Image-Driven Foam Stability Analysis from Microscopy

Linking Bubble Dynamics to Foam Stability with Explainable Machine Learning

Microscopy Hackathon 2025 — AISCIA Informatics Use Case

Author: Hesham Eina Abdalla
Background: Electrical Engineer | Embedded Systems | PCB Design | AI & IoT Enthusiast | Qatar University Alumni

GitHub: https://github.com/hdaw1905
LinkedIn: https://www.linkedin.com/in/hesham-eina/

Executive Summary

This project presents a fully image-driven framework for understanding and comparing foam stability using time-resolved microscopy data. Bubble-level geometric, spatial, and temporal features are extracted directly from segmented microscopy images and analyzed to reveal how microscopic bubble dynamics govern macroscopic foam behavior.

By focusing on interpretable visual descriptors—such as bubble size evolution, population persistence, spatial packing, and size-distribution heterogeneity—this workflow links image-observable patterns to foam stability without relying on chemistry-specific parameters. Explainable machine learning models are then used to quantify the relative importance of these microscopic features and to construct image-derived foam stability proxies.

End-to-End Image-to-Insight Workflow

This project follows a fully image-first workflow that transforms microscopy observations into physical insight and predictive metrics of foam stability:

Microscopy Images
→ Binary Segmentation of Bubble Regions
→ Bubble Geometry and Spatial Feature Extraction
→ Temporal Dynamics Analysis (Coarsening, Entropy, Packing)
→ Image-Derived Foam Stability Proxies
→ Explainable Machine Learning
→ Physical Interpretation of Foam Stability Mechanisms

All results are derived directly from real microscopy images and their time evolution, without using chemical composition or empirical fitting parameters.

📂 Dataset Setup (Required Before Running This Notebook)

This notebook expects the dataset to be located in **Google Drive** at the following path:

`/content/drive/MyDrive/AI_Microscopy_FoamDynamics`

Setup Instructions for Judges

1. Download the dataset from the following Google Drive link:
   https://drive.google.com/drive/folders/1O1y_lesCDsoj3T3hb9_en9Q1v9cR2eQ-?usp=sharing
2. **Unzip** the downloaded dataset on your local machine.
3. Open **Google Drive** in your browser and navigate to **My Drive**.
4. Upload the **entire unzipped folder** named `AI_Microscopy_FoamDynamics`.
5. Ensure the folder name is **exactly** `AI_Microscopy_FoamDynamics` *(case-sensitive, no extra spaces)*.
6. Open this notebook in **Google Colab** and run all cells from top to bottom.

If the dataset is placed correctly, the notebook will automatically set the project root and continue without requiring any code modification.

```
# ============================================
# Google Colab Setup: Mount Drive & Set Project Root
# ============================================

from google.colab import drive
import os
from pathlib import Path

drive.mount('/content/drive')

# 📂 DO NOT CHANGE AFTER SUBMISSION
DATA_ROOT = Path("/content/drive/MyDrive/AI_Microscopy_FoamDynamics")

assert DATA_ROOT.exists(), (
    "Dataset folder not found.\n"
    "Expected: /content/drive/MyDrive/AI_Microscopy_FoamDynamics"
)

os.chdir(DATA_ROOT)
ROOT = Path.cwd()

print("Working directory set to:", ROOT)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
Working directory set to:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics
```

AI Microscopy Foam Dynamics — Image-First ML Pipeline

Goal: link bubble patterns in microscopy images to foam stability (coarsening + distributions + collapse), and compare stability across nanoparticle conditions (baseline vs GO vs NGO).

We use:

- segmented bubble images (binary)
- bubble-center CSVs (x,y per bubble per timestep)
- optional experiment CSVs for reference (breakthrough, mean area, counts)

Core Scientific Python Libraries

```
import os
from pathlib import Path
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from skimage import io, measure, morphology
from scipy import stats
from scipy.spatial import cKDTree, Delaunay
```

Experiment Configuration and Conditions

```
ROOT = Path(".").resolve()  # you run notebook inside
AI_Microscopy_FoamDynamics

EXPERIMENTS = {
    "E130": {"folder": "Experiment-130", "label": "Surfactant + 150
ppm NP (GO)"},
    "E138": {"folder": "Experiment-138", "label": "Surfactant only
(baseline)"},
    "E142": {"folder": "Experiment-142", "label": "Surfactant + 1500
ppm NP (NGO)"},
}

def exp_path(exp_id: str) -> Path:
    return ROOT / EXPERIMENTS[exp_id]["folder"]
```

Dataset Path Resolution Utilities

```
def find_nested_dir(base: Path, name: str) -> Path:
    """
    Finds a directory named `name` inside base, possibly nested one
level (like name/name).
    """
    direct = base / name
    if direct.exists():
        # if it contains another same-named folder, prefer that
        nested = direct / name
        return nested if nested.exists() else direct

    # fallback: search
    hits = list(base.rglob(name))
    hits = [h for h in hits if h.is_dir()]
    if not hits:
        raise FileNotFoundError(f"Could not find directory '{name}'
under {base}")
    # pick the deepest path (usually base/name/name)
    hits.sort(key=lambda p: len(p.parts), reverse=True)
    return hits[0]

def list_images(seg_dir: Path):
    exts = ("*.png", "*.jpg", "*.jpeg", "*.tif", "*.tiff", "*.bmp")
    files = []
    for e in exts:
        files += list(seg_dir.glob(e))
    files = sorted(files)
    if not files:
        raise FileNotFoundError(f"No images found in {seg_dir}")
    return files

def list_center_csvs(center_dir: Path):
    csvs = sorted(center_dir.glob("*.csv"))
    if not csvs:
        raise FileNotFoundError(f"No center CSVs found in
{center_dir}")
    return csvs
```

Dataset Inventory and Consistency Check

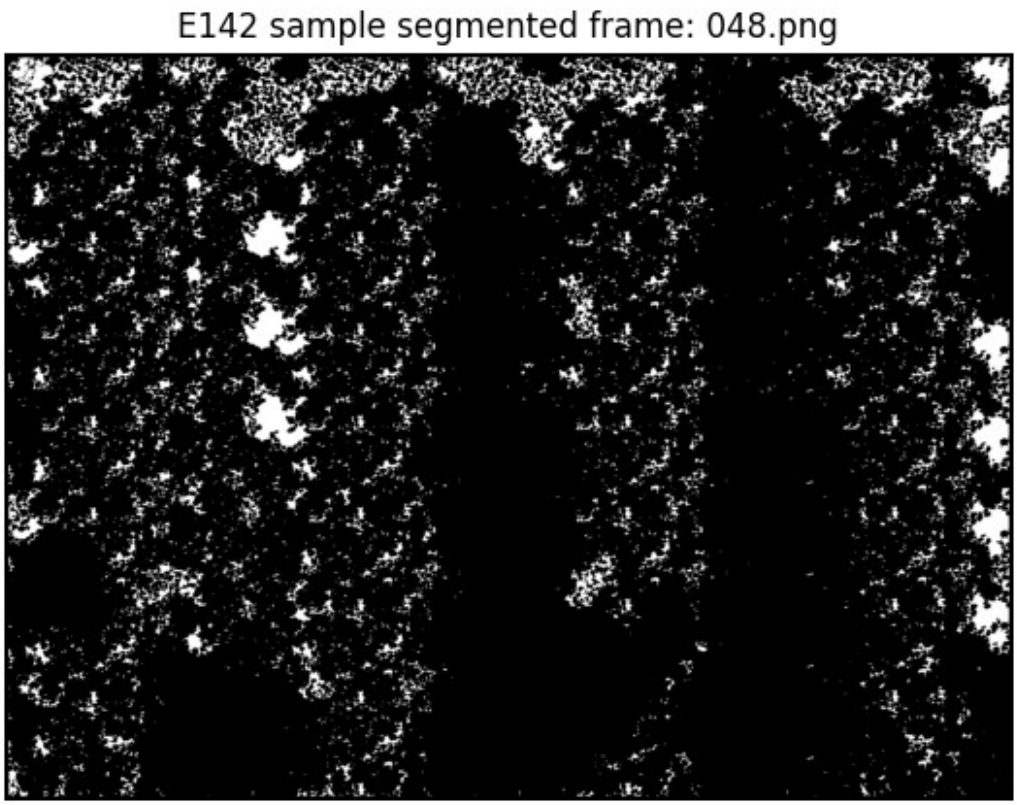
```
for exp_id in EXPERIMENTS:
    p = exp_path(exp_id)

    seg_dir = find_nested_dir(p, "Segmented_pictures")
    cen_dir = find_nested_dir(p, "bubbles_center")

    imgs = list_images(seg_dir)
    csvs = list_center_csvs(cen_dir)
```

```
    print(exp_id, "-", EXPERIMENTS[exp_id]["label"])
    print("  Segmented images:", seg_dir, "count:", len(imgs))
    print("  Center CSVs:", cen_dir, "count:", len(csvs))
```

```
E130 - Surfactant + 150 ppm NP (GO)
  Segmented images:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-130/Segme
nted_pictures/Segmented_pictures count: 166
  Center CSVs:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-130/bubbl
es_center/bubbles_center count: 166
E138 - Surfactant only (baseline)
  Segmented images:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-138/Segme
nted_pictures/Segmented_pictures count: 149
  Center CSVs:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-138/bubbl
es_center/bubbles_center count: 470
E142 - Surfactant + 1500 ppm NP (NGO)
  Segmented images:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-142/Segme
nted_pictures/Segmented_pictures count: 188
  Center CSVs:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-142/bubbl
es_center/bubbles_center count: 188
```

Visual Sanity Check: Segmented Microscopy Frames

```
exp_id = "E142"  # change to E138/E130 to compare
seg_dir = find_nested_dir(exp_path(exp_id), "Segmented_pictures")
imgs = list_images(seg_dir)

img0 = io.imread(imgs[0])
plt.figure(figsize=(10,5))
plt.imshow(img0, cmap="gray")
plt.title(f"{exp_id} sample segmented frame: {imgs[0].name}")
plt.axis("off")
plt.show()

print("Unique values:", np.unique(img0)[:10], " ...")
```




```
Unique values: [  0 255]  ...
```

Image-to-Binary Mask Conversion and Cleaning

```python
# ============================================================
# Binary Mask Utilities (Must Run Before Any Visualization)
# ============================================================

import numpy as np
from skimage import morphology

def to_binary(img):
    if img.ndim == 3:
        img = img[..., 0]

    if img.max() > 1:
        m = img > 127
    else:
        m = img > 0.5

    if m.mean() > 0.5:
        m = ~m

    return m.astype(np.uint8)
```

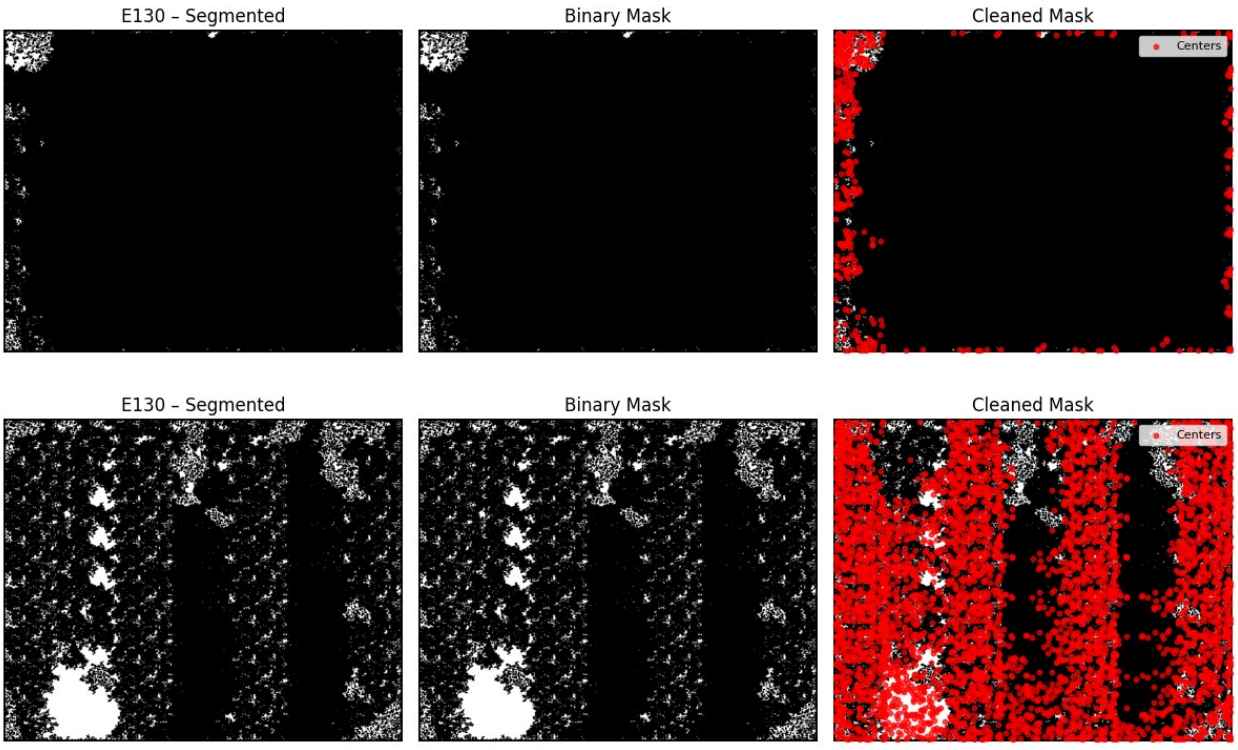
```python
def clean_mask(mask, min_size=30):
    m = mask.astype(bool)
    m = morphology.remove_small_objects(m, min_size=min_size)
    m = morphology.remove_small_holes(m, area_threshold=min_size)
    return m.astype(np.uint8)

print("□ to_binary and clean_mask loaded")
```

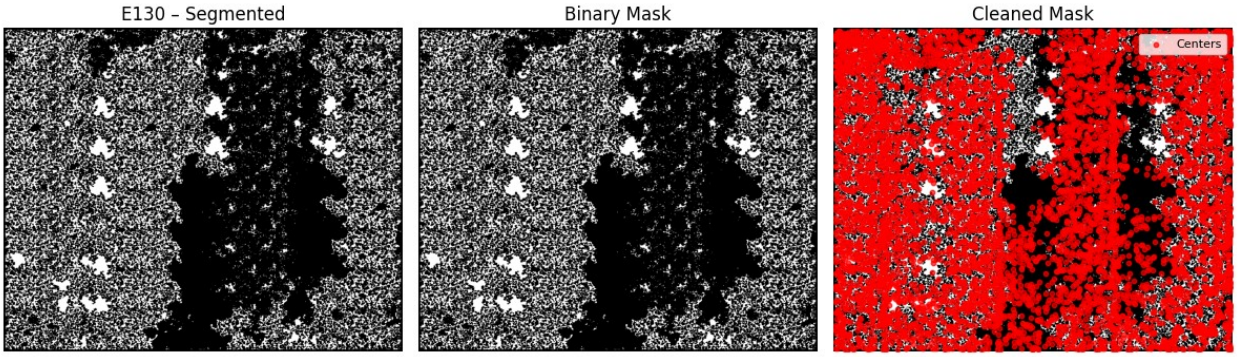
```
□ to_binary and clean_mask loaded
```

```python
import matplotlib.pyplot as plt
from skimage.io import imread
import pandas as pd

MAX_SHOW = 3  # how many frames to visualize per experiment

for exp_id in EXPERIMENTS:
    p = exp_path(exp_id)

    seg_dir = find_nested_dir(p, "Segmented_pictures")
    cen_dir = find_nested_dir(p, "bubbles_center")

    imgs = list_images(seg_dir)
    csvs = list_center_csvs(cen_dir)

    print("=" * 60)
    print(exp_id, "-", EXPERIMENTS[exp_id]["label"])
    print("  Segmented images:", seg_dir, "count:", len(imgs))
    print("  Center CSVs:", cen_dir, "count:", len(csvs))

    for i in range(min(MAX_SHOW, len(imgs))):
        img = imread(imgs[i])
        binary = to_binary(img)
        cleaned = clean_mask(binary)

        fig, ax = plt.subplots(1, 3, figsize=(12, 4))

        ax[0].imshow(img, cmap="gray")
        ax[0].set_title(f"{exp_id} — Segmented")
        ax[0].axis("off")

        ax[1].imshow(binary, cmap="gray")
        ax[1].set_title("Binary Mask")
        ax[1].axis("off")

        ax[2].imshow(cleaned, cmap="gray")
        ax[2].set_title("Cleaned Mask")
        ax[2].axis("off")

        # Overlay bubble centers
```

