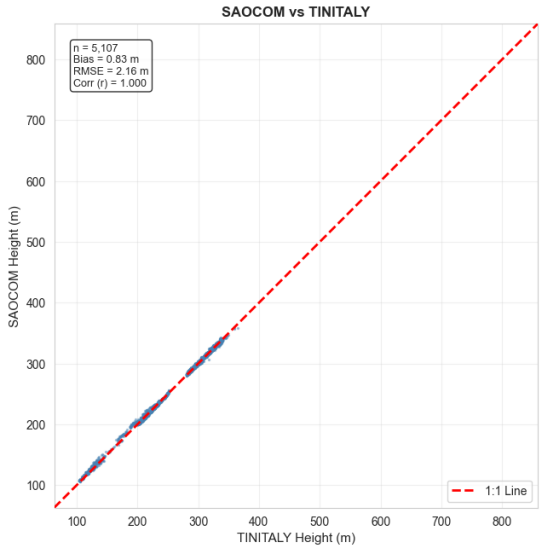


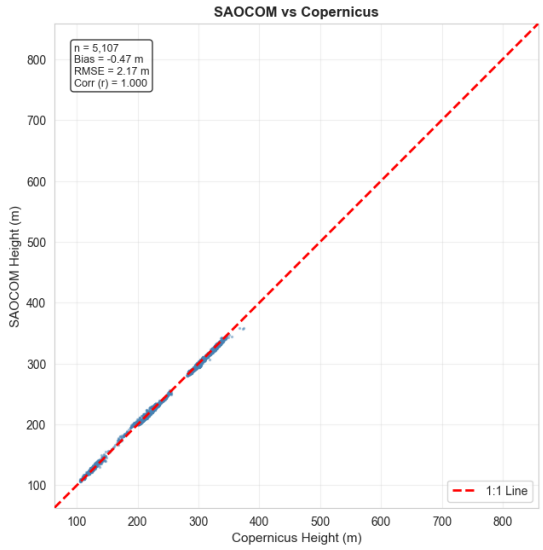
Height Comparison for Land Cover: Coniferous forest



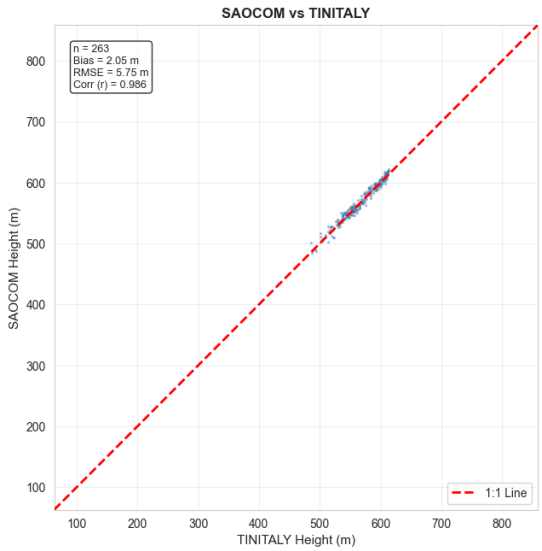


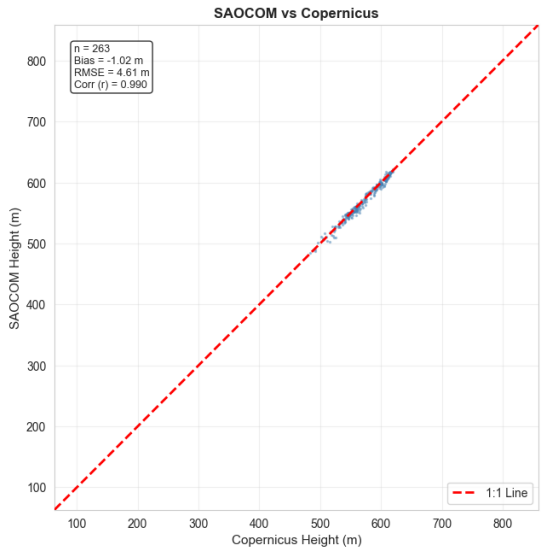
Height Comparison for Land Cover: Discontinuous urban fabric



