SUPPLEMENT

Supplementary Material for "Advanced Micronutrient Imaging in Plants: The POLYX Beamline at the NSRC SOLARIS JU"

M. Pypka[a], A. Barabasz[a], K. Sowa[b], P. Wróbel[b,c], T. Kołodziej[b], P. Korecki[d] and O. Siemianowski[a,*]

[a] *University of Warsaw, Faculty of Biology, Institute of Experimental Plant Biology and Biotechnology, Department of Plant Metal Homeostasis, Miecznikowa 1, 02-096 Warsaw, Poland*
[b] *National Synchrotron Radiation Centre SOLARIS, Jagiellonian University, Czerwone Maki 98, 30-392 Kraków, Poland*
[c] *AGH University of Krakow, Faculty of Physics and Applied Computer Science, Mickiewicza 30, 30-059 Kraków, Poland*
[d] *Jagiellonian University, Institute of Physics, Łojasiewicza 11, 30-348 Kraków, Poland*

Received: 29.30.2025 & Accepted: 02.12.2025

Doi: 10.12693/APhysPolA.149.218.SS1 *e-mail: o.siemianowski@uw.edu.pl

Supplementary material provides additional details that complement and support the results presented in the main text of the article [1]. Included here are quantitative estimates of fluorescence self-absorption in hydrated plant tissue (Table S1), full-resolution elemental maps acquired using polycapillary and monocapillary optics (Fig. S1), anatomical reference images of tobacco roots (Fig. S2), and resolution-dependent co-localization matrices (Fig. S3). The authors also provide the complete Python code used for image normalization, thresholding, co-localization analysis, and visualization. These materials are intended to facilitate reproducibility, enable independent verification of the analytical workflow, and provide deeper insight into the micronutrient distribution patterns obtained at the POLYX beamline.

TABLE SI

Approximate fluorescence self-absorption in hydrated plant tissue ($\sim$ 90% water, $\rho \approx 1$ g/cm^3) for Ca, Mn, Fe, Cu, and Zn Kα lines at effective path lengths of 100, 200, and 400 μm, and at 45° detection geometry (path lengths scaled by $1/\cos(45°) \approx 1.414$ to 141, 283, 566 μm). Values computed using the Beer–Lambert law, $A = 1 - \exp(-\frac{\mu}{\rho}\rho d)$, where: A — absorbance; μ — mass attenuation coefficient; ρ — density of the material; d — sample thickness. Representative water mass attenuation coefficients at the stated energies: Ca 3.69 keV, Mn 5.90 keV, Fe 6.40 keV, Cu 8.04 keV, Zn 8.64 keV (NIST X-Ray Mass Attenuation Coefficients database). Assumptions are as follows: homogeneous tissue, straight-line escape paths, no window/filter losses; numbers are interpretation guides, not calibration constants.

Element Kα energy	100 μm absorption	200 μm absorption	400 μm absorption
Ca (3.69 keV)	$\sim$ 10%	$\sim$ 20%	$\sim$ 36%
Mn (5.90 keV)	$\sim$ 2.5–3%	$\sim$ 5%	$\sim$ 10%
Fe (6.40 keV)	$\sim$ 2%	$\sim$ 4%	$\sim$ 8%
Cu (8.04 keV)	$\sim$ 1–1.5%	$\sim$ 2–3%	$\sim$ 5%
Zn (8.64 keV)	$\sim$ 1%	$\sim$ 2%	$\sim$ 4%

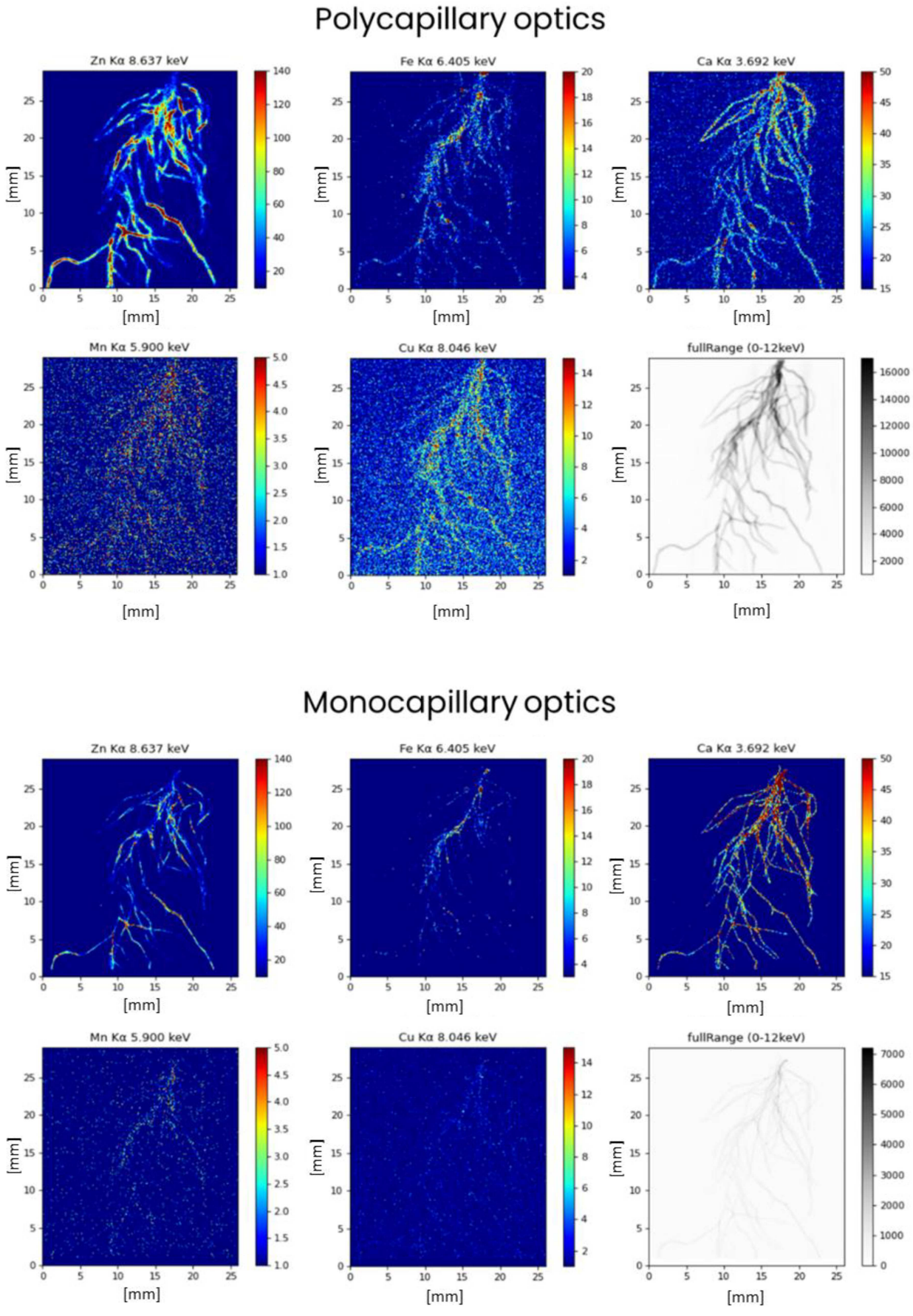

Fig. S1. Full map of the root system made using polycapillary and monocapillary optics.

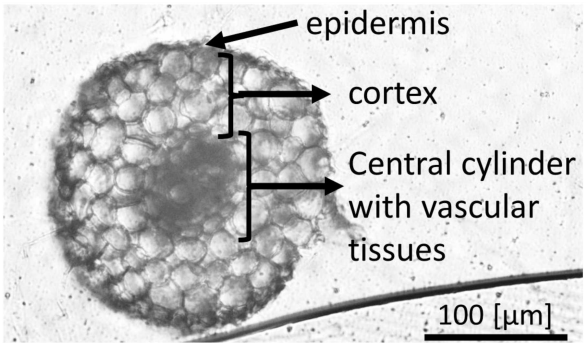

Fig. S2. Transverse section of a 5-week-old tobacco lateral root. Epidermis, cortex, and central cylinder are indicated.

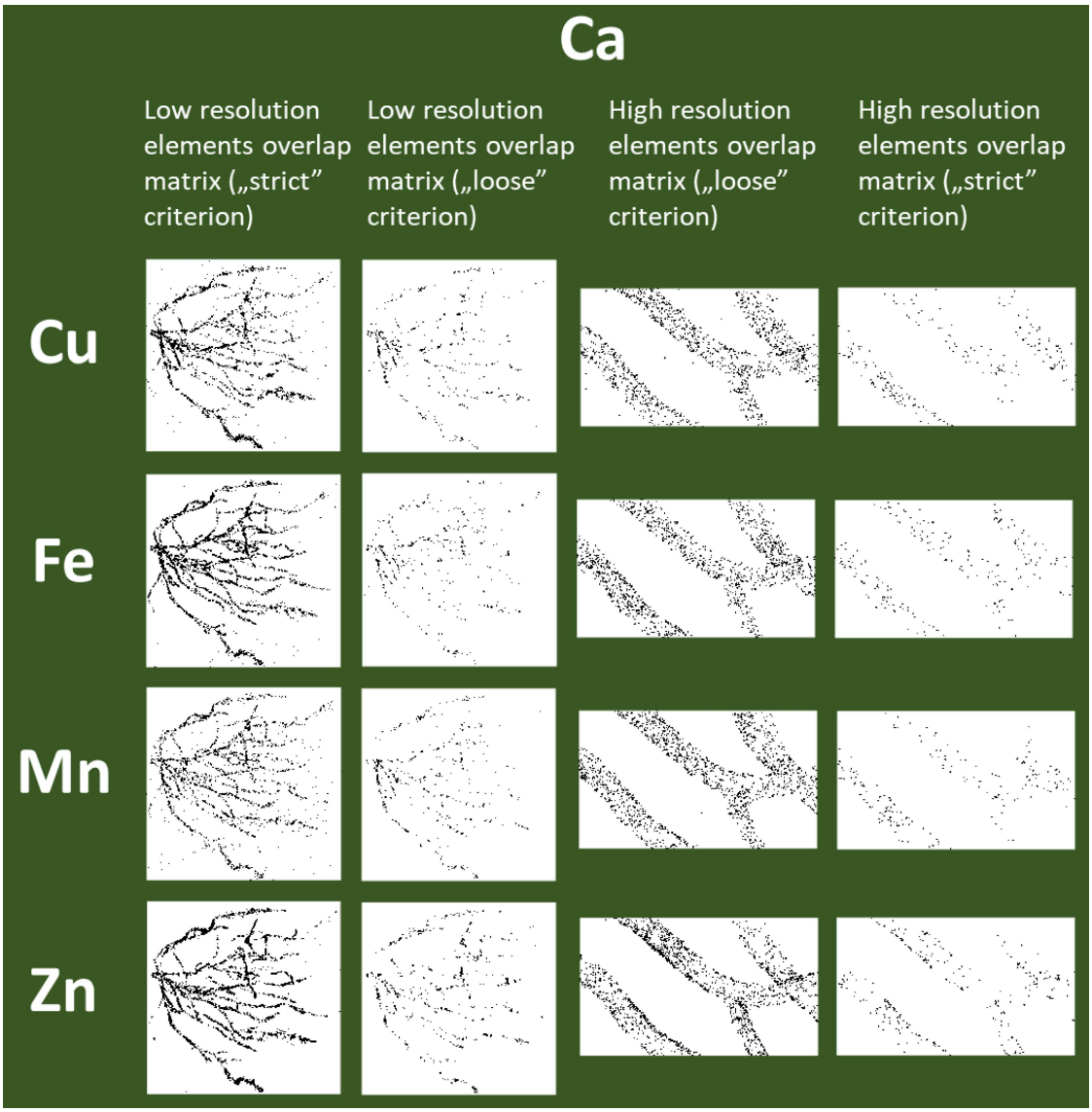

Fig. S3. Resolution-dependent co-localization matrices of Ca with other elements. Matrix overviews of binary overlap masks for Ca paired with Cu, Fe, Mn, and Zn under two imaging resolutions: (a) low (100 μm) and (b) high (5 μm). Each matrix includes masks generated using strict (97%) and loose (92%) pixel-wise overlap thresholds, as indicated. Individual cells show the spatial overlap between Ca and the element specified in the corresponding row, under the resolution and threshold condition defined by the column.

Codes used in the study (all need path and filename adjustment to be used)

Code 1

```
#CODE main to analyse co-localization and make composite .tiff images with overlaps
from pathlib import Path
import itertools
import numpy as np
import tifffile as tiff
import pandas as pd
from scipy.stats import pearsonr, spearmanr

# ==== SETTINGS ====

input_dir = Path(r"C:\...") #Please input full path to your images
output_dir = input_dir / "coloc_results"
output_dir.mkdir(exist_ok = True)

elements_to_use = ["Zn-Ka", "Fe-Kb", "Cu-Ka", "Mn-Ka", "Ca-Ka"] #depending on number of elements to be analysed and if given phrase is present in the file name — please adjust.

low_cut_percent = 5
high_cut_percent = 1
strict_percentile = 97
loose_percentile = 92

# ==== HELPERS ====

def load_tiff(path):
  return tiff.imread(path).astype(np.float32)

def normalize_per_channel(img, low_cut = 5, high_cut = 1):
  low = np.percentile(img, low_cut)
  high = np.percentile(img, 100 - high_cut)
  x = np.clip(img, low, high) - low
  scale = max(high - low, 1e-9)
  return (x / scale).astype(np.float32)

def to_uint8(mask):
  return (mask.astype(np.uint8) * 255)

def save_rgb_tiff(path, r, g, b):
  rgb_cyx = np.stack([to_uint8(r), to_uint8(g), to_uint8(b)], axis = 0)
  tiff.imwrite(path, rgb_cyx, imagej = True,
   metadata = {'axes': 'CYX'})

def manders(a, b, ta, tb):
  mask_a = a >= ta
  mask_b = b >= tb
  M1 = float(a[mask_b].sum() / a.sum()) if a.sum() > 0 else 0.0
  M2 = float(b[mask_a].sum() / b.sum()) if b.sum() > 0 else 0.0
  return M1, M2

# ==== FIND FILES ====

element_files = { }
for f in input_dir.glob("*.tif*"):
  #depending on name used this may be different — our files had _ROI_ — if other phrase is used, rcplease adjust
  name = f.stem
   if "_ROI_" in name:
     element = name.split("_ROI_")[-1]
     if element in elements_to_use:
       element_files[element] = f

elements = sorted(element_files.keys())
print("Found elements:", elements)

# ==== LOOP OVER PAIRS ====

results = []
for el1, el2 in itertools.combinations(elements, 2):
  print(f"Processing: {el1} vs {el2}")

img1 = normalize_per_channel(load_tiff(element_files[el1]), low_cut_percent, high_cut_percent)
img2 = normalize_per_channel(load_tiff(element_files[el2]), low_cut_percent, high_cut_percent)

# Thresholds

thr1_strict = np.percentile(img1, strict_percentile)
thr2_strict = np.percentile(img2, strict_percentile)
thr1_loose = np.percentile(img1, loose_percentile)
thr2_loose = np.percentile(img2, loose_percentile)

# Binary masks

bin1_strict = (img1 >= thr1_strict)
bin2_strict = (img2 >= thr2_strict)
overlap_strict = bin1_strict & bin2_strict
bin1_loose = (img1 >= thr1_loose)
bin2_loose = (img2 >= thr2_loose)
overlap_loose = bin1_loose & bin2_loose

# Save RGB TIFFs

save_rgb_tiff(output_dir / f"{el1}_{el2}_strict_mask_rgb.tiff", bin1_strict, bin2_strict, overlap_strict)
```

```
save_rgb_tiff(output_dir / f"{el1}_{el2}_loose_mask_rgb.tiff", bin1_loose, bin2_loose, overlap_loose)

# Metrics

flat1 = img1.flatten()
flat2 = img2.flatten()
pearson = pearsonr(flat1, flat2)[0]
spearman = spearmanr(flat1, flat2)[0]
M1_strict, M2_strict = manders(img1, img2, thr1_strict, thr2_strict)
M1_loose, M2_loose = manders(img1, img2, thr1_loose, thr2_loose)

results.append({
  "Element1": el1, "Element2": el2,
  "Pearson": pearson,
  "Spearman": spearman,
  "Manders_M1_strict": M1_strict,
  "Manders_M2_strict": M2_strict,
  "Manders_M1_loose": M1_loose,
  "Manders_M2_loose": M2_loose,
  "Thr1_strict": float(thr1_strict),
  "Thr2_strict": float(thr2_strict),
  "Thr1_loose": float(thr1_loose),
  "Thr2_loose": float(thr2_loose),
  "Pixels_used": img1.size
})

# ==== SAVE CSV ====

df = pd.DataFrame(results)
csv_path = output_dir / "co-localization_metrics.csv"
df.to_csv(csv_path, index = False)
print(f"Saved metrics table: {csv_path}")
```

Code 2

```
#CODE that makes heatmaps and graphs to summarize quantification from dataset extracted

from co-localization file
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# === Combined dataset (here past the data from co-localization_metrics.csv generated in #CODE 1) ===
# Each tuple is:
# (Element1, Element2,
# M5_loose, M100_loose,
# M5_strict, M100_strict,
# Pearson_5, Pearson_100,
# Spearman_5, Spearman_100)
pairs = [
  ("Ca","Cu", Manders for 5uM strict,
  Manders for 5uM loose,
  Manders for 100uM strict,
  Manders for 100uM loose,
  Pearson for 5uM,
  Pearson for 100uM,
  Spearman for 5uM,
  Spearman for 100uM),
]
elements = ["Ca","Cu","Fe","Mn","Zn"]

# === Build matrices ===

def empty_matrix():
return pd.DataFrame(np.nan,
index = elements,
columns = elements)

m5_loose, m100_loose = empty_matrix(), empty_matrix()
m5_strict, m100_strict = empty_matrix(), empty_matrix()

p5, p100 = empty_matrix(), empty_matrix()
s5, s100 = empty_matrix(), empty_matrix()
manders_loose_robust, manders_strict_robust, pearson_robust, spearman_robust, combined_stability = {}, {}, {}, {}

for e1,e2,m5l,m100l,m5s,m100s,p5v,p100v,s5v, s100v in pairs:
  # Fill symmetric matrices
  for M,val in [(m5_loose,m5l),(m100_loose,m100l),(m5_strict,m5s),(m100_strict,m100s),(p5,p5v),(p100,p100v),(s5,s5v),(s100,s100v)]:
    M.loc[e1,e2] = M.loc[e2,e1] = val
  key = f"{e1}-{e2}"
  manders_loose_robust[key] = m100l/m5l if m5l != 0 else np.nan
  manders_strict_robust[key] = m100s/m5s if m5s !=0 else np.nan
  pearson_robust[key] = p100v/p5v if p5v != 0 else np.nan
  spearman_robust[key] = s100v/s5v if s5v !=0 else np.nan
  combined_stability[key] = 1 / (abs(manders_loose_robust[key]-1) + abs(manders_strict_robust[key]-1) + abs(pearson_robust[key]-1) + abs(spearman_robust[key]-1))

# Differences

mdiff_loose = m100_loose - m5_loose
mdiff_strict = m100_strict - m5_strict
pdiff = p100 - p5
sdiff = s100 - s5
```

```
# === Combined figure ===

sns.set(style="white")
fig = plt.figure(figsize = (28,18))
gs = fig.add_gridspec(4,4, width_ratios = [1,1,1,0.7])

heatmaps = [
  (m5_loose, "Manders Loose (5µm)", "magma", None, (0,0)),
  (m100_loose, "Manders Loose (100µm)", "magma", None, (0,1)),
  (mdiff_loose, "Manders Loose Δ", "RdBu_r", 0, (0,2)),
  (m5_strict, "Manders Strict (5µm)", "magma", None, (1,0)),
  (m100_strict, "Manders Strict (100µm)", "magma", None, (1,1)),
  (mdiff_strict, "Manders Strict Δ", "RdBu_r", 0, (1,2)),
  (p5, "Pearson (5µm)", "coolwarm", 0, (2,0)),
  (p100, "Pearson (100µm)", "coolwarm", 0, (2,1)),
  (pdiff, "Pearson Δ", "RdBu_r", 0, (2,2)),
  (s5, "Spearman (5µm)", "coolwarm", 0, (3,0)),
  (s100, "Spearman (100µm)", "coolwarm", 0, (3,1)),
  (sdiff, "Spearman Δ", "RdBu_r", 0, (3,2)),
]

for data, title, cmap, center, (r,c) in heatmaps:
  ax = fig.add_subplot(gs[r,c])
  sns.heatmap(data, annot = True, fmt = ".2f", cmap = cmap, center = center,
  xticklabels = elements, yticklabels = elements, square = True,
  cbar_kws = {'label':'Value'}, ax = ax)
  ax.set_title(title)

# Bar charts

ax1 = fig.add_subplot(gs[0,3])
sns.barplot(
  x = list(manders_loose_robust.values()),
  y = list(manders_loose_robust.keys()),
  color = "darkorange", ax = ax1)
ax1.axvline(1.0, color = "gray", linestyle = "–");

ax1.set_title("Manders Loose Robustness")

ax2 = fig.add_subplot(gs[1,3])
sns.barplot(
  x = list(manders_strict_robust.values()),
  y = list(manders_strict_robust.keys()),
  color = "purple", ax = ax2)
ax2.axvline(1.0, color = "gray", linestyle = "–");
ax2.set_title("Manders Strict Robustness")

ax3 = fig.add_subplot(gs[2,3])
sns.barplot(
  x = list(pearson_robust.values()),
  y = list(pearson_robust.keys()),
  color = "steelblue", ax = ax3)
ax3.axvline(1.0, color = "gray", linestyle = "–");
ax3.set_title("Pearson Robustness")

ax4 = fig.add_subplot(gs[3,3])
sns.barplot(
  x = list(spearman_robust.values()),
  y = list(spearman_robust.keys()),
  color = "teal", ax = ax4)
ax4.axvline(1.0, color = "gray", linestyle = "–");
ax4.set_title("Spearman Robustness")

plt.tight_layout()
plt.savefig("coloc_summary_combined1.png", dpi = 300, bbox_inches = "tight")
plt.savefig("coloc_summary_combined1.tiff", dpi = 300, bbox_inches = "tight")
plt.show()

# === Combined stability bar chart ===

plt.figure(figsize = (8,5))
stability_sorted = dict(sorted(combined_stability.items(), key = lambda x:x[1], reverse = True))
sns.barplot(x = list(stability_sorted.values()),
  y = list(stability_sorted.keys()),
  color = "seagreen")
plt.title("Combined Stability Score")
plt.tight_layout()
plt.savefig("coloc_combined.png", dpi = 300)
plt.savefig("coloc_combined.tiff", dpi = 300)
plt.show()
```

Code 3

```
#CODE that takes composite image and extracts overlap channel (third) making overlap matrix but inverted (W background B signal)

import os
import numpy as np
import matplotlib.pyplot as plt
import tifffile as tiff

# === SETTINGS ===

input_dir = r"C:\Users\..."
#Please input full path to your images
```

```
elements = ["Ca-Ka", "Cu-Ka", "Fe-Kb", "Mn-Ka", "Zn-Ka"] #depending on number of elements to be analysed and if given phrase is present in the file name – please adjust.
def create_overlap_matrix(threshold = "strict"):
  n = len(elements)
  fig, axs = plt.subplots(n, n, figsize = (1.6 * n, 1.6 * n))
  for i, e1 in enumerate(elements):
    for j, e2 in enumerate(elements):
    ax = axs[i, j]
    ax.axis("off")
    if i == j:
      ax.text(0.5, 0.5, e1, ha = "center", va = "center", fontsize = 9)
    else:
      el1, el2 = sorted([e1, e2])
      fname = f"{el1}_{el2}_{threshold}_mask_rgb.tiff" #depending on file name!
      fpath = os.path.join(input_dir, fname)
      if os.path.exists(fpath):
        rgb = tiff.imread(fpath)
        # Extract overlap channel (channel index 2)
        if rgb.ndim == 3 and rgb.shape[0] == 3: # CYX
          overlap = rgb[2] > 0
        elif rgb.ndim == 3 and rgb.shape[-1] == 3: # HWC
          overlap = rgb[..., 2] > 0
        else:
          continue
        # Invert the grayscale image
        inverted = np.logical_not(overlap)
        ax.imshow(inverted, cmap = "gray")
      else:
        ax.text(0.5, 0.5, "N/A", ha = "center", va = "center", fontsize = 8, color = "gray")

  fig.suptitle(f"Overlap Matrix ({threshold} )", y = 0.995)
  fig.tight_layout()
  out_png = os.path.join(input_dir, f"overlap_matrix_{threshold}_overlap.png")
  out_tif = os.path.join(input_dir, f"overlap_matrix_{threshold}_overlap.tiff")
  fig.savefig(out_png, dpi = 300, bbox_inches = "tight")
  fig.savefig(out_tif, dpi = 300, bbox_inches = "tight")
  plt.show()

# === CREATE BOTH STRICT & LOOSE MATRICES ===

create_overlap_matrix("strict")
create_overlap_matrix("loose")
```

References

[1] M. Pypka, A. Barabasz, K. Sowa, P. Wróbel, T. Kołodziej, P. Korecki, O. Siemianowski, *Acta Phys. Pol. A* **149**, 218 (2026).