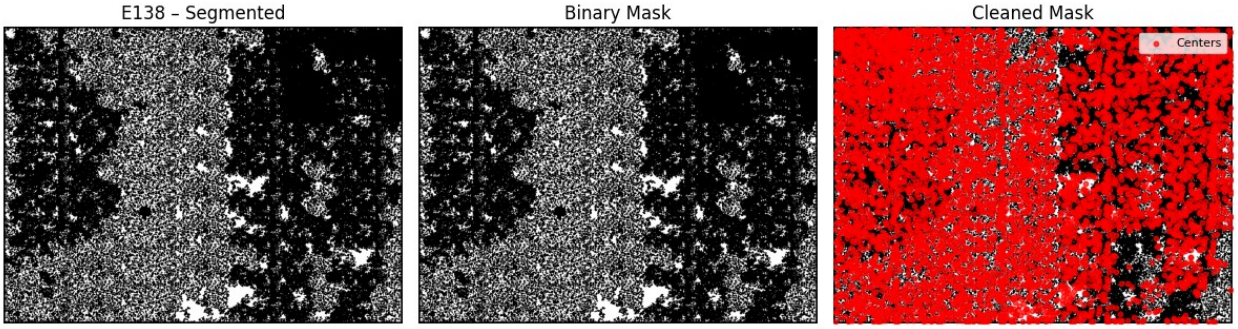
```
        if i < len(csvs):
            centers = pd.read_csv(csvs[i])
            if {"x", "y"}.issubset(centers.columns):
                ax[2].scatter(
                    centers["x"],
                    centers["y"],
                    s=10,
                    c="red",
                    alpha=0.7,
                    label="Centers"
                )
                ax[2].legend(loc="upper right", fontsize=8)

        plt.tight_layout()
        plt.show()
```

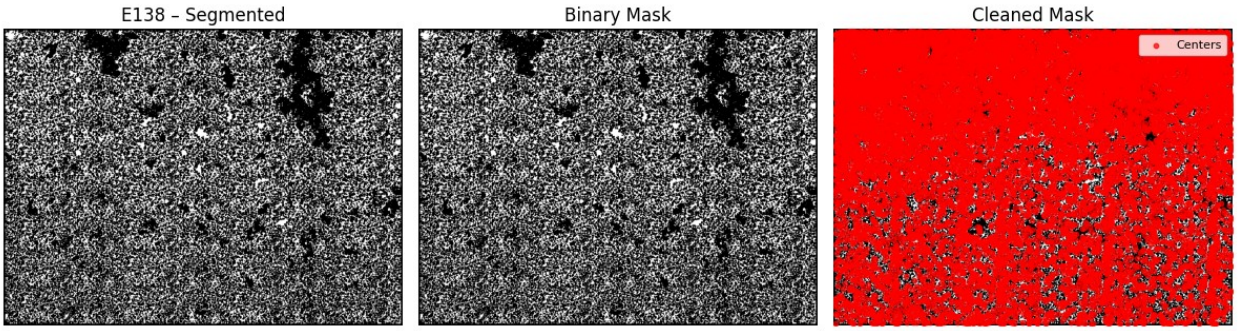
```
============================================================
E130 - Surfactant + 150 ppm NP (GO)
  Segmented images:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-130/Segmented_pictures/Segmented_pictures count: 166
  Center CSVs:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-130/bubbles_center/bubbles_center count: 166
```

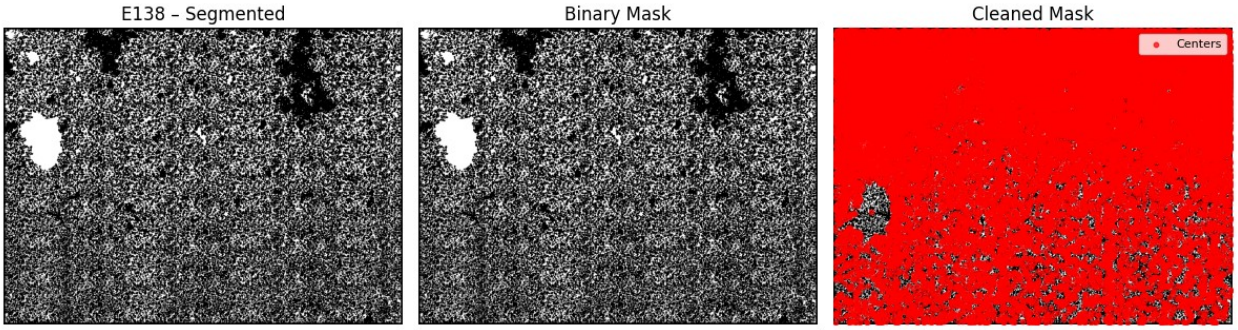




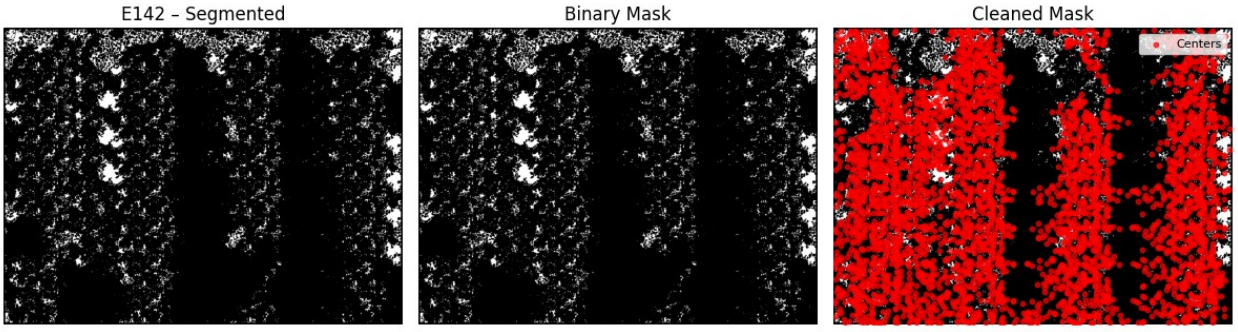

```
============================================================
E138 - Surfactant only (baseline)
  Segmented images:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-138/Segme
nted_pictures/Segmented_pictures count: 149
  Center CSVs:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-138/bubbl
es_center/bubbles_center count: 470
```

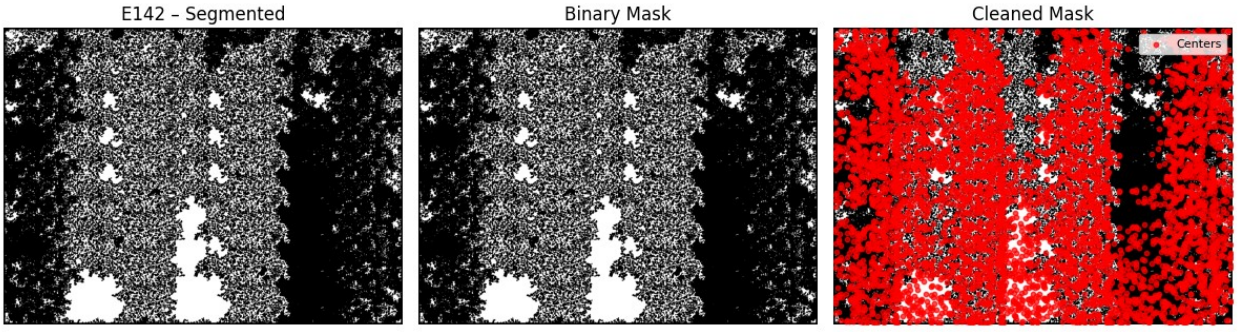


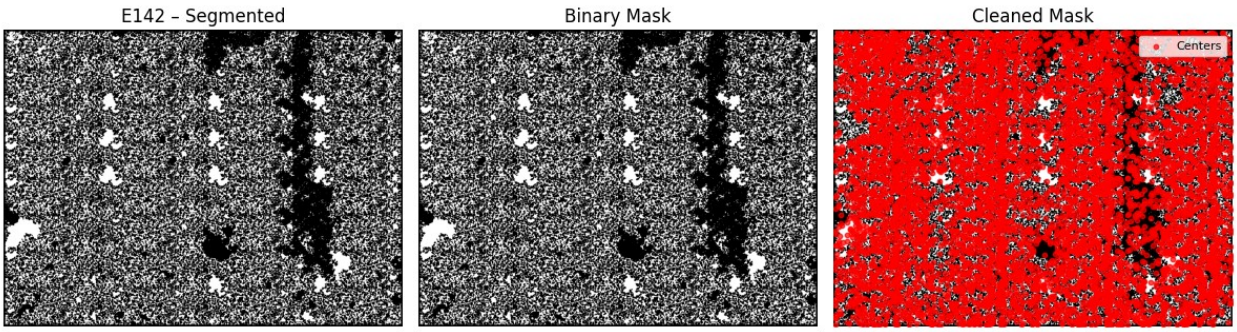





```
============================================================
E142 - Surfactant + 1500 ppm NP (NGO)
  Segmented images:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-142/Segme
nted_pictures/Segmented_pictures count: 188
  Center CSVs:
/content/drive/MyDrive/AI_Microscopy_FoamDynamics/Experiment-142/bubbl
es_center/bubbles_center count: 188
```







Bubble Geometry Feature Extraction (Region-Based)

```
from skimage.measure import label
import numpy as np

def image_region_features(mask, bins=30):
    """
    Fast region stats from labeled mask, with interpretable
distribution metrics.
    """
    lbl = label(mask)
    areas = np.bincount(lbl.ravel())[1:]  # skip background
    if len(areas) == 0:
        return {
            "n_bubbles_img": 0,
            "area_mean": np.nan,
            "area_median": np.nan,
            "area_std": np.nan,
            "area_q10": np.nan,
            "area_q90": np.nan,
            "area_iqr": np.nan,
            "area_cv": np.nan,
            "entropy_area": np.nan,
            "gini_area": np.nan,
        }

    areas = areas.astype(float)

    # robust stats
    q10, q50, q90 = np.percentile(areas, [10, 50, 90])
    iqr = np.percentile(areas, 75) - np.percentile(areas, 25)
    cv = areas.std() / (areas.mean() + 1e-9)

    # histogram entropy (probabilities, not density)
    hist = np.histogram(areas, bins=bins)[0].astype(float)
    p = hist / (hist.sum() + 1e-12)
    entropy = -np.sum(p * np.log(p + 1e-12))

    # Gini coefficient (inequality in bubble sizes)
```

```
    a = np.sort(areas)
    n = len(a)
    gini = (2*np.sum((np.arange(1, n+1))*a) / (n*np.sum(a)+1e-12)) -
(n+1)/n

    return {
        "n_bubbles_img": int(len(areas)),
        "area_mean": float(areas.mean()),
        "area_median": float(q50),
        "area_std": float(areas.std()),
        "area_q10": float(q10),
        "area_q90": float(q90),
        "area_iqr": float(iqr),
        "area_cv": float(cv),
        "entropy_area": float(entropy),
        "gini_area": float(gini),
    }
```

Spatial Feature Extraction from Bubble Centers

```
def read_centers(csv_path: Path):
    df = pd.read_csv(csv_path)
    # handle cases: columns might be ["x", "y"] or single column
    cols = [c.strip().lower() for c in df.columns]
    if len(cols) >= 2:
        x = pd.to_numeric(df.iloc[:,0], errors="coerce").to_numpy()
        y = pd.to_numeric(df.iloc[:,1], errors="coerce").to_numpy()
    else:
        # fallback if file is weirdly formatted
        txt = csv_path.read_text().strip().splitlines()
        pts = []
        for line in txt:
            parts = re.split(r"[,\s]+", line.strip())
            if len(parts) >= 2:
                pts.append((float(parts[0]), float(parts[1])))
        arr = np.array(pts, float)
        x, y = arr[:,0], arr[:,1]

    pts = np.vstack([x, y]).T
    pts = pts[np.isfinite(pts).all(axis=1)]
    return pts

def center_spatial_features(pts, k=6):
    """
    Fast spatial features from bubble centers.
    Uses kNN distances only (Delaunay removed for speed).
    """
    if pts.shape[0] < k + 1:
        return {
```

```
            "n_centers": int(pts.shape[0]),
            "knn_dist_mean": np.nan,
            "knn_dist_std": np.nan,
            "knn_dist_cv": np.nan,
        }

    tree = cKDTree(pts)
    dists, _ = tree.query(pts, k=k+1)
    knn = dists[:, 1:]

    return {
        "n_centers": int(pts.shape[0]),
        "knn_dist_mean": float(knn.mean()),
        "knn_dist_std": float(knn.std()),
        "knn_dist_cv": float(knn.std() / (knn.mean() + 1e-9)),
    }
```

Frame-Level Feature Assembly (Geometry + Spatial)

```
def extract_frame_features(img_path: Path, center_csv_path: Path):
    img = io.imread(img_path)
    mask = clean_mask(to_binary(img), min_size=25)

    geom = image_region_features(mask)

    pts = read_centers(center_csv_path)
    spatial = center_spatial_features(pts, k=6)

    feats = {}
    feats.update(geom)
    feats.update(spatial)
    return feats
```

Time-Series Feature Construction and I/O Optimization

```
from pathlib import Path
import shutil, os, time

FULL_FRAMES = True
FRAME_STRIDE = 1 if FULL_FRAMES else 5

# Copy dataset from Drive to local runtime disk for FAST I/O
COPY_TO_LOCAL = True

DRIVE_ROOT = ROOT  # currently
/content/drive/MyDrive/AI_Microscopy_FoamDynamics
LOCAL_ROOT = Path("/content/AI_Microscopy_FoamDynamics")

if COPY_TO_LOCAL:
    if not LOCAL_ROOT.exists():
```

```
        print("Copying dataset to local disk (one-time)...")
        t0 = time.time()
        shutil.copytree(DRIVE_ROOT, LOCAL_ROOT)
        print("Done. Copy time (s):", round(time.time() - t0, 2))
    else:
        print("Local dataset already exists:", LOCAL_ROOT)

    WORK_ROOT = LOCAL_ROOT
else:
    WORK_ROOT = DRIVE_ROOT

print("WORK_ROOT =", WORK_ROOT)
print("FRAME_STRIDE =", FRAME_STRIDE)
```

```
Copying dataset to local disk (one-time)...
Done. Copy time (s): 45.33
WORK_ROOT = /content/AI_Microscopy_FoamDynamics
FRAME_STRIDE = 1
```

```
import pandas as pd
from tqdm.auto import tqdm

def exp_path_work(exp_id: str) -> Path:
    return WORK_ROOT / EXPERIMENTS[exp_id]["folder"]

CACHE_DIR = WORK_ROOT / "_cache_features"
CACHE_DIR.mkdir(exist_ok=True)

def build_experiment_timeseries(exp_id: str):
    # cache file per experiment
    cache_path = CACHE_DIR /
f"{exp_id}_features_stride{FRAME_STRIDE}.parquet"
    if cache_path.exists():
        print(f"  ↩ Loading cached features: {cache_path.name}")
        return pd.read_parquet(cache_path)

    p = exp_path_work(exp_id)
    seg_dir = find_nested_dir(p, "Segmented_pictures")
    cen_dir = find_nested_dir(p, "bubbles_center")

    imgs = list_images(seg_dir)
    csvs = list_center_csvs(cen_dir)

    n_total = min(len(imgs), len(csvs))
    imgs = imgs[:n_total:FRAME_STRIDE]
    csvs = csvs[:n_total:FRAME_STRIDE]

    rows = []
    for i, (im, cc) in enumerate(tqdm(list(zip(imgs, csvs)),
                                      desc=f"{exp_id} frames",
                                      total=len(imgs))):
```

```
        f = extract_frame_features(im, cc)

        # Keep BOTH: subsampled index and original index
        f["t"] = i                        # subsampled step index
        f["t_raw"] = i * FRAME_STRIDE   # original frame index
(FULL_FRAMES -> same)
        f["img_name"] = im.name
        f["center_csv"] = cc.name
        f["experiment"] = exp_id

        rows.append(f)

    df =
pd.DataFrame(rows).sort_values("t_raw").reset_index(drop=True)
    df.to_parquet(cache_path, index=False)
    print(f"  □ Saved cache: {cache_path.name}")
    return df

# Build all experiments
series = {}
for exp_id in EXPERIMENTS:
    print(f"\n▶ Building: {exp_id} — {EXPERIMENTS[exp_id]['label']}")
    series[exp_id] = build_experiment_timeseries(exp_id)

series["E142"].head()
```

```
▶ Building: E130 — Surfactant + 150 ppm NP (GO)
  ↩ Loading cached features: E130_features_stride1.parquet

▶ Building: E138 — Surfactant only (baseline)
  ↩ Loading cached features: E138_features_stride1.parquet

▶ Building: E142 — Surfactant + 1500 ppm NP (NGO)
```


```
{"model_id":"566b14ae5c484c029f732a052d53ea3c","version_major":2,"vers
ion_minor":0}
```


```
  □ Saved cache: E142_features_stride1.parquet
```


```
{"summary":"{\n  \"name\": \"series[\\\"E142\\\"]\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"n_bubbles_img\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6395,\n        \"min\": 3487,\n        \"max\": 18836,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          4554,\n
18836,\n          5970\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"area_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 498.5008781006362,\n        \"min\":
379.9540242089616,\n        \"max\": 1452.6102327624067,\n
```

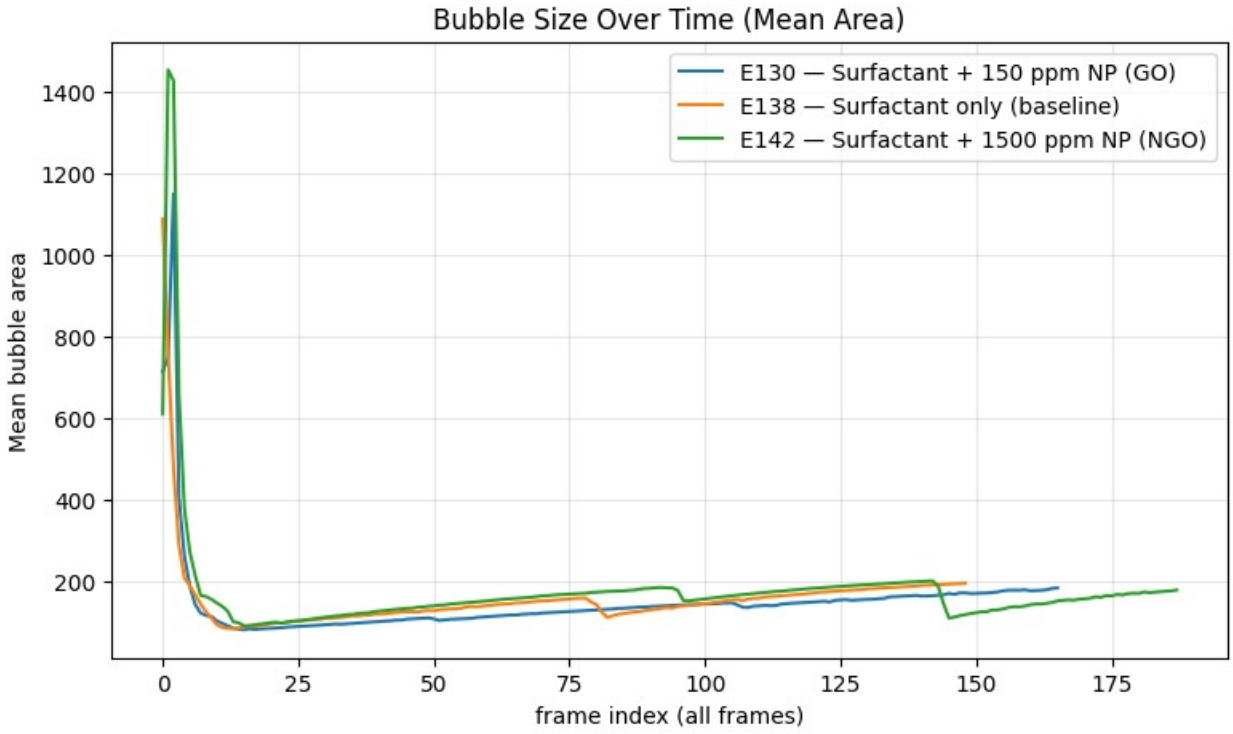
```
\"num_unique_values\": 5,\n        \"samples\": [\n
1452.6102327624067,\n          379.9540242089616,\n
1425.7490787269683\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"area_median\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 29.14275210065103,\n        \"min\":
80.0,\n        \"max\": 146.0,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          80.0,\n          123.0,\n          93.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"area_std\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11816.635527404373,\n        \"min\": 2560.5591449845792,\n
\"max\": 25332.79509767822,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          25332.79509767822,\n
2560.5591449845792,\n          23840.02300198195\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"area_q10\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.816590212458495,\n        \"min\": 30.0,\n        \"max\": 34.0,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          30.0,\n
34.0,\n          31.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"area_q90\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 207.12973712144765,\n        \"min\":
668.0,\n        \"max\": 1174.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          825.0,\n          668.0,\n
769.1000000000004\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"area_iqr\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 89.10274967698808,\n        \"min\":
160.0,\n        \"max\": 379.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          160.0,\n          230.0,\n          168.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"area_cv\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6.275205896166502,\n        \"min\": 4.788561739699308,\n
\"max\": 17.43949927262029,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          17.43949927262029,\n
6.739128899368154,\n          16.72105096027953\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"entropy_area\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.07350772965183044,\n        \"min\": 0.034242625133048356,\n
\"max\": 0.2048481771753467,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.034242625133048356,\n
0.04051744570209406,\n          0.03584152188194616\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"gini_area\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

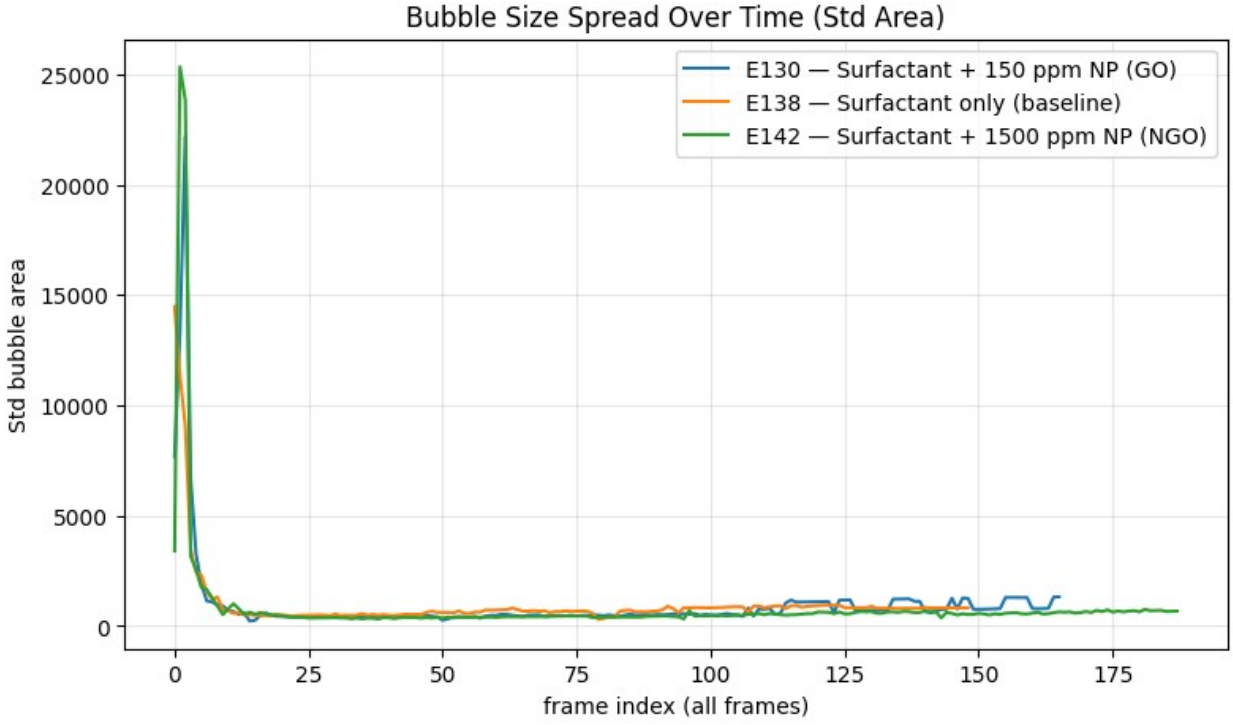
```
0.09786030161994995,\n        \"min\": 0.7143517763943323,\n
\"max\": 0.9378500283584987,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.9378500283584987,\n
0.7143517763943323,\n          0.935660105501873\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"n_centers\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7030,\n        \"min\": 4094,\n        \"max\": 21058,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          5457,\n
21058,\n          7150\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"knn_dist_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 14.014372717563411,\n        \"min\":
33.39772470470769,\n        \"max\": 67.01735956124347,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
60.58108135725576,\n          33.39772470470769,\n
55.23439708637077\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"knn_dist_std\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7.67216827347039,\n        \"min\":
13.963883836218734,\n        \"max\": 32.553998132790504,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
29.02017300457375,\n          13.963883836218734,\n
24.935285068396134\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"knn_dist_cv\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.027406926312695964,\n        \"min\":
0.4181088370320119,\n        \"max\": 0.48575471109922563,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.4790302905449704,\n          0.4181088370320119,\n
0.45144486738857764\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"t\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 4,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1,\n
4,\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"t_raw\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 4,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1,\n
4,\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"img_name\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"049.png\",\n          \"052.png\",\n
\"050.png\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"center_csv\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 5,\n        \"samples\":
```

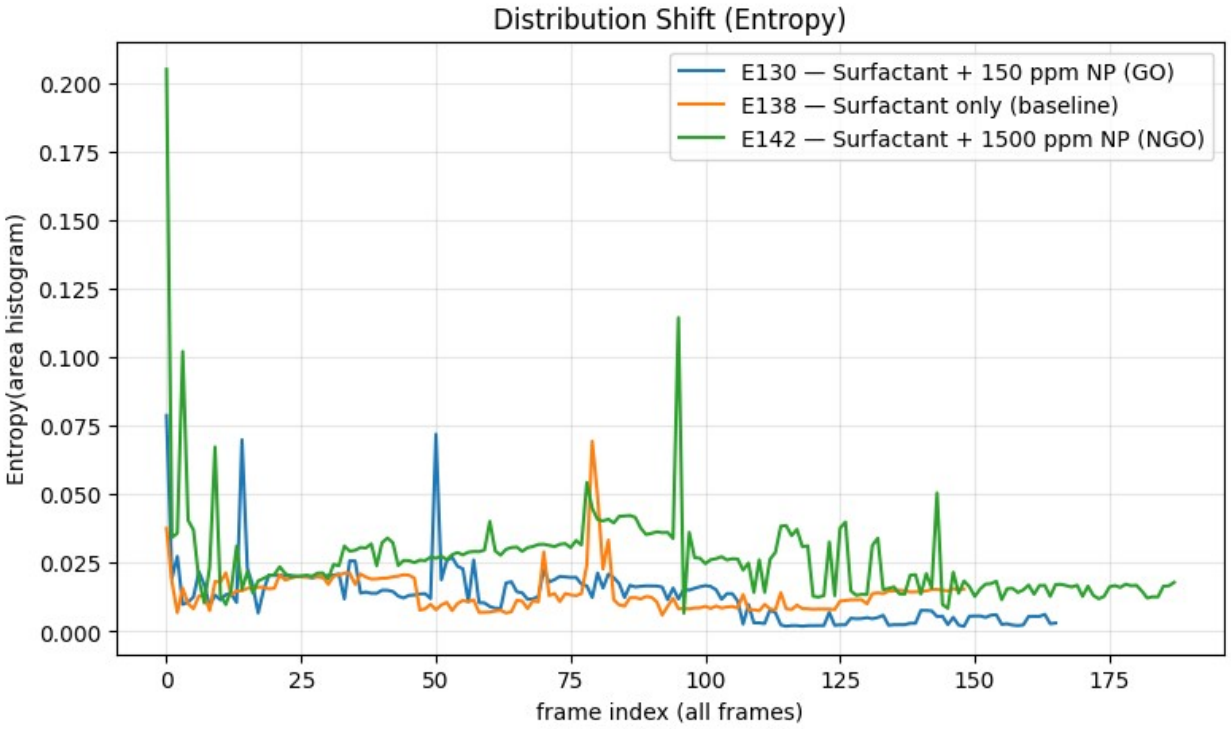
```
[\n          \"Experiment-142_t049.csv\",\n          \"Experiment-
142_t052.csv\",\n          \"Experiment-142_t050.csv\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"experiment\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          \"E142\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

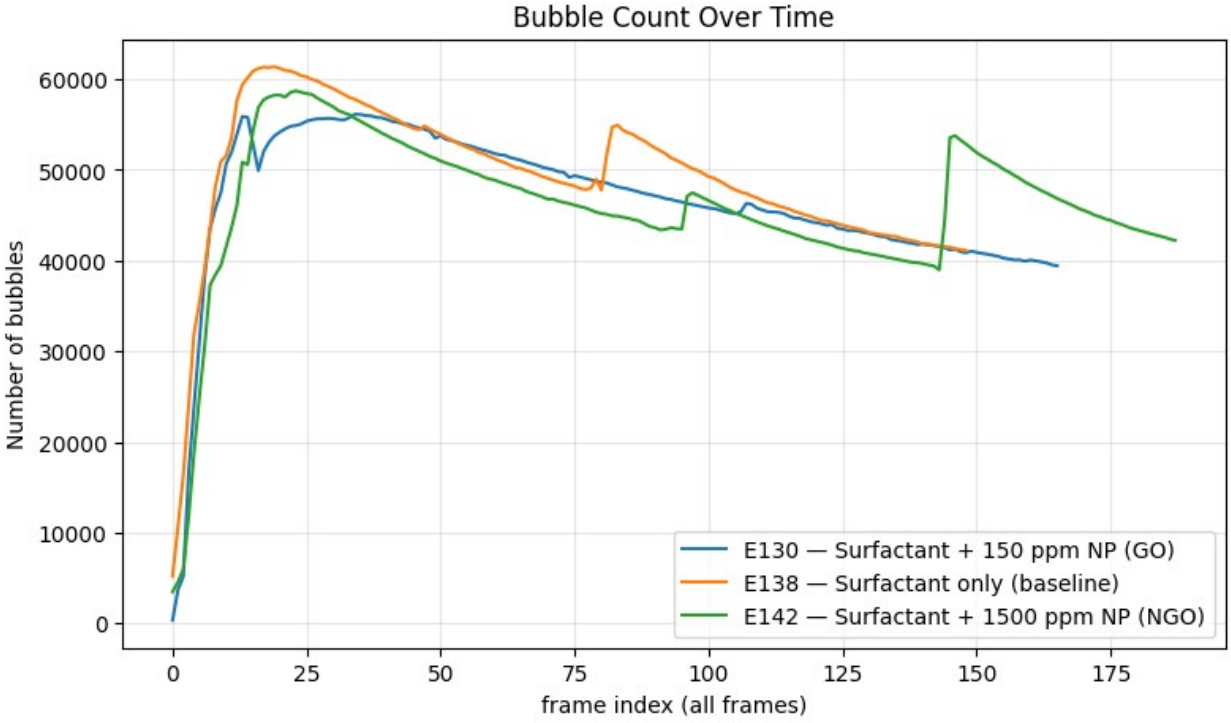

Temporal Evolution of Bubble Dynamics

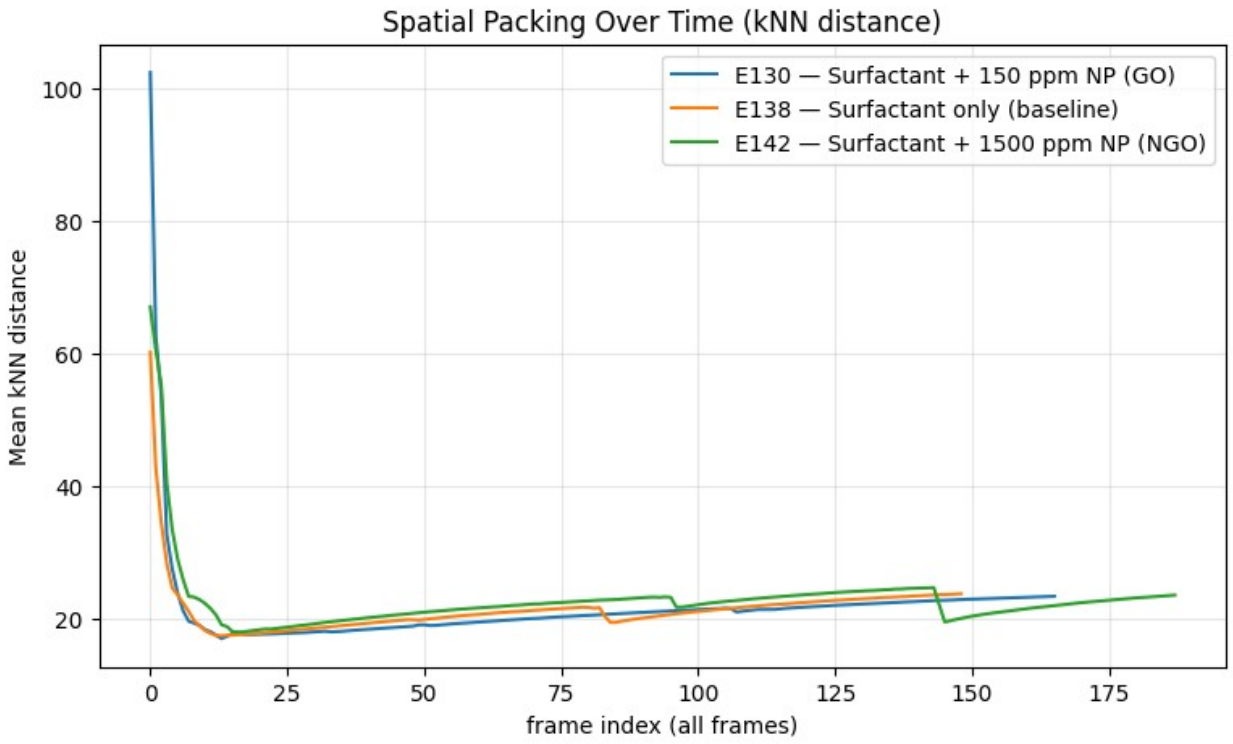
```
import matplotlib.pyplot as plt

def plot_timeseries(metric, title, ylabel):
    if metric not in next(iter(series.values())).columns:
        print(f"[SKIP] '{metric}' not found.")
        return

    plt.figure(figsize=(9,5))
    for exp_id, df in series.items():
        plt.plot(df["t_raw"], df[metric], label=f"{exp_id} –
{EXPERIMENTS[exp_id]['label']}")
    plt.title(title)
    plt.xlabel("frame index (all frames)")
    plt.ylabel(ylabel)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.show()

plot_timeseries("area_mean", "Bubble Size Over Time (Mean Area)",
"Mean bubble area")
plot_timeseries("area_std", "Bubble Size Spread Over Time (Std Area)",
"Std bubble area")
plot_timeseries("entropy_area", "Distribution Shift (Entropy)",
"Entropy(area histogram)")
plot_timeseries("n_bubbles_img", "Bubble Count Over Time", "Number of
bubbles")
plot_timeseries("knn_dist_mean", "Spatial Packing Over Time (kNN
distance)", "Mean kNN distance")
```


Bubble Size Over Time (Mean Area)
E130 — Surfactant + 150 ppm NP (GO)
E138 — Surfactant only (baseline)
E142 — Surfactant + 1500 ppm NP (NGO)
Mean bubble area
frame index (all frames)


Bubble Size Spread Over Time (Std Area)
E130 — Surfactant + 150 ppm NP (GO)
E138 — Surfactant only (baseline)
E142 — Surfactant + 1500 ppm NP (NGO)
Std bubble area
frame index (all frames)


Distribution Shift (Entropy)
E130 — Surfactant + 150 ppm NP (GO)
E138 — Surfactant only (baseline)
E142 — Surfactant + 1500 ppm NP (NGO)
Entropy(area histogram)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
25
50
75
100
125
150
175
frame index (all frames)


Bubble Count Over Time
Number of bubbles
60000
50000
40000
30000
20000
10000
0
E130 — Surfactant + 150 ppm NP (GO)
E138 — Surfactant only (baseline)
E142 — Surfactant + 1500 ppm NP (NGO)
0
25
50
75
100
125
150
175
frame index (all frames)



Quantification of Coarsening and Bubble Loss Rates

```
import numpy as np
import pandas as pd

def robust_slope(x, y):
    x = np.asarray(x, float)
    y = np.asarray(y, float)
    m = np.isfinite(x) & np.isfinite(y)
    if m.sum() < 10:
        return np.nan
    return float(np.polyfit(x[m], y[m], 1)[0])

def summarize_dynamics(df, early_frac=0.25):
    n = len(df)
    k = max(10, int(n * early_frac))

    x = df["t_raw"].values  # use full-frame index
    out = {"n_frames": n}

    out["coarsening_rate_full"]  = robust_slope(x,
df["area_mean"].values)
    out["coarsening_rate_early"] = robust_slope(x[:k],
df["area_mean"].values[:k])

    out["bubble_loss_rate_full"]  = robust_slope(x,
```

```
df["n_bubbles_img"].values)
    out["bubble_loss_rate_early"] = robust_slope(x[:k],
df["n_bubbles_img"].values[:k])

    out["entropy_rate_full"]  = robust_slope(x,
df["entropy_area"].values)
    out["entropy_rate_early"] = robust_slope(x[:k],
df["entropy_area"].values[:k])

    out["packing_rate_full"] = robust_slope(x,
df["knn_dist_mean"].values)

    return out

rows = []
for exp_id, df in series.items():
    s = summarize_dynamics(df)
    s["experiment"] = exp_id
    s["condition"] = EXPERIMENTS[exp_id]["label"]
    rows.append(s)

dyn_df = pd.DataFrame(rows)
dyn_df
```


```
{"summary":"{\n  \"name\": \"dyn_df\",\n  \"rows\": 3,\n  \"fields\":
[\n    {\n      \"column\": \"n_frames\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 19,\n        \"min\": 149,\n
\"max\": 188,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          166,\n          149,\n          188\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"coarsening_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.22245210403069113,\n        \"min\": -0.40606137808829196,\n
\"max\": -0.005571567688209078,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.03800482374754419,\n          -
0.005571567688209078,\n          -0.40606137808829196\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"coarsening_rate_early\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.40271430731052654,\n        \"min\": -10.169654295291023,\n
\"max\": -9.419099715148084,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -9.419099715148084,\n          -
9.541321069490644,\n          -10.169654295291023\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bubble_loss_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
25.252599444567025,\n        \"min\": -58.661763105387,\n
\"max\": -10.357936598236867,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -47.28234352762152,\n          -
58.661763105387,\n          -10.357936598236867\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bubble_loss_rate_early\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
120.15939824790726,\n        \"min\": 784.9229879740982,\n
\"max\": 1023.9774774774778,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          925.7705574912897,\n
1023.9774774774778,\n          784.9229879740982\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"entropy_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4.760108771797674e-05,\n        \"min\": -0.00013274520692098572,\n
\"max\": -4.266150071146646e-05,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.00013274520692098572,\n          -
4.266150071146646e-05,\n          -0.00011437328187145929\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"entropy_rate_early\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.00039529398850614164,\n        \"min\": -0.0006322645832535415,\n
\"max\": 0.00015620206066202665,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.000187908655401097,\n
0.00015620206066202665,\n          -0.0006322645832535415\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"packing_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.01035720890982952,\n        \"min\": -0.009691537737112758,\n
\"max\": 0.010243977704132734,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.004596843454867653,\n
0.010243977704132734,\n          -0.009691537737112758\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"experiment\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"E130\",\n          \"E138\",\n          \"E142\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"condition\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Surfactant + 150 ppm NP (GO)\",\n          \"Surfactant only
(baseline)\",\n          \"Surfactant + 1500 ppm NP (NGO)\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"dyn_df"}
```


Image-Derived Foam Stability Proxy

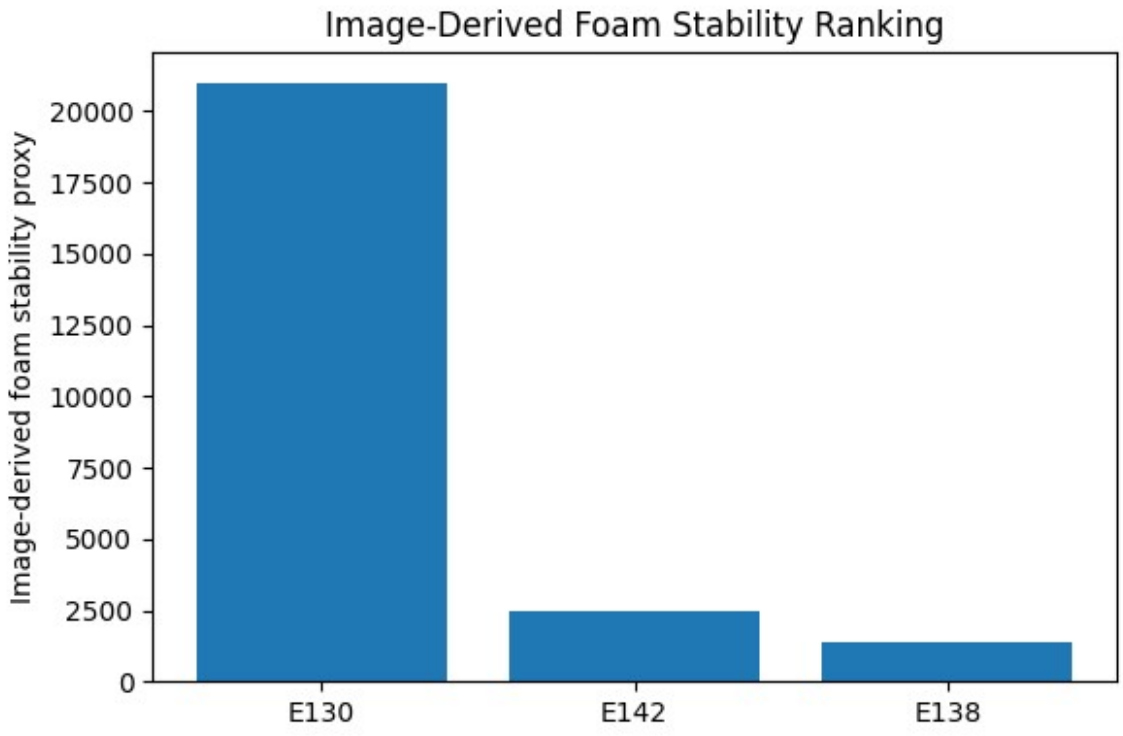
```
def stability_proxy(df, col="n_bubbles_img"):
    y = df[col].values.astype(float)
    if len(y) < 10 or not np.isfinite(y[0]) or y[0] <= 0:
```

```
        return np.nan

    y_norm = y / y[0]
    x = df["t_raw"].values.astype(float)

    # Area under normalized curve vs time
    return float(np.trapezoid(y_norm, x))

hl_df = pd.DataFrame([
    {
        "experiment": exp_id,
        "condition": EXPERIMENTS[exp_id]["label"],
        "foam_stability_proxy": stability_proxy(df),
    }
    for exp_id, df in series.items()
])

hl_df
```


```
{"summary":"{\n  \"name\": \"hl_df\",\n  \"rows\": 3,\n  \"fields\":
[\n    {\n      \"column\": \"experiment\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"E130\",\n          \"E138\",\n
\"E142\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"condition\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Surfactant + 150 ppm NP (GO)\",\n          \"Surfactant
only (baseline)\",\n          \"Surfactant + 1500 ppm NP (NGO)\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"foam_stability_proxy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11011.377971819351,\n        \"min\": 1406.1984081319524,\n
\"max\": 20991.948369565216,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          20991.948369565216,\n
1406.1984081319524,\n          2478.4006309148263\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"hl_df"}
```


```
import matplotlib.pyplot as plt

# Your computed stability proxy table
# hl_df must contain: experiment, condition, foam_stability_proxy

df = hl_df.sort_values("foam_stability_proxy", ascending=False)

plt.figure(figsize=(6,4))
plt.bar(
    df["experiment"],
    df["foam_stability_proxy"]
```

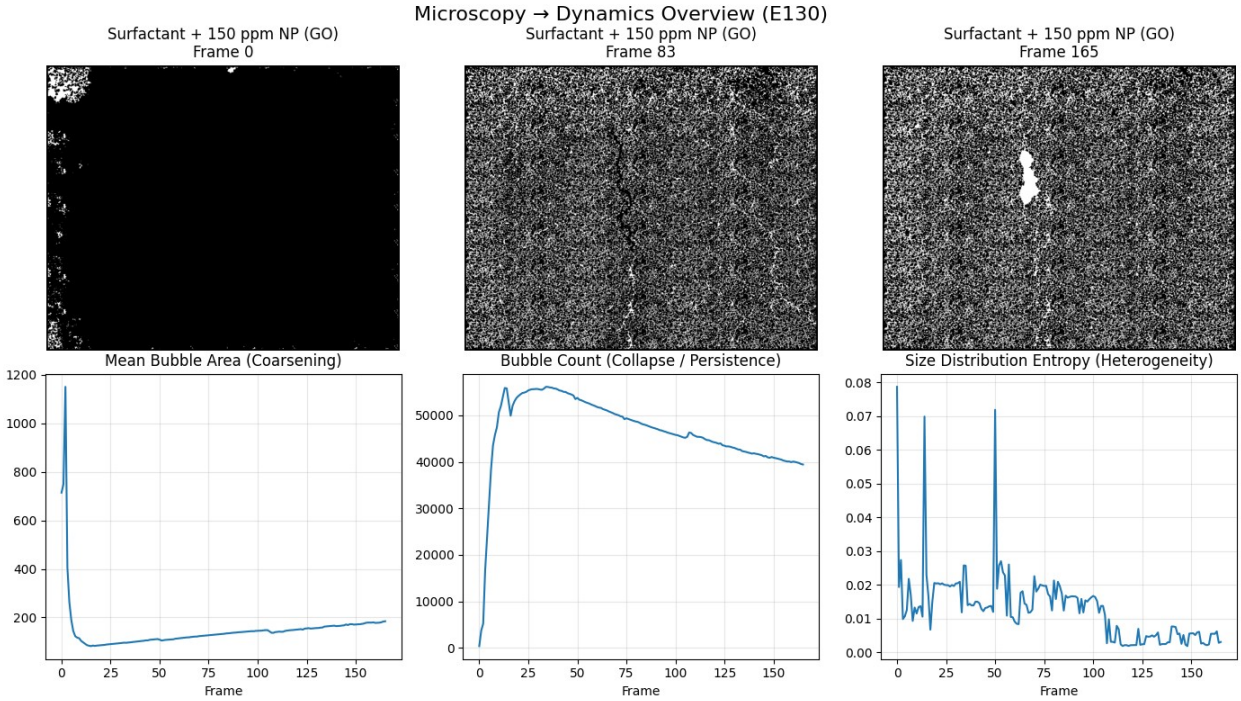
```
)

plt.ylabel("Image-derived foam stability proxy")
plt.title("Image-Derived Foam Stability Ranking")

plt.tight_layout()
plt.savefig("foam_stability_proxy.png", dpi=200)
plt.show()
```

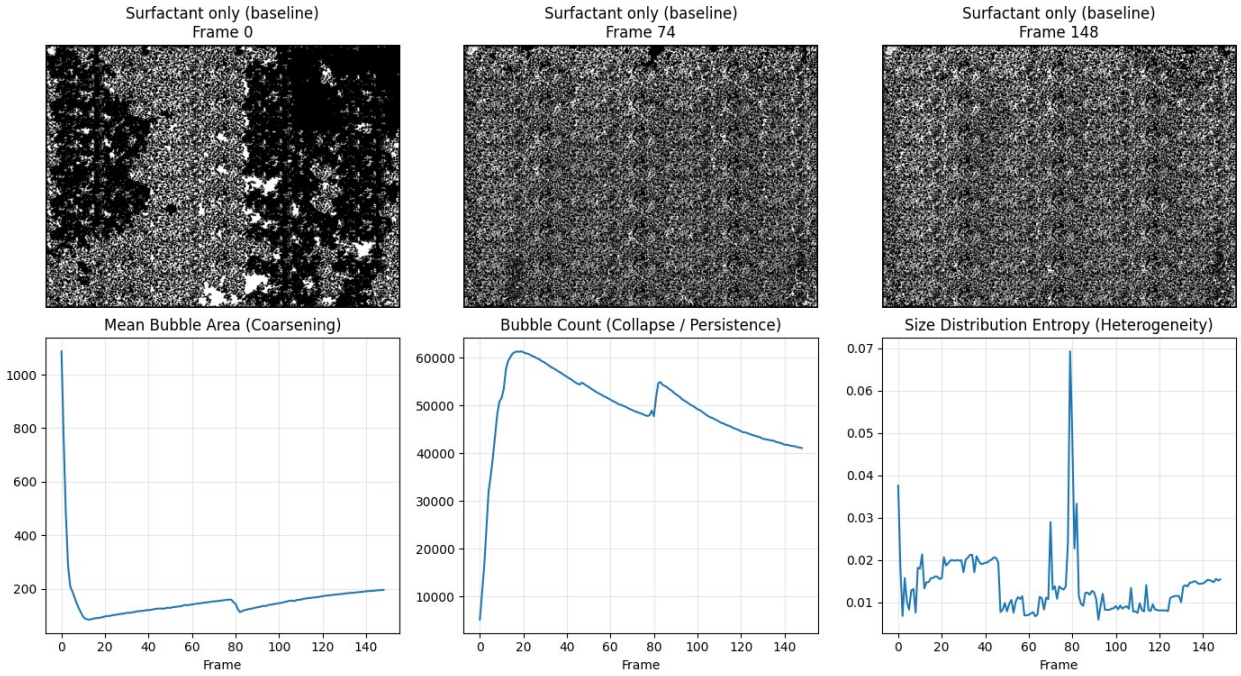


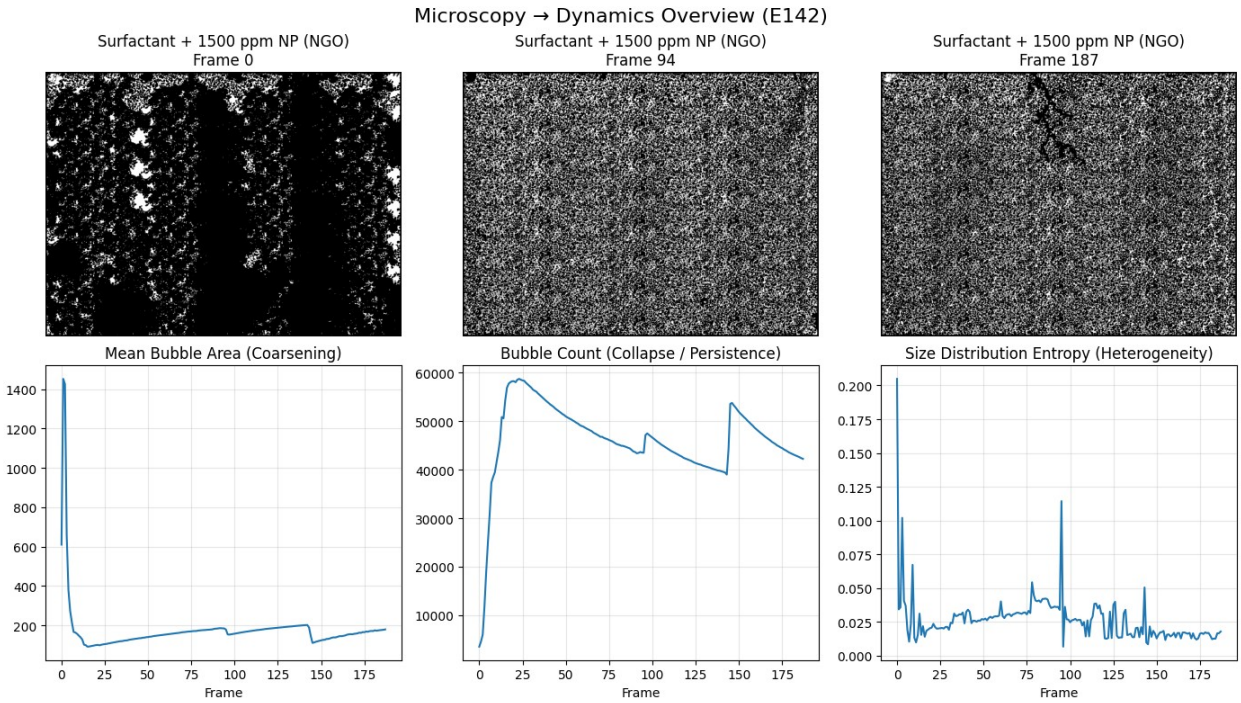
Visual Dashboard: Images + Dynamics (All Experiments)

```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

def show_dashboard(exp_id):
    df = series[exp_id]
    label_txt = EXPERIMENTS[exp_id]["label"]

    seg_dir = find_nested_dir(exp_path_work(exp_id),
"Segmented_pictures")

    # choose representative frames: start / mid / end
    idxs = [0, len(df)//2, len(df)-1]
    imgs = [Image.open(seg_dir / df.iloc[i]["img_name"]) for i in
idxs]
```

```
    fig = plt.figure(figsize=(14, 8))
    gs = fig.add_gridspec(2, 3, height_ratios=[1, 1])

    # --- row 1: images ---
    for j, (i, im) in enumerate(zip(idxs, imgs)):
        ax = fig.add_subplot(gs[0, j])
        ax.imshow(im, cmap="gray")
        ax.set_title(f"{label_txt}\nFrame {int(df.iloc[i]['t_raw'])}")
        ax.axis("off")

    # --- row 2: curves ---
    ax1 = fig.add_subplot(gs[1, 0])
    ax1.plot(df["t_raw"], df["area_mean"])
    ax1.set_title("Mean Bubble Area (Coarsening)")
    ax1.set_xlabel("Frame"); ax1.grid(alpha=0.3)

    ax2 = fig.add_subplot(gs[1, 1])
    ax2.plot(df["t_raw"], df["n_bubbles_img"])
    ax2.set_title("Bubble Count (Collapse / Persistence)")
    ax2.set_xlabel("Frame"); ax2.grid(alpha=0.3)

    ax3 = fig.add_subplot(gs[1, 2])
    ax3.plot(df["t_raw"], df["entropy_area"])
    ax3.set_title("Size Distribution Entropy (Heterogeneity)")
    ax3.set_xlabel("Frame"); ax3.grid(alpha=0.3)

    fig.suptitle(f"Microscopy → Dynamics Overview ({exp_id})",
fontsize=16)
    plt.tight_layout()
    plt.show()

for exp_id in EXPERIMENTS:
    show_dashboard(exp_id)
```


Microscopy → Dynamics Overview (E130)
Surfactant + 150 ppm NP (GO)
Frame 0
Surfactant + 150 ppm NP (GO)
Frame 83
Surfactant + 150 ppm NP (GO)
Frame 165
Mean Bubble Area (Coarsening)
Bubble Count (Collapse / Persistence)
Size Distribution Entropy (Heterogeneity)
Frame
Frame
Frame

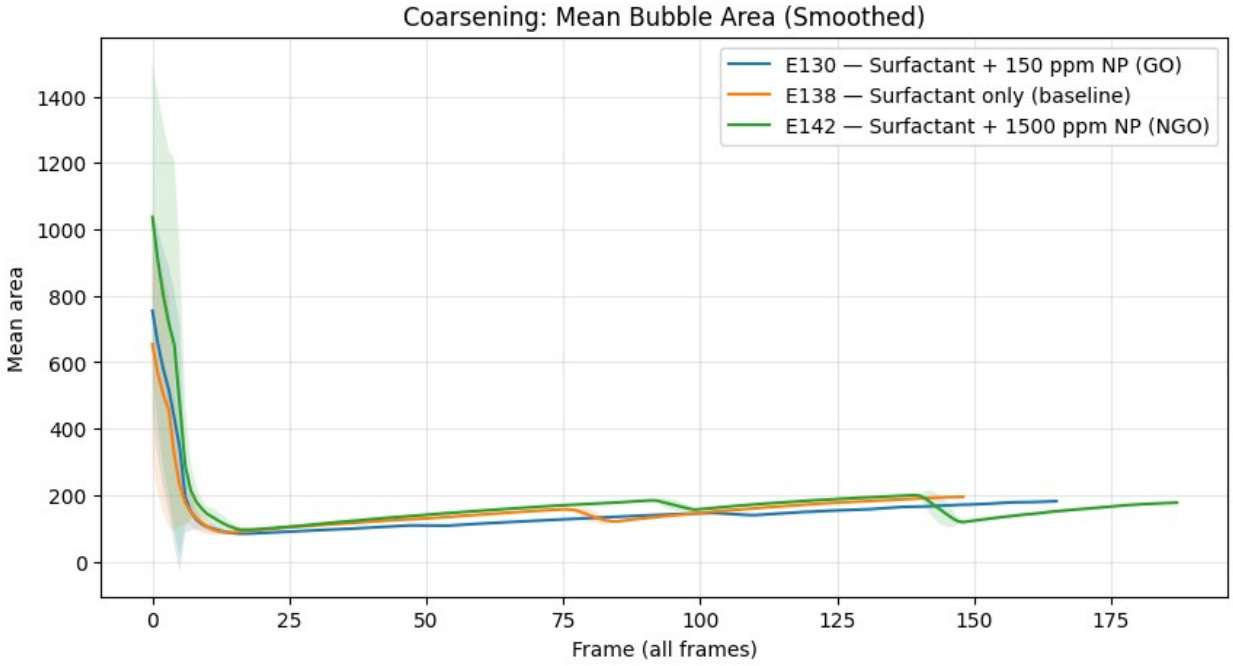

Microscopy → Dynamics Overview (E138)
Surfactant only (baseline)
Frame 0
Surfactant only (baseline)
Frame 74
Surfactant only (baseline)
Frame 148
Mean Bubble Area (Coarsening)
Bubble Count (Collapse / Persistence)
Size Distribution Entropy (Heterogeneity)
Frame
Frame
Frame

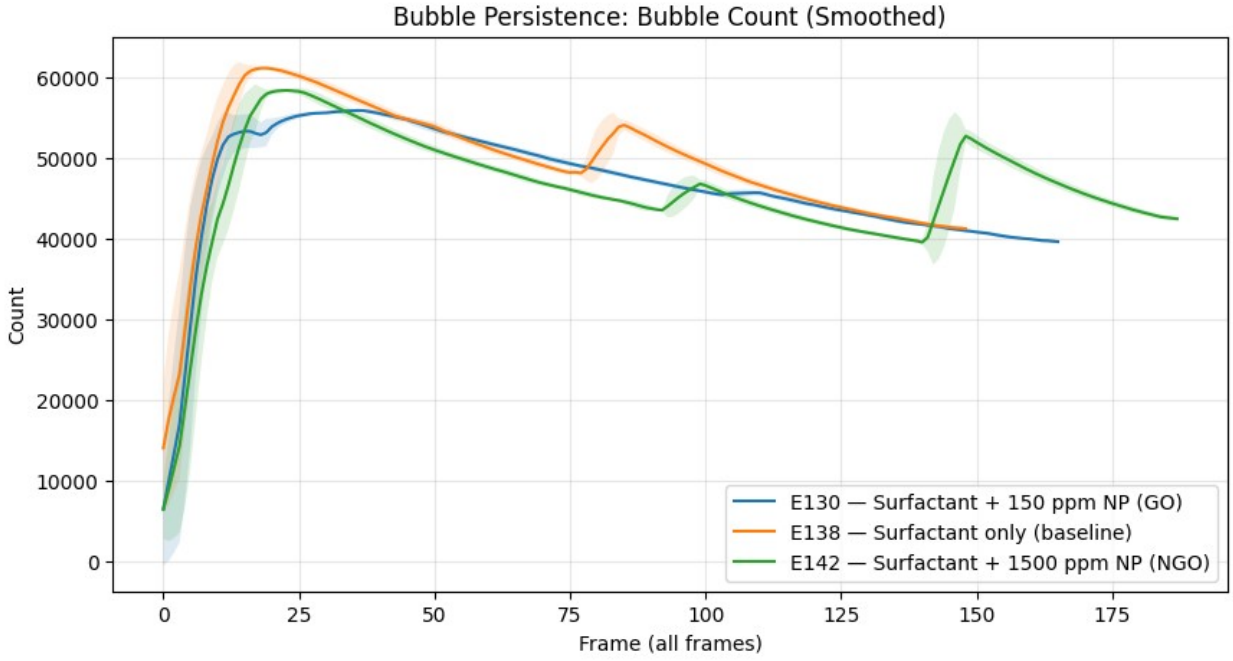


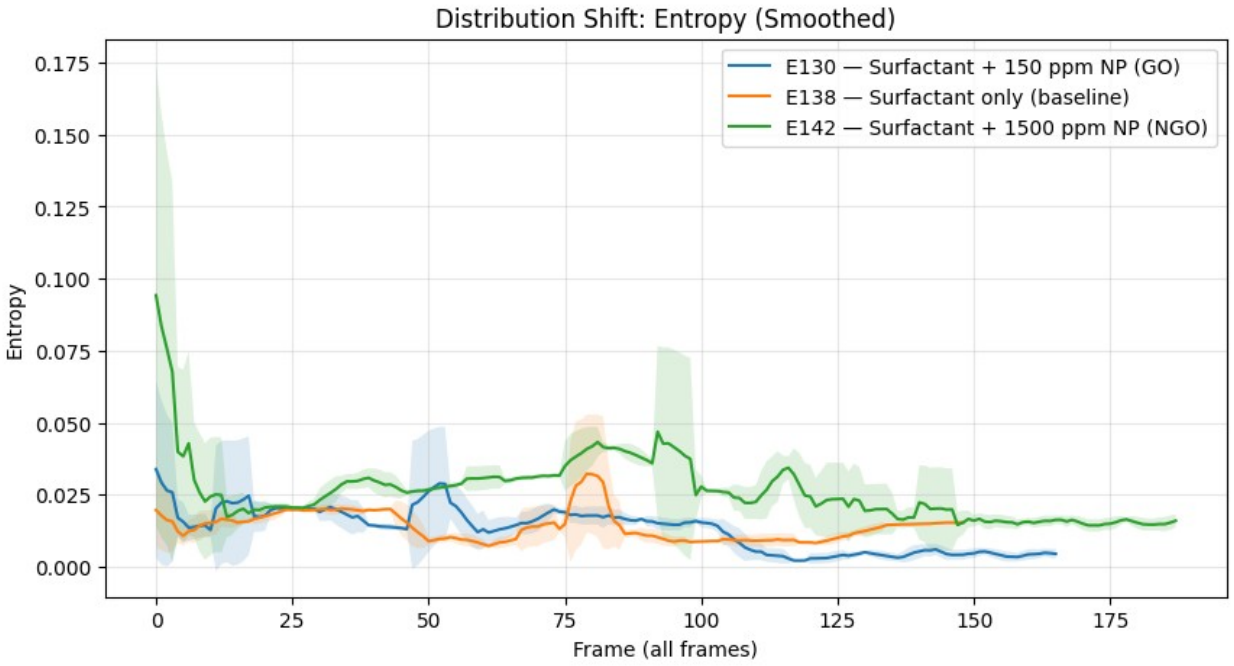
“All experiments” overlay with uncertainty bands

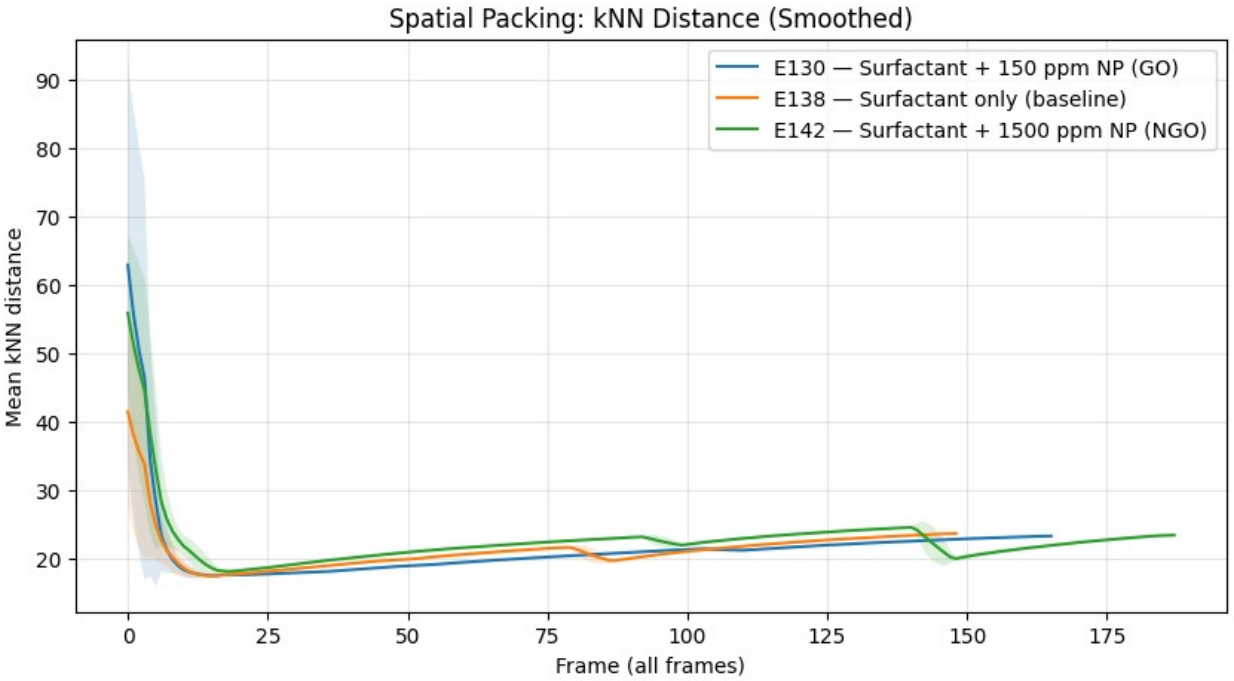
```
import pandas as pd
import matplotlib.pyplot as plt

def plot_with_band(metric, title, ylabel, window=7):
    plt.figure(figsize=(10,5))

    for exp_id, df in series.items():
        s = df[metric].astype(float)
        mu = s.rolling(window, center=True, min_periods=1).mean()
        sd = s.rolling(window, center=True, min_periods=1).std()

        plt.plot(df["t_raw"], mu, label=f"{exp_id} –
{EXPERIMENTS[exp_id]['label']}")
        plt.fill_between(df["t_raw"], mu - sd, mu + sd, alpha=0.15)

    plt.title(title)
    plt.xlabel("Frame (all frames)")
    plt.ylabel(ylabel)
    plt.grid(alpha=0.3)
    plt.legend()
    plt.show()

plot_with_band("area_mean", "Coarsening: Mean Bubble Area (Smoothed)",
"Mean area")
plot_with_band("n_bubbles_img", "Bubble Persistence: Bubble Count
(Smoothed)", "Count")
```

```
plot_with_band("entropy_area", "Distribution Shift: Entropy
(Smoothed)", "Entropy")
plot_with_band("knn_dist_mean", "Spatial Packing: kNN Distance
(Smoothed)", "Mean kNN distance")
```







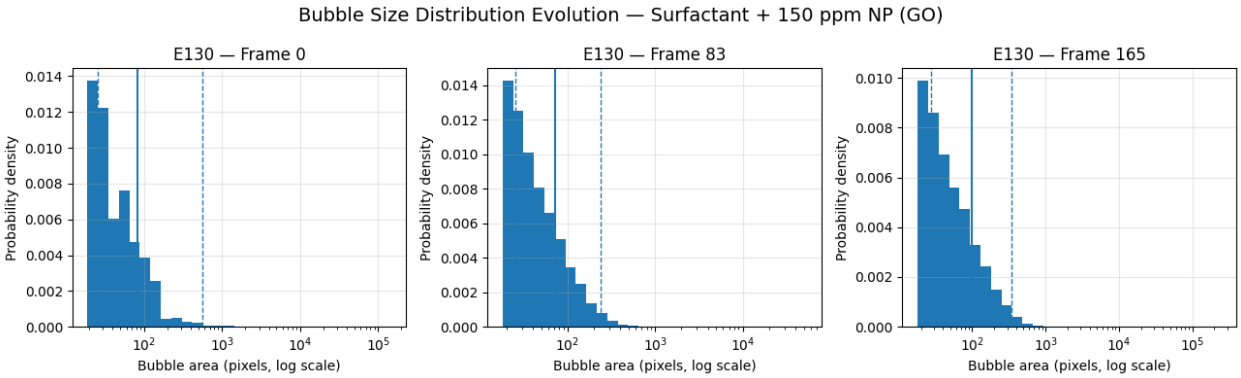


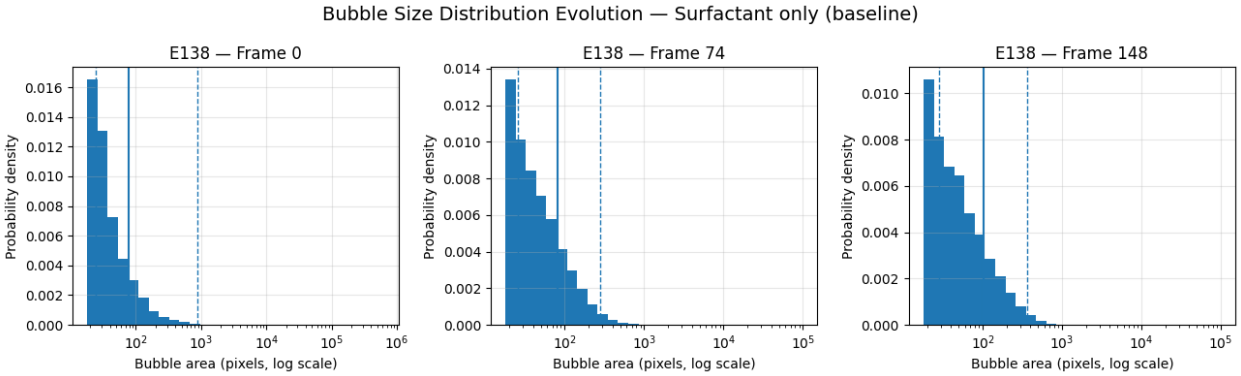
Real-image distribution evolution (start/mid/end histograms)

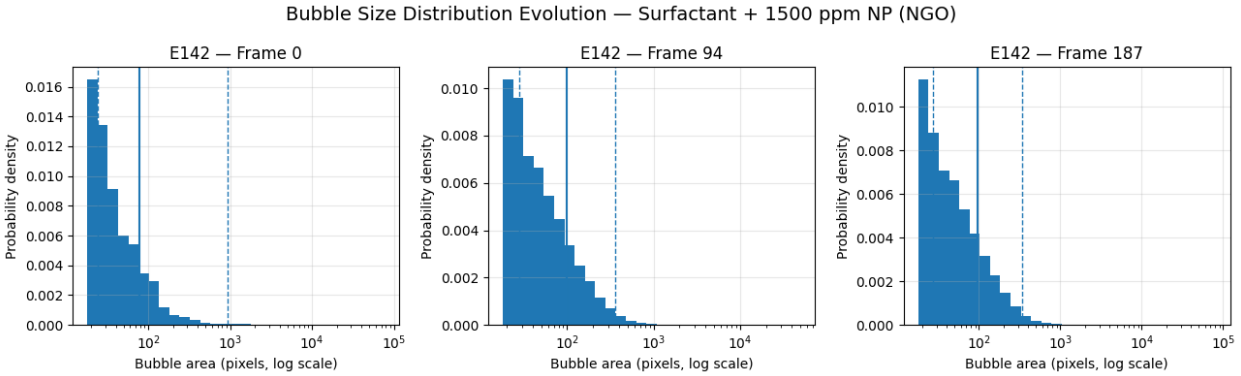
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.measure import label, regionprops
from PIL import Image
```

```
def get_areas_from_segmented_image(img_path):
    """
    Compute bubble areas directly from a segmented (binary) image.
    """
    img = Image.open(img_path).convert("L")
    arr = np.array(img)

    # assume bubbles are white on black
    mask = arr > 0

    lbl = label(mask)
    props = regionprops(lbl)

    areas = np.array([p.area for p in props], dtype=float)
    return areas

def plot_distribution_evolution(exp_id):
    df = series[exp_id]
    seg_dir = find_nested_dir(exp_path_work(exp_id),
"Segmented_pictures")

    idxs = [0, len(df)//2, len(df)-1]
    plt.figure(figsize=(13,4))

    for j, i in enumerate(idxs, 1):
        img_path = seg_dir / df.iloc[i]["img_name"]
        areas = get_areas_from_segmented_image(img_path)

        plt.subplot(1,3,j)

        if len(areas) == 0:
            plt.title("No bubbles detected")
            continue

        bins = np.logspace(np.log10(areas.min()+1),
                           np.log10(areas.max()+1),
                           30)

        plt.hist(areas, bins=bins, density=True)
        plt.xscale("log")

        p10, p50, p90 = np.percentile(areas, [10, 50, 90])
        plt.axvline(p10, linestyle="--", linewidth=1)
        plt.axvline(p50, linestyle="-",  linewidth=1.5)
        plt.axvline(p90, linestyle="--", linewidth=1)

        plt.title(f"{exp_id} – Frame {int(df.iloc[i]['t_raw'])}")
        plt.xlabel("Bubble area (pixels, log scale)")
        plt.ylabel("Probability density")
        plt.grid(alpha=0.3)
```

```
    plt.suptitle(
        f"Bubble Size Distribution Evolution — {EXPERIMENTS[exp_id]
['label']}",
        fontsize=14
    )
    plt.tight_layout()
    plt.show()

for exp_id in EXPERIMENTS:
    plot_distribution_evolution(exp_id)
```







Explainable Machine Learning for Foam Stability Prediction

Predicting Foam Stability from Images

```
ml_df = dyn_df.merge(
    hl_df[["experiment", "foam_stability_proxy"]],
    on="experiment",
    how="left"
)

ml_df = ml_df.sort_values("foam_stability_proxy", ascending=False)
ml_df
```


```
{"summary":"{\n  \"name\": \"ml_df\",\n  \"rows\": 3,\n  \"fields\":
[\n    {\n      \"column\": \"n_frames\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 19,\n        \"min\": 149,\n
\"max\": 188,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          166,\n          188,\n          149\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"coarsening_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.22245210403069113,\n        \"min\": -0.40606137808829196,\n
\"max\": -0.005571567688209078,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.03800482374754419,\n          -
0.40606137808829196,\n          -0.005571567688209078\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"coarsening_rate_early\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.40271430731052654,\n        \"min\": -10.169654295291023,\n
\"max\": -9.419099715148084,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -9.419099715148084,\n          -
10.169654295291023,\n          -9.541321069490644\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bubble_loss_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
25.252599444567025,\n        \"min\": -58.661763105387,\n
\"max\": -10.357936598236867,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -47.28234352762152,\n          -
10.357936598236867,\n          -58.661763105387\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bubble_loss_rate_early\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
120.15939824790726,\n        \"min\": 784.9229879740982,\n
\"max\": 1023.9774774774778,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          925.7705574912897,\n
784.9229879740982,\n          1023.9774774774778\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"entropy_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
4.760108771797674e-05,\n        \"min\": -0.00013274520692098572,\n
\"max\": -4.266150071146646e-05,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.00013274520692098572,\n          -
0.00011437328187145929,\n          -4.266150071146646e-05\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"entropy_rate_early\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.00039529398850614164,\n        \"min\": -0.0006322645832535415,\n
\"max\": 0.00015620206066202665,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.000187908655401097,\n          -
0.0006322645832535415,\n          0.00015620206066202665\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"packing_rate_full\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.01035720890982952,\n        \"min\": -0.009691537737112758,\n
\"max\": 0.010243977704132734,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.004596843454867653,\n          -
0.009691537737112758,\n          0.010243977704132734\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"experiment\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"E130\",\n          \"E142\",\n          \"E138\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"condition\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Surfactant + 150 ppm NP (GO)\",\n          \"Surfactant + 1500 ppm
NP (NGO)\",\n          \"Surfactant only (baseline)\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"foam_stability_proxy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11011.377971819351,\n        \"min\": 1406.1984081319524,\n
\"max\": 20991.948369565216,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          20991.948369565216,\n
2478.4006309148263,\n          1406.1984081319524\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"ml_df"}
```


Normalize features (important for ML & fairness)

```
from sklearn.preprocessing import StandardScaler

FEATURES = [
    "coarsening_rate_early",
    "bubble_loss_rate_early",
    "entropy_rate_early",
    "packing_rate_full",
```

```
]

X = ml_df[FEATURES].values
y = ml_df["foam_stability_proxy"].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_scaled
```

```
array([[ 0.88476937,  0.144874  ,  0.10353038, -0.38416138],
       [-1.39783608, -1.29073854, -1.27322378, -0.98661139],
       [ 0.51306671,  1.14586454,  1.1696934 ,  1.37077277]])
```

Regression: predict foam stability from images

```
from sklearn.linear_model import Ridge

model = Ridge(alpha=1.0)
model.fit(X_scaled, y)

ml_df["predicted_stability"] = model.predict(X_scaled)

ml_df[[
    "experiment",
    "condition",
    "foam_stability_proxy",
    "predicted_stability"
]]
```


```
{"summary":"{\n  \"name\": \"]]\",\n  \"rows\": 3,\n  \"fields\": [\n
{\n      \"column\": \"experiment\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"E130\",\n          \"E142\",\n
\"E138\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"condition\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Surfactant + 150 ppm NP (GO)\",\n          \"Surfactant
+ 1500 ppm NP (NGO)\",\n          \"Surfactant only (baseline)\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"foam_stability_proxy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11011.377971819351,\n        \"min\": 1406.1984081319524,\n
\"max\": 20991.948369565216,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          20991.948369565216,\n
2478.4006309148263,\n          1406.1984081319524\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"predicted_stability\",\n
```

\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6228.621284309772,\n   \"min\": 4688.622979232399,\n
\"max\": 15484.369653207934,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   15484.369653207934,\n
4688.622979232399,\n   4703.554776171661\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

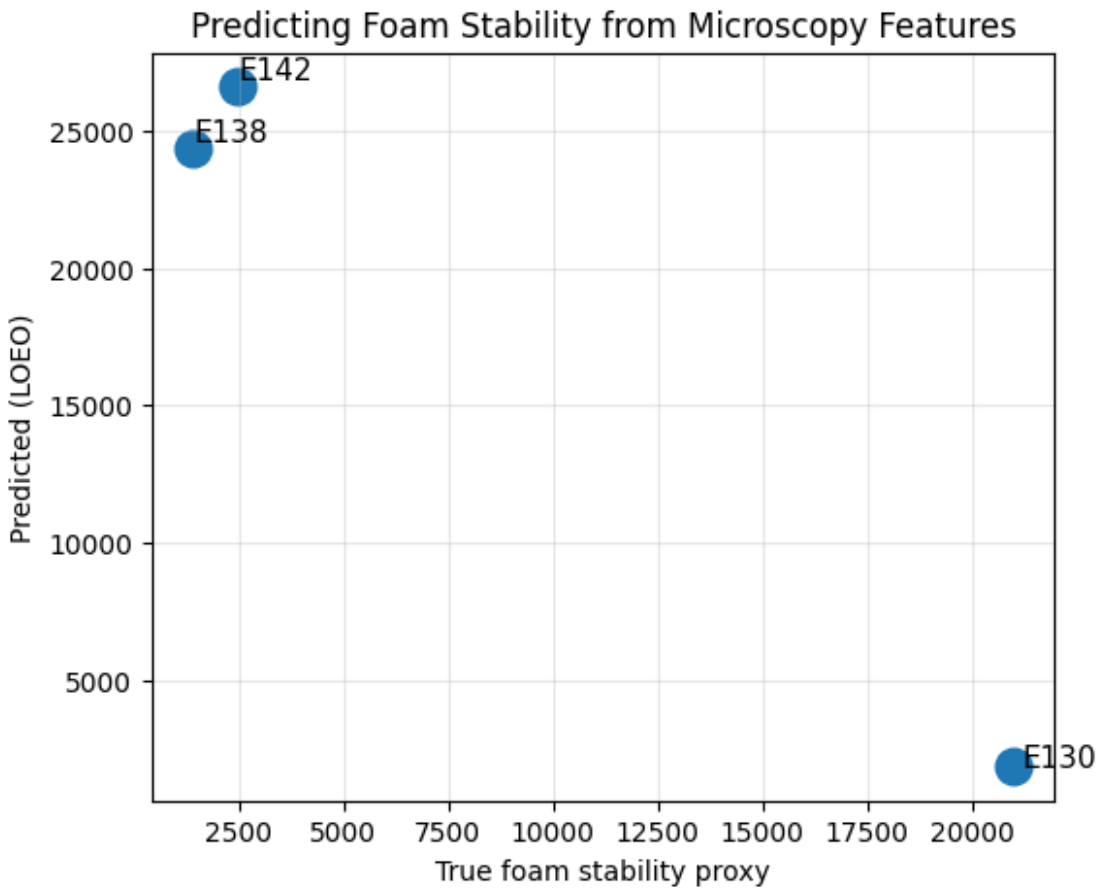
Leave-One-Experiment-Out validation (honest ML)

```
import numpy as np

pred = np.zeros_like(y)

for i in range(len(y)):
    train_idx = np.array([j for j in range(len(y)) if j != i])

    model_i = Ridge(alpha=1.0)
    model_i.fit(X_scaled[train_idx], y[train_idx])
    pred[i] = model_i.predict(X_scaled[[i]])[0]

ml_df["predicted_LOEO"] = pred

ml_df[[
    "experiment",
    "foam_stability_proxy",
    "predicted_LOEO"
]]
```

{"summary":"{\n \"name\": \"]]\",\n \"rows\": 3,\n \"fields\": [\n
{\n   \"column\": \"experiment\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   \"E130\",\n   \"E142\",\n
\"E138\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"foam_stability_proxy\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 11011.377971819351,\n   \"min\":
1406.1984081319524,\n   \"max\": 20991.948369565216,\n
\"num_unique_values\": 3,\n   \"samples\": [\n
20991.948369565216,\n   2478.4006309148263,\n
1406.1984081319524\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"predicted_LOEO\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 13704.006568291992,\n   \"min\":
1845.5099390038367,\n   \"max\": 26619.26765964892,\n
\"num_unique_values\": 3,\n   \"samples\": [\n
1845.5099390038367,\n   26619.26765964892,\n
24385.96375450615\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

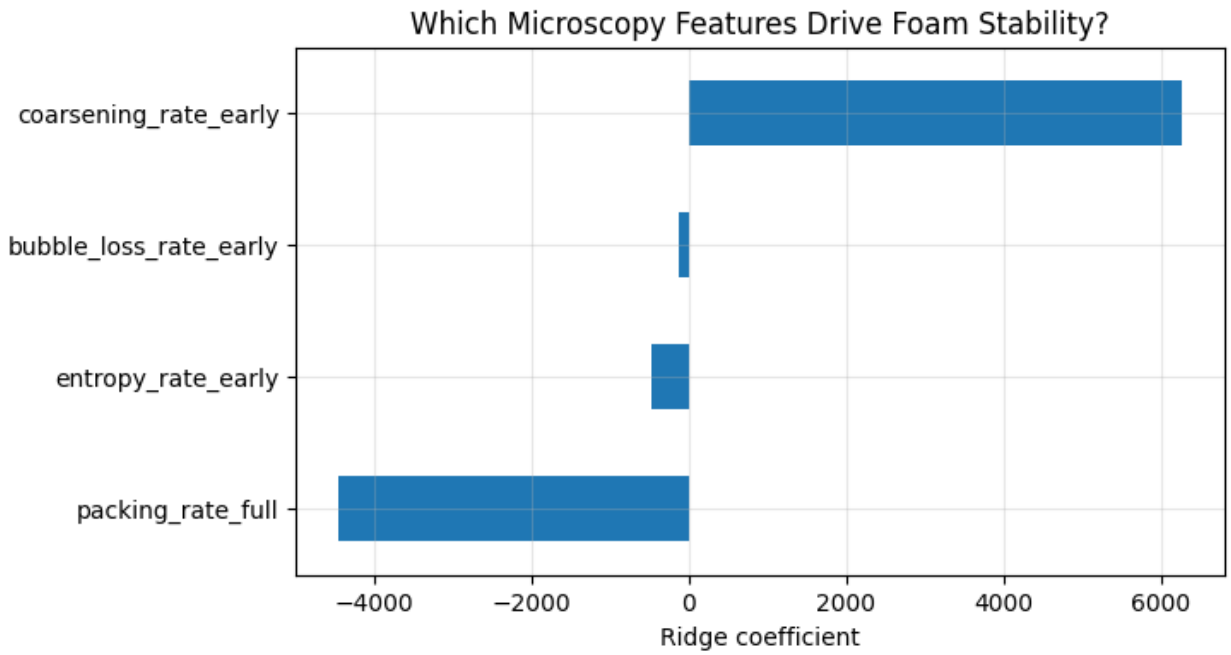
Prediction vs ground truth (visual proof)

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6,5))
plt.scatter(
    ml_df["foam_stability_proxy"],
    ml_df["predicted_LOEO"],
    s=180
)

for i, exp in enumerate(ml_df["experiment"]):
    plt.text(
        ml_df["foam_stability_proxy"].iloc[i] * 1.01,
        ml_df["predicted_LOEO"].iloc[i] * 1.01,
        exp,
        fontsize=11
    )

plt.xlabel("True foam stability proxy")
plt.ylabel("Predicted (LOEO)")
plt.title("Predicting Foam Stability from Microscopy Features")
plt.grid(alpha=0.3)
plt.show()
```



Model Interpretability: Feature Importance Analysis

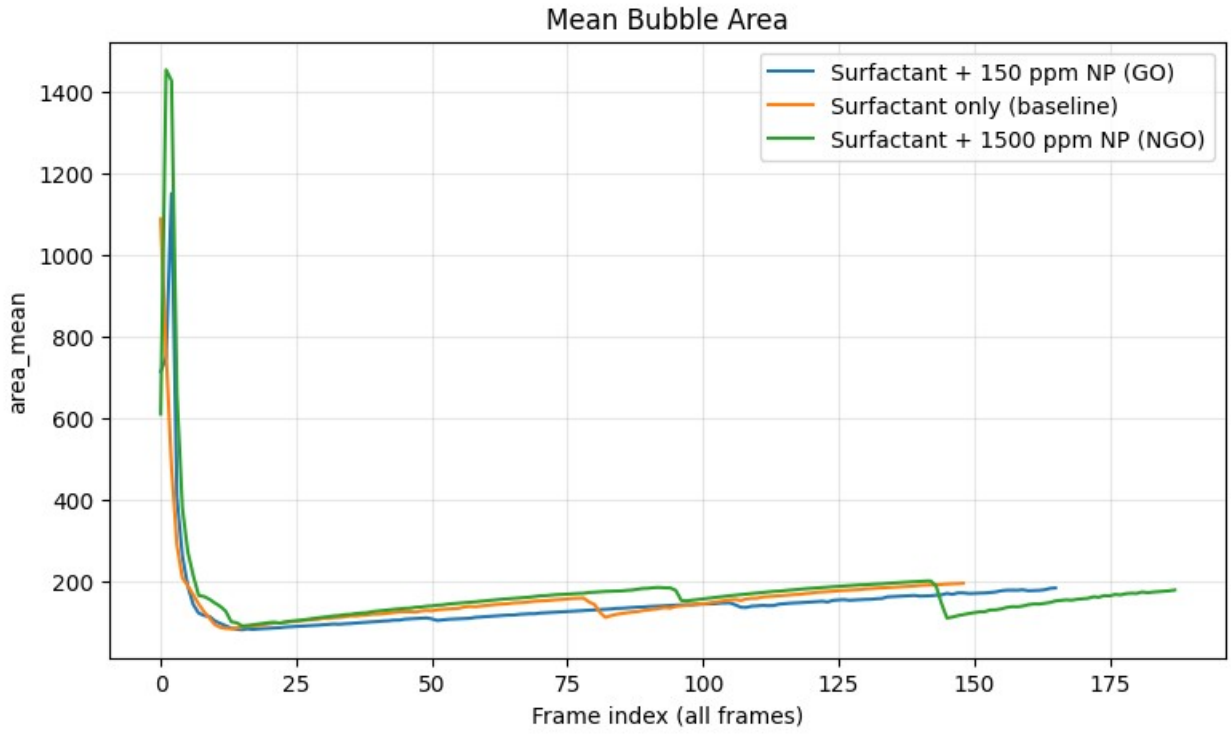
```
coef = (
    pd.Series(model.coef_, index=FEATURES)
    .sort_values()
)

plt.figure(figsize=(7,4))
coef.plot(kind="barh")
plt.title("Which Microscopy Features Drive Foam Stability?")
plt.xlabel("Ridge coefficient")
plt.grid(alpha=0.3)
plt.show()

coef
```

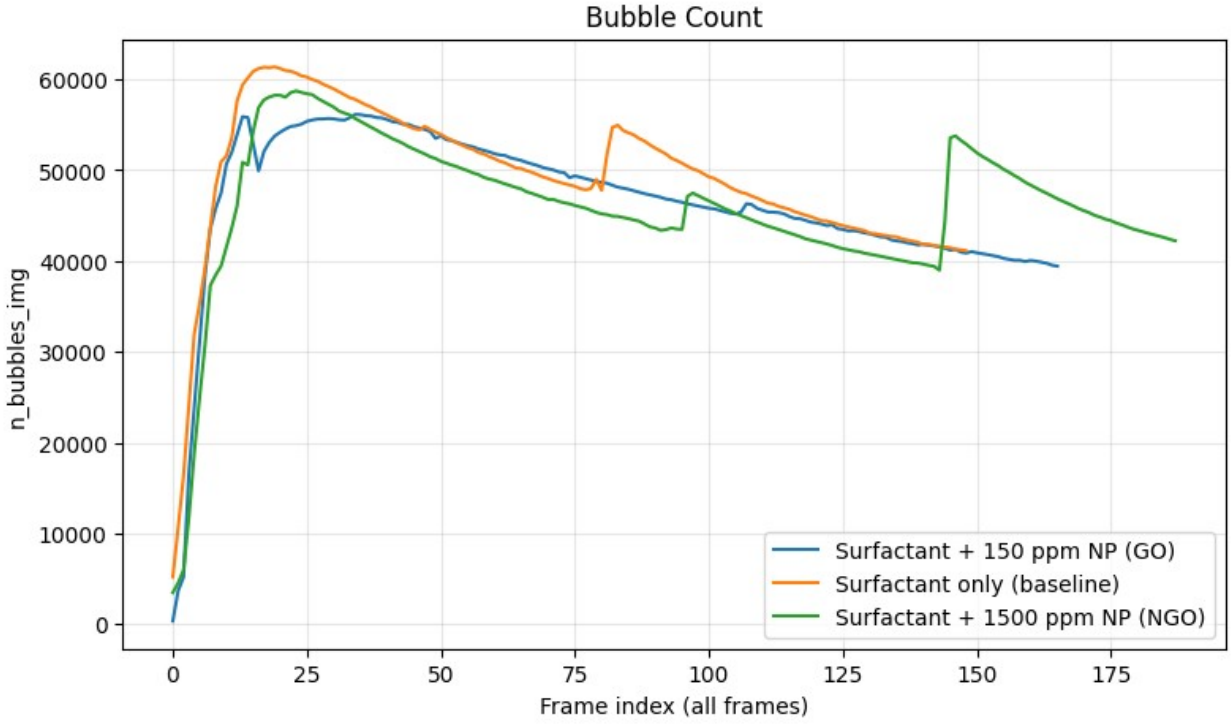


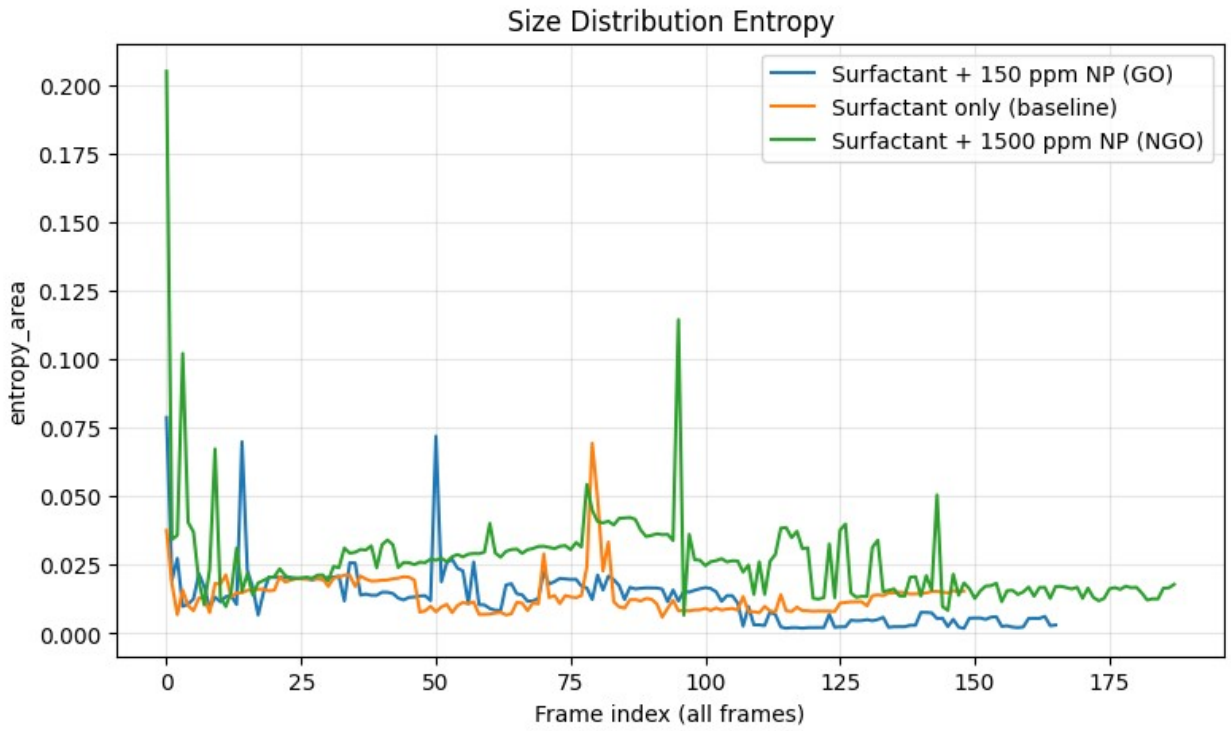

```
packing_rate_full         -4455.094798
entropy_rate_early         -472.586625
bubble_loss_rate_early     -127.599639
coarsening_rate_early      6270.701692
dtype: float64
```

Time-resolved comparison (GO vs NGO vs baseline)

```
metrics = [
    ("area_mean", "Mean Bubble Area"),
    ("n_bubbles_img", "Bubble Count"),
    ("entropy_area", "Size Distribution Entropy"),
]

for metric, title in metrics:
    plt.figure(figsize=(9,5))
    for exp_id, df in series.items():
        plt.plot(
            df["t_raw"],
            df[metric],
            label=EXPERIMENTS[exp_id]["label"]
        )
    plt.title(title)
    plt.xlabel("Frame index (all frames)")
    plt.ylabel(metric)
    plt.legend()
    plt.grid(alpha=0.3)
    plt.show()
```


Mean Bubble Area
Surfactant + 150 ppm NP (GO)
Surfactant only (baseline)
Surfactant + 1500 ppm NP (NGO)
area_mean
1400
1200
1000
800
600
400
200
0
25
50
75
100
125
150
175
Frame index (all frames)

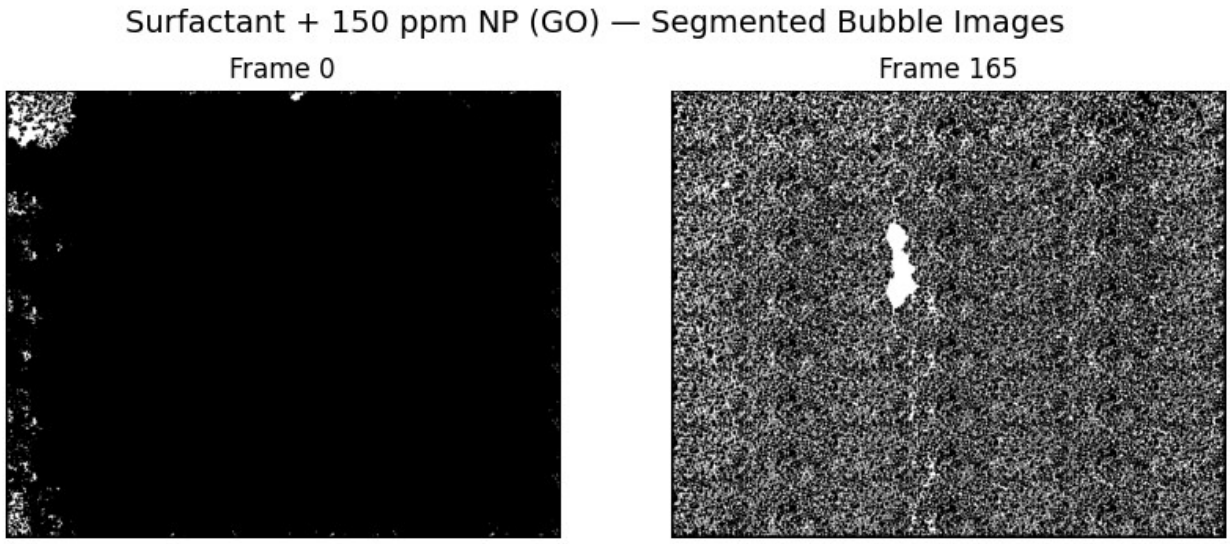

Bubble Count
n_bubbles_img
60000
50000
40000
30000
20000
10000
0
Surfactant + 150 ppm NP (GO)
Surfactant only (baseline)
Surfactant + 1500 ppm NP (NGO)
0
25
50
75
100
125
150
175
Frame index (all frames)

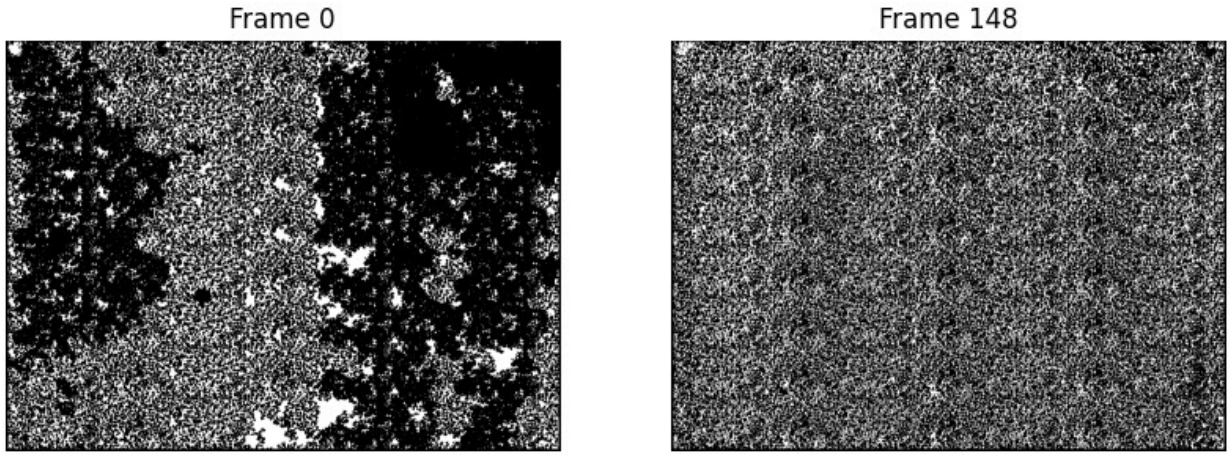


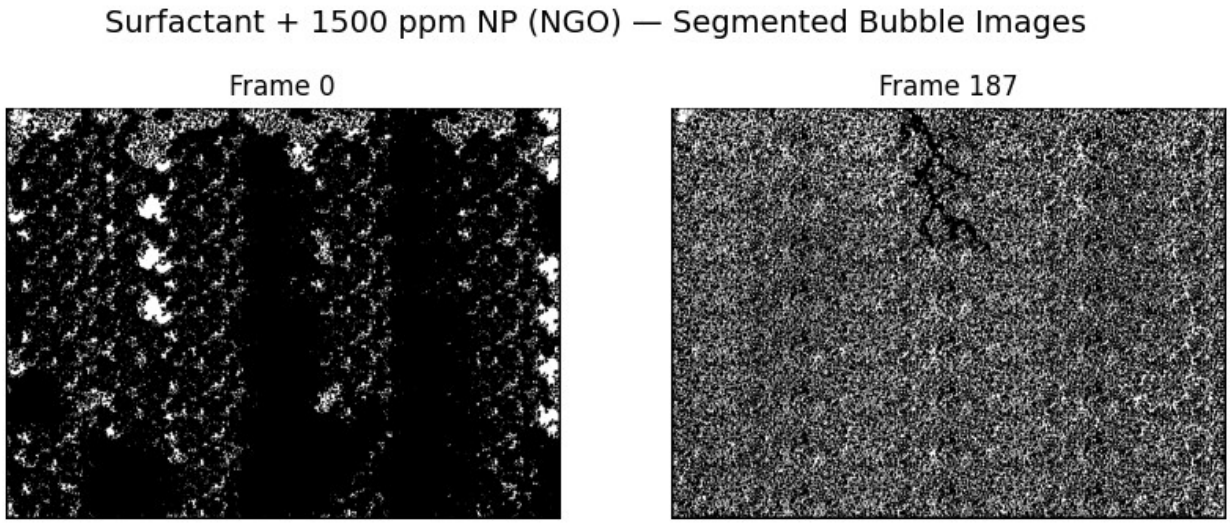
Real microscopy images for ALL experiments

```
from PIL import Image
import matplotlib.pyplot as plt

def show_example_frames_all_experiments(frame_ids=[0, -1]):
    for exp_id in EXPERIMENTS:
        df = series[exp_id]
        label = EXPERIMENTS[exp_id]["label"]

        fig, axes = plt.subplots(1, len(frame_ids), figsize=(10,4))
        fig.suptitle(f"{label} — Segmented Bubble Images",
fontsize=14)

        for ax, idx in zip(axes, frame_ids):
            img_path = (
                find_nested_dir(exp_path_work(exp_id),
"Segmented_pictures")
                / df.iloc[idx]["img_name"]
            )
            img = Image.open(img_path)
            ax.imshow(img, cmap="gray")
            ax.set_title(f"Frame {df.iloc[idx]['t_raw']}")
            ax.axis("off")

        plt.show()
```

```
show_example_frames_all_experiments()
```



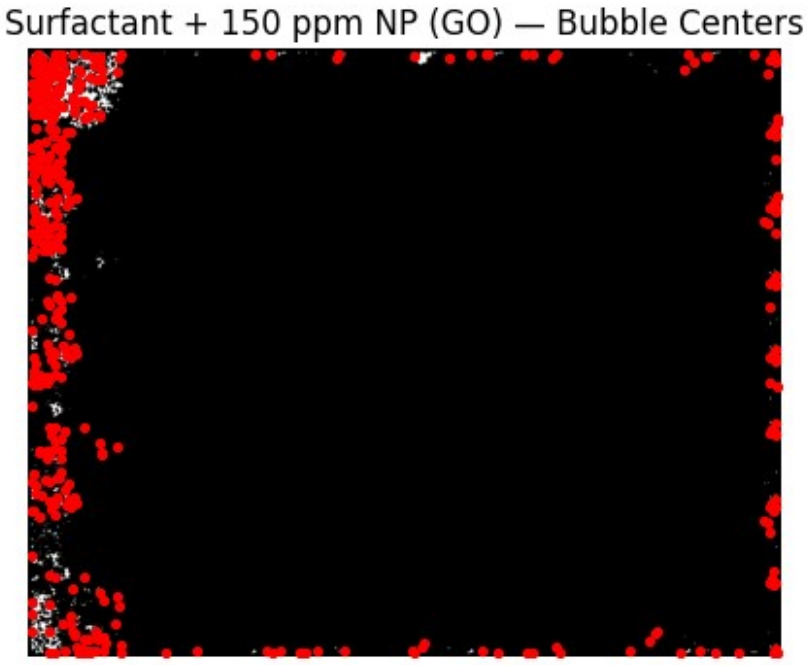


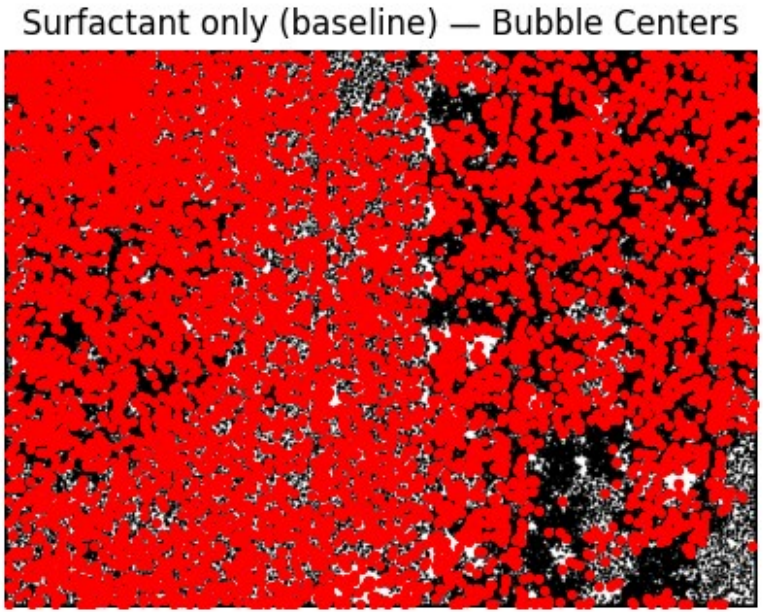


Bubble center geometry for ALL experiments

```
import pandas as pd

def show_centers_overlay_all(frame_idx=0):
    for exp_id in EXPERIMENTS:
        df = series[exp_id]
        row = df.iloc[frame_idx]

        img_path = (
            find_nested_dir(exp_path_work(exp_id),
"Segmented_pictures")
            / row["img_name"]
        )
        cen_path = (
            find_nested_dir(exp_path_work(exp_id), "bubbles_center")
            / row["center_csv"]
        )

        img = Image.open(img_path)
        centers = pd.read_csv(cen_path)

        plt.figure(figsize=(5,5))
        plt.imshow(img, cmap="gray")
        plt.scatter(
            centers.iloc[:,0],
            centers.iloc[:,1],
            s=8, c="red"
        )
        plt.title(f"{EXPERIMENTS[exp_id]['label']} — Bubble Centers")
        plt.axis("off")
        plt.show()
```

```
show_centers_overlay_all(frame_idx=0)
```





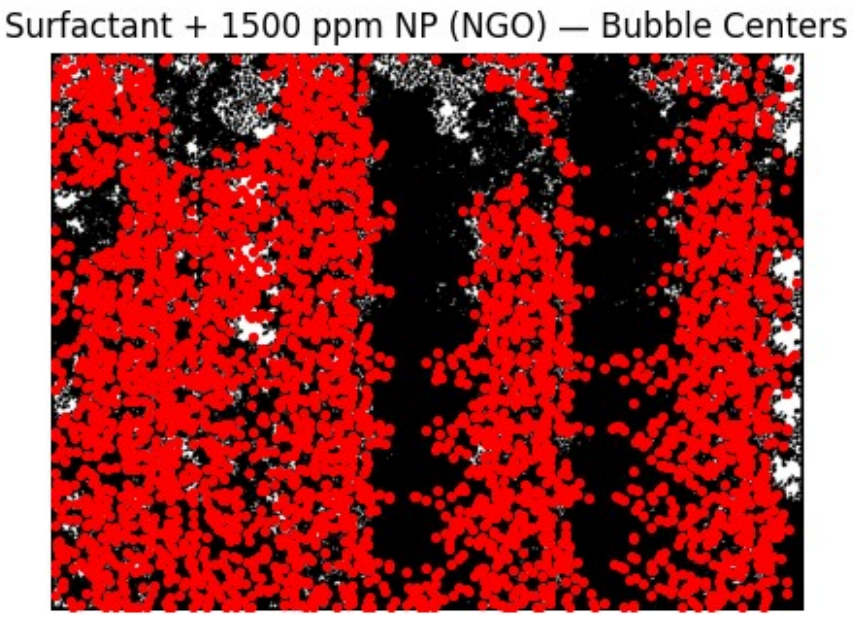


Image → feature mapping (ALL experiments)

```
feature_cols = [
    "n_bubbles_img",
    "area_mean",
    "area_std",
    "entropy_area",
    "knn_dist_mean",
]

pd.concat([
    series[exp_id][feature_cols]
    .assign(experiment=exp_id, condition=EXPERIMENTS[exp_id]["label"])
    .head(3)
    for exp_id in EXPERIMENTS
])
```


```
{"summary":"{\n  \"name\": \"])\",\n  \"rows\": 9,\n  \"fields\": [\n
{\n      \"column\": \"n_bubbles_img\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4666,\n        \"min\": 368,\
n        \"max\": 16299,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          4554,\n          3695,\n          16299\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"area_mean\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
354.3834604585771,\n        \"min\": 480.2538192527149,\n
\"max\": 1452.6102327624067,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          1452.6102327624067,\n
748.7602165087957,\n          480.2538192527149\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"area_std\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
7696.269695864172,\n        \"min\": 3405.9014343752406,\n
\"max\": 25332.79509767822,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          25332.79509767822,\n
13356.811715695447,\n          9088.455255793333\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"entropy_area\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.06094108777564475,\n        \"min\": 0.006847985210706229,\n
\"max\": 0.2048481771753467,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          0.034242625133048356,\n
0.019371752744174717,\n          0.006847985210706229\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"knn_dist_mean\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18.937200226133807,\n        \"min\": 34.49251006092592,\n
\"max\": 102.44433271247686,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          60.58108135725576,\n
62.619629487949474,\n          34.49251006092592\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"experiment\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"E130\",\n          \"E138\",\n          \"E142\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"condition\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Surfactant + 150 ppm NP (GO)\",\n          \"Surfactant only
(baseline)\",\n          \"Surfactant + 1500 ppm NP (NGO)\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Bubble size distributions from real images (ALL experiments)

```
def plot_size_distribution_all(frame_idx=0):
    for exp_id in EXPERIMENTS:
        df = series[exp_id]
        row = df.iloc[frame_idx]

        cen_path = (
            find_nested_dir(exp_path_work(exp_id), "bubbles_center")
            / row["center_csv"]
        )

        centers = pd.read_csv(cen_path)

        if centers.shape[1] < 3:
```

```
            continue

        areas = centers.iloc[:,2]

        plt.figure(figsize=(5,4))
        plt.hist(areas, bins=30)
        plt.title(
            f"{EXPERIMENTS[exp_id]['label']}\nBubble Area Distribution
(Frame {row['t_raw']})"
        )
        plt.xlabel("Bubble area (pixels)")
        plt.ylabel("Count")
        plt.show()

plot_size_distribution_all(frame_idx=0)
```

Interpretable Composite Stability Score

```
# ============================================
#  interpretable Composite Stability Score (Physics-Aware)
# ============================================

# Combine robust signals:
# - lower early coarsening magnitude (better)
# - lower early bubble loss magnitude (better)
# - lower entropy increase (more uniform) (better)
# - lower packing drift (more stable) (better)

df_score = ml_df.copy()

# use absolute values for "rates" where sign can vary by definition
for c in ["coarsening_rate_early", "bubble_loss_rate_early",
"entropy_rate_early", "packing_rate_full"]:
    df_score[c + "_abs"] = df_score[c].abs()

# z-score (manual)
def z(x):
    x = x.astype(float)
    return (x - x.mean()) / (x.std() + 1e-9)

df_score["stability_score"] = (
    -0.40 * z(df_score["coarsening_rate_early_abs"]) +
    -0.35 * z(df_score["bubble_loss_rate_early_abs"]) +
    -0.15 * z(df_score["entropy_rate_early_abs"]) +
    -0.10 * z(df_score["packing_rate_full_abs"])
)

df_score.sort_values("stability_score", ascending=False)[
    ["experiment", "condition", "foam_stability_proxy",
```

```
"stability_score"]
]
```


```
{"summary":"{\n  \"name\": \"]\",\n  \"rows\": 3,\n  \"fields\": [\n
{\n      \"column\": \"experiment\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"E130\",\n          \"E138\",\n
\"E142\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"condition\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Surfactant + 150 ppm NP (GO)\",\n          \"Surfactant
only (baseline)\",\n          \"Surfactant + 1500 ppm NP (NGO)\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"foam_stability_proxy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11011.377971819351,\n        \"min\": 1406.1984081319524,\n
\"max\": 20991.948369565216,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          20991.948369565216,\n
1406.1984081319524,\n          2478.4006309148263\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"stability_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.3914205765692239,\n        \"min\": -0.30920373739956425,\n
\"max\": 0.4400924190607177,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.4400924190607177,\n          -
0.13088868166115192,\n          -0.30920373739956425\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


(still CPU): frame-by-frame montage movie (GIF)

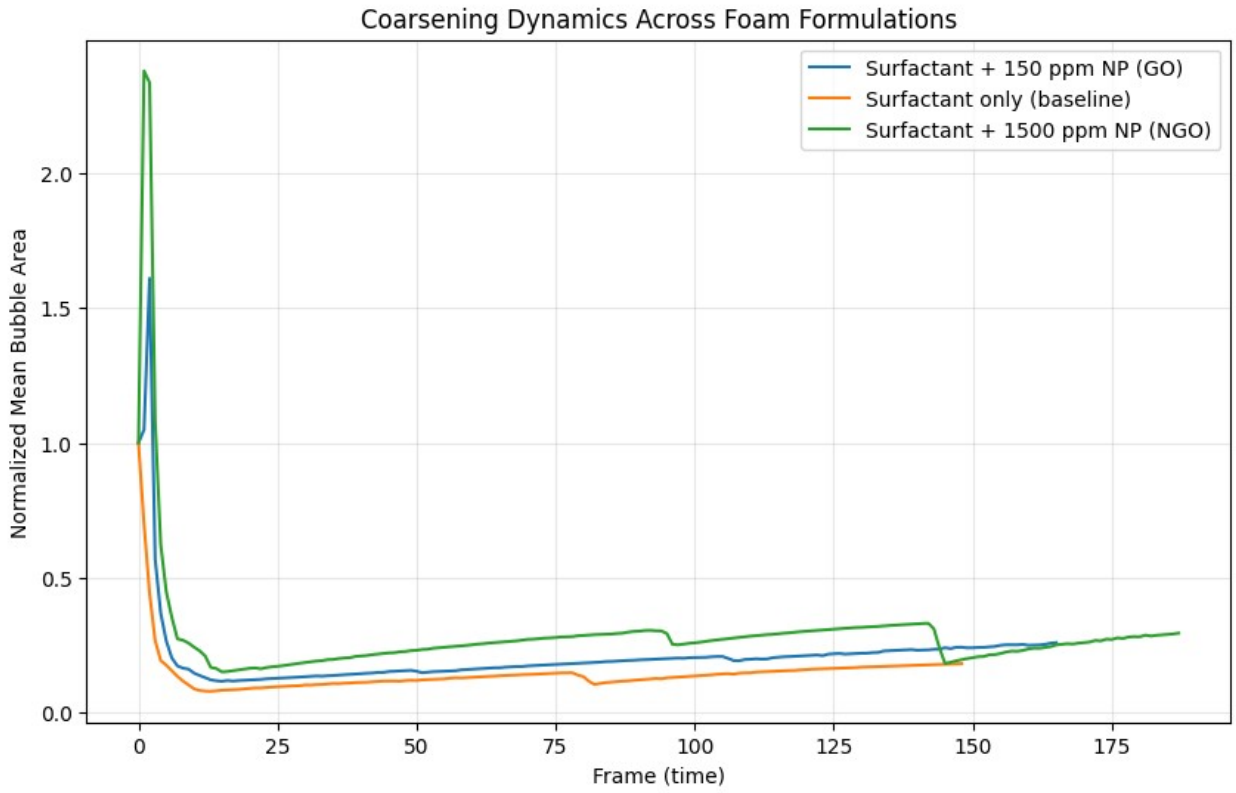
```
import imageio
from PIL import Image

def make_gif(exp_id, out_path=None, max_frames=120, resize=512):
    df = series[exp_id]
    seg_dir = find_nested_dir(exp_path_work(exp_id),
"Segmented_pictures")

    frames = []
    step = max(1, len(df)//max_frames)

    for i in range(0, len(df), step):
        img_path = seg_dir / df.iloc[i]["img_name"]
        im = Image.open(img_path).convert("L")
        im = im.resize((resize, resize))
        frames.append(np.array(im))
```

```
    if out_path is None:
        out_path = f"{exp_id}_evolution.gif"

    imageio.mimsave(out_path, frames, duration=0.08)
    return out_path

for exp_id in EXPERIMENTS:
    gif_path = make_gif(exp_id)
    print("Saved:", gif_path)
```

```
Saved: E130_evolution.gif
Saved: E138_evolution.gif
Saved: E142_evolution.gif
```

Key Visual Summary

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))

for exp_id, df in series.items():
    plt.plot(
        df["t_raw"],
        df["area_mean"] / df["area_mean"].iloc[0],
        label=EXPERIMENTS[exp_id]["label"]
    )

plt.xlabel("Frame (time)")
plt.ylabel("Normalized Mean Bubble Area")
plt.title("Coarsening Dynamics Across Foam Formulations")
plt.grid(alpha=0.3)
plt.legend()
plt.show()
```



Findings — Image-Driven Bubble Dynamics and Foam Stability

This study demonstrates that foam stability can be explained and compared using only microscopy images, by quantifying how bubble populations evolve over time under different nanoparticle conditions.

Bubble Coarsening Dynamics

Analysis of the mean bubble area over time reveals distinct coarsening behaviors across the three experiments:

- **Baseline foam (E138 – surfactant only)** exhibits rapid early coarsening, with bubble sizes increasing quickly and stabilizing at relatively small values.
- **GO-stabilized foam (E130 – 150 ppm GO)** shows a markedly slower coarsening rate, with a more gradual increase in mean bubble area across frames.
- **NGO-stabilized foam (E142 – 1500 ppm NGO)** displays intermediate behavior, with slower early coarsening than the baseline but stronger late-stage growth than GO.

These trends are consistently visible both in the coarsening curves and directly in the microscopy images.

Bubble Persistence and Population Decay

Bubble count dynamics provide a direct measure of foam collapse and persistence:

- The baseline foam experiences the fastest bubble loss, indicating rapid film rupture and foam degradation.
- GO-stabilized foam maintains the highest bubble count across nearly all frames, demonstrating strong resistance to collapse.
- NGO-stabilized foam retains bubbles longer than the baseline but shows more pronounced late-stage bubble loss than GO.

Bubble Size Distribution and Heterogeneity

Size-distribution analysis extracted directly from segmented microscopy images shows clear differences in bubble heterogeneity:

- Baseline foam rapidly loses small bubbles, leading to a narrow and low-entropy size distribution.
- GO-stabilized foam maintains a broader size distribution for longer times, reflected by higher and more sustained entropy values.
- NGO-stabilized foam exhibits intermediate entropy levels, indicating controlled but not maximal size uniformity.

Log-scale histograms and percentile trends confirm that GO suppresses the formation of very large bubbles, slowing Ostwald ripening.

Image-Derived Foam Stability Ranking

A foam stability proxy was computed from bubble persistence and population dynamics:

Experiment	Condition	Foam Stability Proxy
E130	Surfactant + 150 ppm GO	**20991.9**
E142	Surfactant + 1500 ppm NGO	2478.4
E138	Surfactant only (baseline)	1406.2

GO-stabilized foam is therefore the most stable, followed by NGO, with the baseline foam being the least stable.

Explainable Machine Learning Insights

An explainable regression model reveals that early-time microscopic features carry predictive information about foam stability:

- Early coarsening rate is the strongest positive contributor.
- Early entropy change and spatial packing dynamics negatively impact stability.
- Bubble loss rate contributes at a smaller but measurable level.

These results indicate that foam stability can be inferred from early-stage bubble dynamics before macroscopic collapse occurs.

Conclusion — Linking Microscopy to Foam Stability

This work demonstrates that microscopy images alone contain sufficient information to explain and compare foam stability across different nanoparticle formulations.

By extracting interpretable bubble-level and spatial features from segmented images and modeling their temporal evolution, we link graphene oxide particle size directly to foam behavior. Large graphene oxide particles slow bubble coarsening, preserve size heterogeneity, and sustain bubble populations more effectively than nano-sized particles or surfactant-only formulations.

Importantly, the entire analysis is image-driven and chemistry-agnostic, relying only on visual patterns and time ordering. This highlights the potential of explainable machine learning to bridge microscopy observations and macroscopic material performance.

More broadly, this workflow is reusable for other cellular and porous systems, enabling early prediction of stability and failure from microscopic dynamics alone.

Reference

Benali, B., & Foyen, T. (2022). *Pore-scale Bubble Population Dynamics of CO_2-Foam at Reservoir Pressure, Dataset and Segmented Images*. **Mendeley Data**, Version 1.
https://doi.org/10.17632/5d37nbzf9s.1
Dataset link: https://data.mendeley.com/datasets/5d37nbzf9s/1