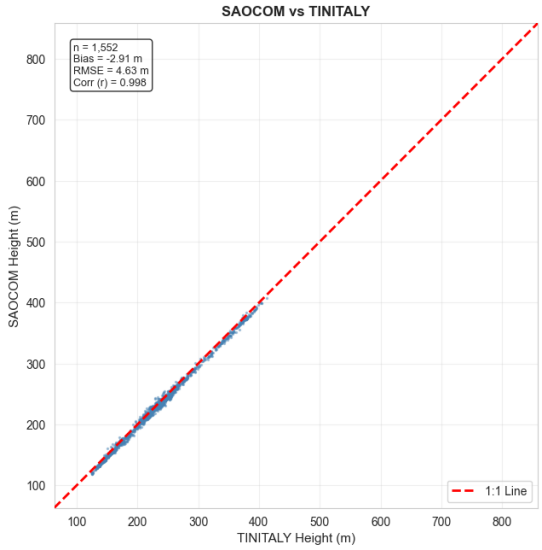


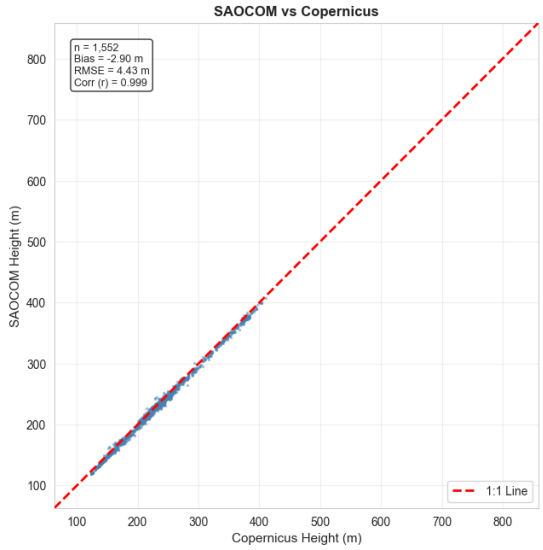
Height Comparison for Land Cover: Mixed forest



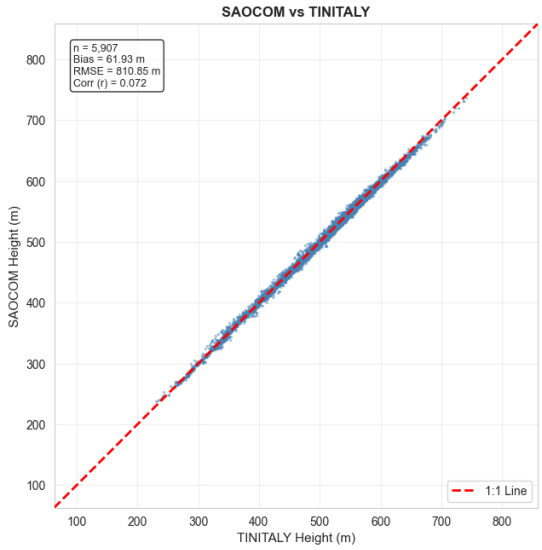


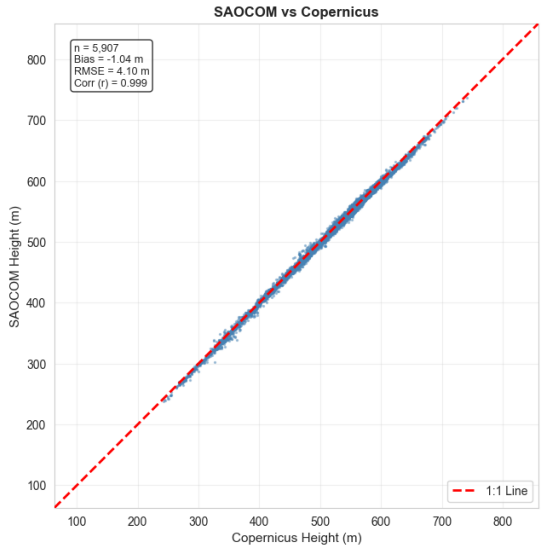
Height Comparison for Land Cover: Olive groves



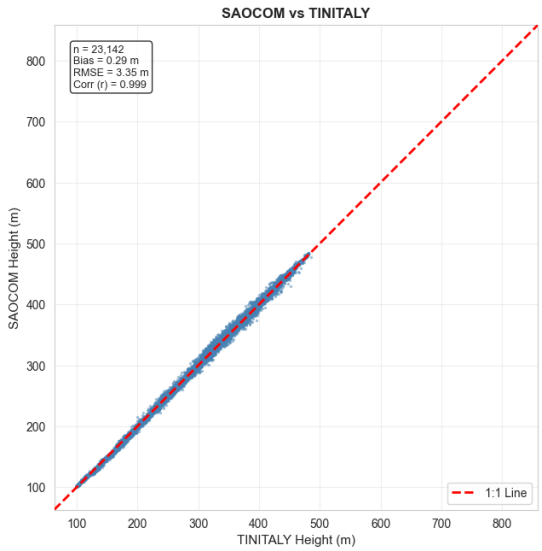


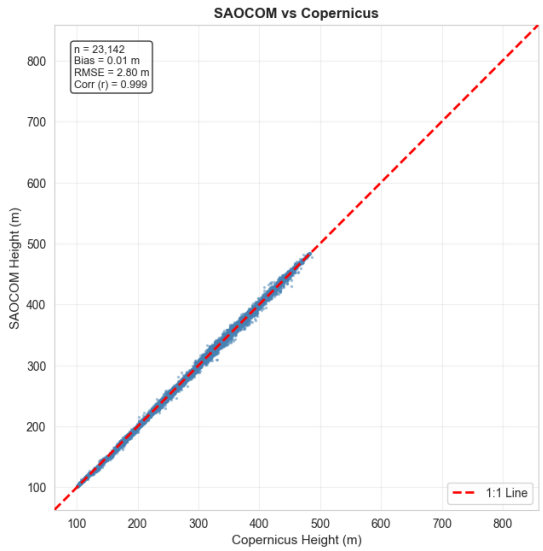
Height Comparison for Land Cover: Pastures





Height Comparison for Land Cover: Vineyards





In [ ]: