Library importation

```
import json  # Provides functions for parsing JSON data
import numpy as np  # Provides support for large, multi-dimensional
arrays and matrices
import matplotlib.pyplot as plt  # Used for creating static, animated,
and interactive visualizations
from pathlib import Path  # Provides an object-oriented interface for
working with file system paths
from PIL import Image  # Provides functions for opening, manipulating,
and saving image files
import tensorflow as tf  # Open-source machine learning framework
import pandas as pd  # Provides data structures and data analysis
tools
from tensorflow.keras import layers  # Provides building blocks for
creating neural networks
import os  # Provides functions for interacting with the operating
system
import cv2  # Open-source computer vision library for real-time image
processing
from skimage.draw import polygon as sk_polygon  # Provides functions
for drawing polygons on images
from tensorflow.keras.preprocessing.image import load_img,
img_to_array  # Provides functions to load and convert images to
arrays
import shutil  # Provides functions for high-level file operations

# Define paths
json_path = Path(r"C:\Users\isaac\Downloads\coco - Copy6\annotations\
instances_default.json")
image_folder = Path(r"C:\Users\isaac\Downloads\coco - Copy6\images\
default")
```

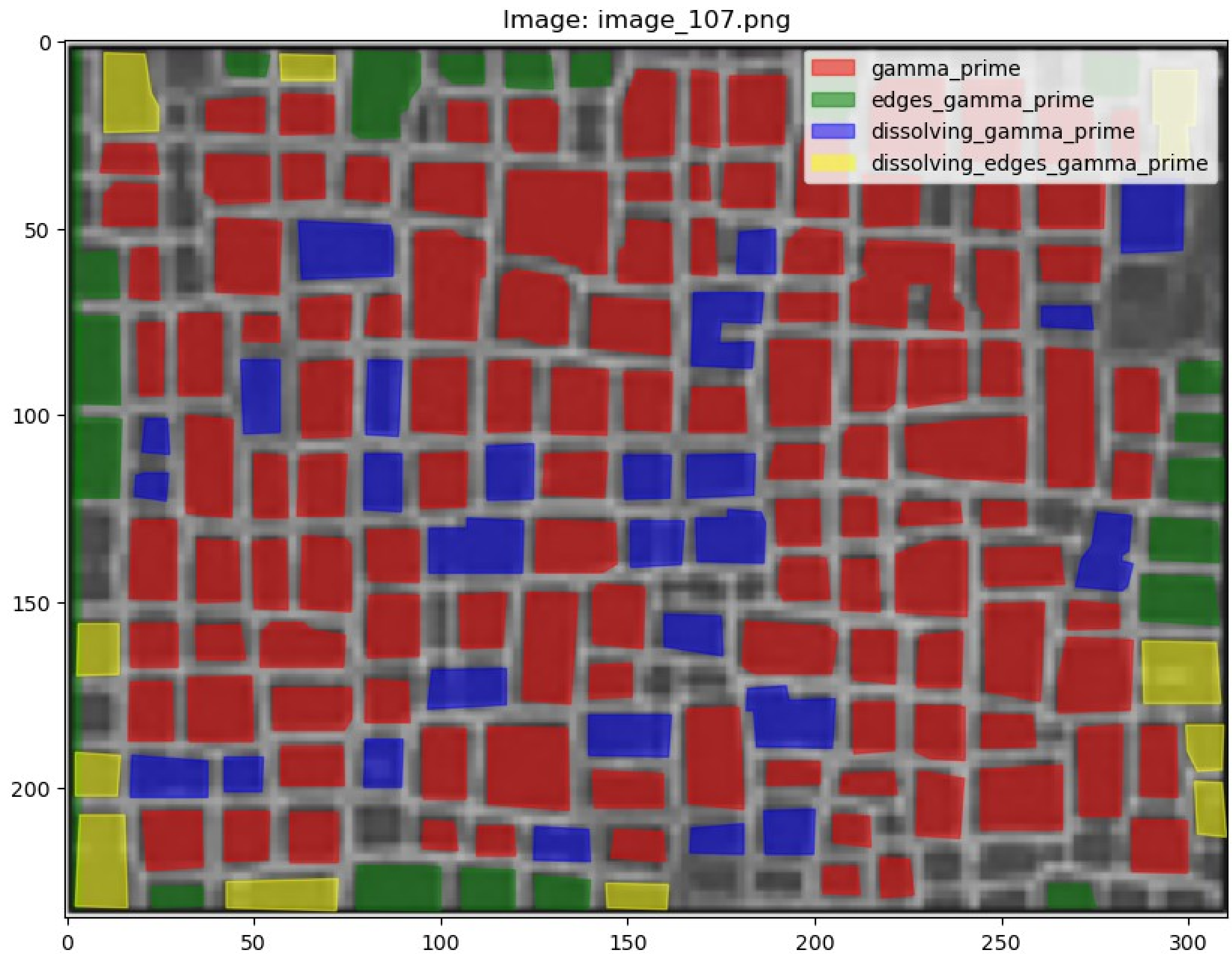
Loads image annotations from a JSON file, maps category IDs to label names and colors,

and visualizes image annotations by displaying segmentation polygons on the corresponding image.

```
# Load the annotations from the JSON file
with open(json_path) as f:
    data = json.load(f)

# Create a dictionary to map category IDs to label names and colors
category_map = {
    1: ('gamma_prime', 'red'),
    2: ('dissolving_gamma_prime', 'blue'),
    3: ('edges_gamma_prime', 'green'),
```

```
    4: ('dissolving_edges_gamma_prime', 'yellow')
}

# Function to display images with annotations
def visualize_image_annotations(image_id):
    # Find the image details
    image_info = next(img for img in data['images'] if img['id'] ==
image_id)
    image_filename = image_info['file_name']
    image_path = image_folder / image_filename

    # Load the image
    img = Image.open(image_path)
    img_array = np.array(img)

    # Create the plot
    fig, ax = plt.subplots(1, figsize=(10, 10))
    ax.imshow(img_array)

    # Filter annotations for this image
    annotations = [ann for ann in data['annotations'] if
ann['image_id'] == image_id]

    # Keep track of which labels have been added to the legend
    added_labels = set()

    # Loop over each annotation and draw the segmentation polygon with
corresponding color
    for ann in annotations:
        # Get the category ID and label with color
        category_id = ann['category_id']
        label, color = category_map.get(category_id, ('Unknown',
'black'))

        # Only add the label to the legend once
        if label not in added_labels:
            added_labels.add(label)
            # Draw the segmentation polygons
            if 'segmentation' in ann:
                for seg in ann['segmentation']:
                    poly = np.array(seg).reshape((len(seg) // 2, 2))
                    ax.fill(poly[:, 0], poly[:, 1], alpha=0.5,
label=label, color=color)
        else:
            # Draw the segmentation polygons without adding a new
label to the legend
            if 'segmentation' in ann:
                for seg in ann['segmentation']:
                    poly = np.array(seg).reshape((len(seg) // 2, 2))
```

```
                    ax.fill(poly[:, 0], poly[:, 1], alpha=0.5,
color=color)

    ax.set_title(f"Image: {image_filename}")
    plt.legend(loc='upper right')
    plt.show()

# Example: Visualize annotations for the ___ image
image_id = data['images'][10]['id']
visualize_image_annotations(image_id)
```



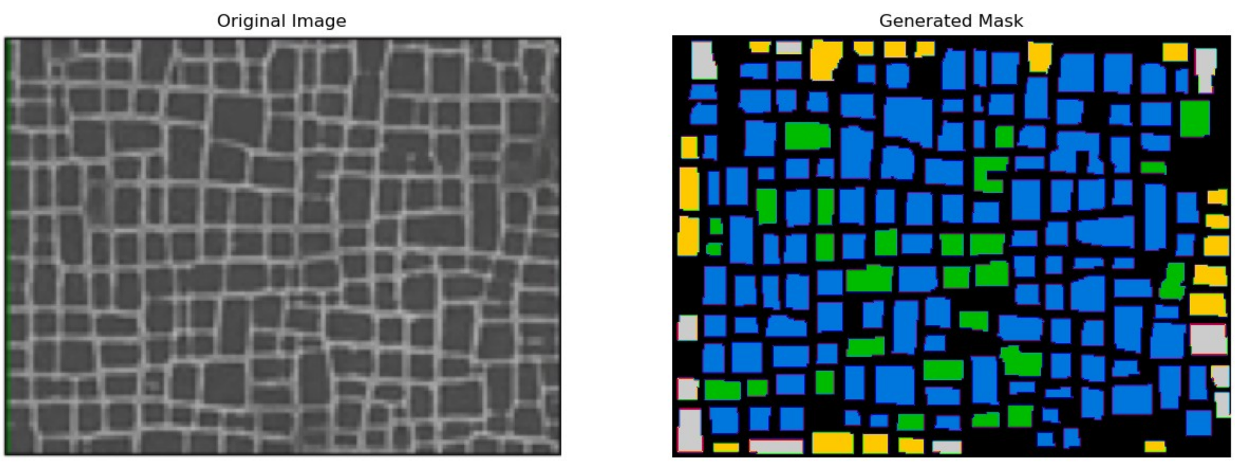
Generating masks from the image annotations by converting segmentation polygons into a binary mask

and then visualizes both the original image and the generated mask side by side.

```
def create_mask(image_id, image_folder, json_data, category_map):
    # Find the image details
```

```
    image_info = next(img for img in json_data['images'] if img['id']
== image_id)
    image_filename = image_info['file_name']
    image_path = image_folder / image_filename

    # Load the image to get dimensions (height, width)
    img = Image.open(image_path)
    img_width, img_height = img.size

    # Create an empty mask of the same size (all zeros)
    mask = np.zeros((img_height, img_width), dtype=np.uint8)

    # Filter annotations for this image
    annotations = [ann for ann in json_data['annotations'] if
ann['image_id'] == image_id]

    # Loop over each annotation and draw the segmentation polygon
    for ann in annotations:
        category_id = ann['category_id']
        label, color = category_map.get(category_id, ('Unknown',
'black'))

        # Convert segmentation into a polygon and fill the mask with
the category ID
        if 'segmentation' in ann:
            for seg in ann['segmentation']:
                poly = np.array(seg).reshape((len(seg) // 2, 2))

                # Convert polygon to raster (draw it on the mask)
                rr, cc = sk_polygon(poly[:, 1], poly[:, 0],
shape=mask.shape)  # Convert polygon to raster

                # Ensure the indices are within bounds
                rr = np.clip(rr, 0, mask.shape[0] - 1)
                cc = np.clip(cc, 0, mask.shape[1] - 1)

                # Set the pixels inside the polygon to the category ID
                mask[rr, cc] = category_id

    return mask

# Example: Generate mask for a specific image
image_id = data['images'][10]['id']
mask = create_mask(image_id, image_folder, data, category_map)

# Function to visualize the mask
def visualize_mask(image_id, image_folder, mask):
    # Find the image details
    image_info = next(img for img in data['images'] if img['id'] ==
image_id)
```

```
    image_filename = image_info['file_name']
    image_path = image_folder / image_filename

    # Load the original image for comparison
    img = Image.open(image_path)

    # Create the plot
    fig, axs = plt.subplots(1, 2, figsize=(15, 10))

    # Show original image
    axs[0].imshow(img)
    axs[0].set_title("Original Image")
    axs[0].axis('off')

    # Show the generated mask
    axs[1].imshow(mask, cmap='nipy_spectral')  # Using a colormap to differentiate the regions
    axs[1].set_title("Generated Mask")
    axs[1].axis('off')

    plt.show()

# Example: Visualize the mask for a specific image
visualize_mask(image_id, image_folder, mask)
```



Generates and saves masks for all images in the dataset by creating masks

from image annotations and storing them in a specified output folder.

```
# Function to save all the masks
def save_all_masks(image_folder, json_data, category_map, output_folder):
    # Create the output folder if it doesn't exist
```

```
    Path(output_folder).mkdir(parents=True, exist_ok=True)

    # Iterate over all images
    for image_info in json_data['images']:
        image_id = image_info['id']

        # Generate the mask for each image
        mask = create_mask(image_id, image_folder, json_data, 
category_map)

        # Generate the mask file name
        image_filename = image_info['file_name']
        mask_filename = f"mask_{Path(image_filename).stem}.png"  # 
Using the image's name to name the mask file

        # Save the mask to the output folder
        mask_path = Path(output_folder) / mask_filename
        cv2.imwrite(str(mask_path), mask)

        #print(f"Mask for {image_filename} saved to {mask_path}")

# Define the output folder
output_folder = r"C:\Users\isaac\Downloads\coco - Copy6\mask$$$"

# Call the function to save all the masks
save_all_masks(image_folder, data, category_map, output_folder)
```

Visualizing the image annotations for a specific class by displaying segmentation polygons

for that class on the image, with each class shown separately.

```
# Function to display images with annotations for a specific class
def visualize_class_annotations(image_id, class_id):
    # Find the image details
    image_info = next(img for img in data['images'] if img['id'] == 
image_id)
    image_filename = image_info['file_name']
    image_path = image_folder / image_filename

    # Load the image
    img = Image.open(image_path)
    img_array = np.array(img)

    # Create the plot
    fig, ax = plt.subplots(1, figsize=(10, 10))
    ax.imshow(img_array)

    # Filter annotations for this image
```

```
    annotations = [ann for ann in data['annotations'] if
ann['image_id'] == image_id and ann['category_id'] == class_id]

    # Keep track of which labels have been added to the legend
    added_labels = set()

    # Loop over each annotation and draw the segmentation polygon with
corresponding color
    for ann in annotations:
        # Get the category ID and label with color
        category_id = ann['category_id']
        label, color = category_map.get(category_id, ('Unknown',
'black'))

        # Draw the segmentation polygons
        if 'segmentation' in ann:
            for seg in ann['segmentation']:
                poly = np.array(seg).reshape((len(seg) // 2, 2))
                # Only add the label to the legend once
                if label not in added_labels:
                    ax.fill(poly[:, 0], poly[:, 1], alpha=0.5,
label=label, color=color)
                    added_labels.add(label)
                else:
                    ax.fill(poly[:, 0], poly[:, 1], alpha=0.5,
color=color)

    ax.set_title(f"Image: {image_filename} - Class: {label}")
    plt.legend(loc='upper right')
    plt.show()

# Example: Visualize annotations for each class in the image
image_id = data['images'][10]['id']

# Visualize each class separately
for class_id in category_map.keys():
    visualize_class_annotations(image_id, class_id)
```

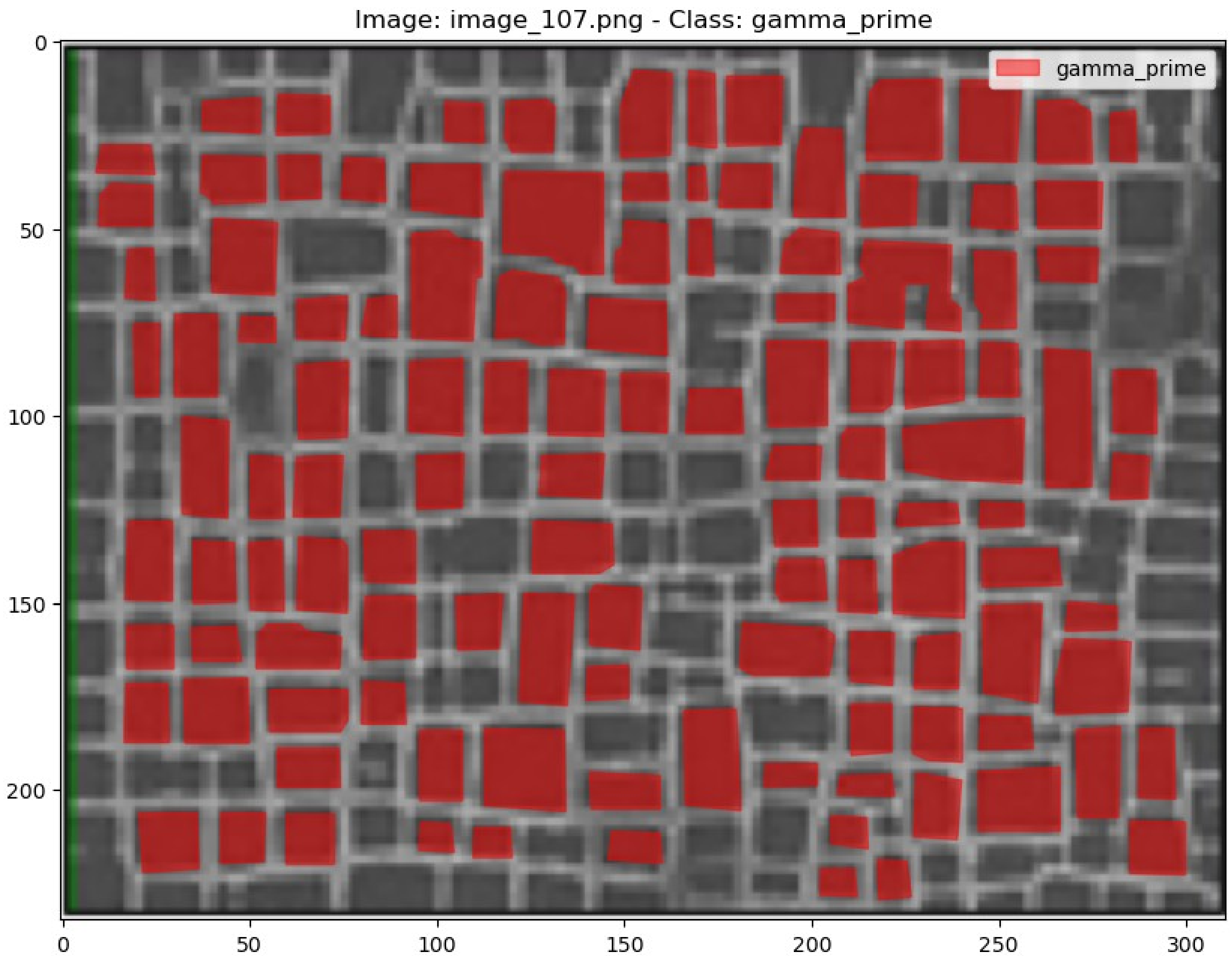


Image: image_107.png - Class: gamma_prime
gamma_prime
0
50
100
150
200
0
50
100
150
200
250
300

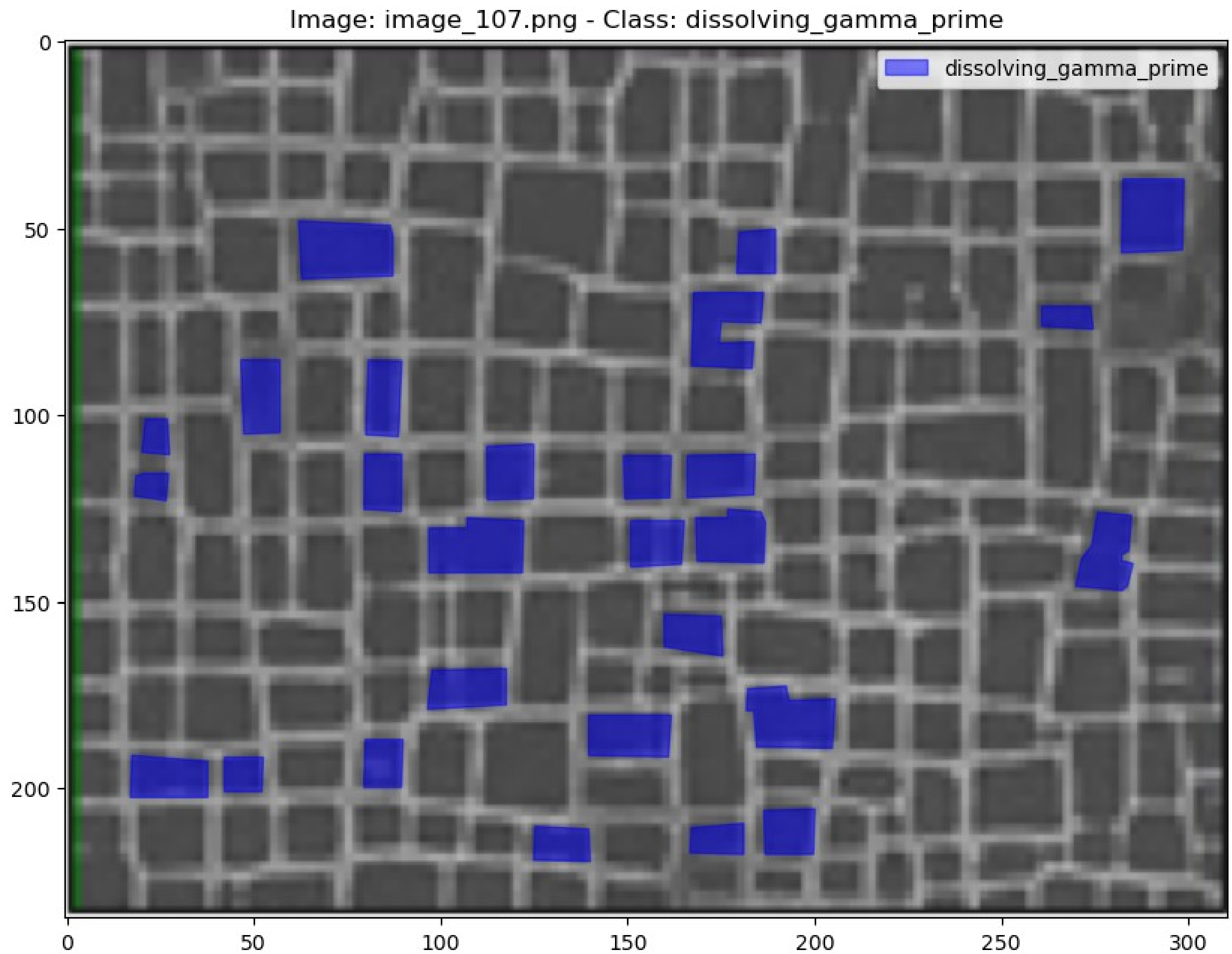


Image: image_107.png - Class: dissolving_gamma_prime
dissolving_gamma_prime
0
50
100
150
200
250
300

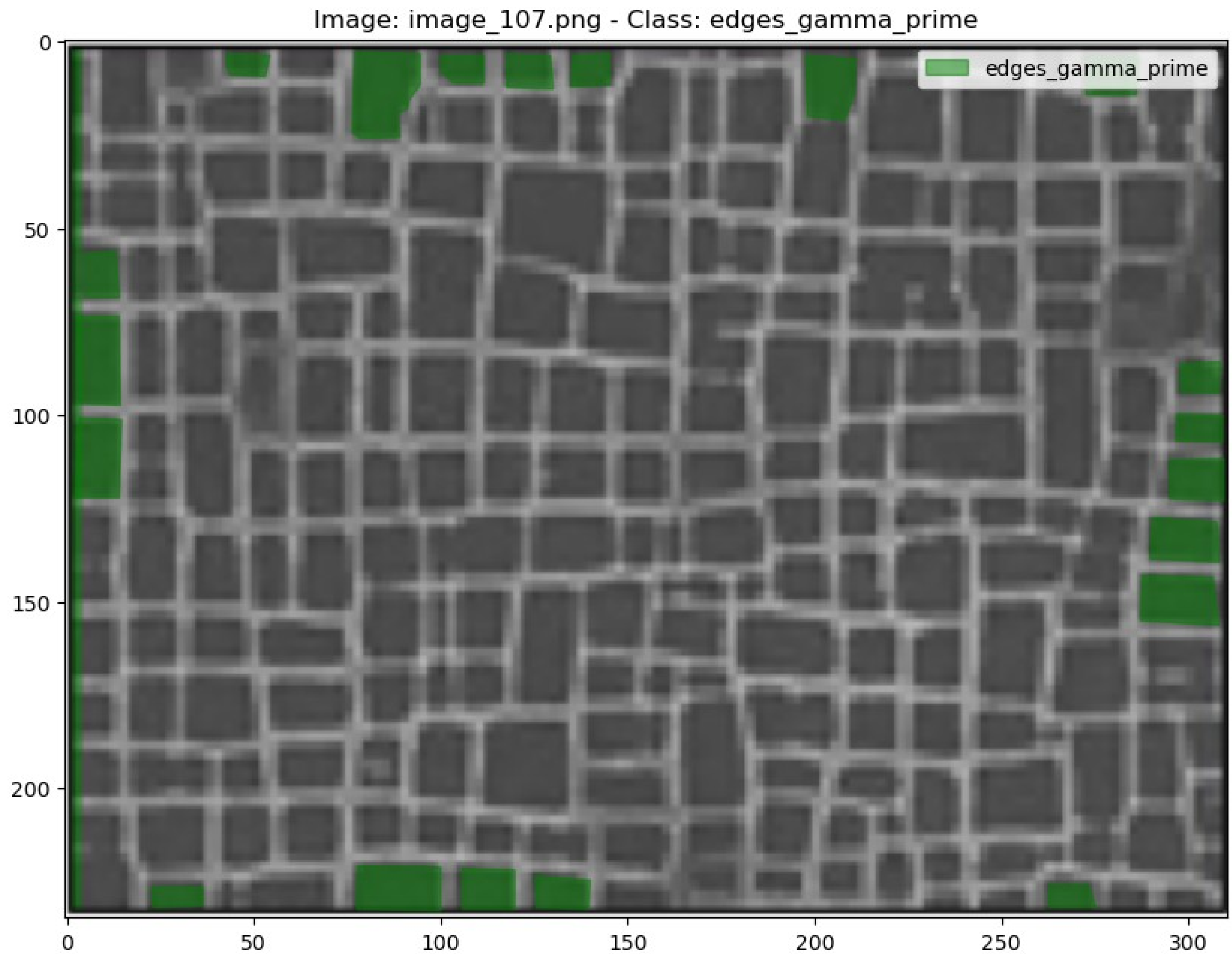
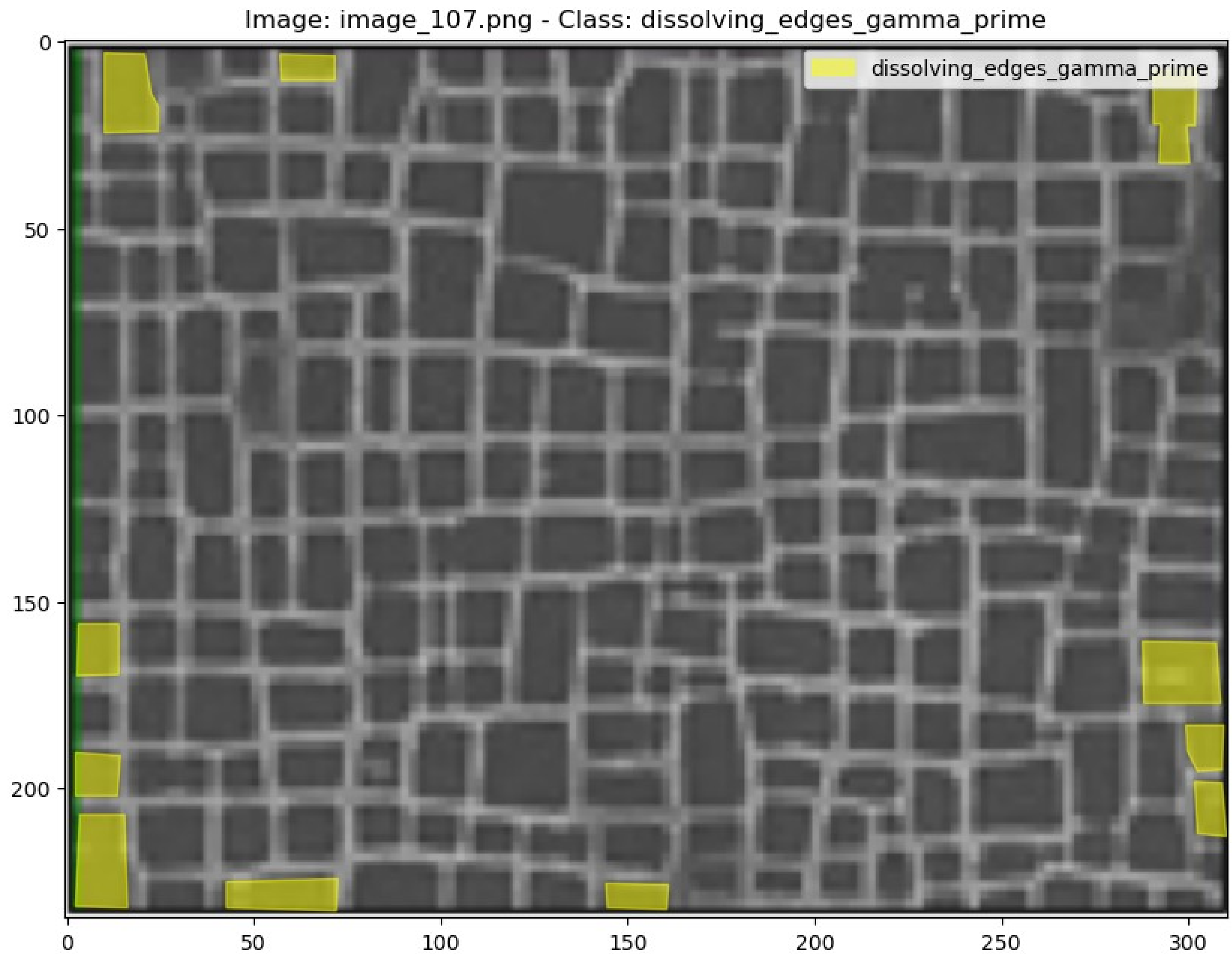


Image: image_107.png - Class: edges_gamma_prime
edges_gamma_prime
0
50
100
150
200
0
50
100
150
200
250
300



Preprocesses images (denoising, enhancing contrast), segments them, calculates particle and precipitate sizes,

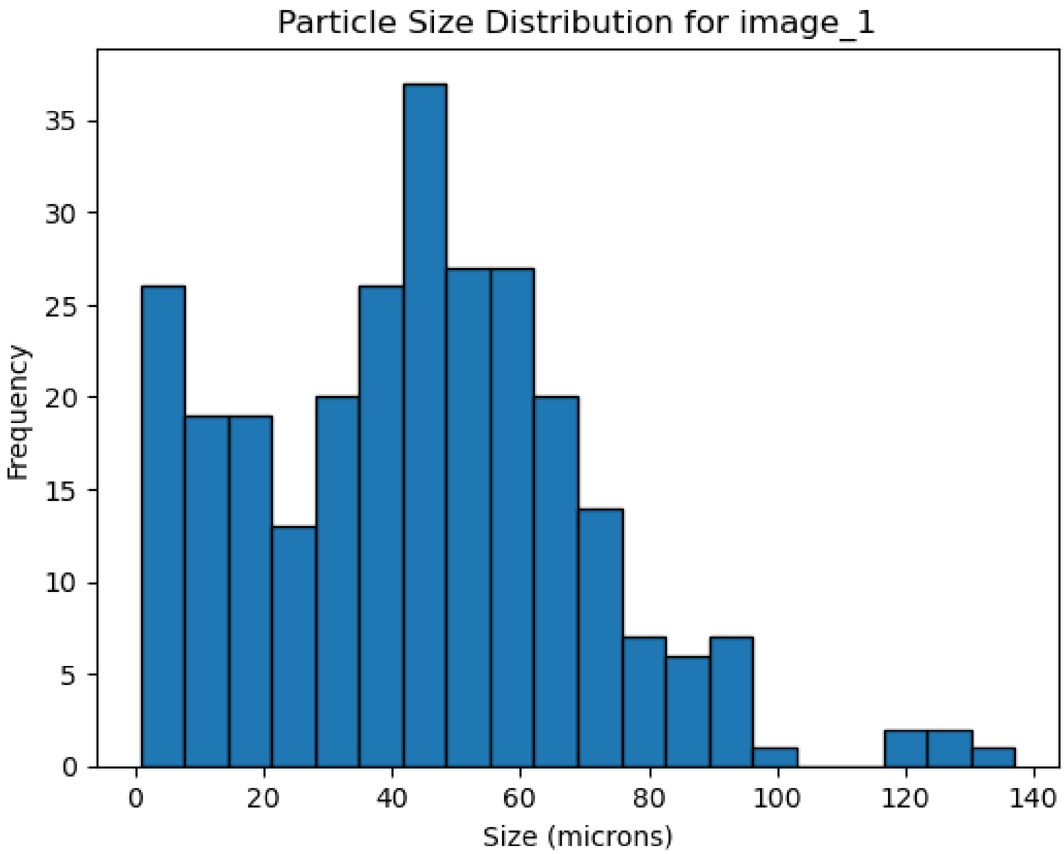
detects defects, and performs statistical analysis while visualizing size distributions and interpreting material properties.

```
# Function to preprocess the image (denoise and enhance contrast)
def preprocess_image(image_path):
    image = cv2.imread(str(image_path))
    if image is None:
        print(f"Error: Unable to open image at {image_path}")
        return None

    # Denoising using Gaussian blur
    image = cv2.GaussianBlur(image, (5, 5), 0)

    # Convert to grayscale for further processing
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Enhance contrast using histogram equalization
```

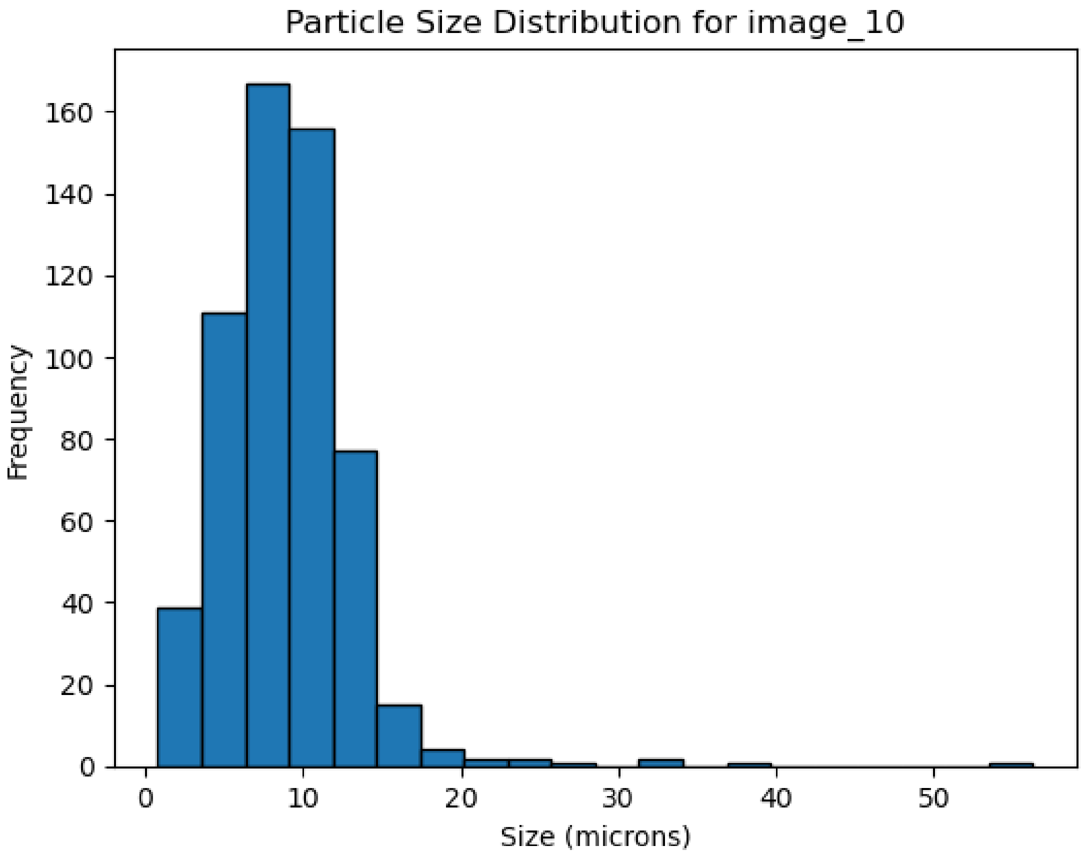
```
    enhanced_image = cv2.equalizeHist(gray_image)

    return enhanced_image

# Function for segmentation using thresholding
def segment_image(image):
    # Use Otsu's thresholding to separate phases
    _, segmented = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV
+ cv2.THRESH_OTSU)

    return segmented

# Function to calculate the area and equivalent diameter (for particle
or precipitate size)
def calculate_area_and_diameter(segmentation):
    contours, _ = cv2.findContours(segmentation, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    areas = []
    diameters = []

    for contour in contours:
        area = cv2.contourArea(contour)
        if area > 0:
            equivalent_diameter = np.sqrt(4 * area / np.pi)  #
Equivalent diameter
            areas.append(area)
            diameters.append(equivalent_diameter)

    return areas, diameters

# Function to detect cracks using Canny edge detection
def detect_cracks(image):
    edges = cv2.Canny(image, 100, 200)
    return edges

# Function to detect voids/tears using morphological operations
def detect_voids(image):
    kernel = np.ones((5, 5), np.uint8)
    voids = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
    return voids

# Function to plot size distribution of particles or precipitates
def plot_size_distribution(sizes, title):
    plt.hist(sizes, bins=20, edgecolor='black')
    plt.title(title)
    plt.xlabel('Size (microns)')
    plt.ylabel('Frequency')
    plt.show()

# Function to compute basic statistics (mean, median, std)
```

```
def compute_statistics(sizes):
    mean_size = np.mean(sizes)
    median_size = np.median(sizes)
    std_dev = np.std(sizes)

    return mean_size, median_size, std_dev

# Function to interpret the results and assess material properties
def interpret_results(particle_size, precipitate_size, defects):
    if particle_size < 10:
        print("Fine-particle microstructure: Likely to have higher
strength.")
    else:
        print("Coarse-particle microstructure: Likely to have lower
strength.")

    if len(defects) > 5:
        print("High number of defects detected: Material integrity may
be compromised.")
    else:
        print("Low number of defects: Material is likely strong and
reliable.")

# Process all images in the directory
image_paths = list(image_folder.glob("*.png"))  # images are .png
format
all_particle_sizes = []
all_precipitate_sizes = []
all_defects = []

for image_path in image_paths:
    # Preprocess the image
    preprocessed_image = preprocess_image(image_path)
    if preprocessed_image is None:
        continue  # Skip image if it couldn't be processed

    # Segment the image
    segmented_image = segment_image(preprocessed_image)

    # Extract geometrical features (particle sizes, precipitate sizes)
    particle_areas, particle_diameters =
calculate_area_and_diameter(segmented_image)
    precipitate_areas, precipitate_diameters =
calculate_area_and_diameter(segmented_image)  # If needed for
precipitates

    all_particle_sizes.extend(particle_diameters)  # Collect particle+
sizes
    all_precipitate_sizes.extend(precipitate_diameters)  # Collect
precipitate sizes
```

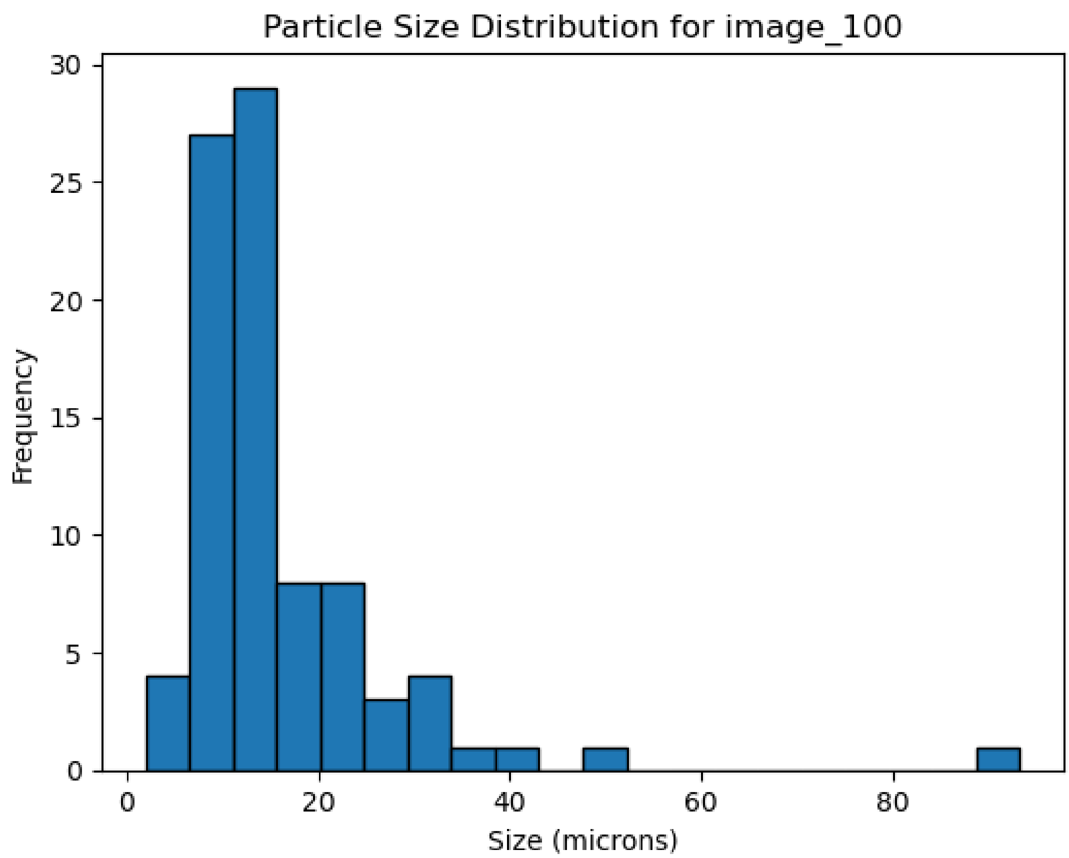
```
    # Detect defects (cracks and voids)
    cracks = detect_cracks(preprocessed_image)
    voids = detect_voids(segmented_image)
    all_defects.append((cracks, voids))  # Store detected defects for
later analysis

    # Plot size distribution
    plot_size_distribution(particle_diameters, f"Particle Size
Distribution for {image_path.stem}")

    # Perform statistical analysis
    mean, median, std_dev = compute_statistics(particle_diameters)
    print(f"Particle Size Stats for {image_path.stem} - Mean: {mean},
Median: {median}, Std Dev: {std_dev}")

    # Interpret the results
    interpret_results(mean, None, cracks)  # Interpret based on
particle size and cracks

# After processing all images, plot combined statistics
plot_size_distribution(all_particle_sizes, "Combined Particle Size
Distribution")
plot_size_distribution(all_precipitate_sizes, "Combined Precipitate
Size Distribution")
```



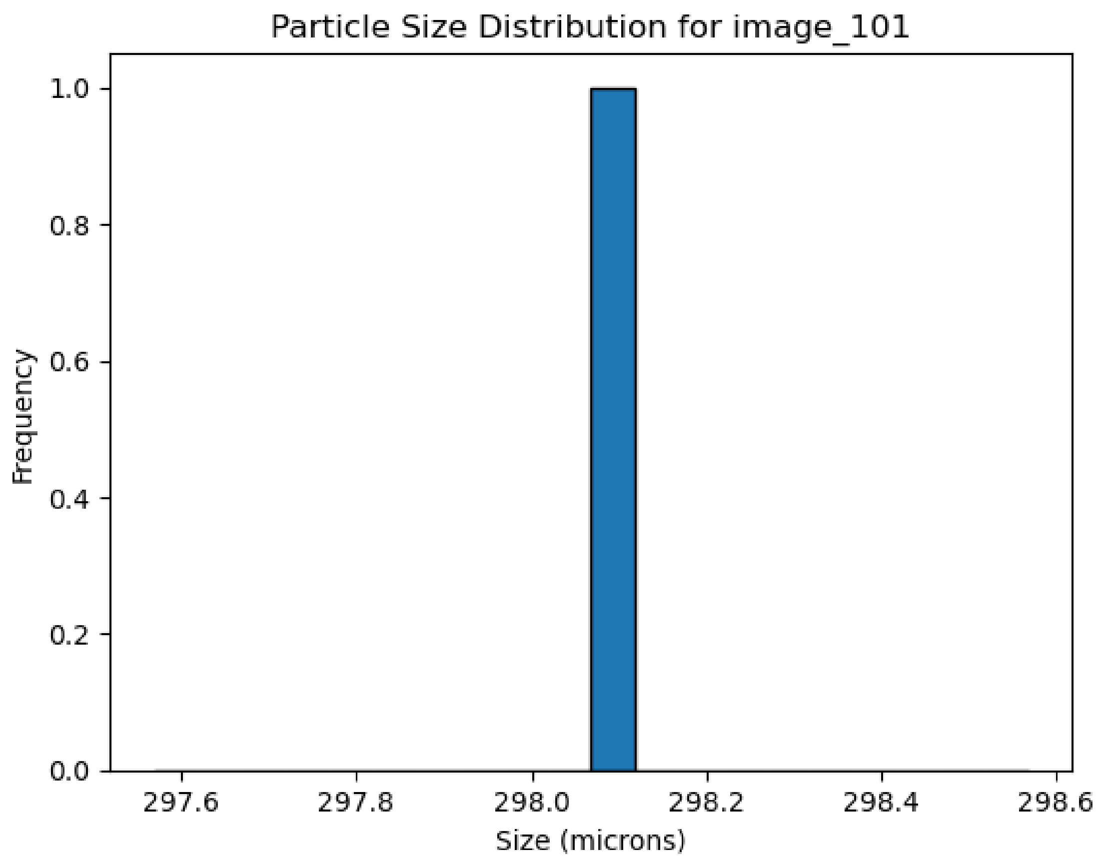

```
Particle Size Stats for image_1 - Mean: 43.79884113313701, Median:
44.56740556478078, Std Dev: 26.015830373295056
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



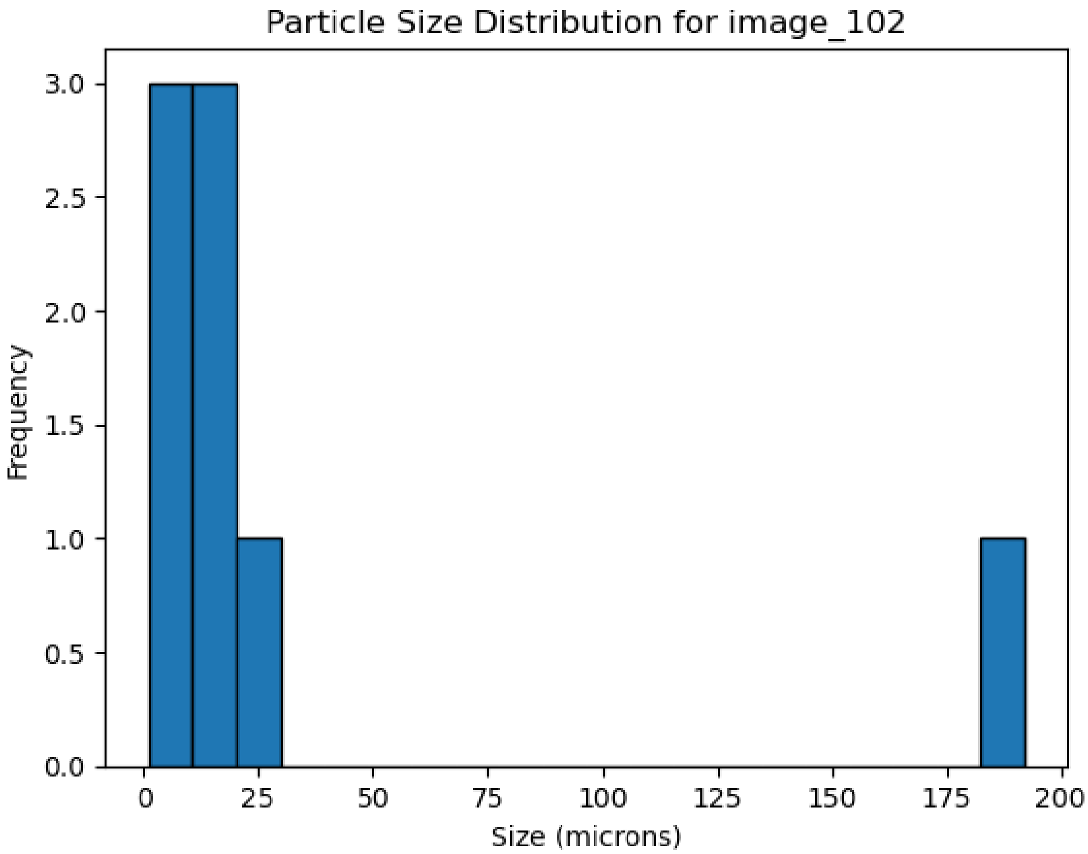

```
Particle Size Stats for image_10 - Mean: 8.94930712519118, Median:
8.51907589177983, Std Dev: 4.494512866931588
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



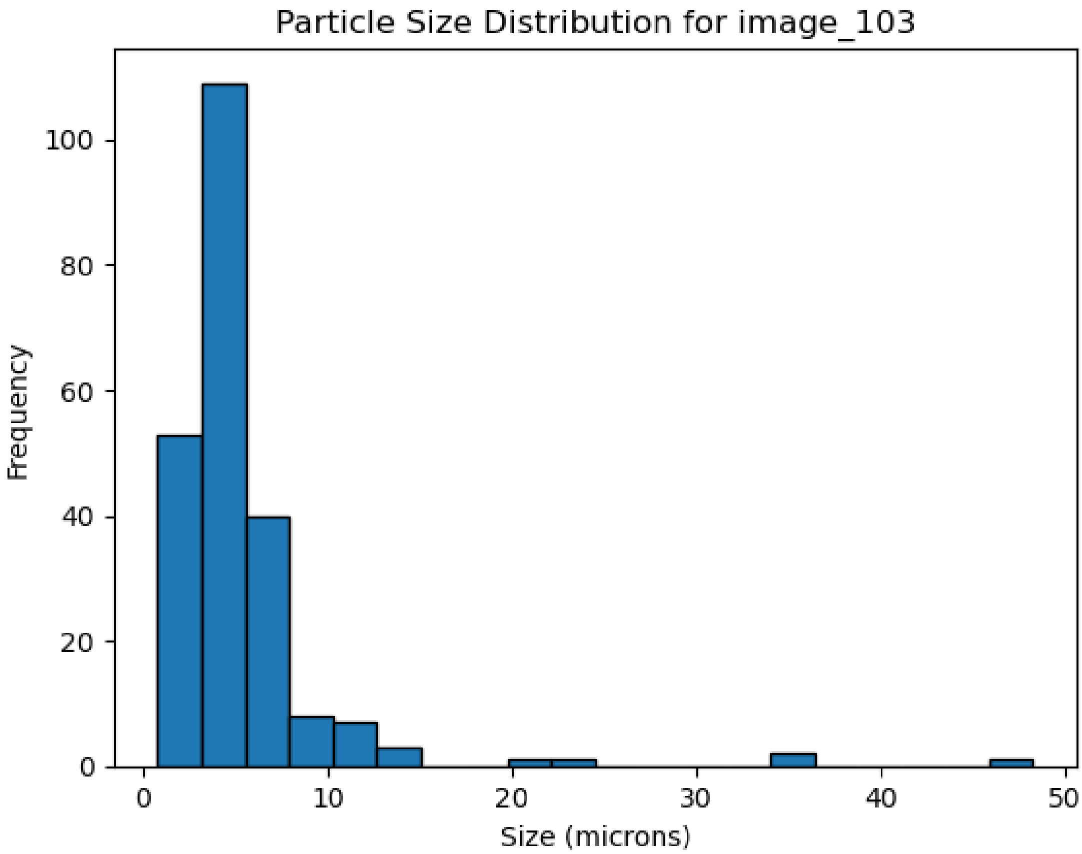

```
Particle Size Stats for image_100 - Mean: 15.868381924410874, Median:
12.939513133525306, Std Dev: 11.690034782337873
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



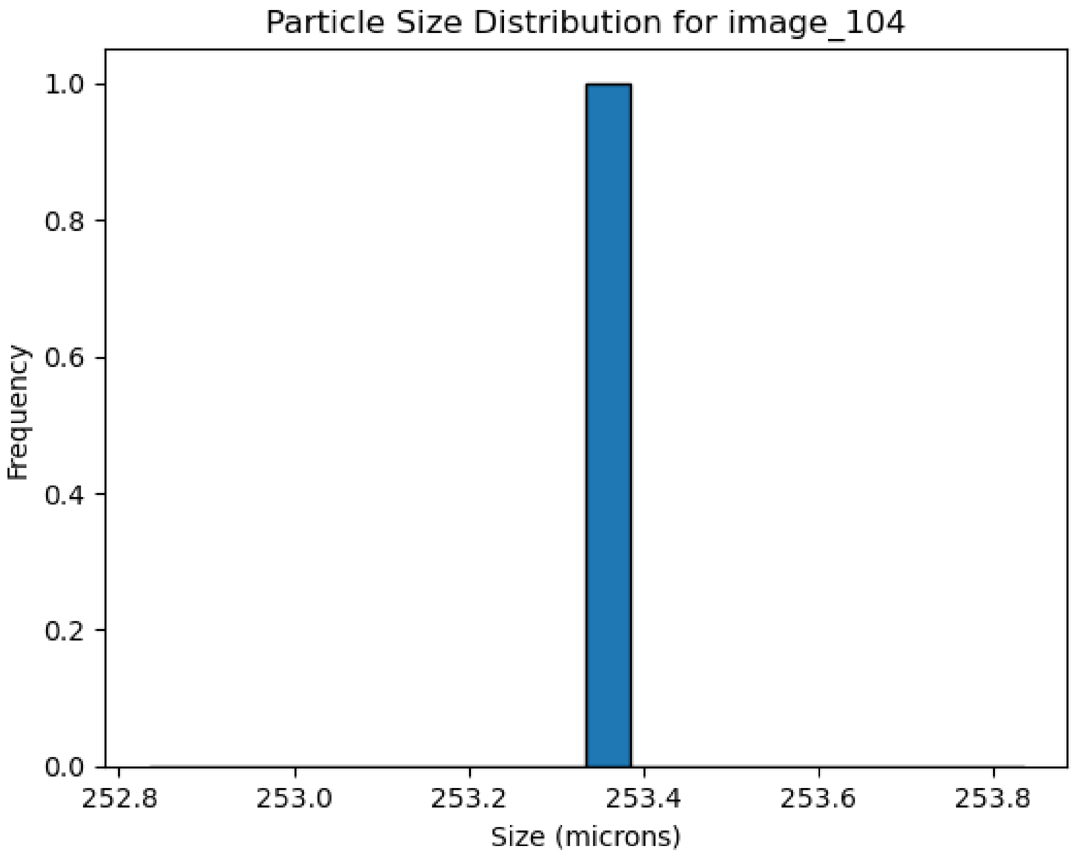

```
Particle Size Stats for image_101 - Mean: 298.0683572140836, Median:
298.0683572140836, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



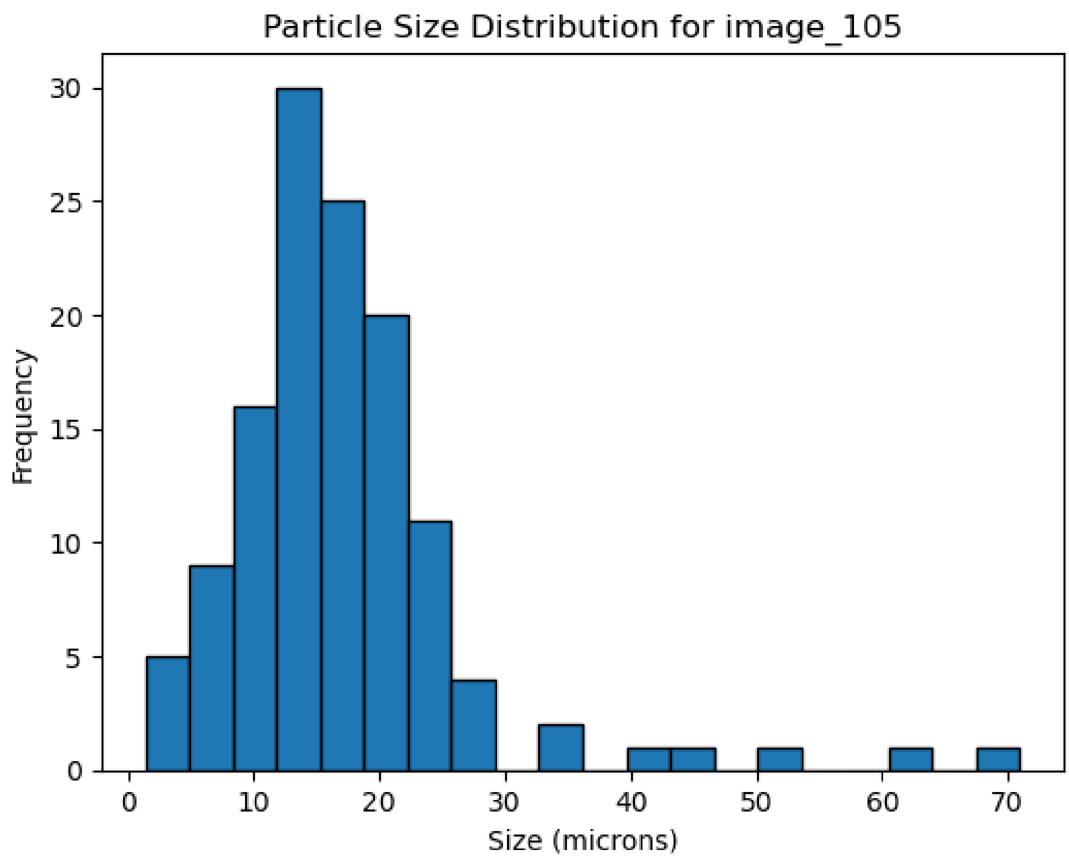

```
Particle Size Stats for image_102 - Mean: 33.56548676708502, Median:
13.499004245710108, Std Dev: 60.33941015958282
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



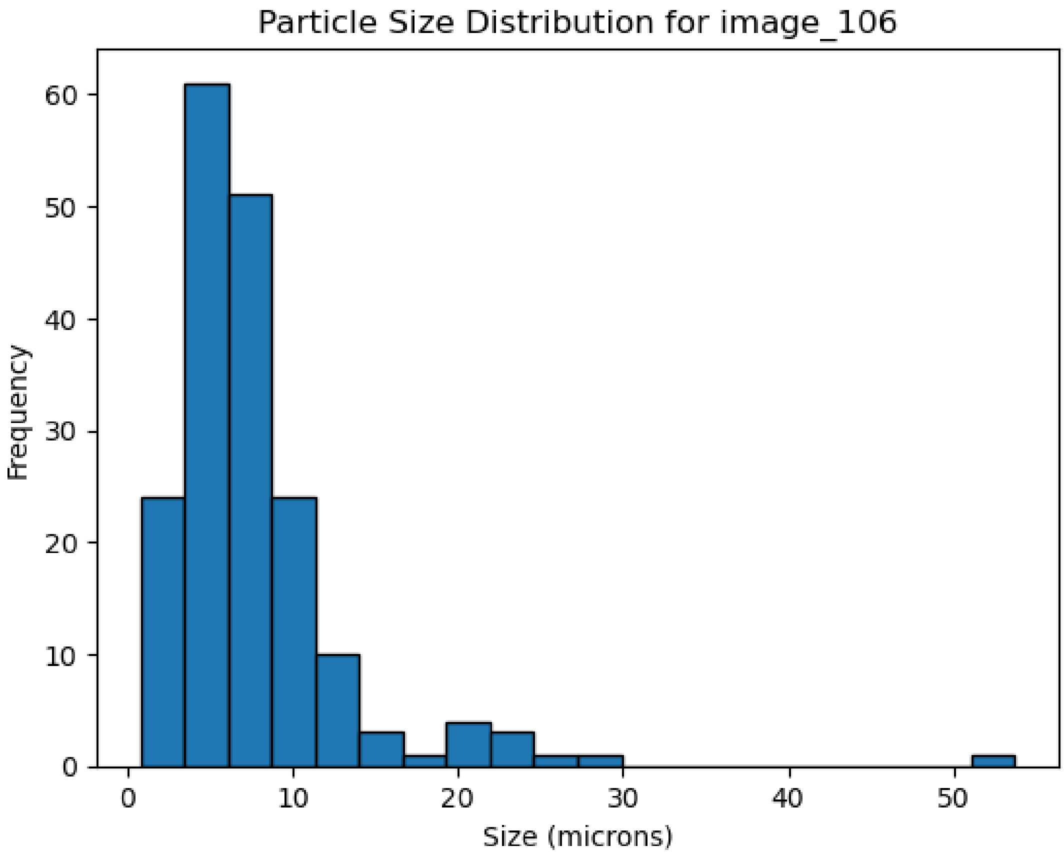

```
Particle Size Stats for image_103 - Mean: 5.34737349920857, Median:
4.370193722368317, Std Dev: 5.037820527955586
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



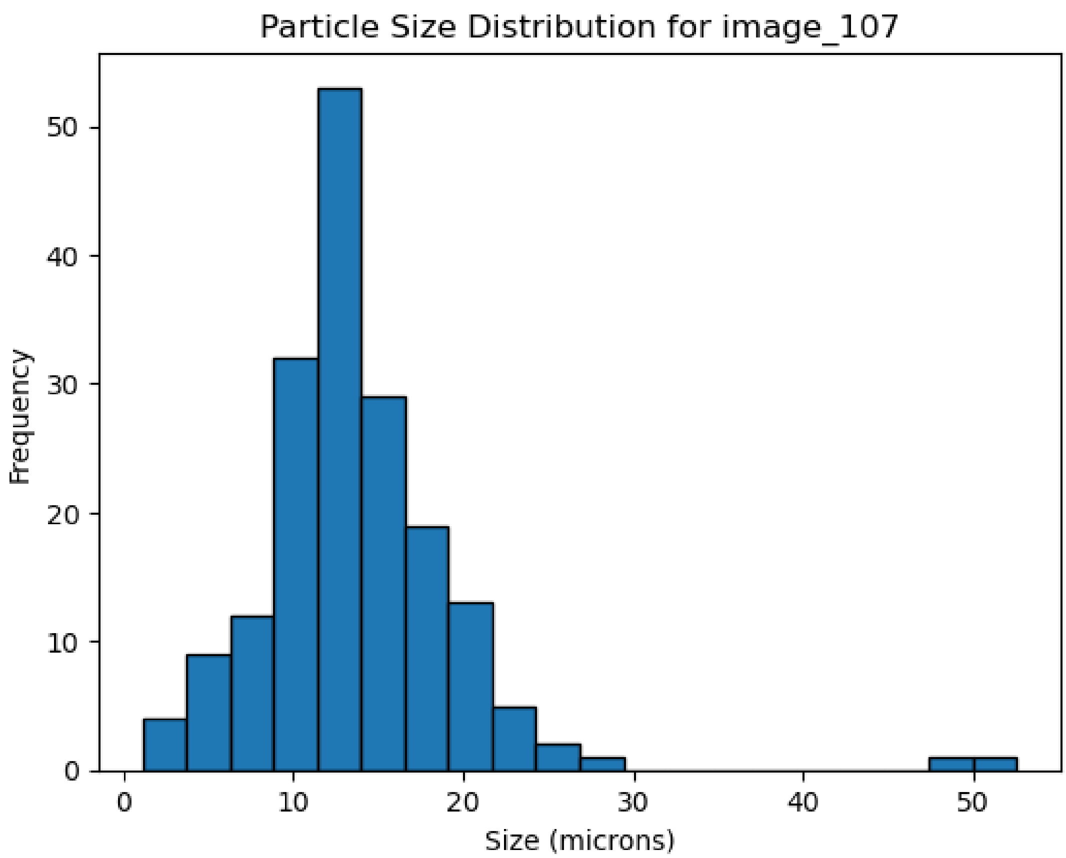

```
Particle Size Stats for image_104 - Mean: 253.33431641242024, Median:
253.33431641242024, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



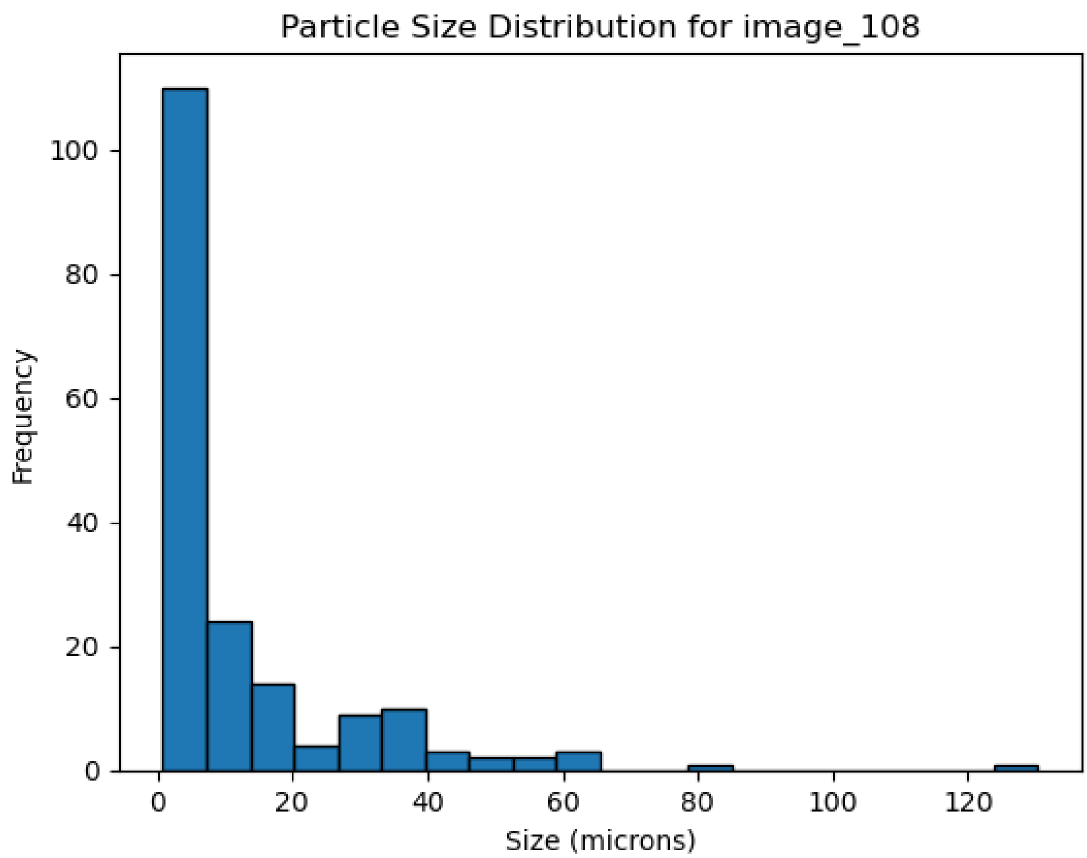

```
Particle Size Stats for image_105 - Mean: 17.365027126945535, Median:
15.635280380893438, Std Dev: 9.848087283156296
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



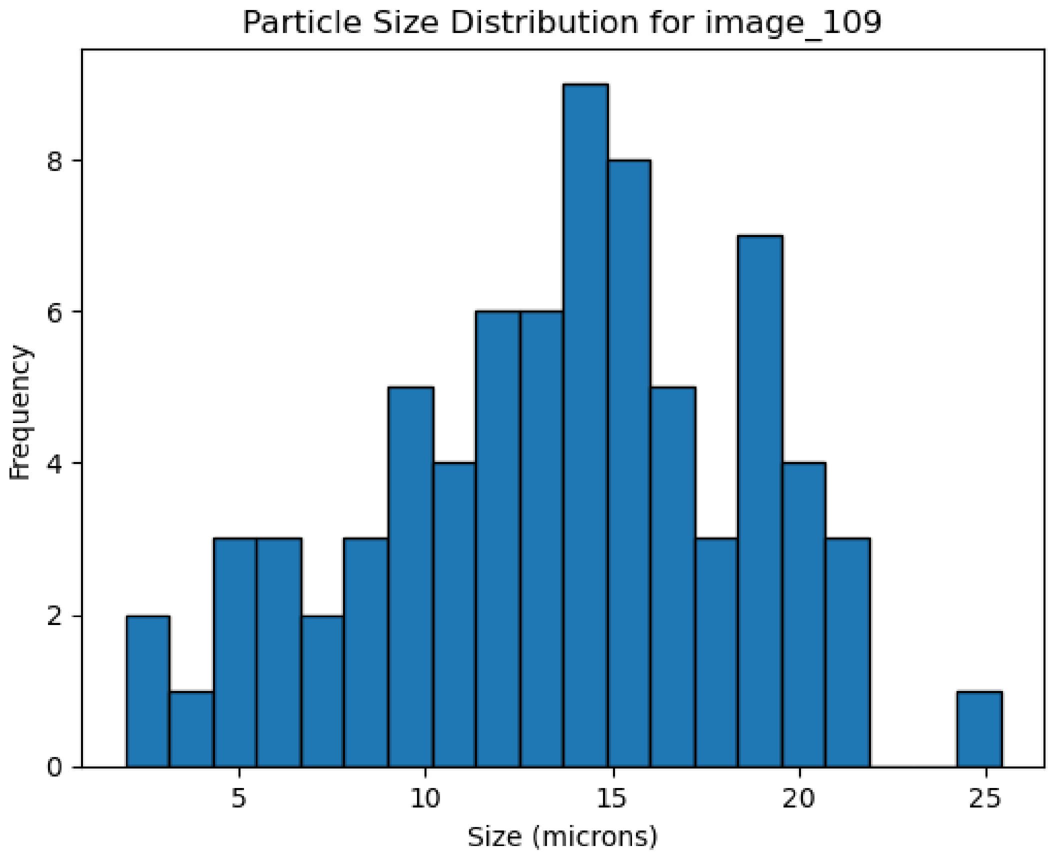

```
Particle Size Stats for image_106 - Mean: 7.628225015720425, Median:
6.358044427445026, Std Dev: 5.903396218027281
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



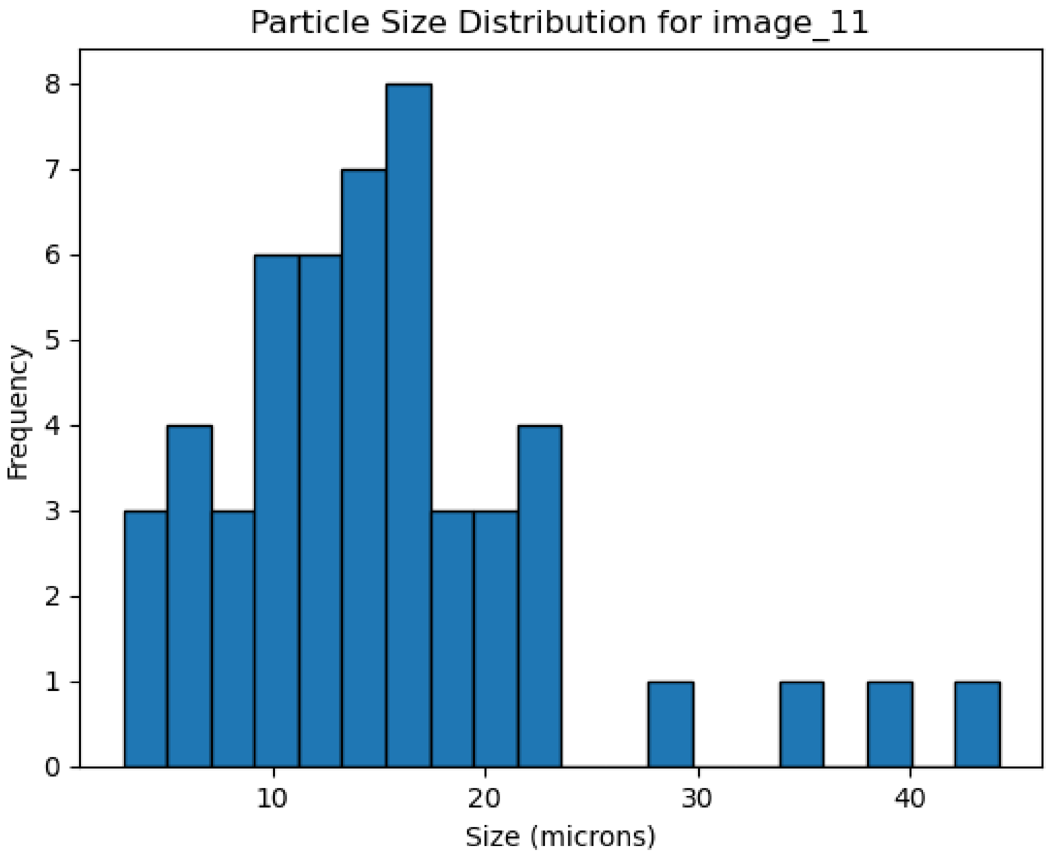

```
Particle Size Stats for image_107 - Mean: 13.788328580707493, Median:
13.110581167104948, Std Dev: 6.088005054332201
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



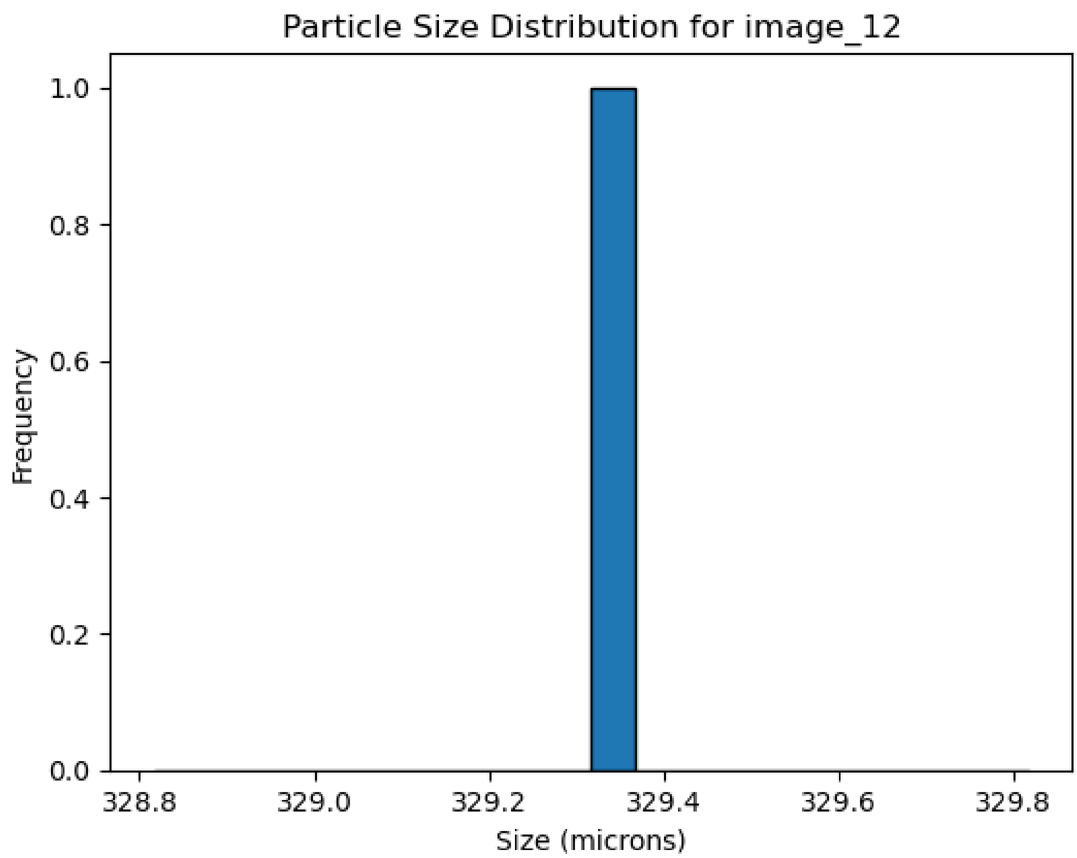

```
Particle Size Stats for image_108 - Mean: 12.589657158720849, Median:
5.046265044040321, Std Dev: 17.341554436669327
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



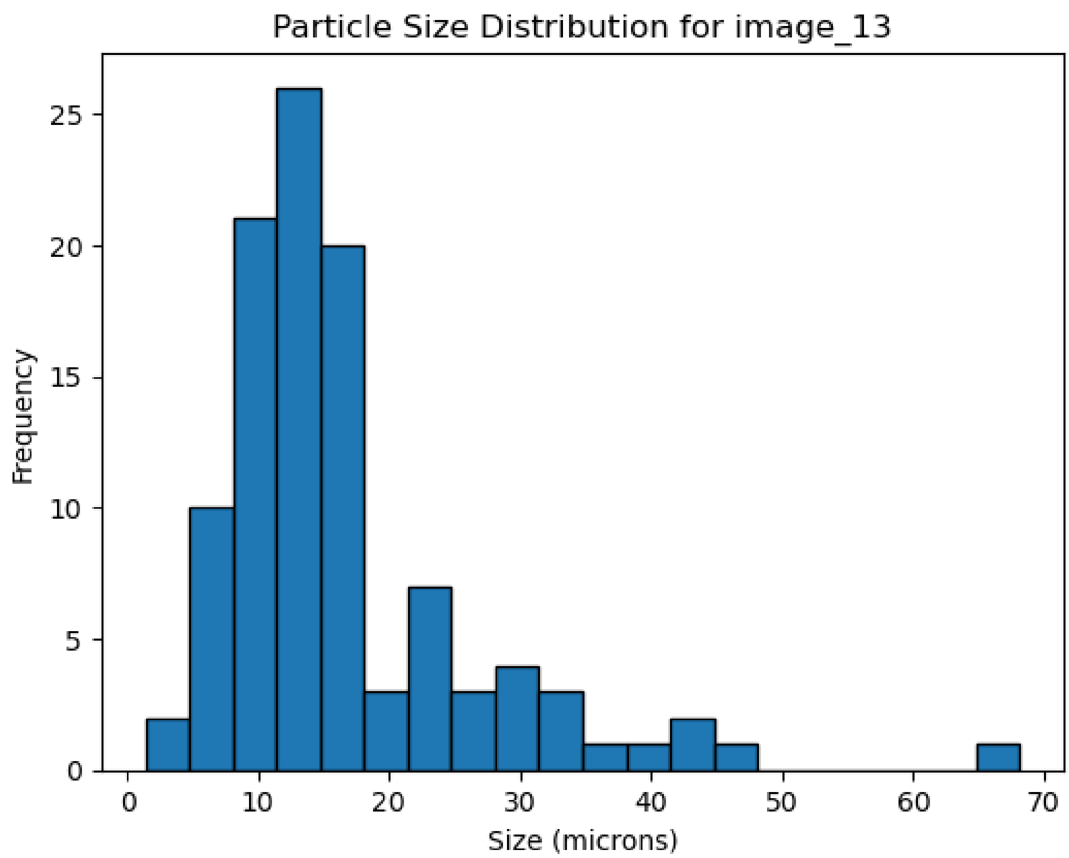

```
Particle Size Stats for image_109 - Mean: 13.3753229002125, Median:
13.888692920047486, Std Dev: 4.902005299358755
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



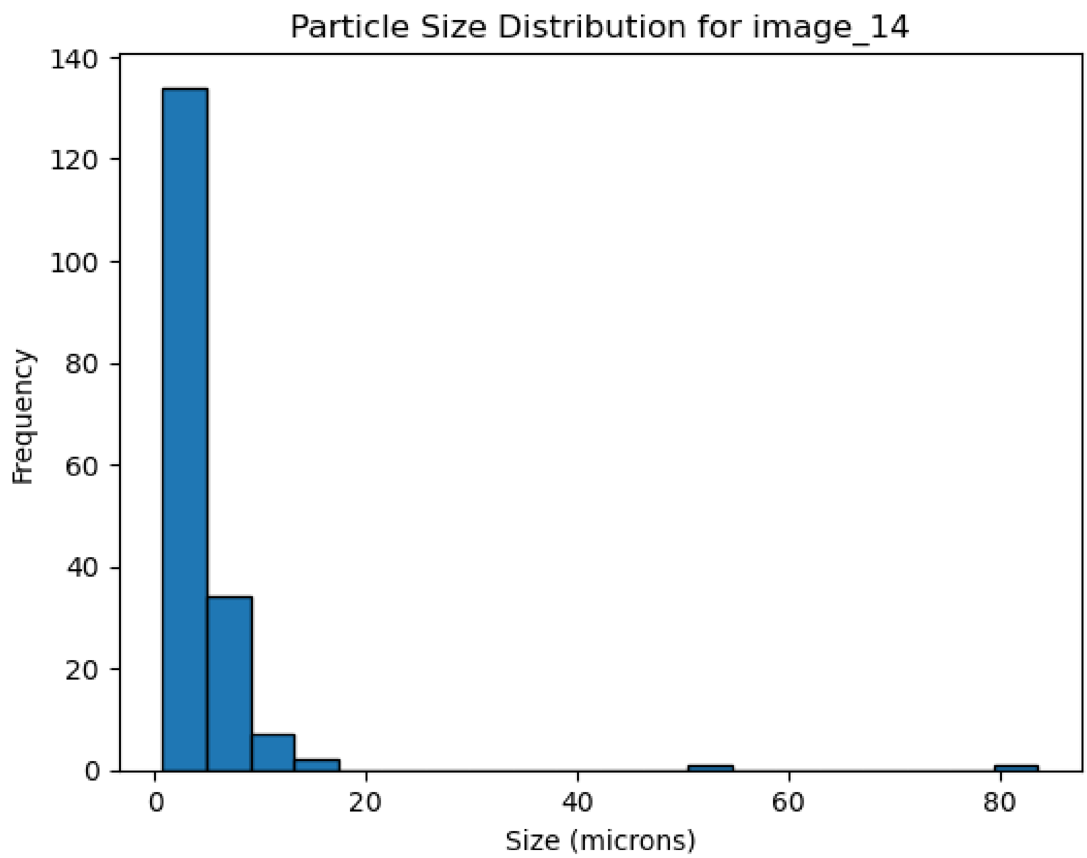

```
Particle Size Stats for image_11 - Mean: 15.488810799787663, Median:
14.60368527951557, Std Dev: 8.280714134183167
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



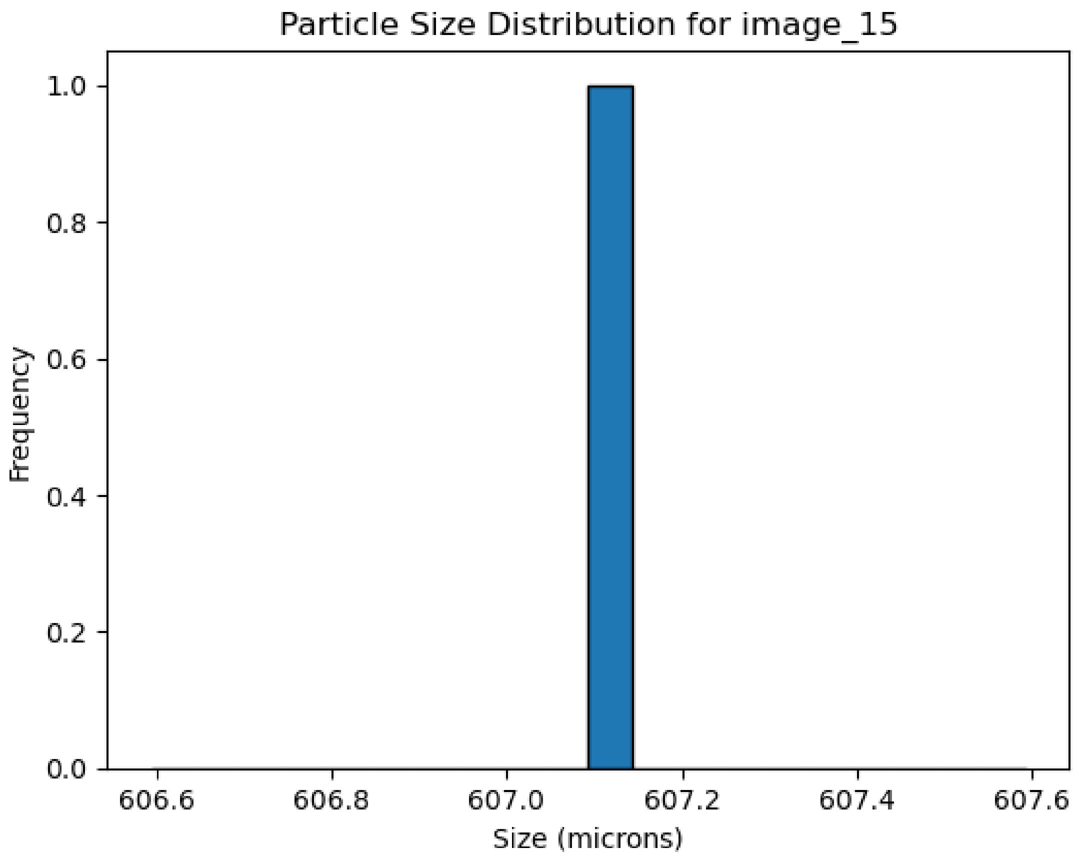

```
Particle Size Stats for image_12 - Mean: 329.31664316029065, Median:
329.31664316029065, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



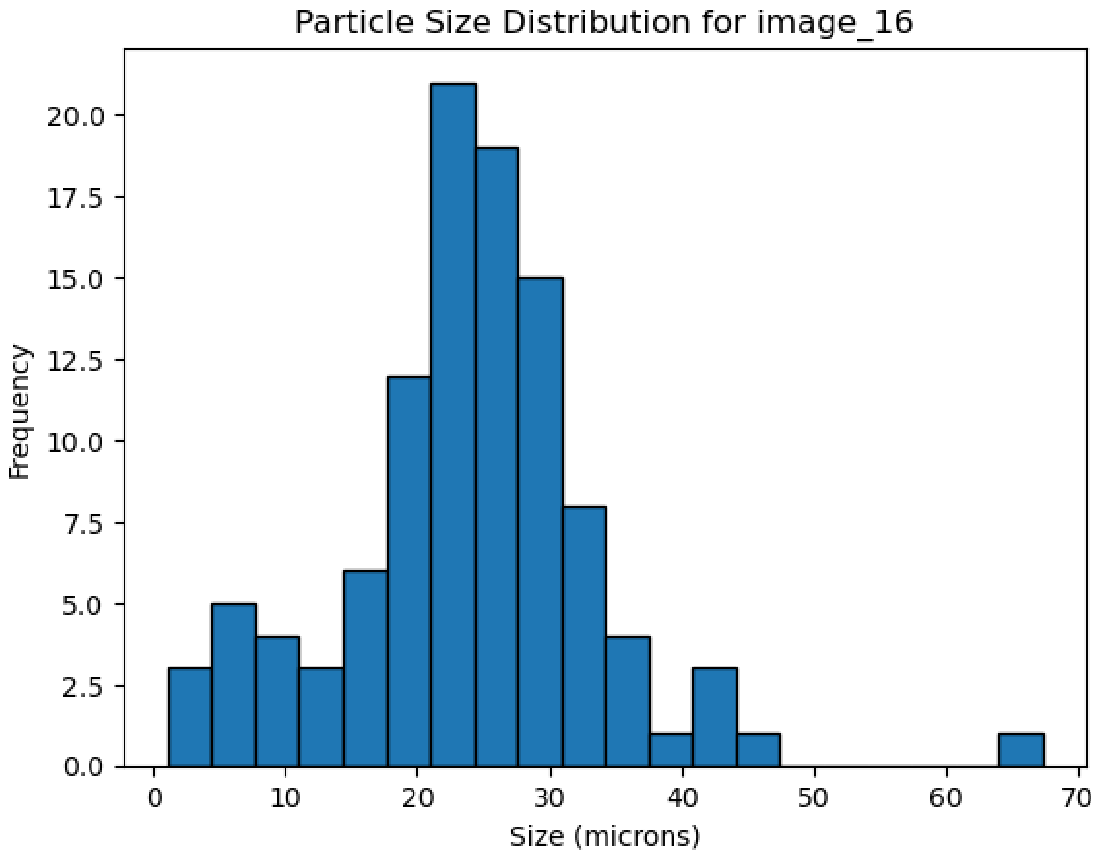

```
Particle Size Stats for image_13 - Mean: 16.66515415639093, Median:
14.1160188704554, Std Dev: 10.167770055927502
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



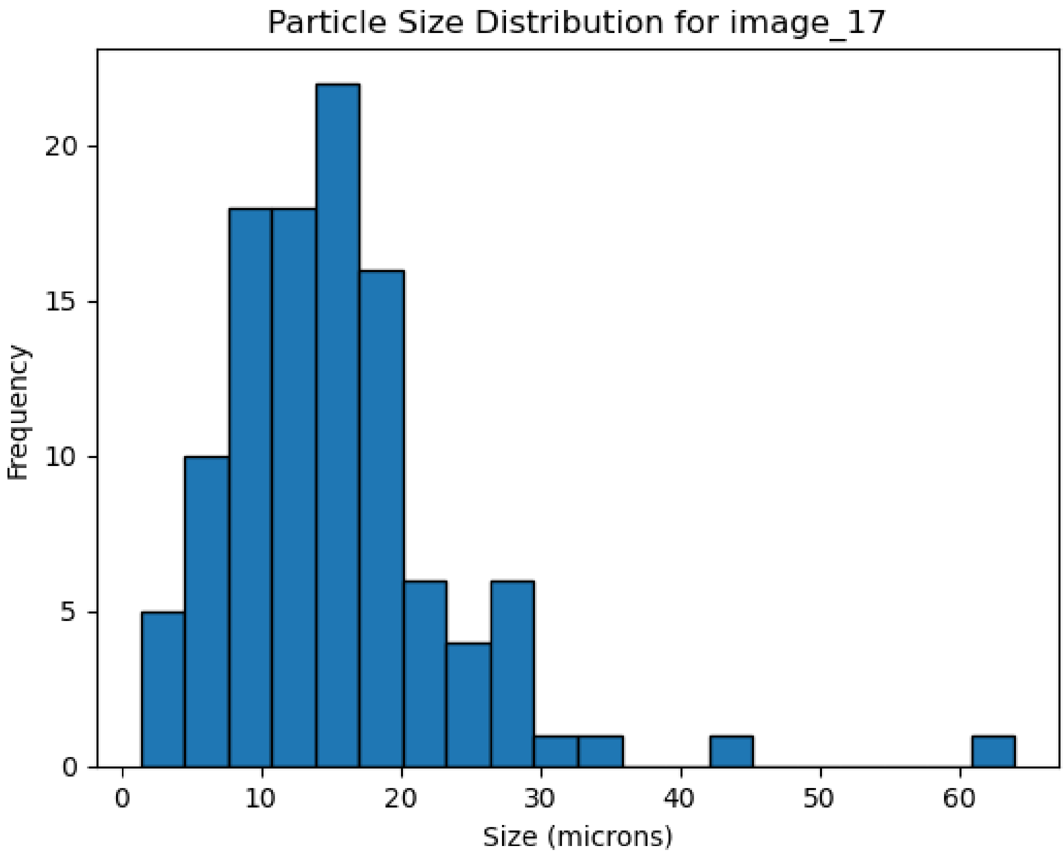

```
Particle Size Stats for image_14 - Mean: 4.738626200411483, Median:
3.5682482323055424, Std Dev: 7.303337592835731
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



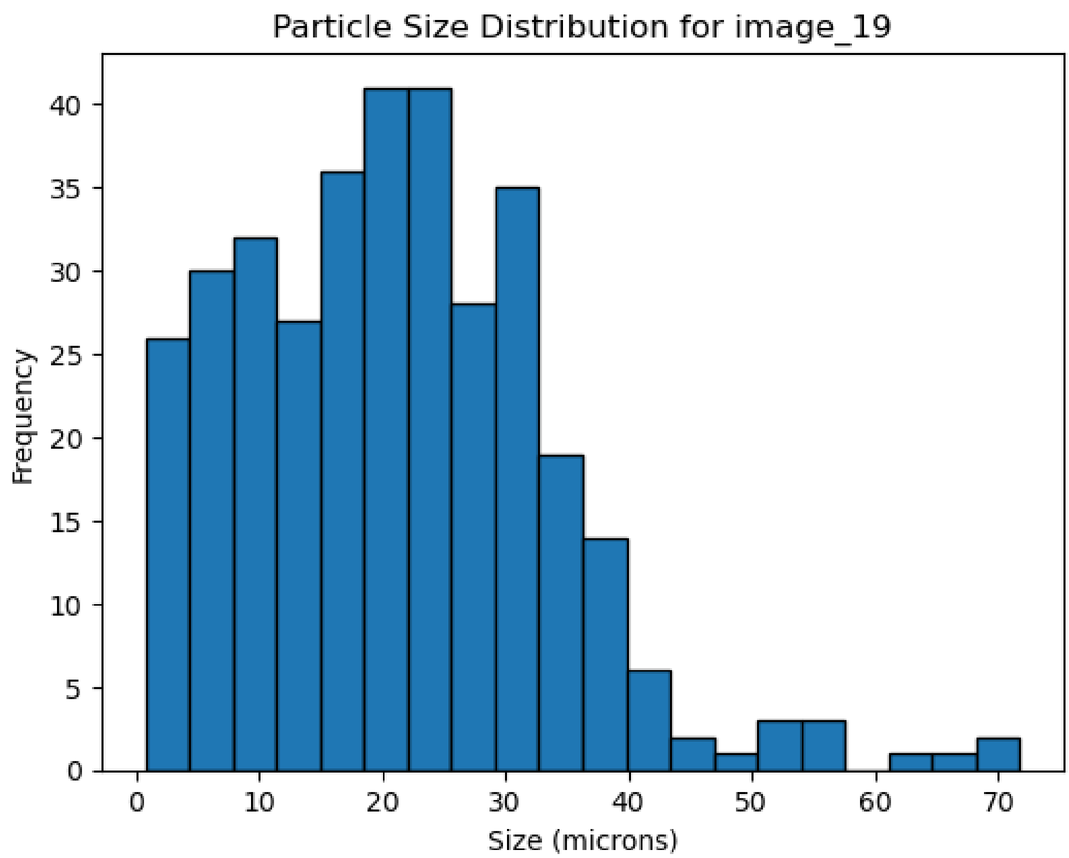

```
Particle Size Stats for image_15 - Mean: 607.0931596842432, Median:
607.0931596842432, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



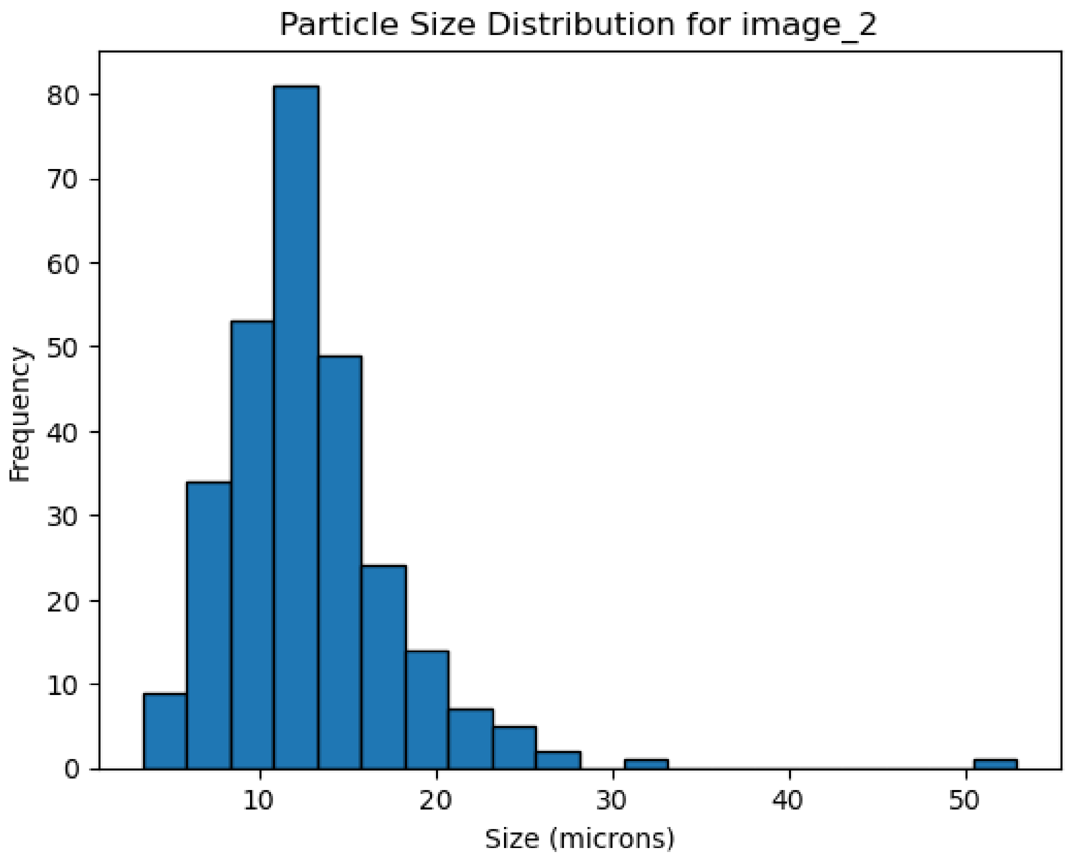

```
Particle Size Stats for image_16 - Mean: 23.943782379726045, Median:
24.194351504345313, Std Dev: 9.913379473112917
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



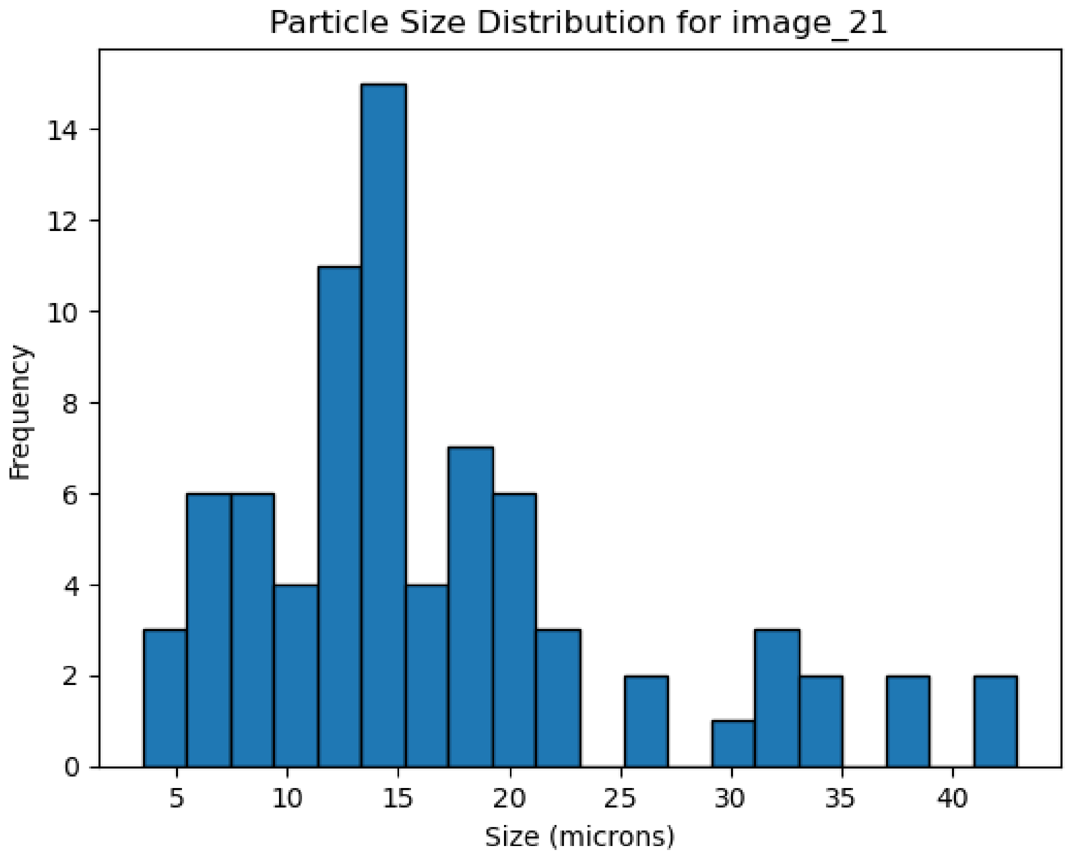

```
Particle Size Stats for image_17 - Mean: 15.318858063409655, Median:
14.339741506551949, Std Dev: 8.534144291221672
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



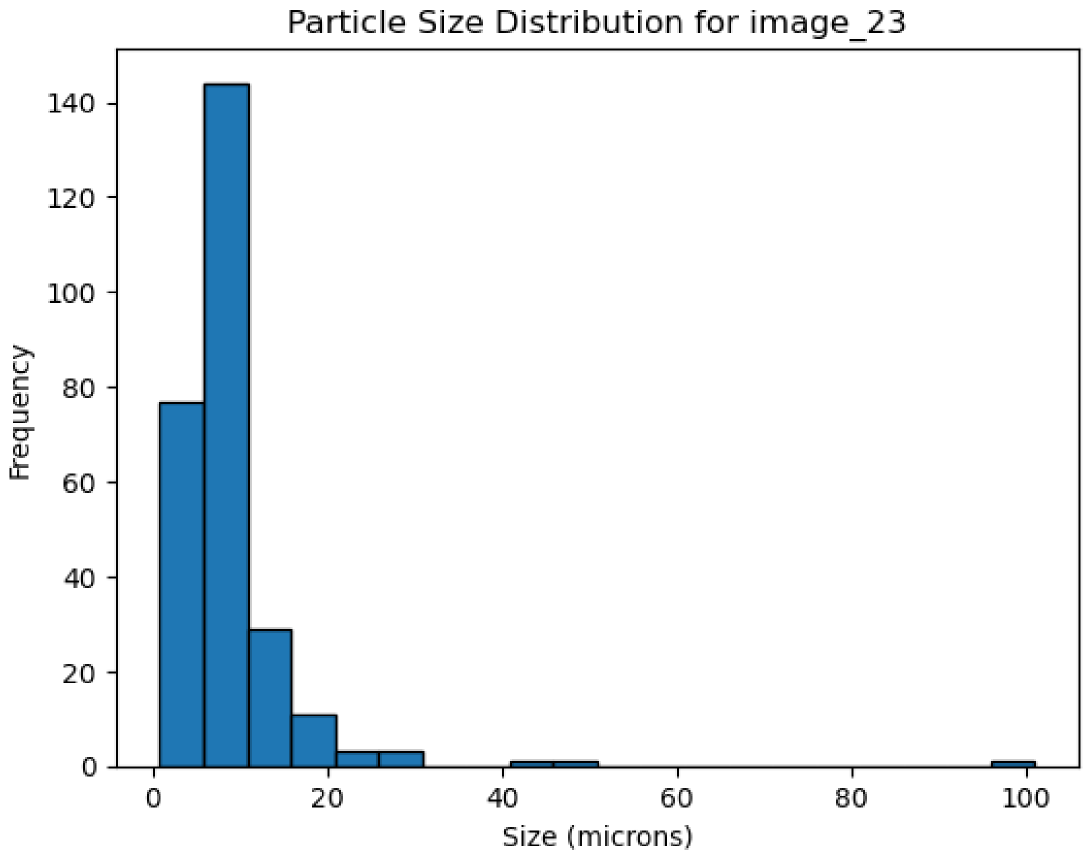

```
Particle Size Stats for image_19 - Mean: 21.082090760561748, Median:
20.528913240563227, Std Dev: 12.526504553136302
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



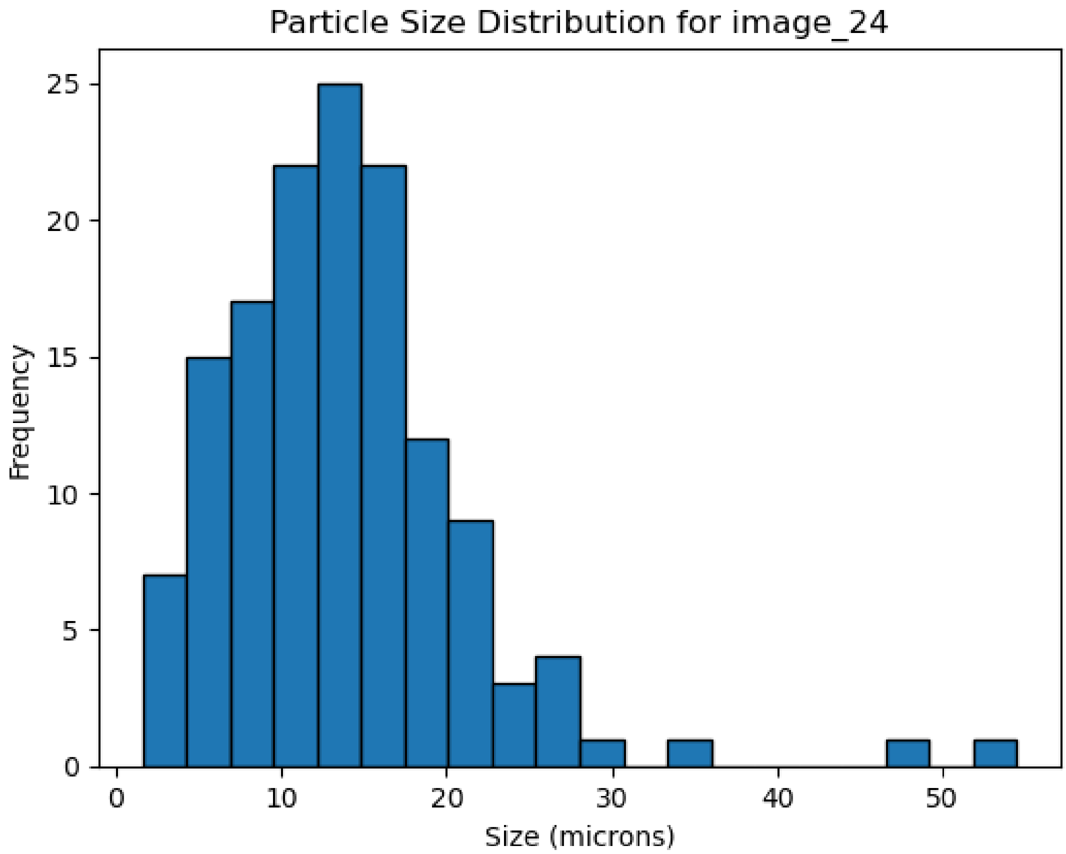

```
Particle Size Stats for image_2 - Mean: 12.780700866474952, Median:
12.060988510717074, Std Dev: 5.014522085852893
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



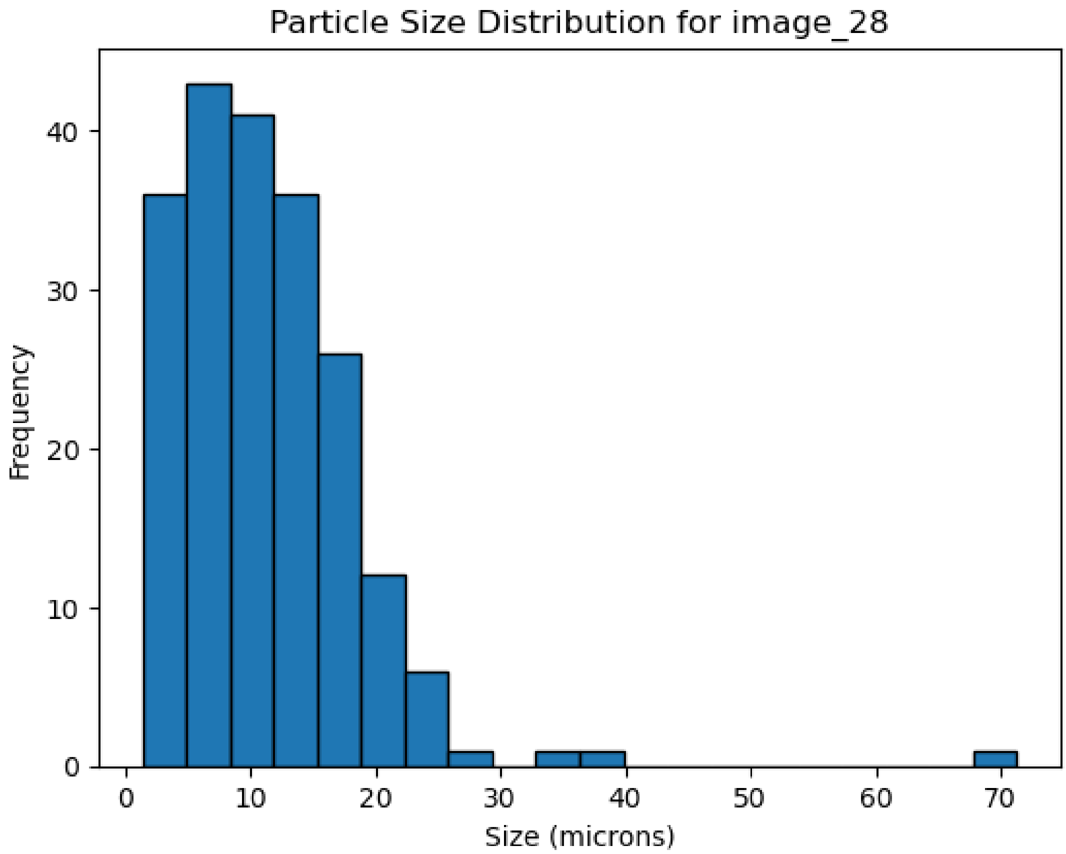

```
Particle Size Stats for image_21 - Mean: 16.729407338607608, Median:
14.38406847937898, Std Dev: 8.808692648626398
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



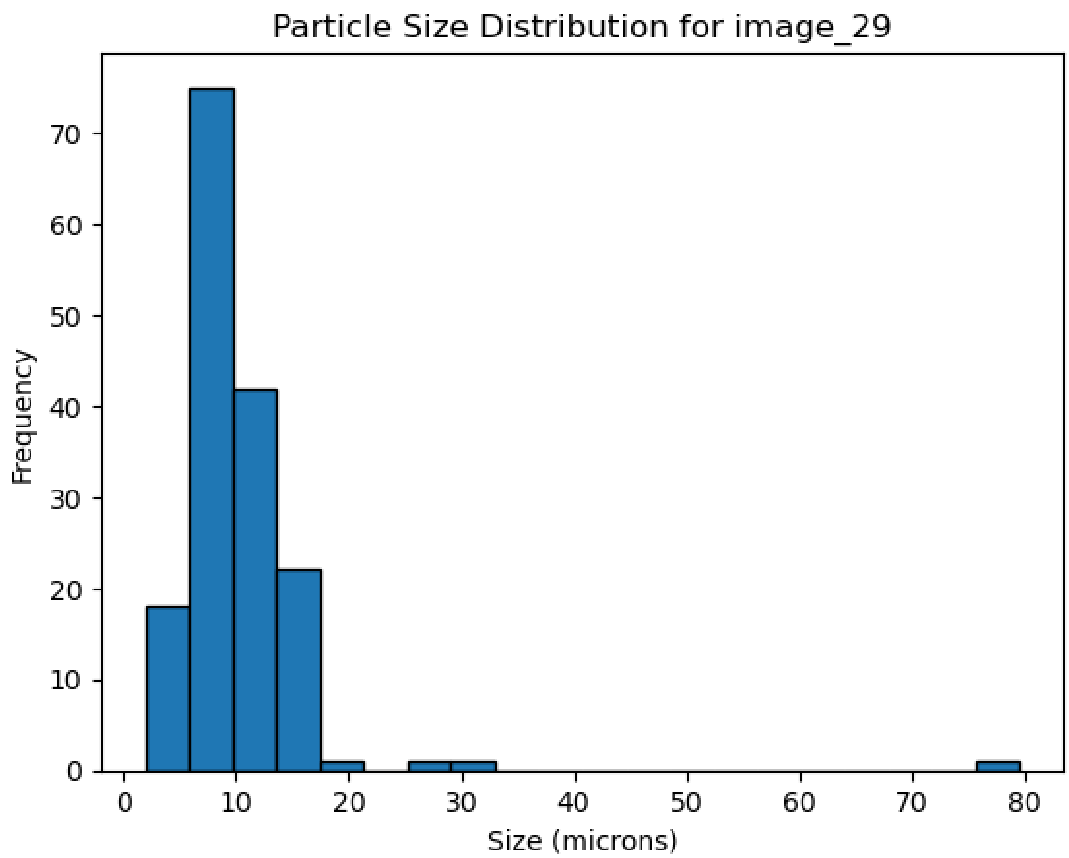

```
Particle Size Stats for image_23 - Mean: 8.850618027135797, Median:
7.673779954648447, Std Dev: 7.898297838209322
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



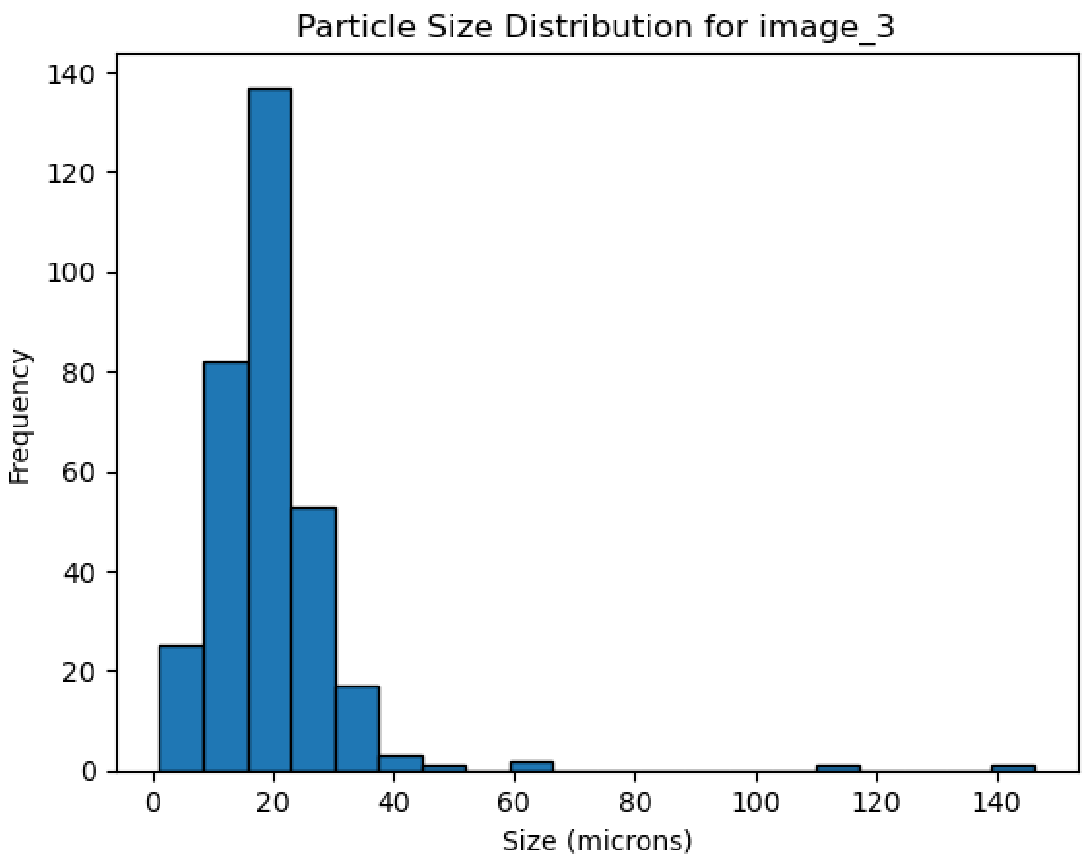

```
Particle Size Stats for image_24 - Mean: 13.817627210616957, Median:
12.85311937238519, Std Dev: 7.513925374022
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



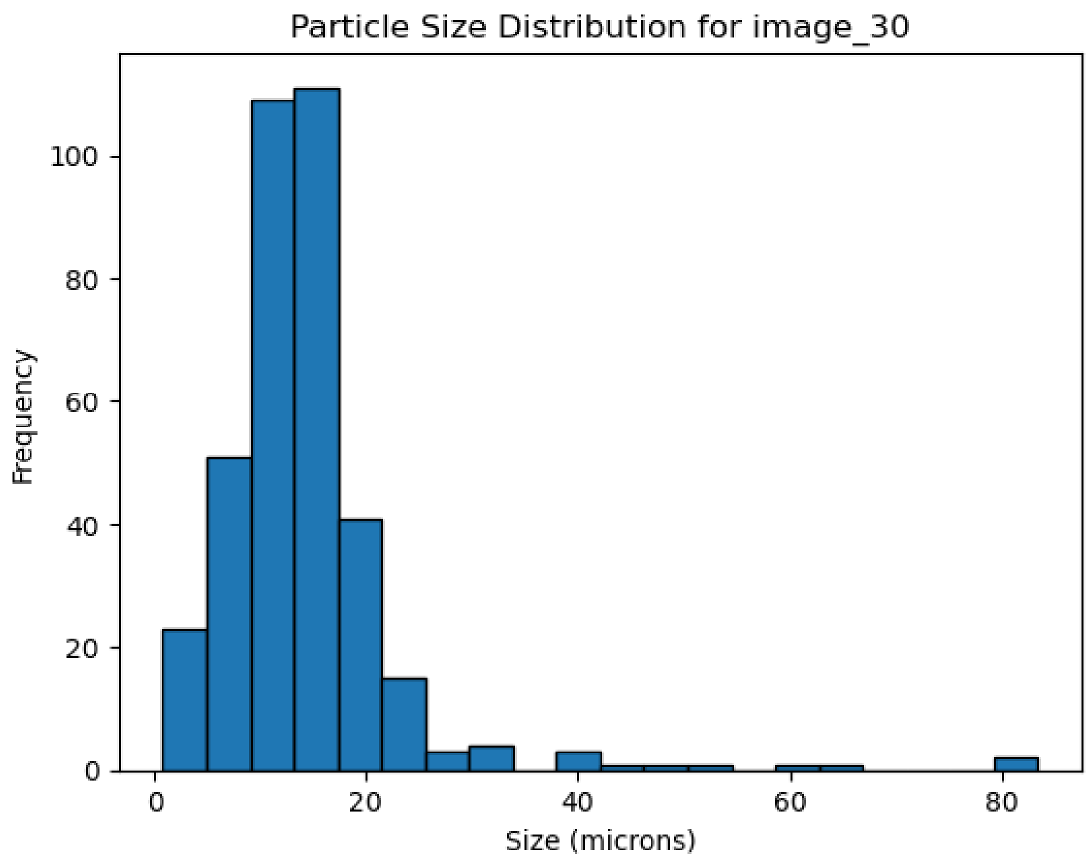

```
Particle Size Stats for image_28 - Mean: 11.33263710763989, Median:
10.124020118074668, Std Dev: 7.557226494882106
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



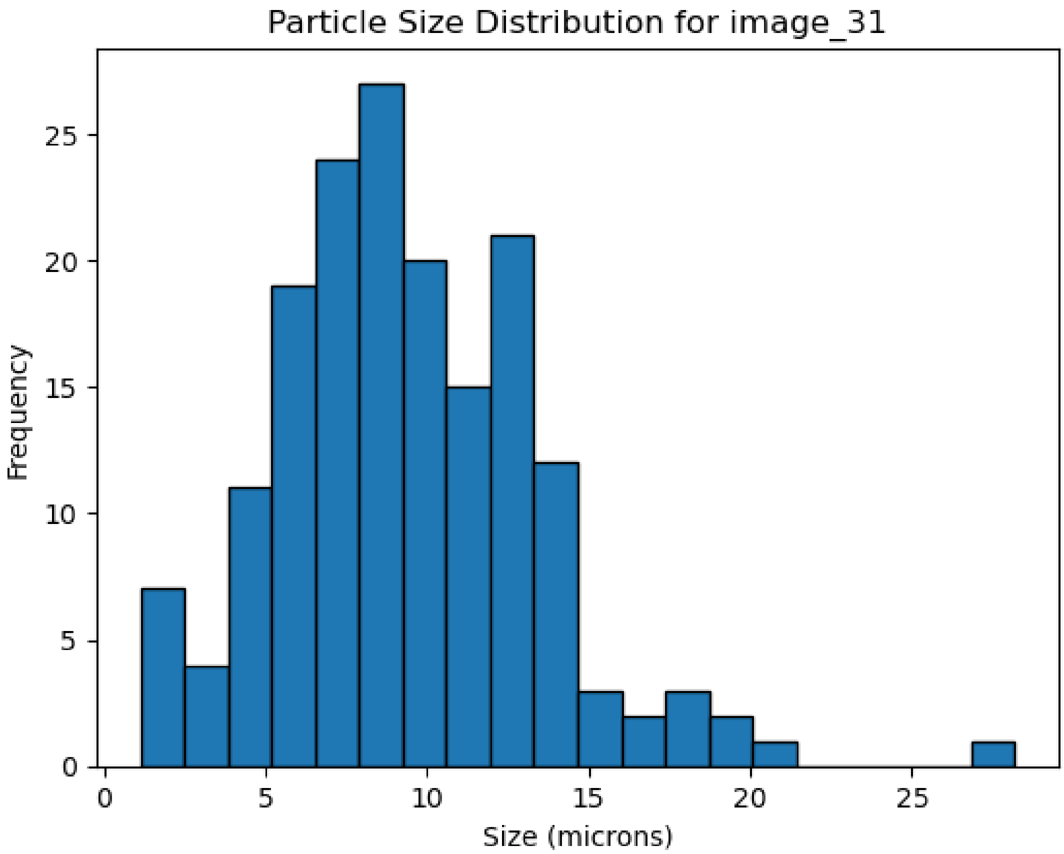

```
Particle Size Stats for image_29 - Mean: 10.105585920129105, Median:
9.09728368293446, Std Dev: 6.7566266436516695
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



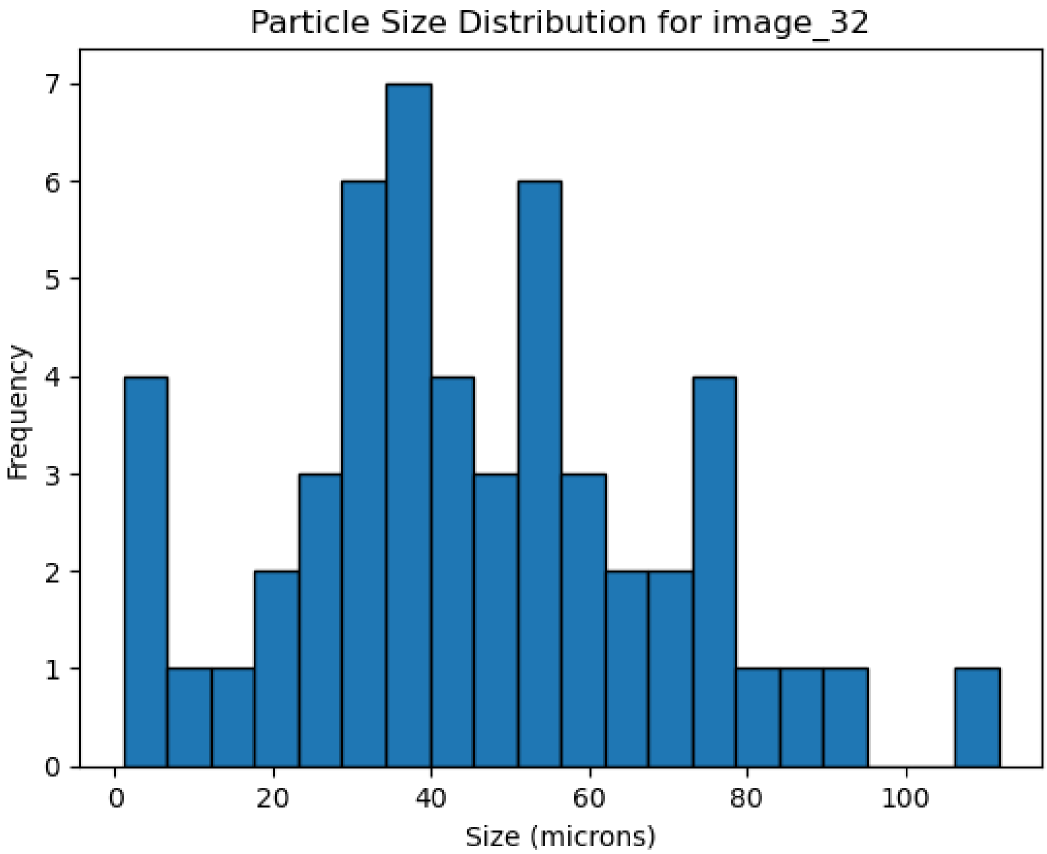

```
Particle Size Stats for image_3 - Mean: 19.649623416637585, Median:
18.864528956575565, Std Dev: 11.943141584032952
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



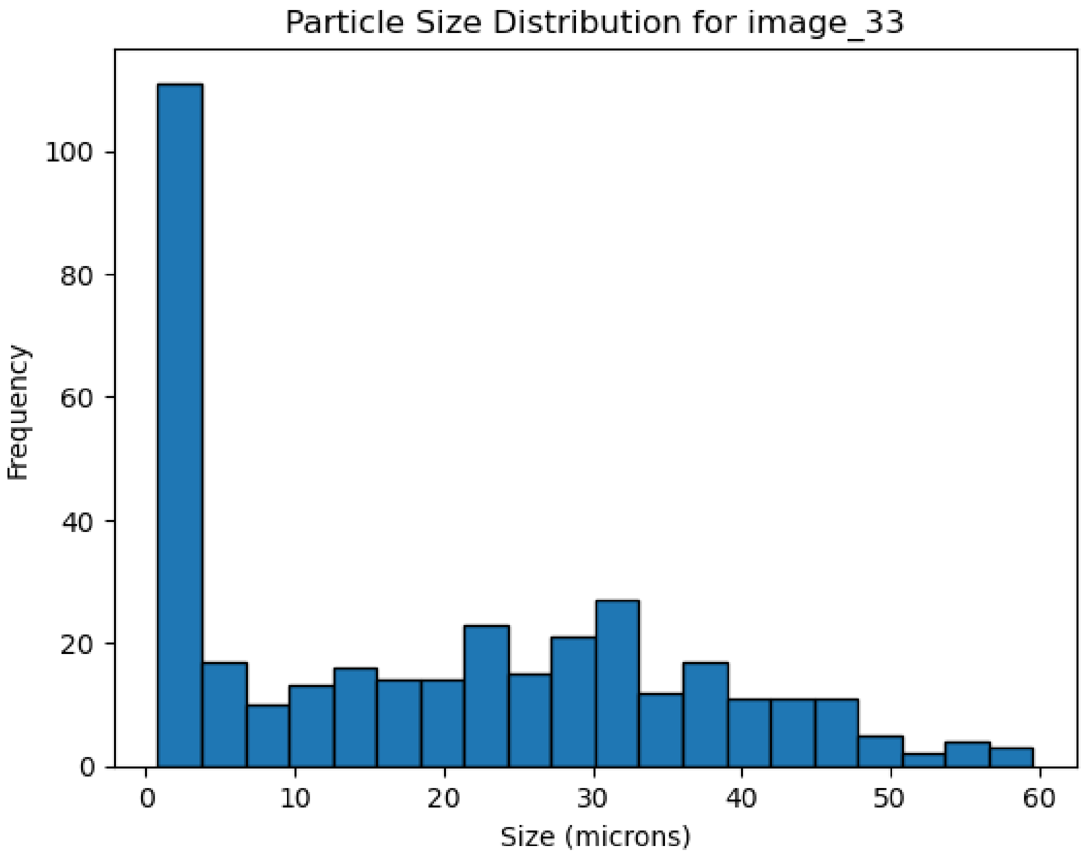

```
Particle Size Stats for image_30 - Mean: 14.328648297457764, Median:
13.231418571003069, Std Dev: 9.078896582123013
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



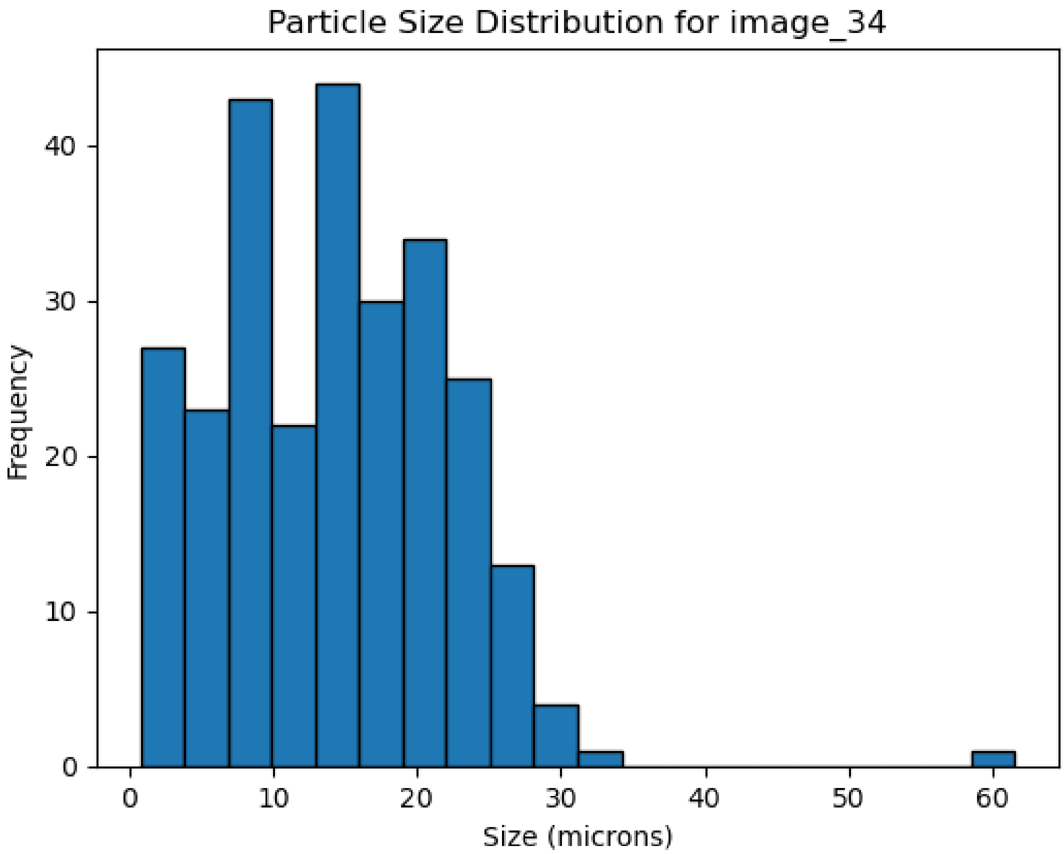

```
Particle Size Stats for image_31 - Mean: 9.322373365995338, Median:
8.884866446580956, Std Dev: 4.029039650341205
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



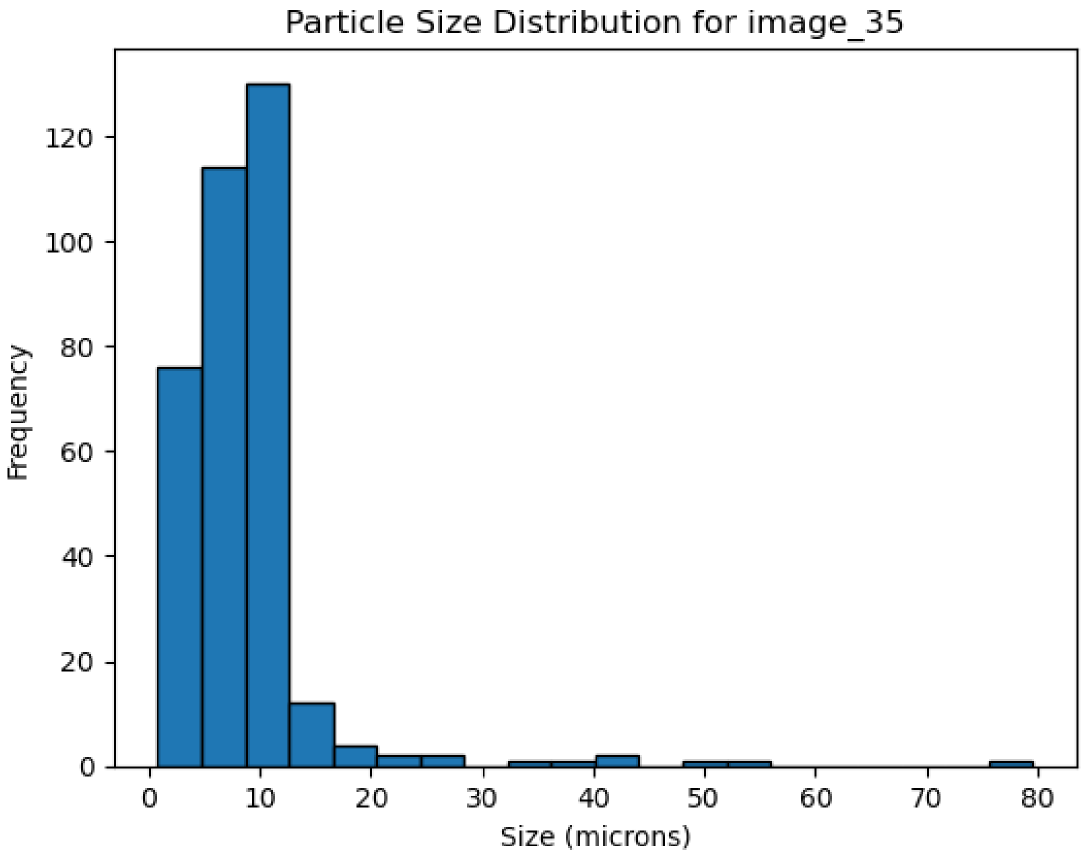

```
Particle Size Stats for image_32 - Mean: 45.183670577247696, Median:
43.587334207041266, Std Dev: 23.765137210113036
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



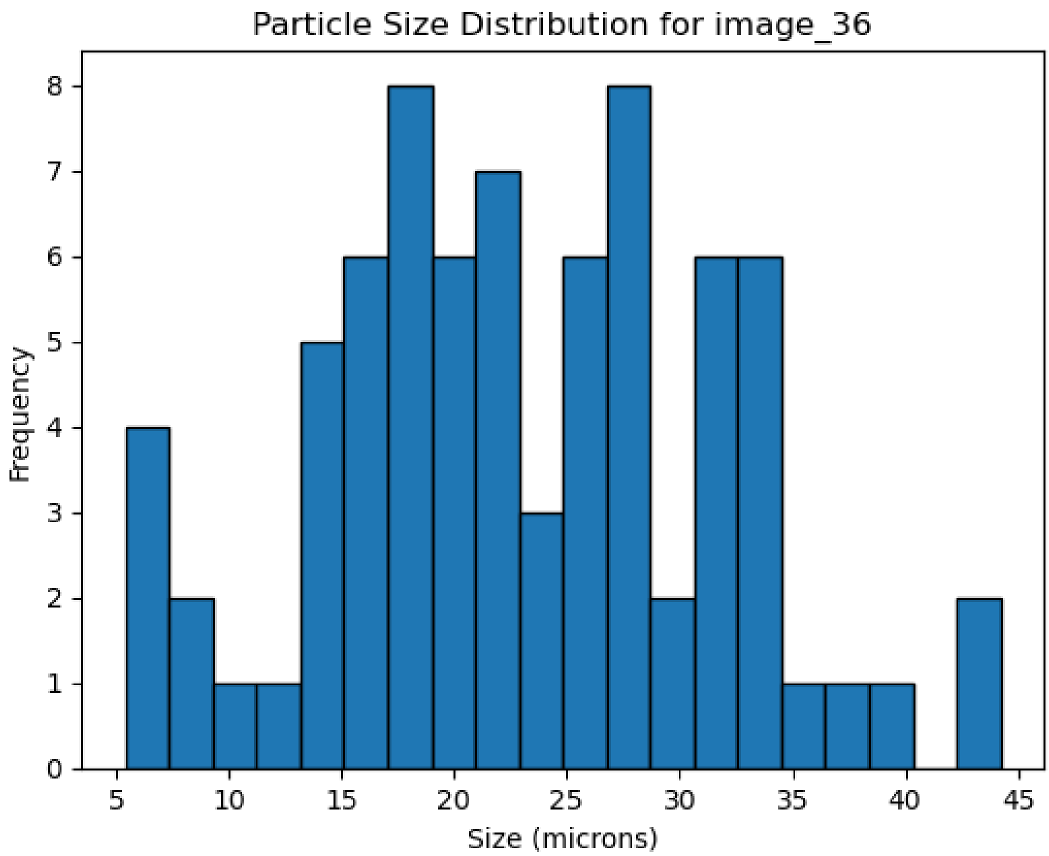

```
Particle Size Stats for image_33 - Mean: 19.11041742809208, Median:
17.89468483938439, Std Dev: 16.008718560369708
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



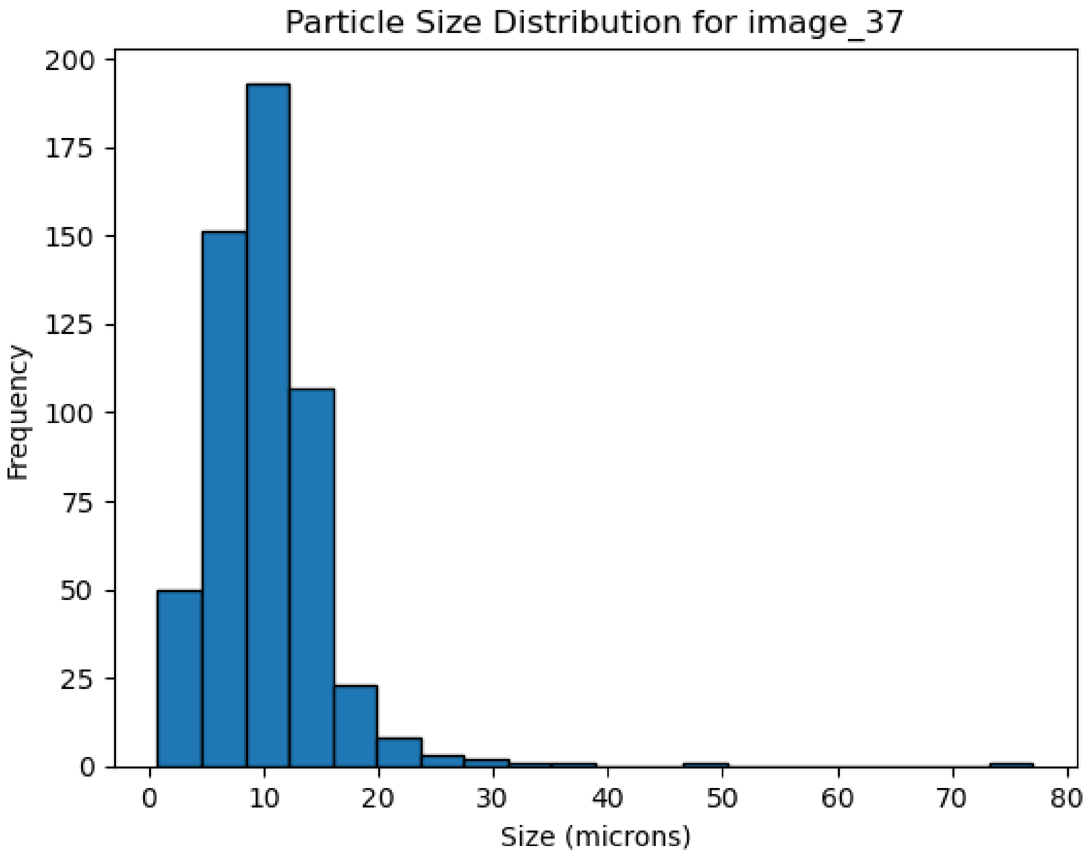

```
Particle Size Stats for image_34 - Mean: 14.33577027200145, Median: 
14.1160188704554, Std Dev: 7.899221718094443
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be 
compromised.
```



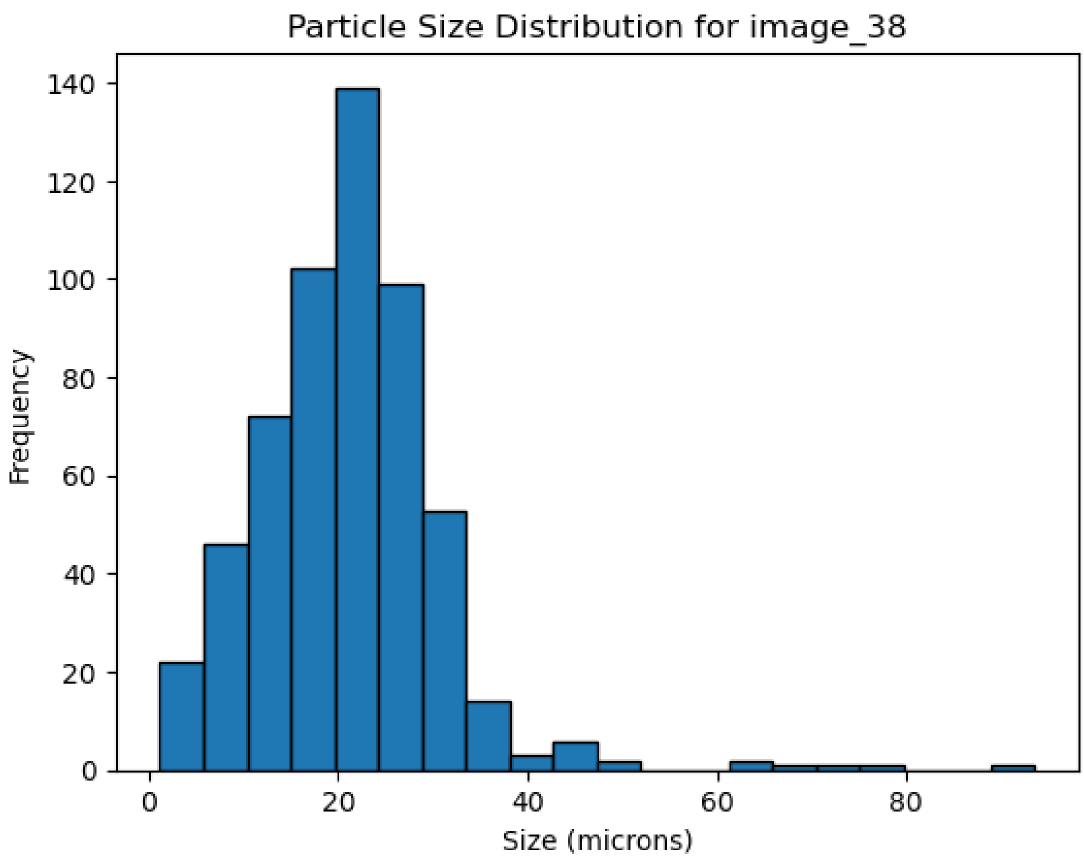

```
Particle Size Stats for image_35 - Mean: 8.503368176911932, Median:
8.018640596081465, Std Dev: 7.187822929171411
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



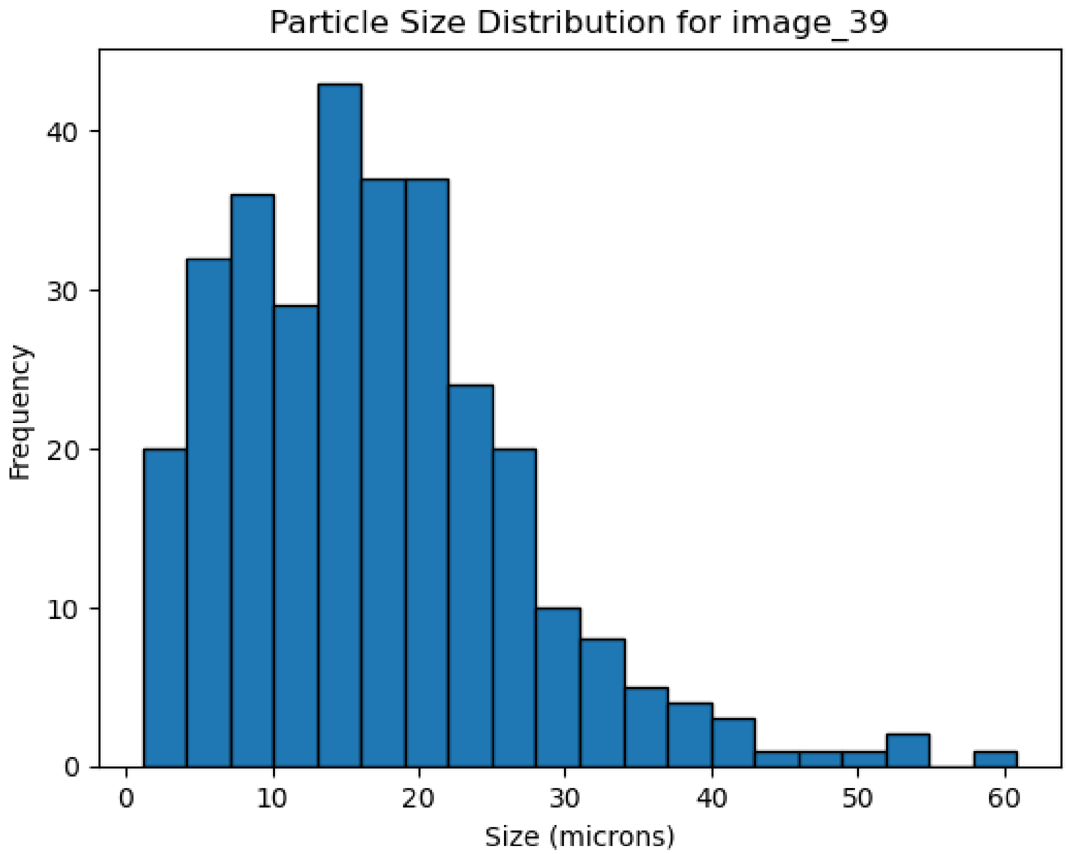

```
Particle Size Stats for image_36 - Mean: 23.045049166392165, Median:
22.38347070615472, Std Dev: 8.570089978366589
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



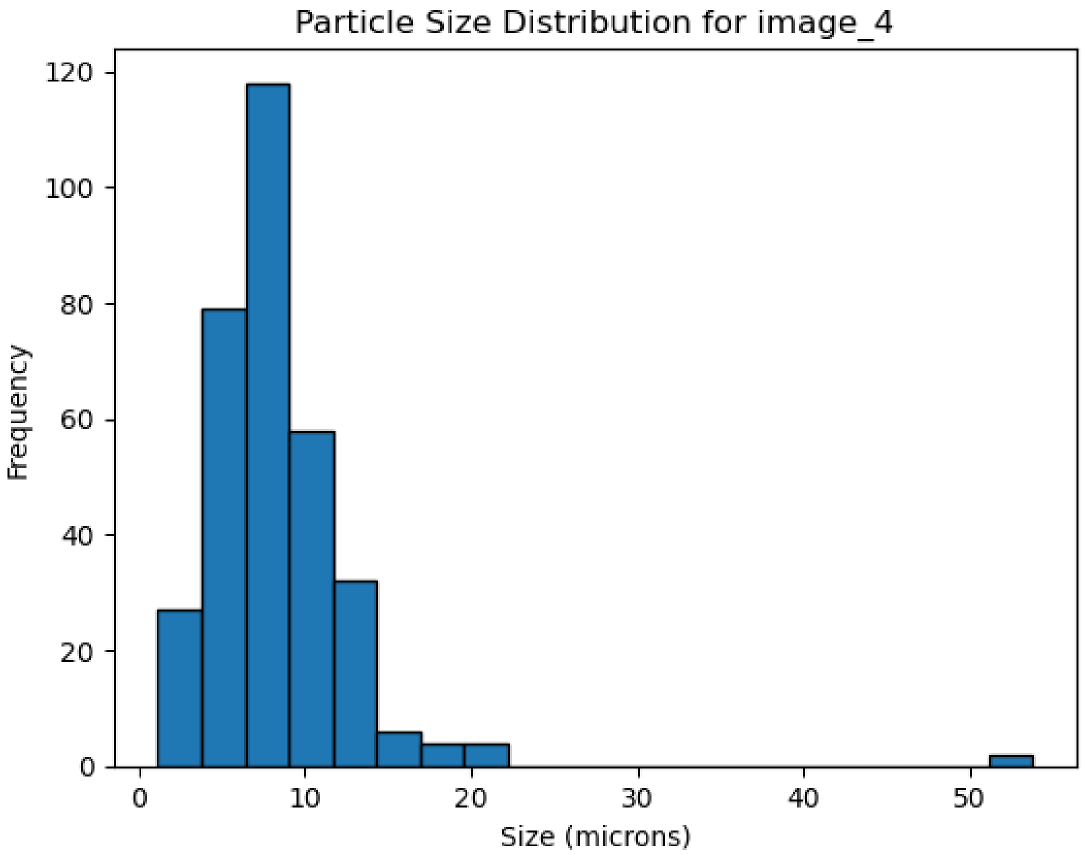

```
Particle Size Stats for image_37 - Mean: 10.261920769748393, Median:
10.060941497019325, Std Dev: 5.709177653525803
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



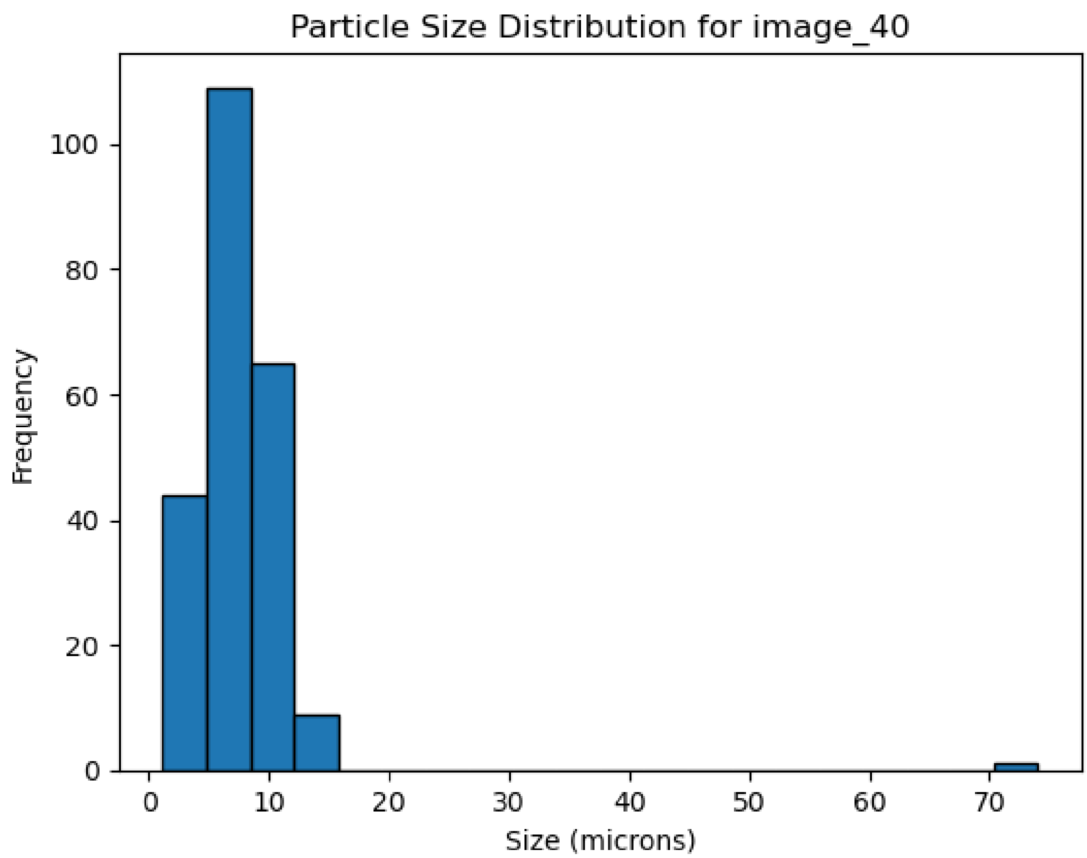

```
Particle Size Stats for image_38 - Mean: 21.066432324894066, Median:
20.867383439632196, Std Dev: 9.8939295678287
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



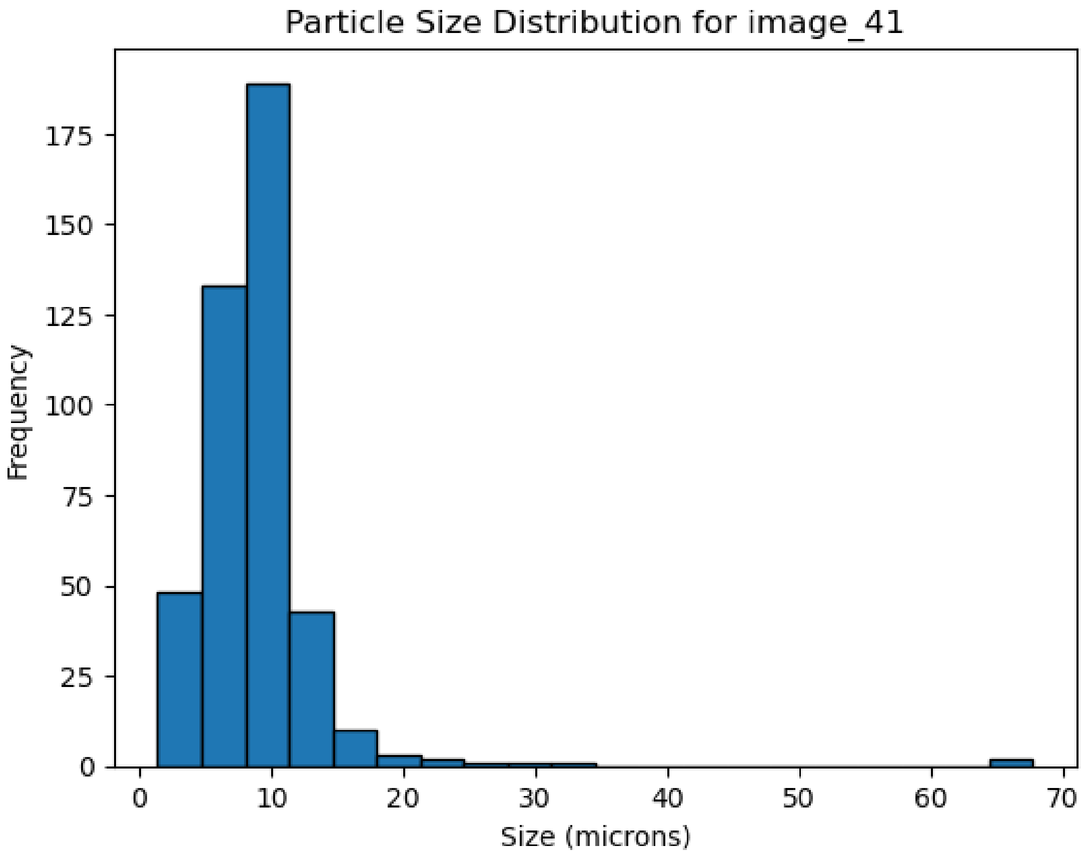

```
Particle Size Stats for image_39 - Mean: 16.95226751251302, Median:
15.827475174162174, Std Dev: 9.964543250537885
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



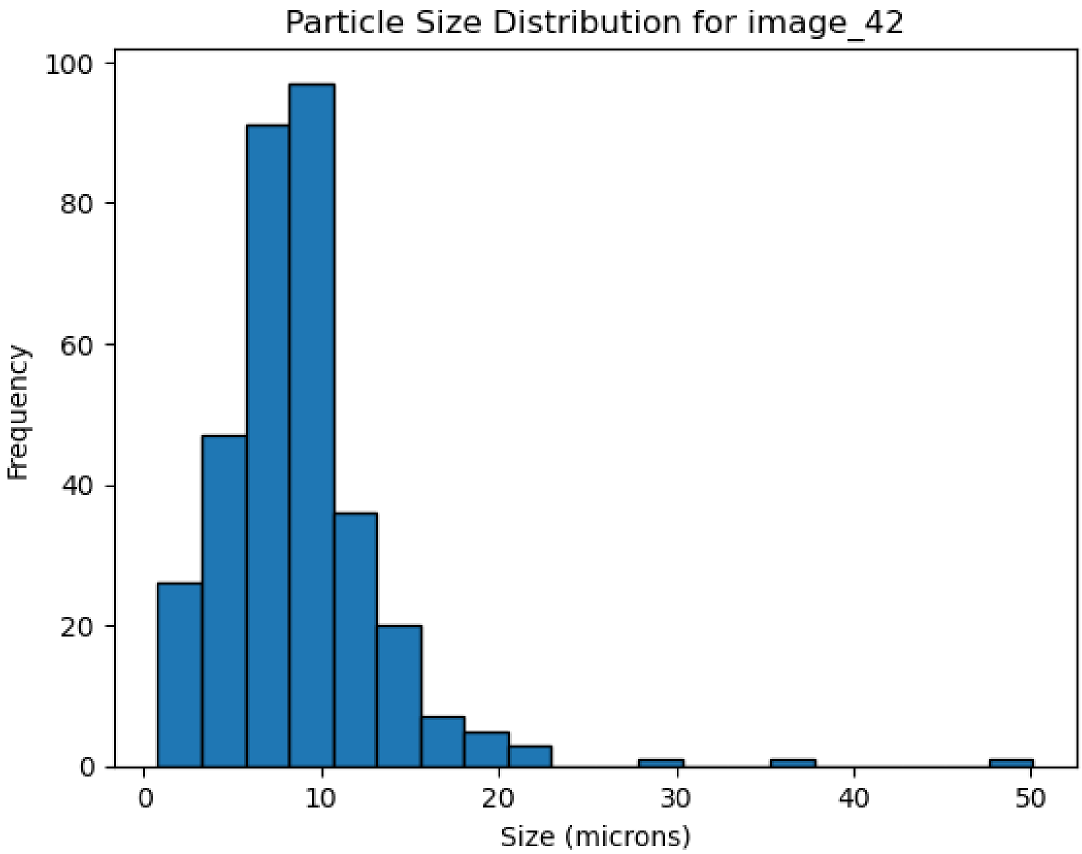

```
Particle Size Stats for image_4 - Mean: 8.3366099091851, Median:
7.611333607551958, Std Dev: 4.915405150455609
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



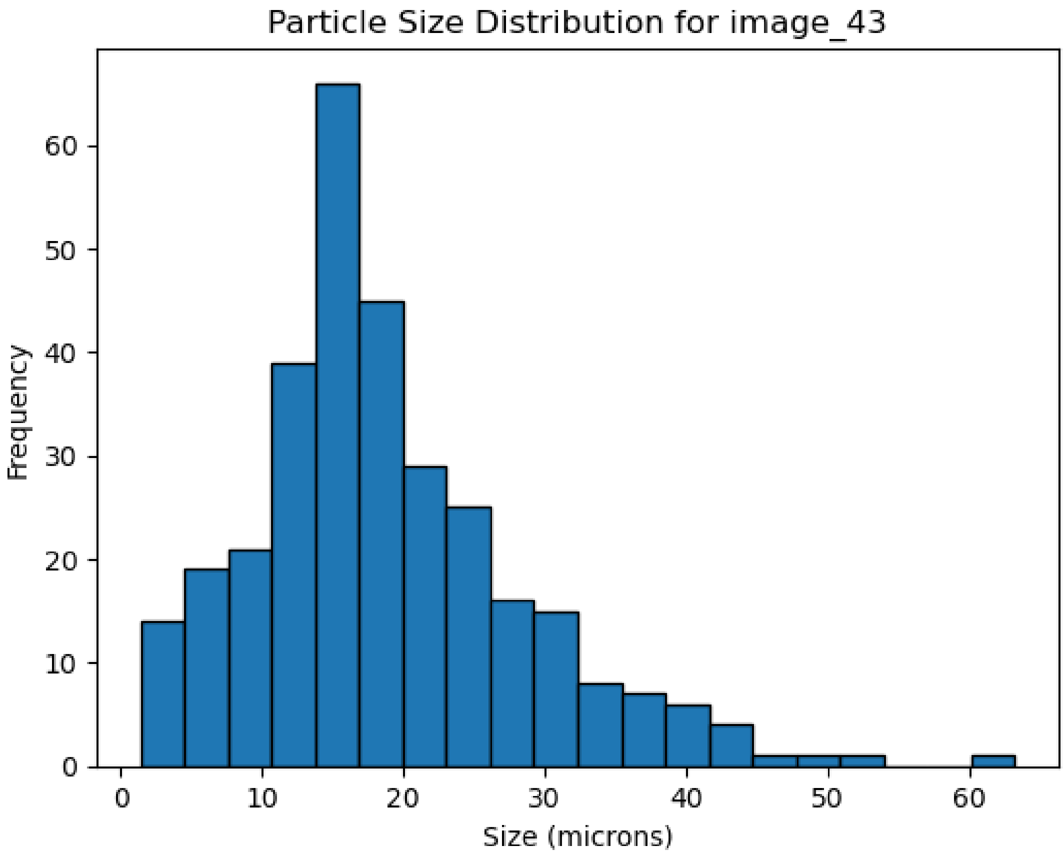

```
Particle Size Stats for image_40 - Mean: 7.459214440925882, Median:
7.136496464611085, Std Dev: 5.25923790914554
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



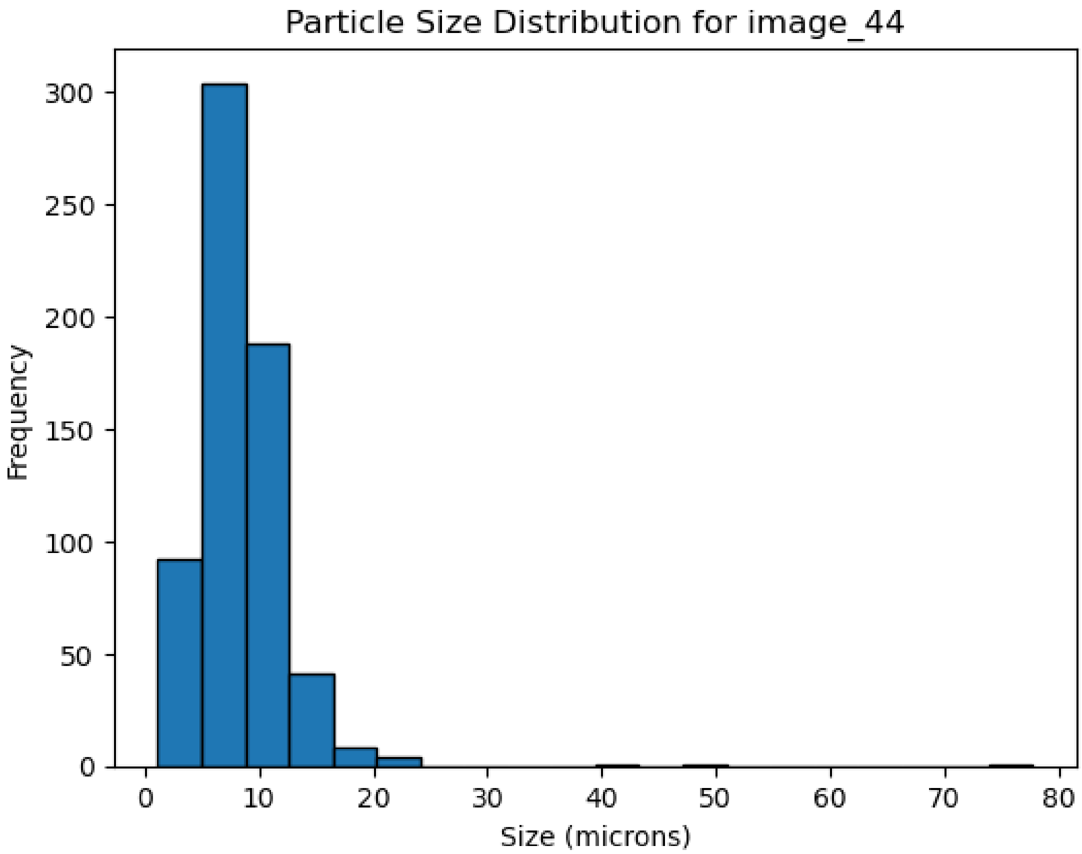

```
Particle Size Stats for image_41 - Mean: 8.90904912325856, Median:
8.667244841319219, Std Dev: 5.356398685833067
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



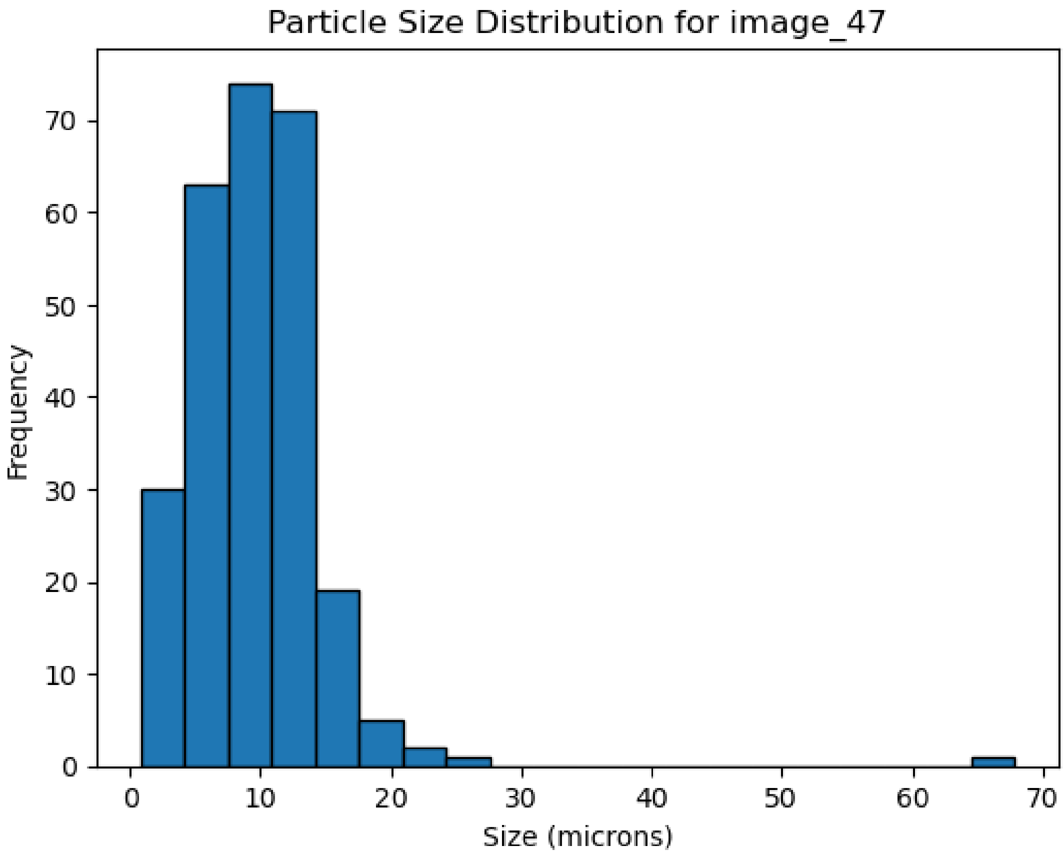

```
Particle Size Stats for image_42 - Mean: 8.769693928335725, Median:
8.481629223064205, Std Dev: 4.744435957706066
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



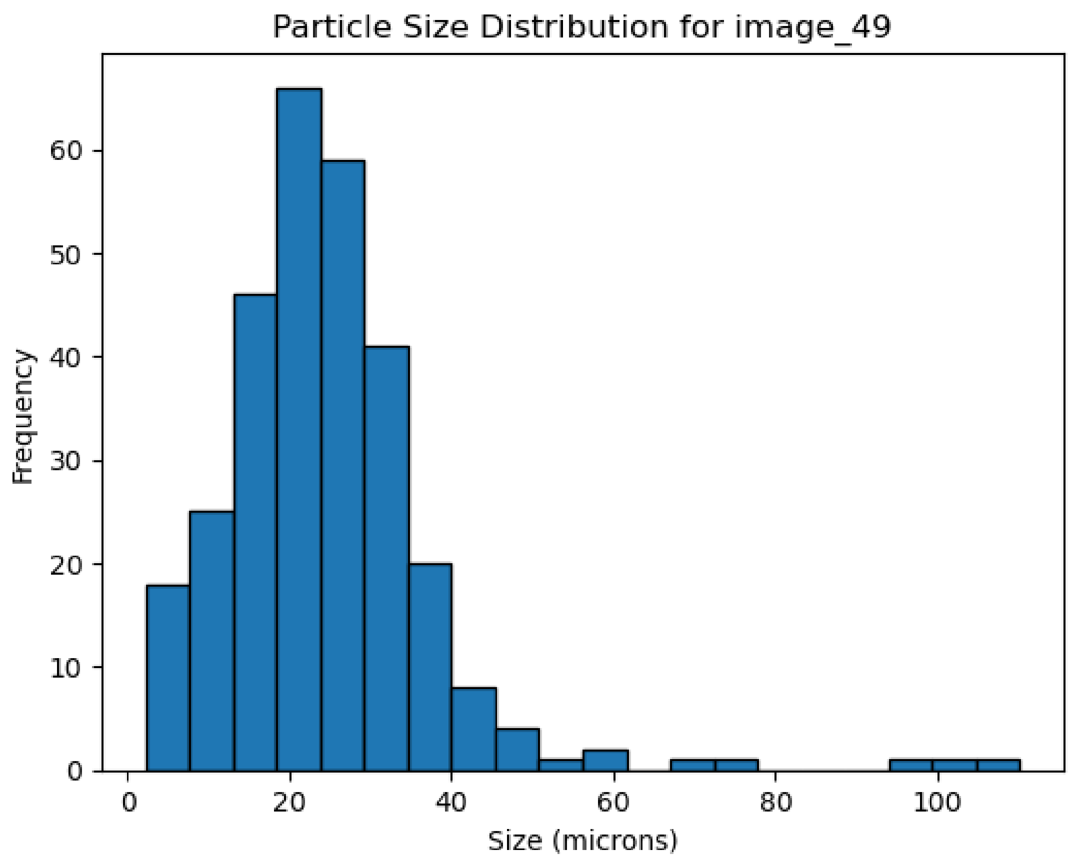

```
Particle Size Stats for image_43 - Mean: 18.70228241329626, Median:
16.85960740296317, Std Dev: 9.627012944144095
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



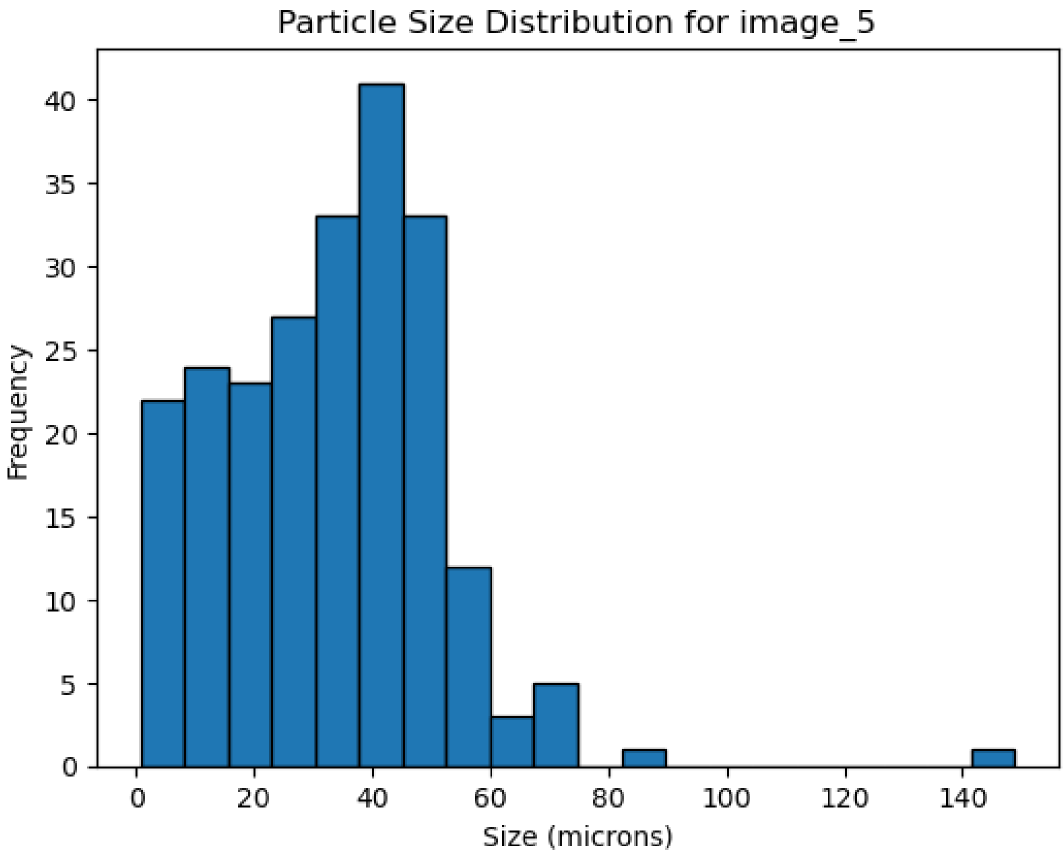

```
Particle Size Stats for image_44 - Mean: 8.342205843225017, Median:
7.898654169668588, Std Dev: 4.746311983977216
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



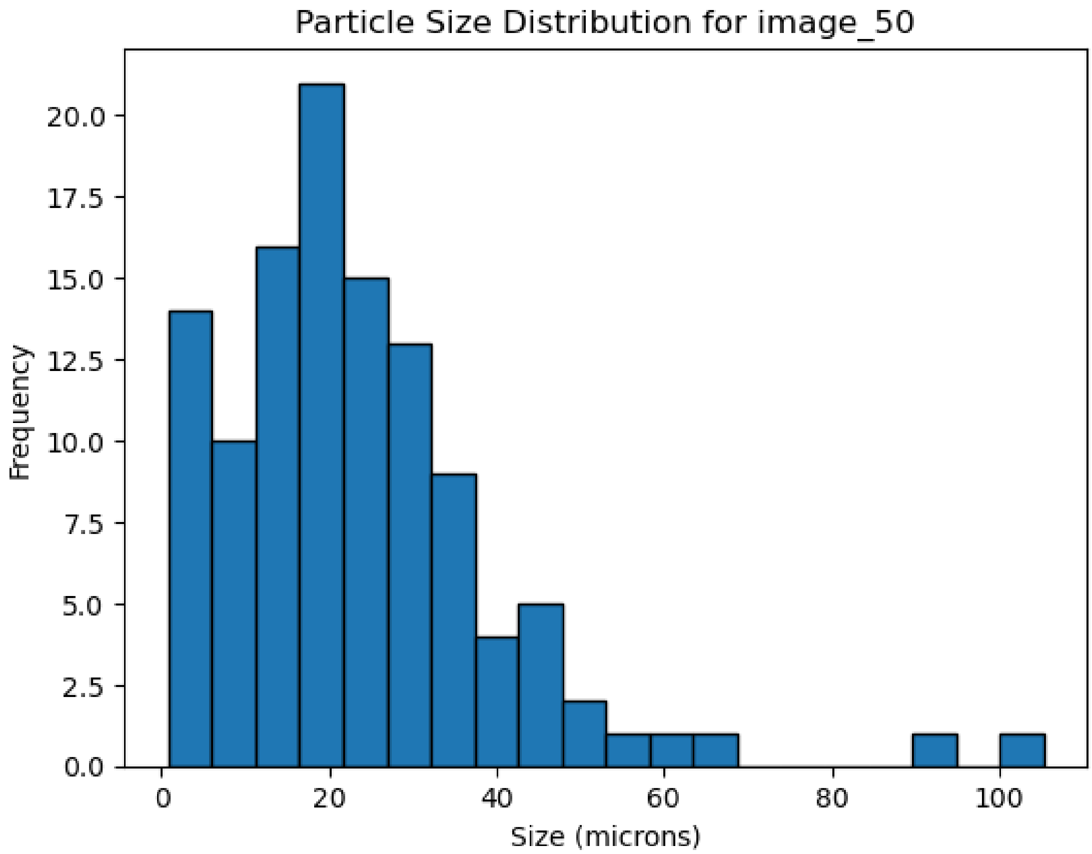

```
Particle Size Stats for image_47 - Mean: 9.535691391786715, Median:
9.338999347593866, Std Dev: 5.492028008457396
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



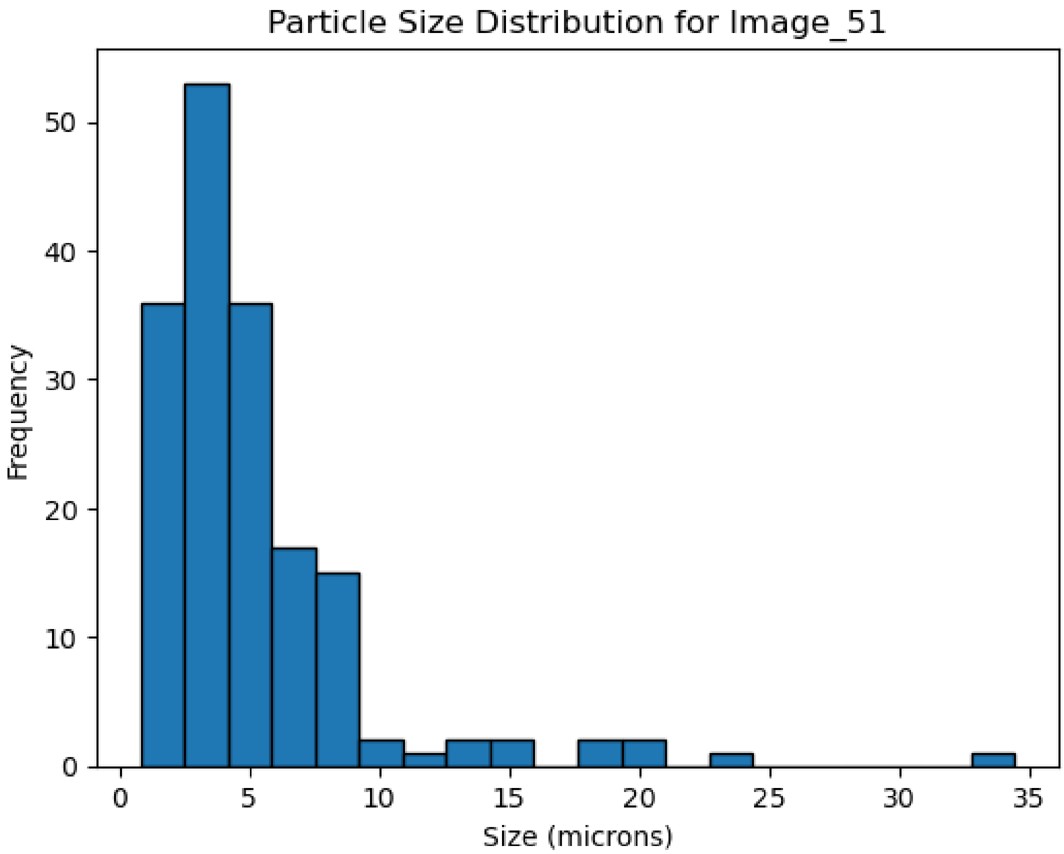

```
Particle Size Stats for image_49 - Mean: 24.626519629122814, Median:
23.330450175131254, Std Dev: 13.311506544186054
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



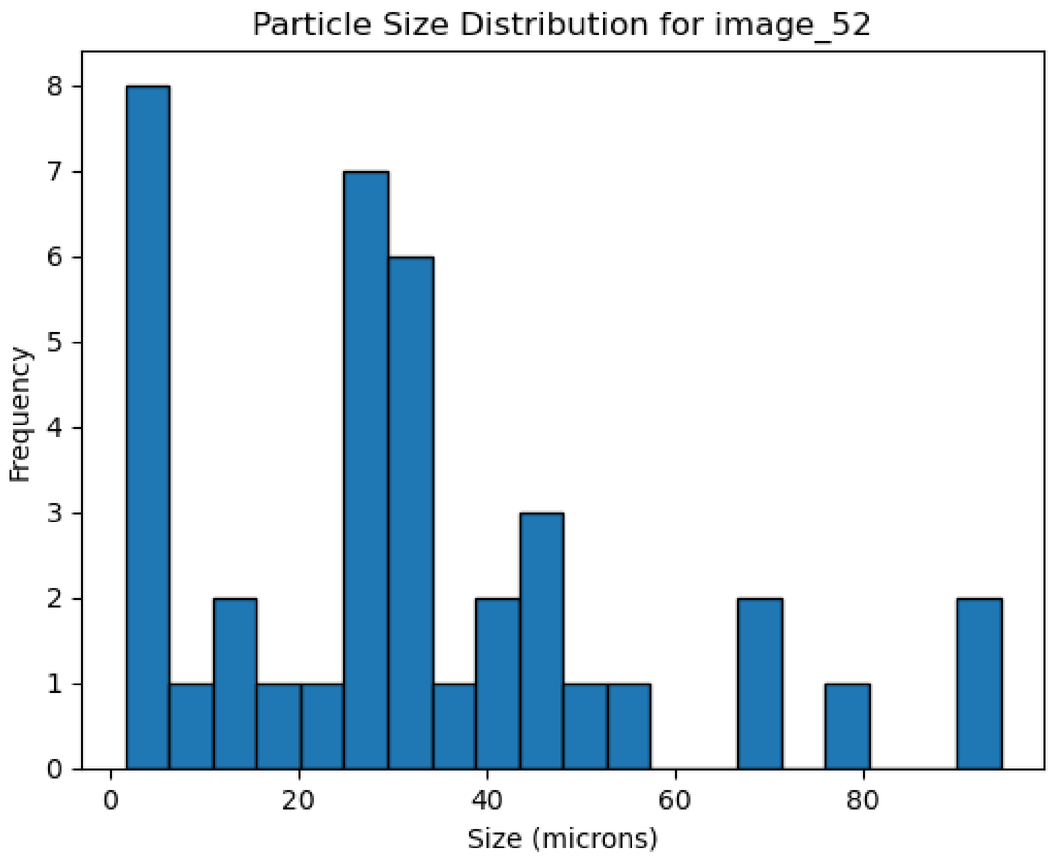

```
Particle Size Stats for image_5 - Mean: 32.99405648389687, Median:
34.383177824142415, Std Dev: 18.845790703814806
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



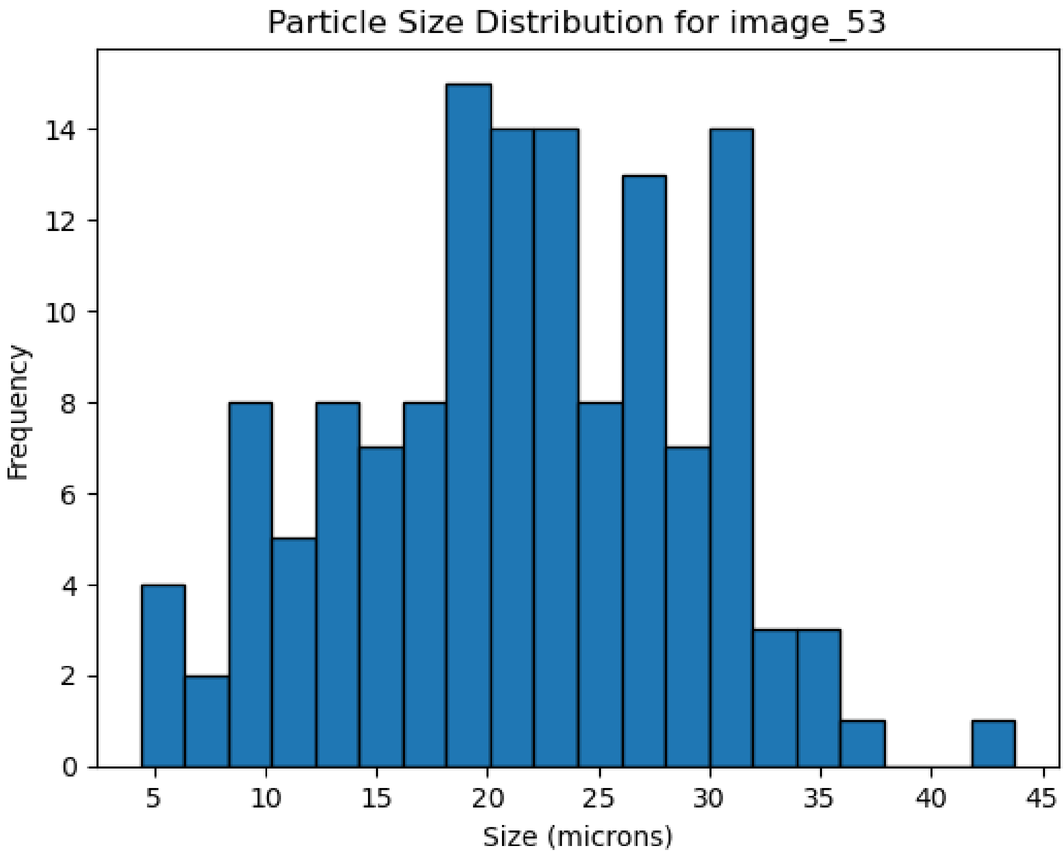

```
Particle Size Stats for image_50 - Mean: 23.540960330454617, Median:
20.668130783112442, Std Dev: 16.794711918117468
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



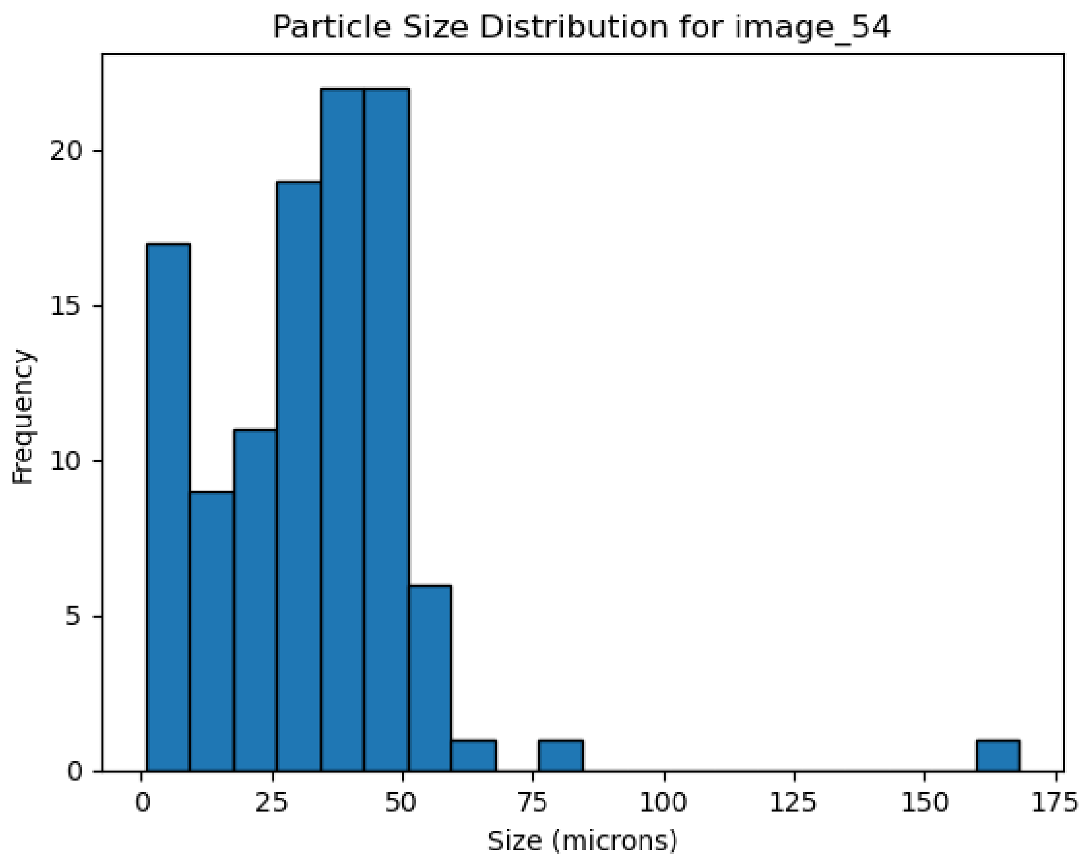

```
Particle Size Stats for Image_51 - Mean: 5.147624959329655, Median:
4.028925874571273, Std Dev: 4.371087040170622
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



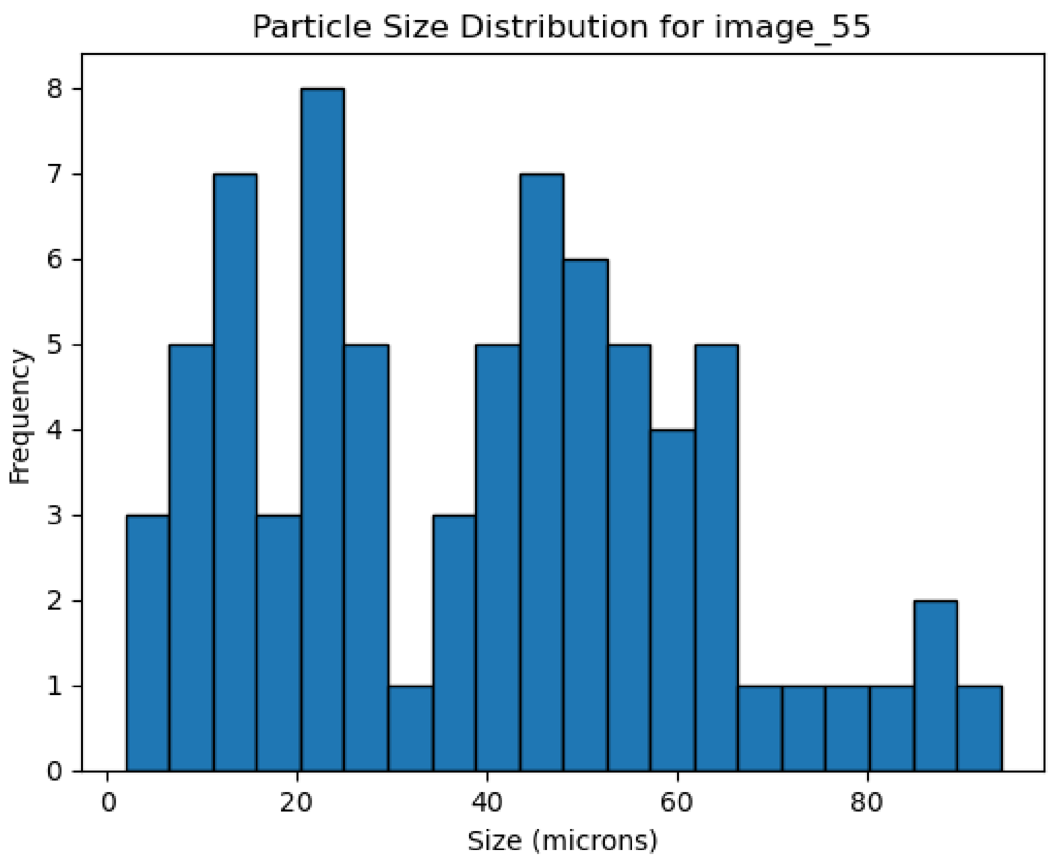

```
Particle Size Stats for image_52 - Mean: 32.012872148229334, Median:
29.130982760044088, Std Dev: 24.15316756297349
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



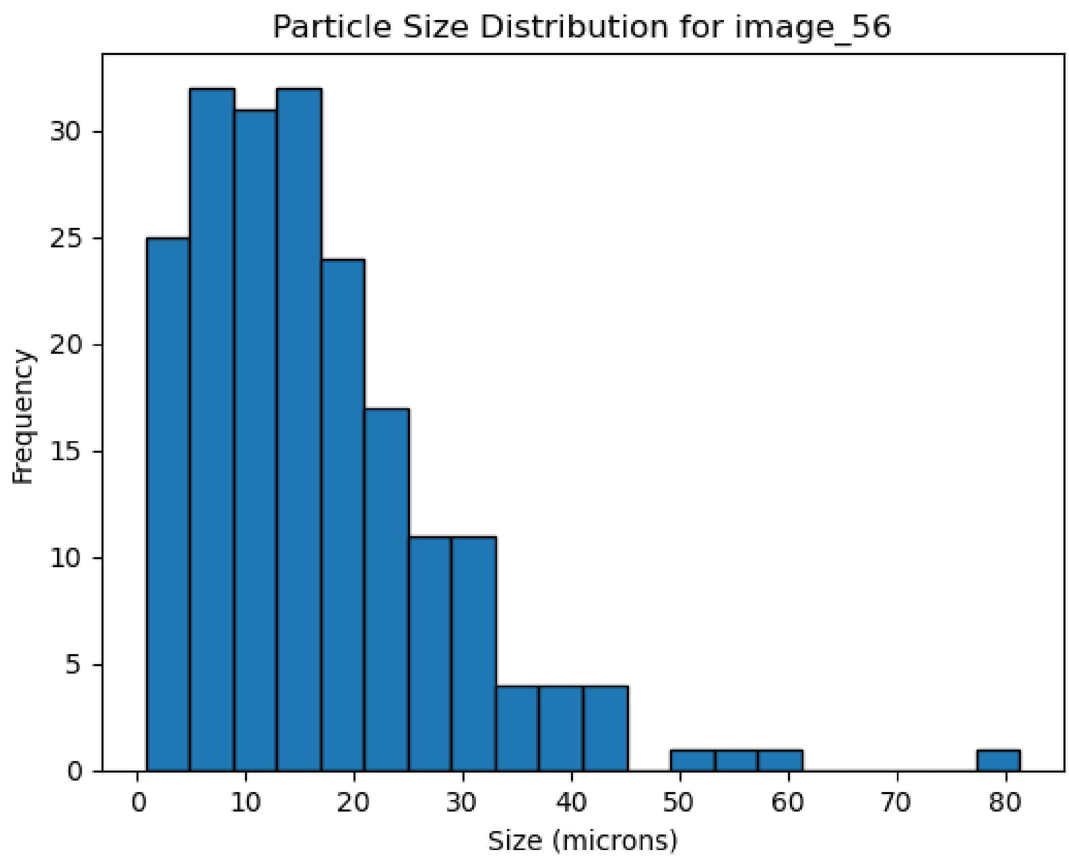

```
Particle Size Stats for image_53 - Mean: 21.579996992965565, Median:
21.778010249190537, Std Dev: 7.755142187941122
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



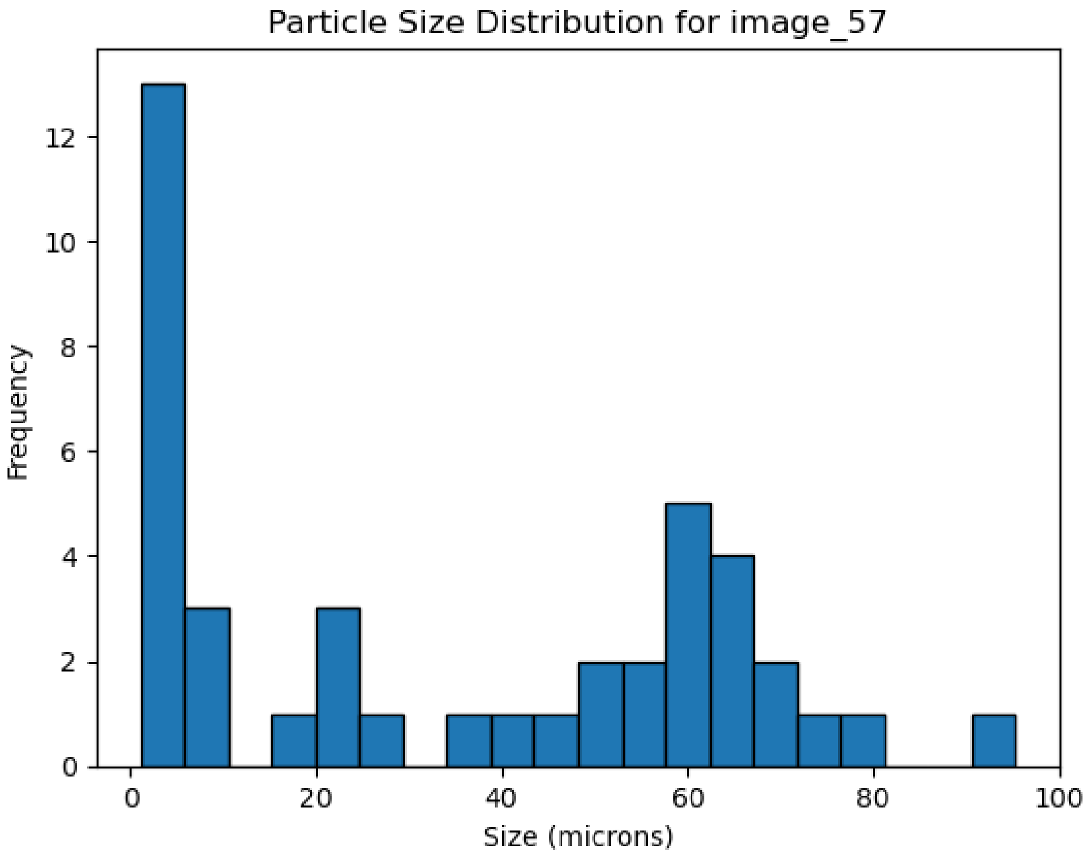

```
Particle Size Stats for image_54 - Mean: 32.41516471730654, Median:
33.982764264767994, Std Dev: 21.117532554398235
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



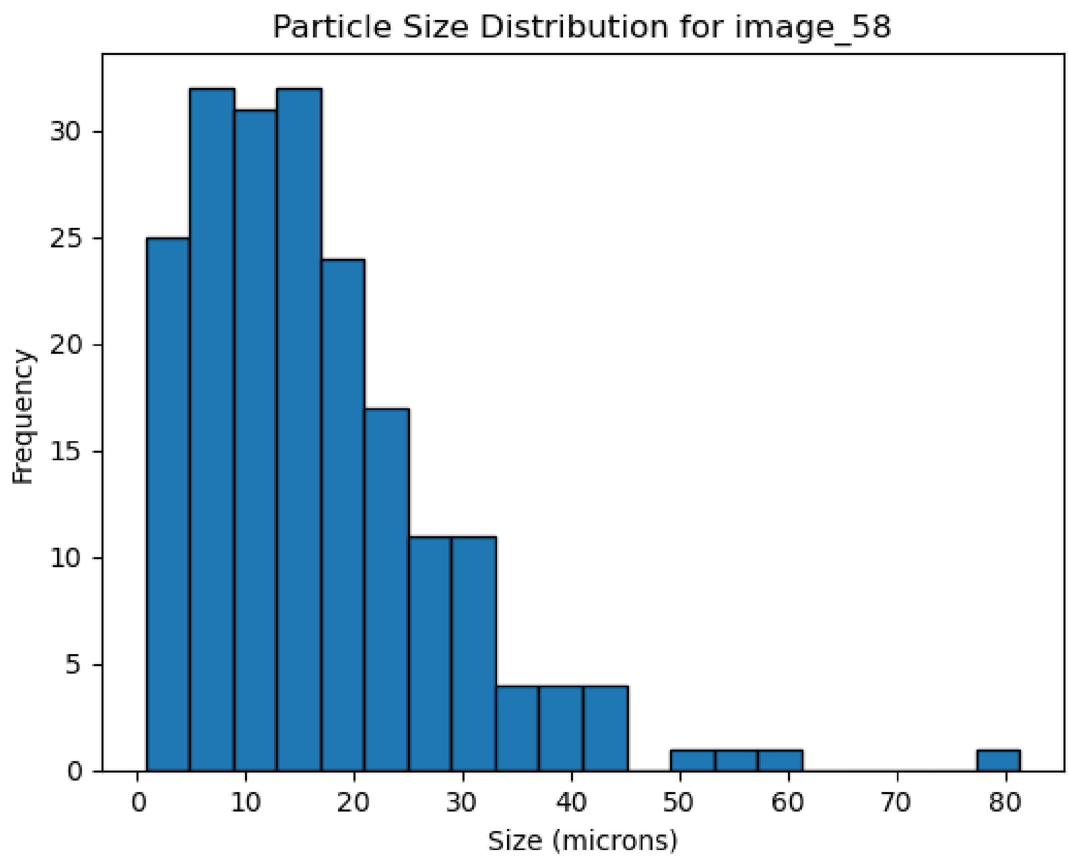

```
Particle Size Stats for image_55 - Mean: 39.179815162740866, Median:
41.05779647062597, Std Dev: 22.55776757236994
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



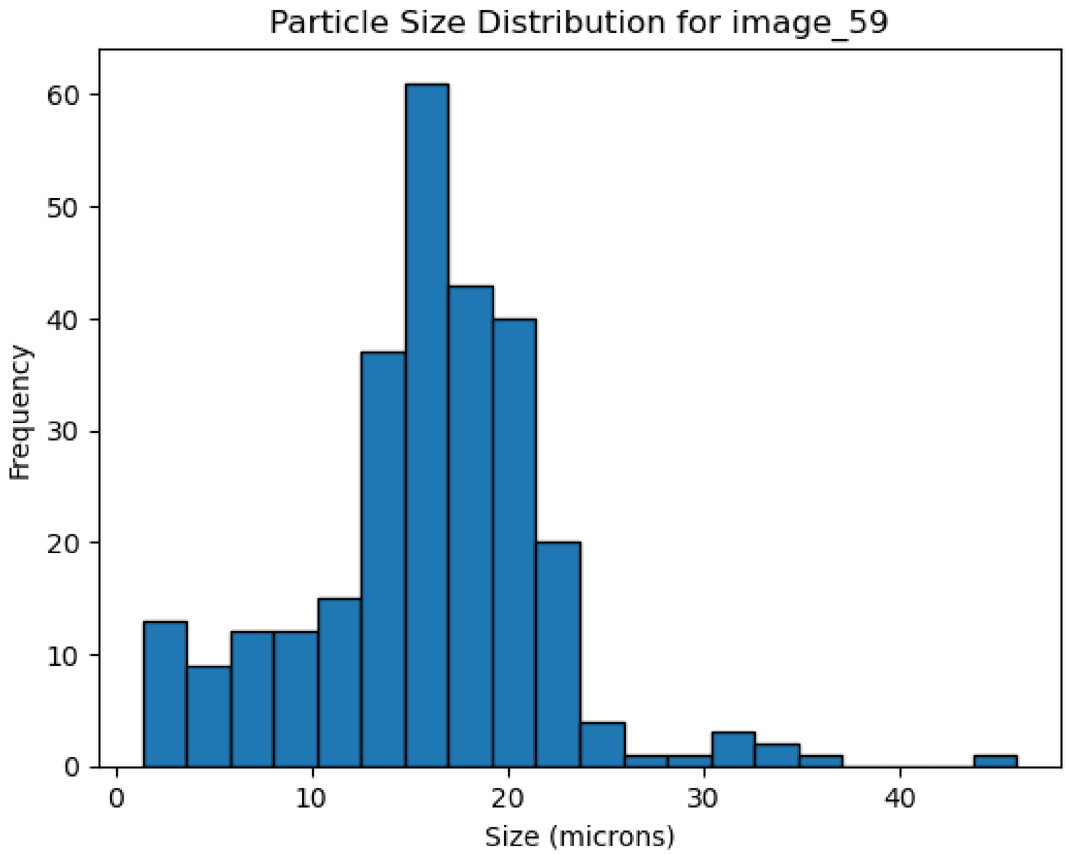

```
Particle Size Stats for image_56 - Mean: 16.58858488712706, Median:
14.317526556719258, Std Dev: 11.955102900438472
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



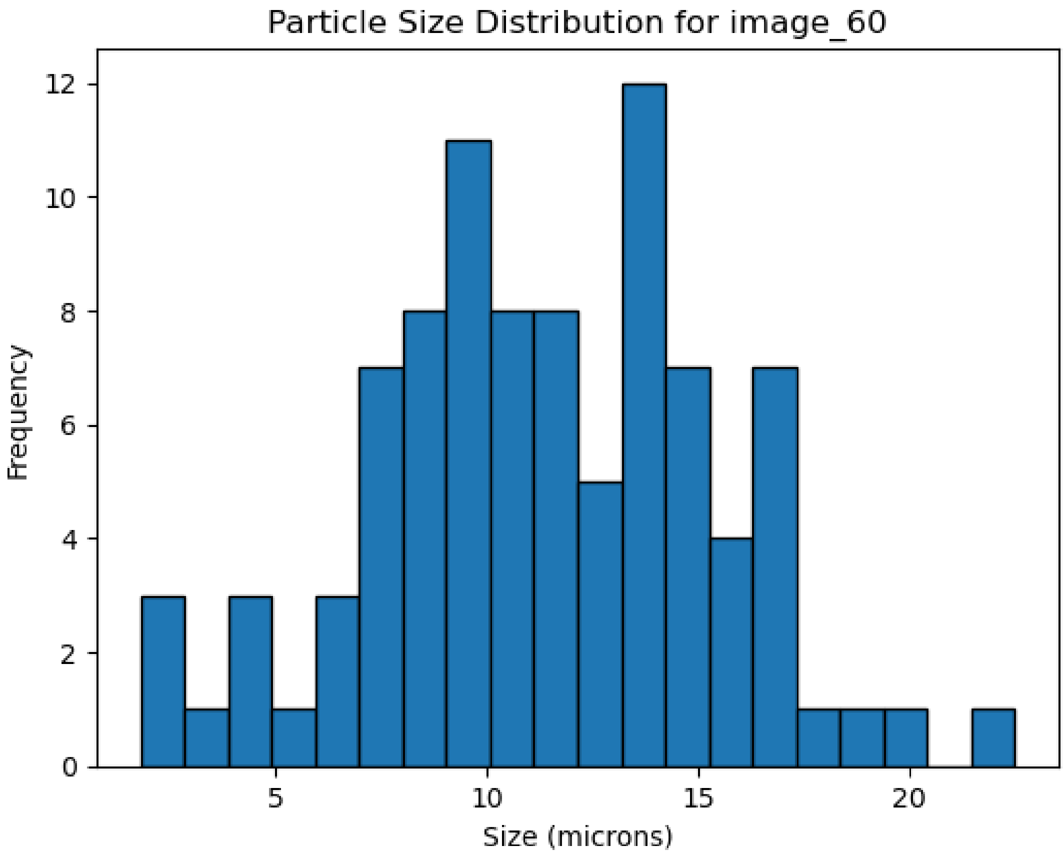

```
Particle Size Stats for image_57 - Mean: 34.75700022612697, Median:
32.13305137588334, Std Dev: 28.25426730138875
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



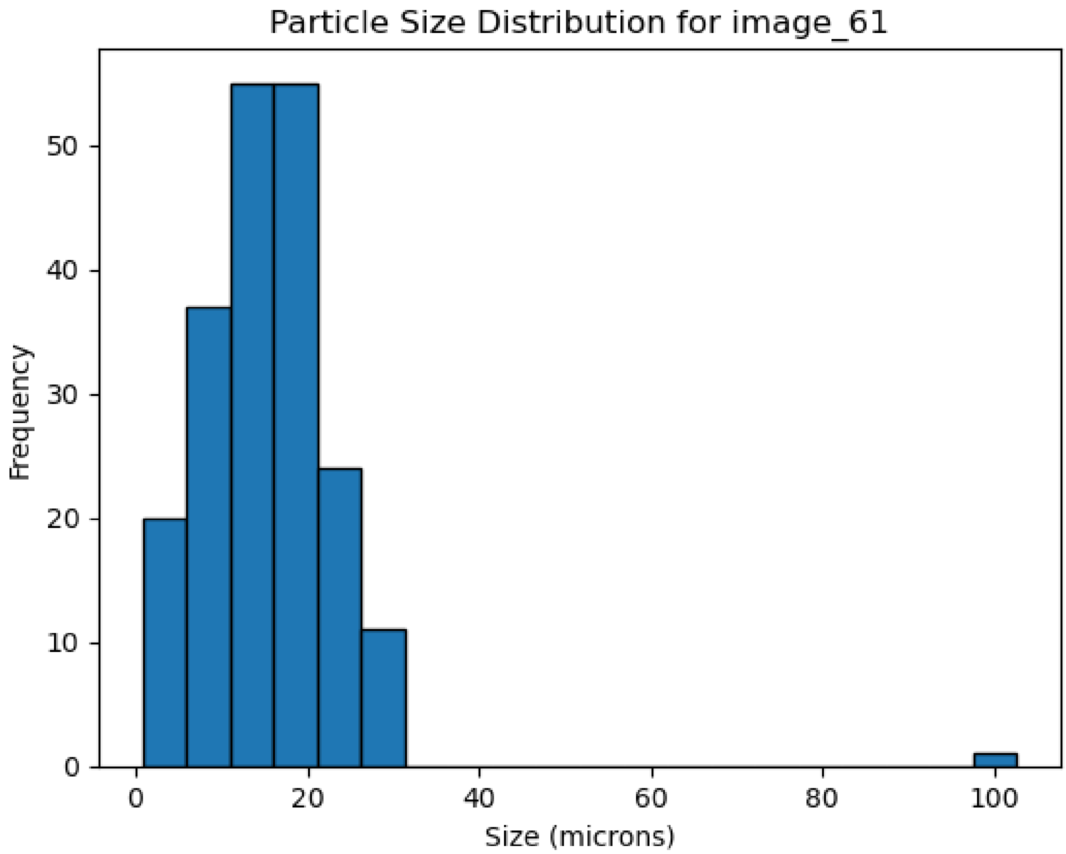

```
Particle Size Stats for image_58 - Mean: 16.58858488712706, Median:
14.317526556719258, Std Dev: 11.955102900438472
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



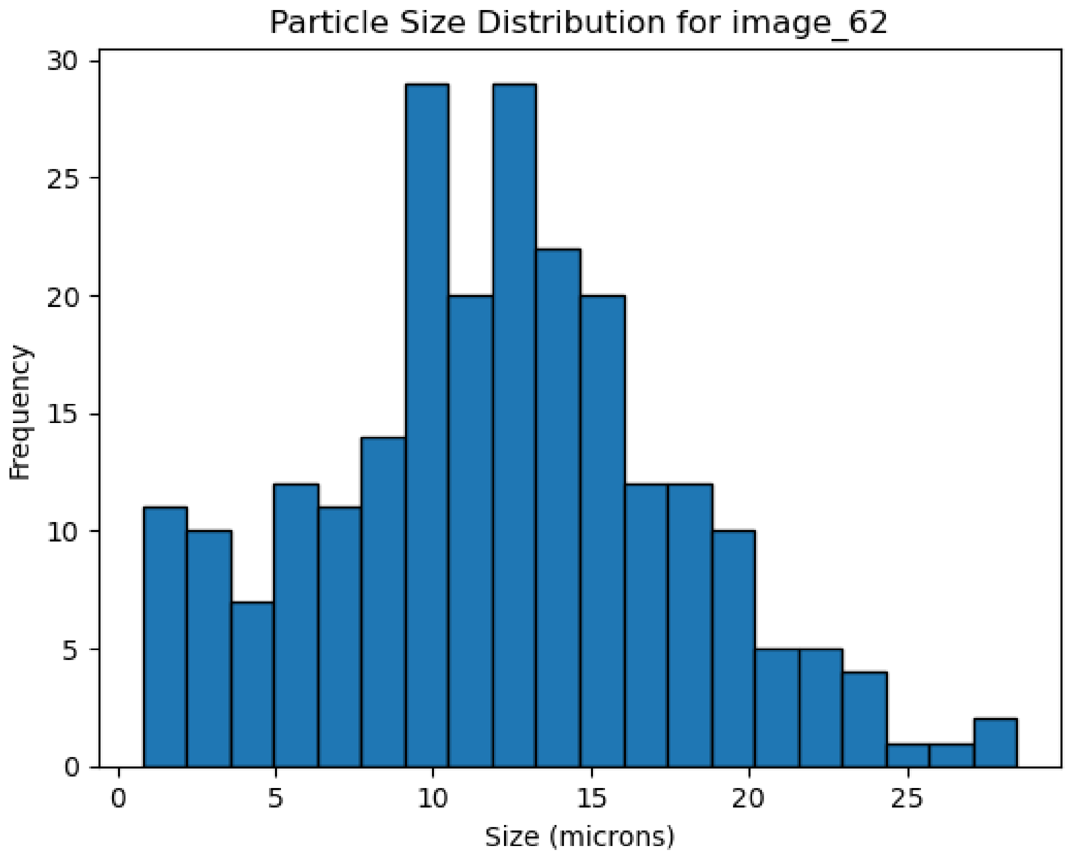

```
Particle Size Stats for image_59 - Mean: 15.804755632005554, Median:
16.096715421277896, Std Dev: 6.175953775732848
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



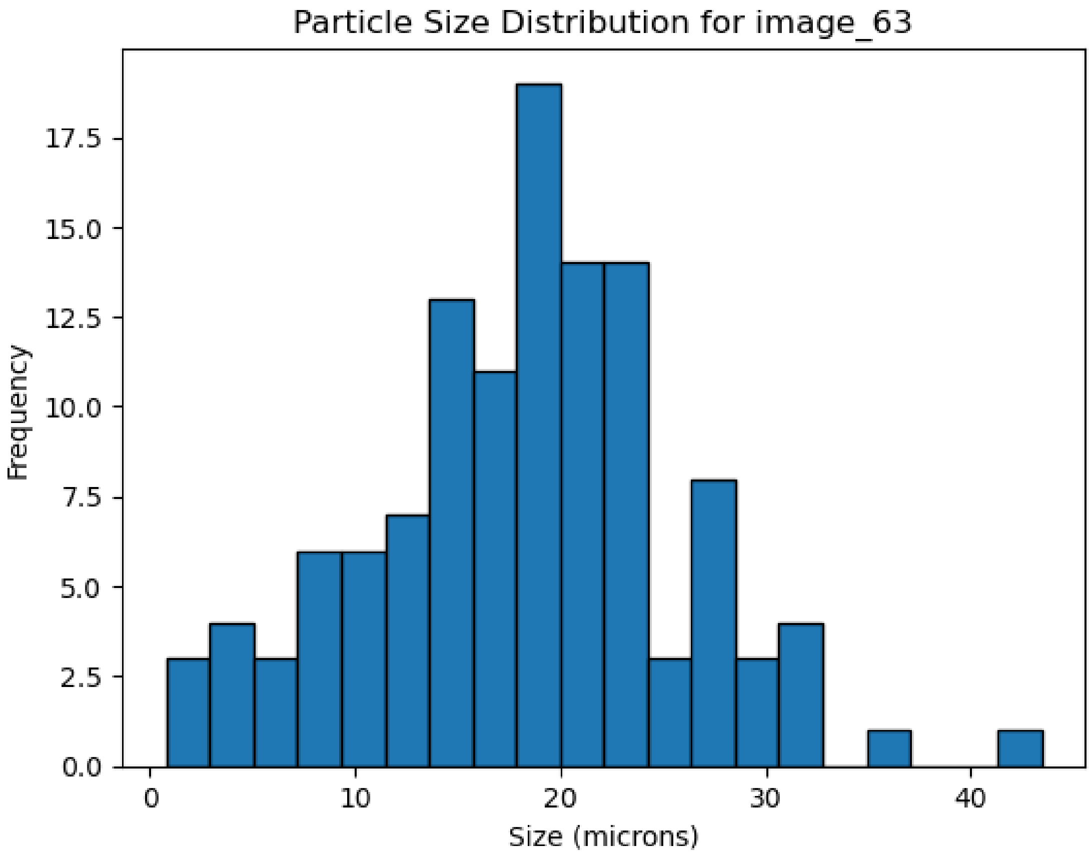

```
Particle Size Stats for image_60 - Mean: 11.355817271740738, Median:
11.170383851240114, Std Dev: 4.063623023464038
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



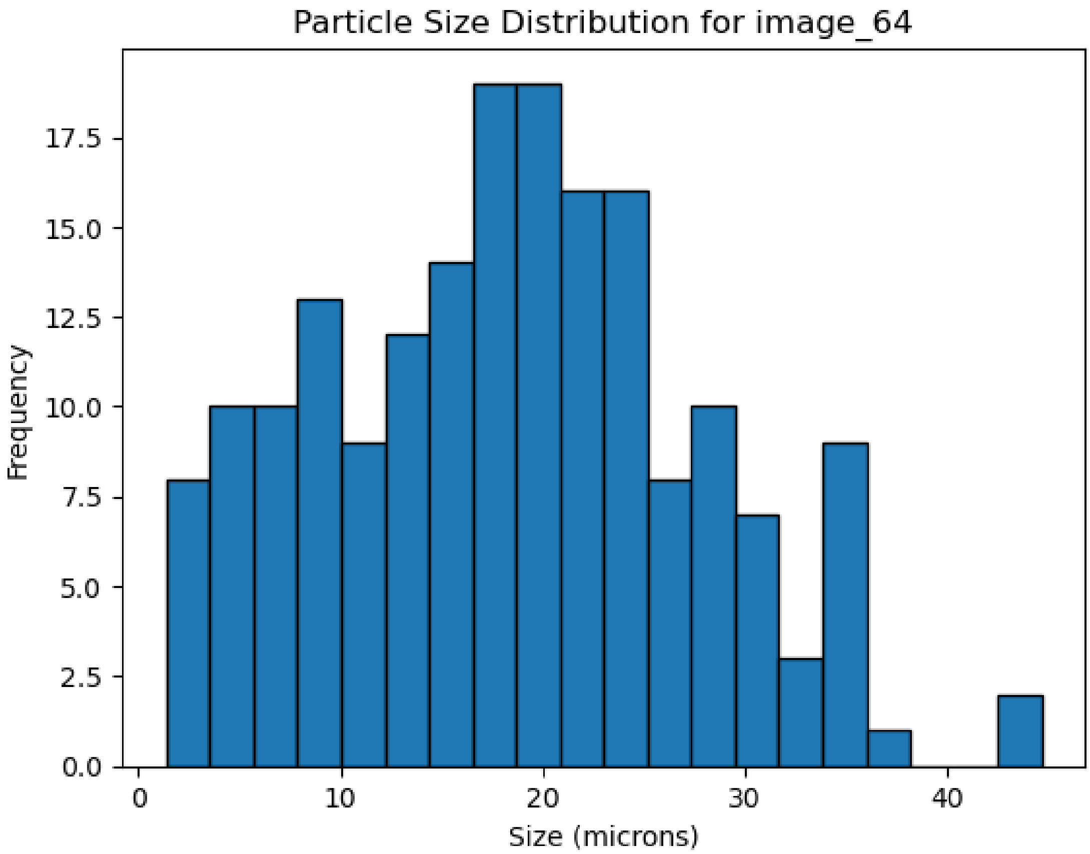

```
Particle Size Stats for image_61 - Mean: 15.476118562957529, Median:
15.471555655790098, Std Dev: 8.934491108756136
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



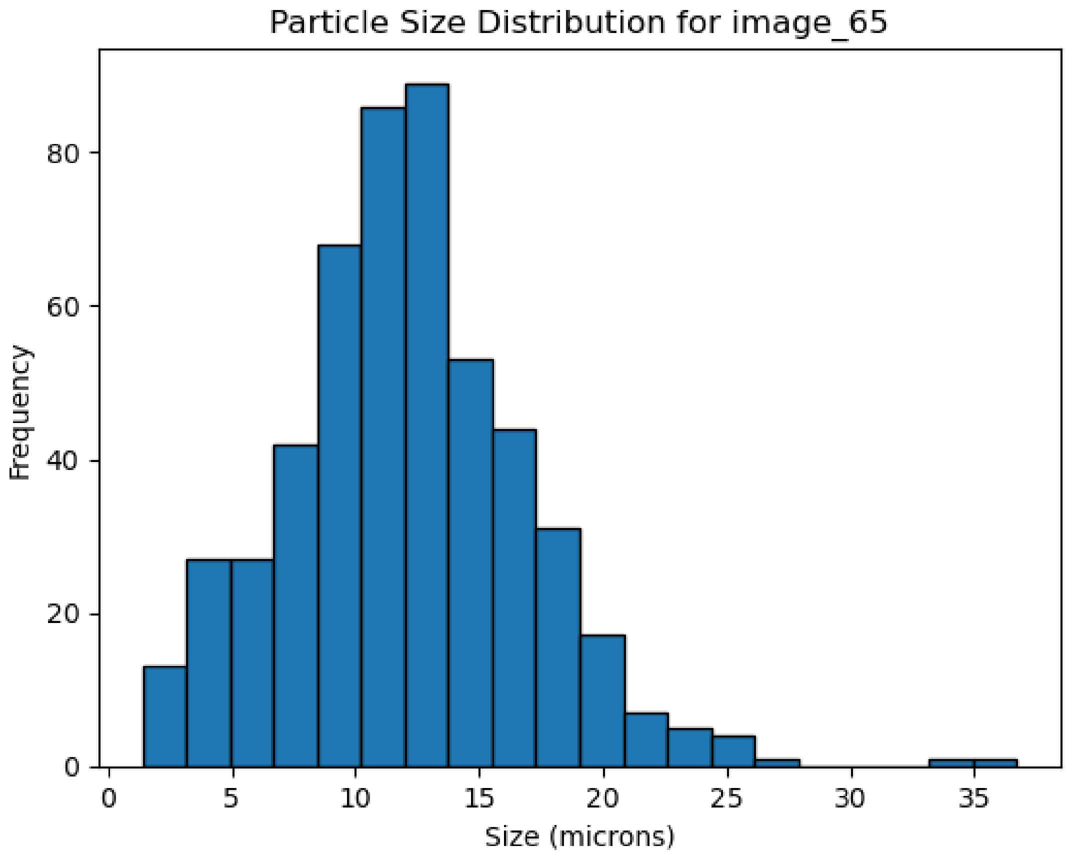

```
Particle Size Stats for image_62 - Mean: 12.056783636418235, Median:
12.100518486599809, Std Dev: 5.646730661176433
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



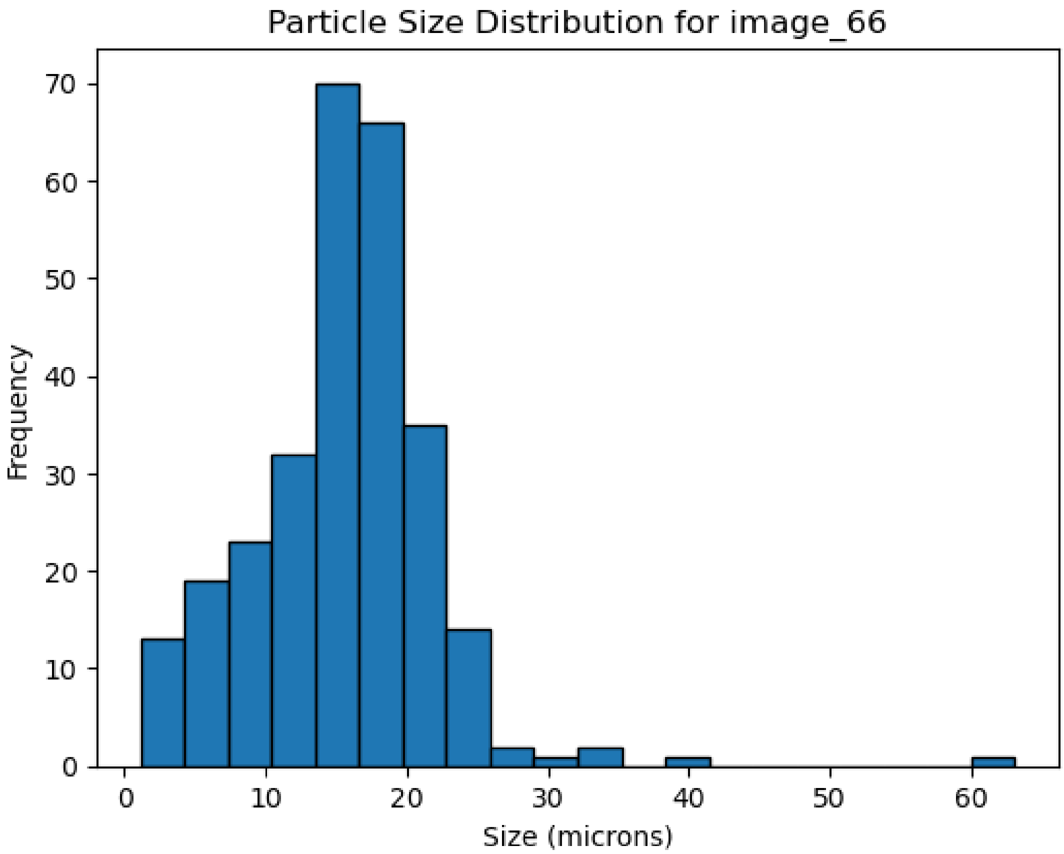

```
Particle Size Stats for image_63 - Mean: 18.177895267072063, Median:
18.686422627102235, Std Dev: 7.520989514792041
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



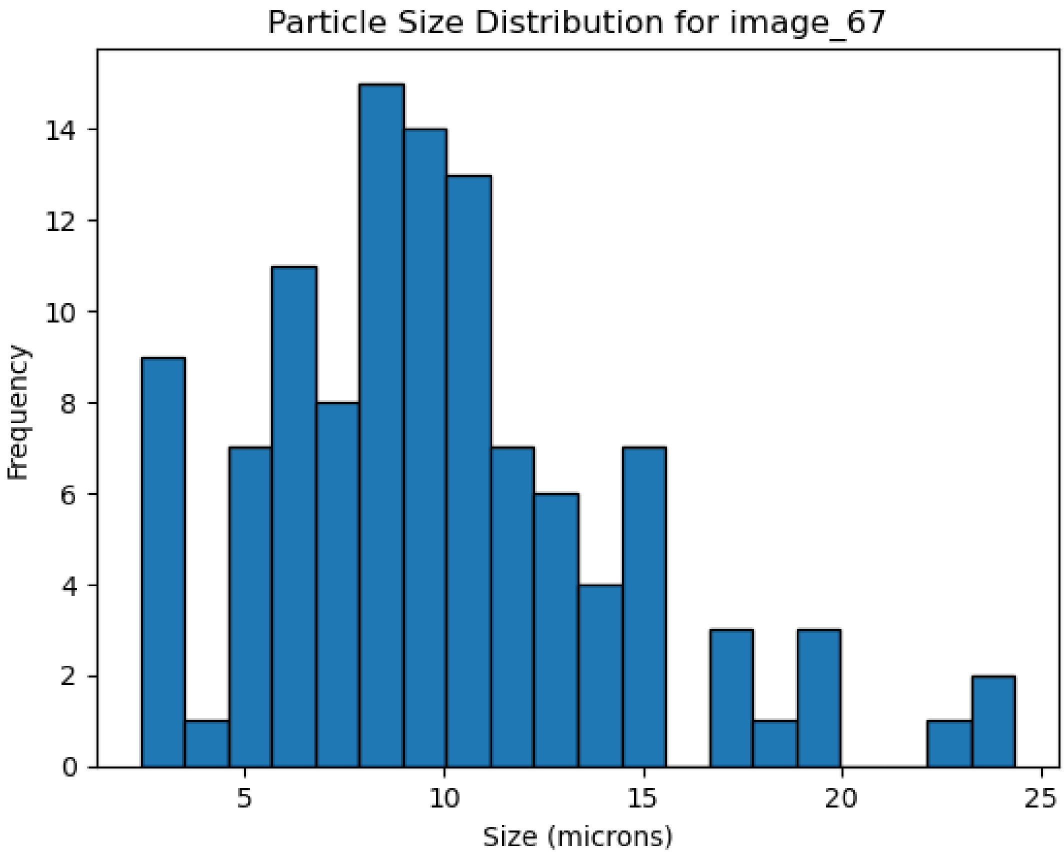

```
Particle Size Stats for image_64 - Mean: 18.397603920208752, Median:
18.359997090726495, Std Dev: 9.052462422177891
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



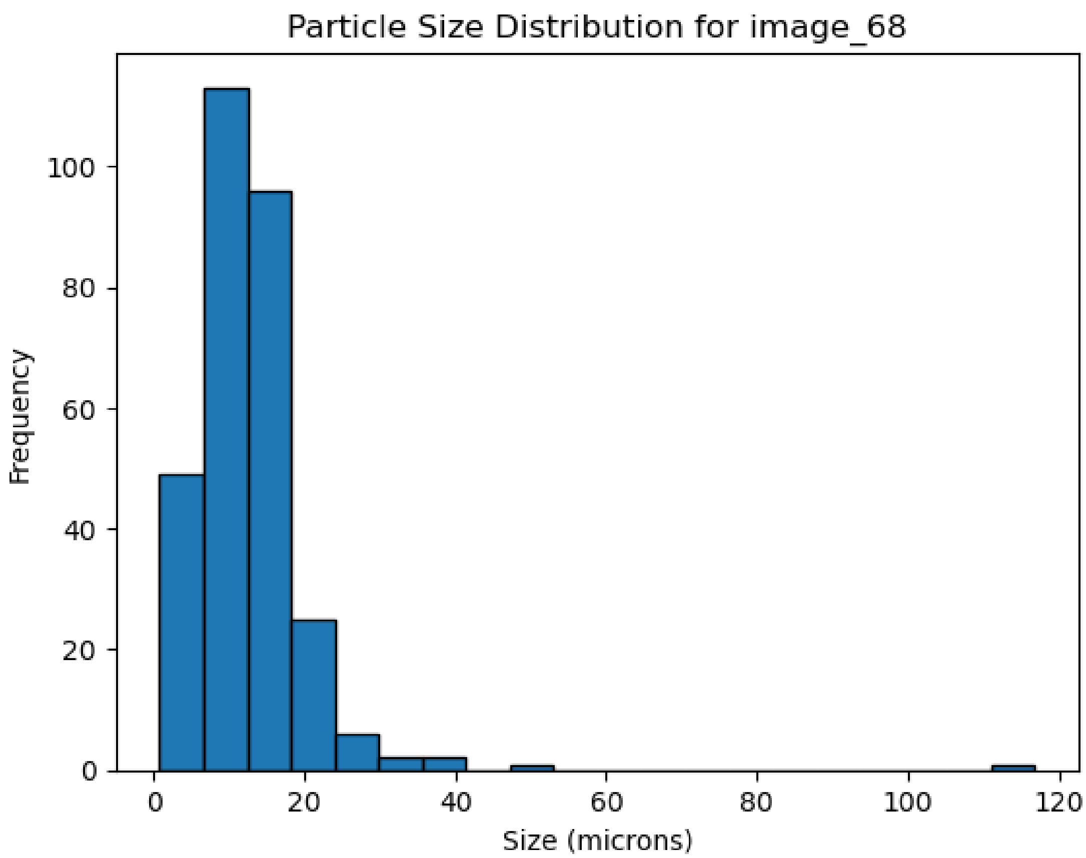

```
Particle Size Stats for image_65 - Mean: 12.070511313532311, Median:
11.928300008730007, Std Dev: 4.822597789034326
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



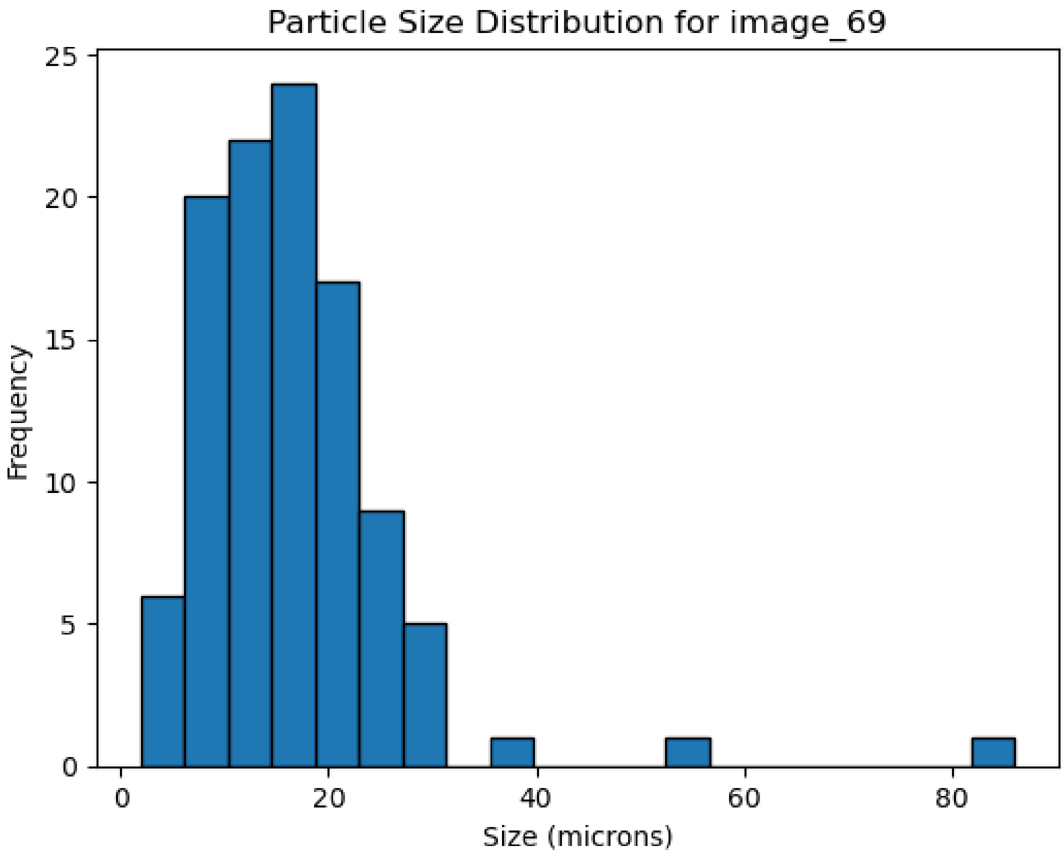

```
Particle Size Stats for image_66 - Mean: 15.489288935245598, Median:
15.797308339337176, Std Dev: 6.581498204739113
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



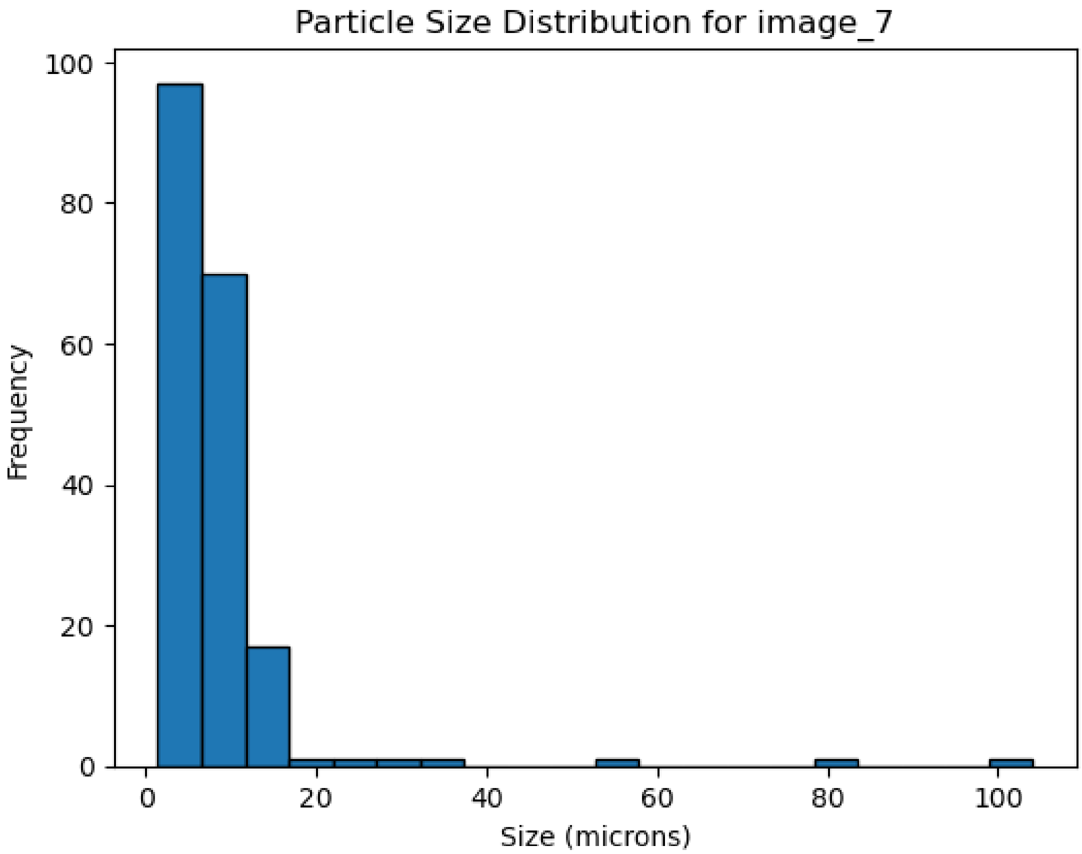

```
Particle Size Stats for image_67 - Mean: 9.896329637463756, Median:
9.423687768377217, Std Dev: 4.504669355623453
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



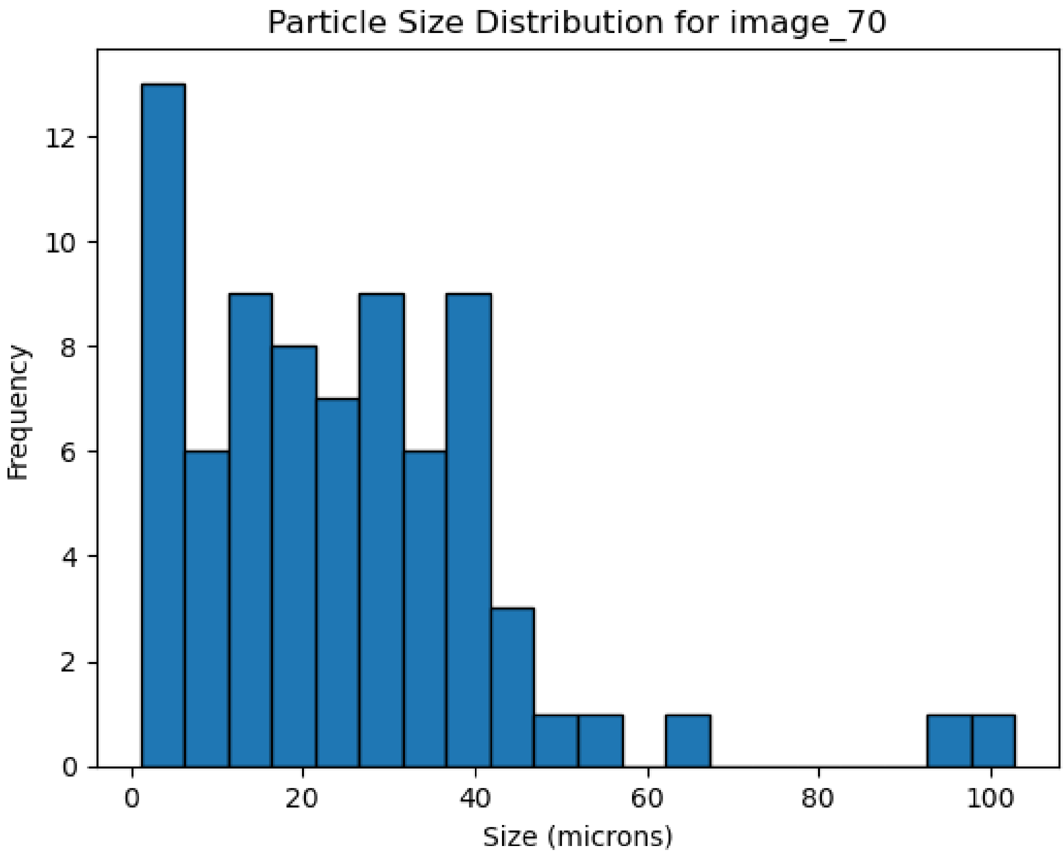

```
Particle Size Stats for image_68 - Mean: 12.542835859736048, Median:
11.617374563210365, Std Dev: 8.692730957151156
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



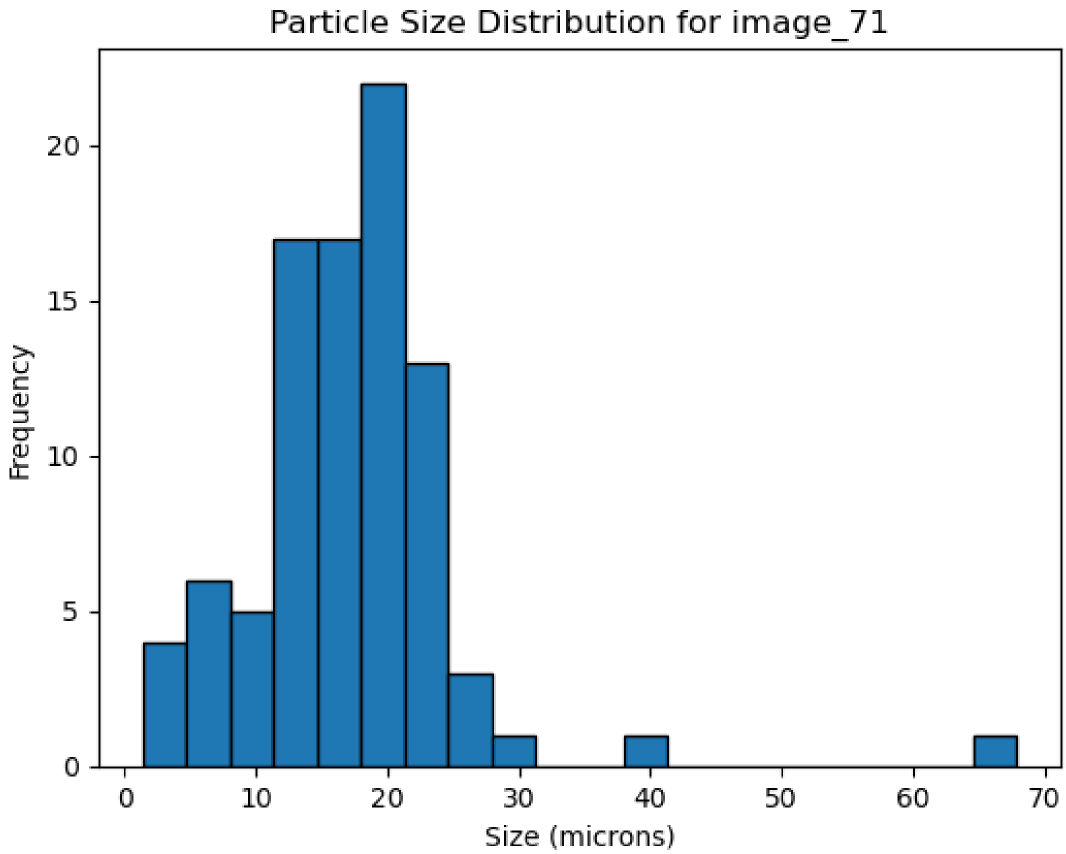

```
Particle Size Stats for image_69 - Mean: 16.61541056287725, Median:
15.47143253838414, Std Dev: 10.209621719424648
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



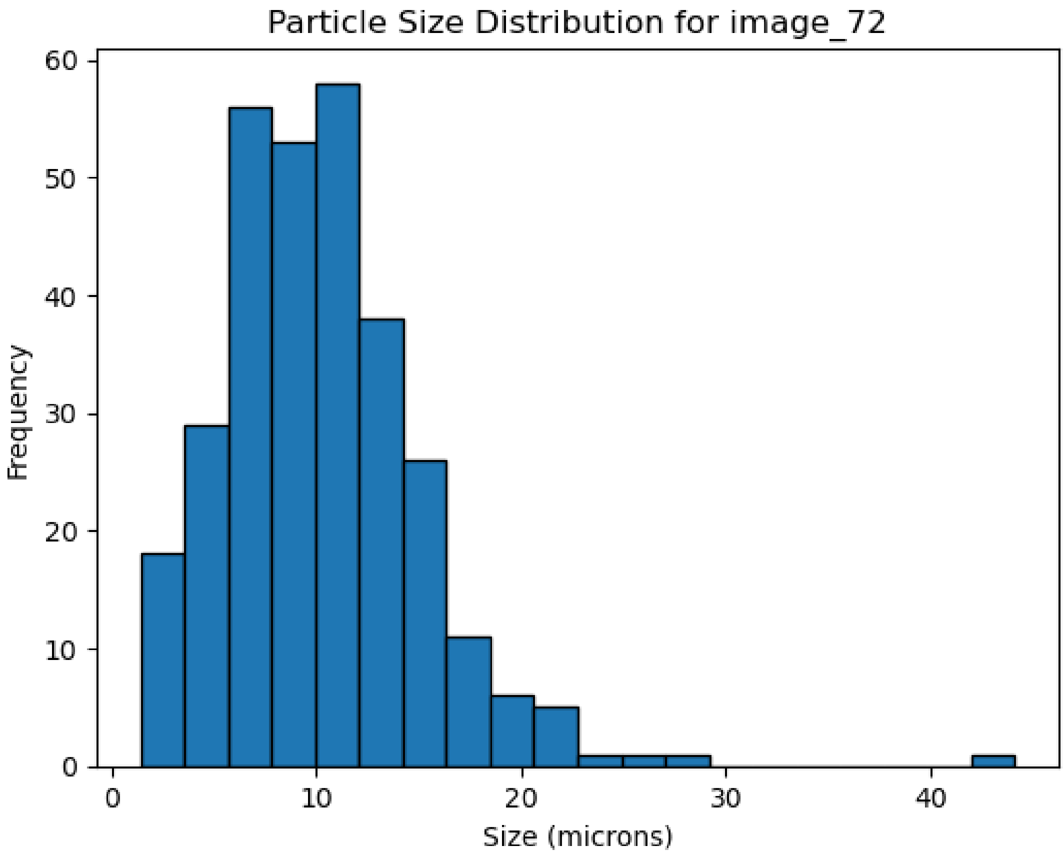

```
Particle Size Stats for image_7 - Mean: 8.382340474680445, Median:
6.4820448144285745, Std Dev: 10.439819479928266
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



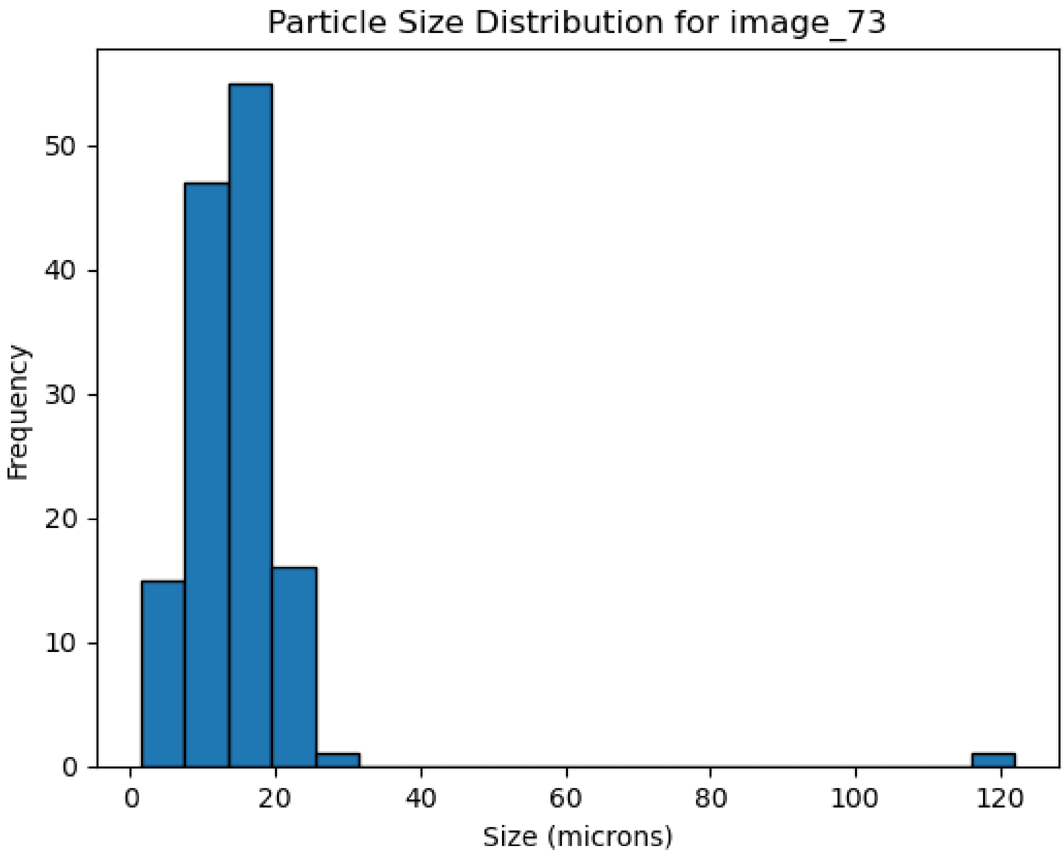

```
Particle Size Stats for image_70 - Mean: 24.83408505546361, Median:
22.03953008383946, Std Dev: 19.06526725141327
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



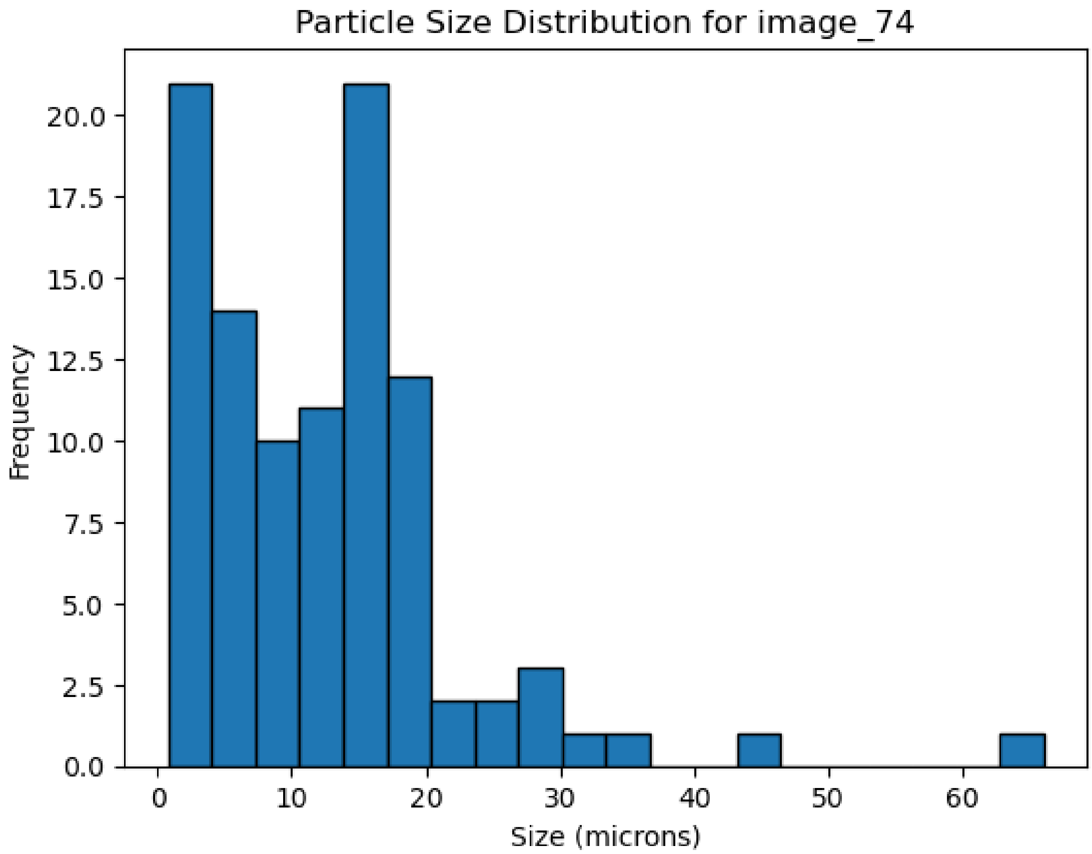

```
Particle Size Stats for image_71 - Mean: 17.124487465115273, Median:
16.65014572277187, Std Dev: 8.364915026760302
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



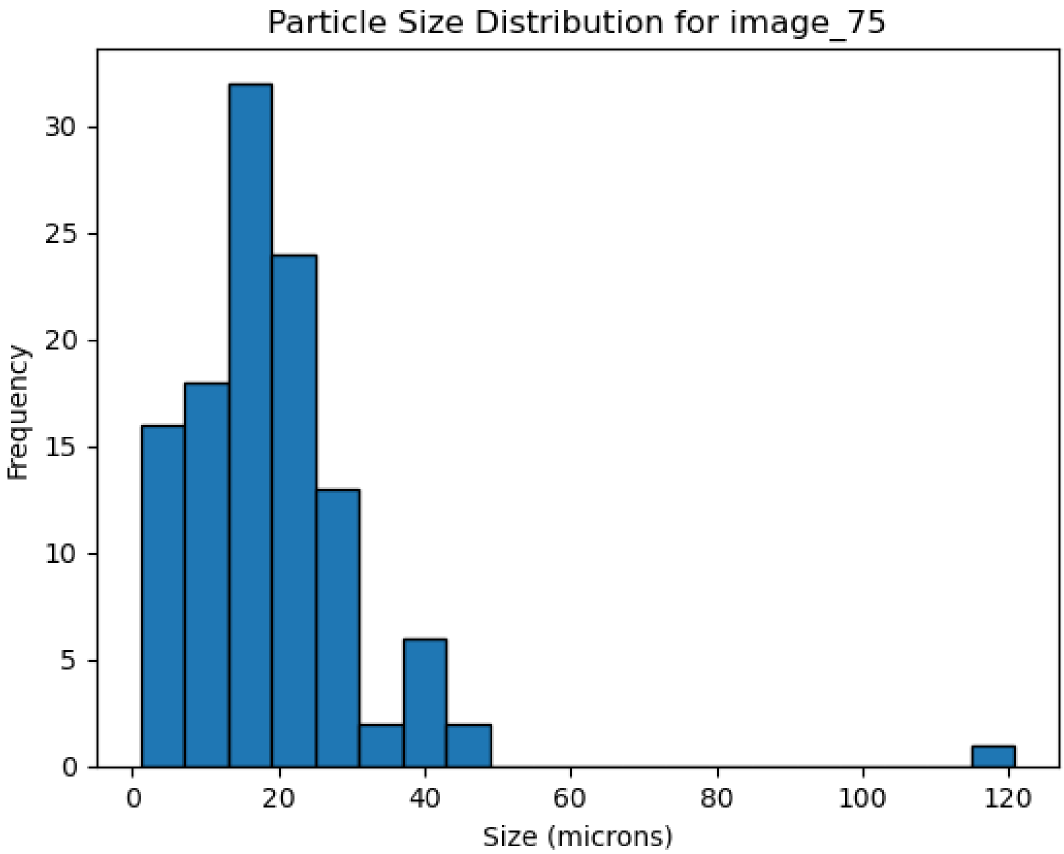

```
Particle Size Stats for image_72 - Mean: 10.219169772331474, Median:
9.624339115834097, Std Dev: 4.961662854722917
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



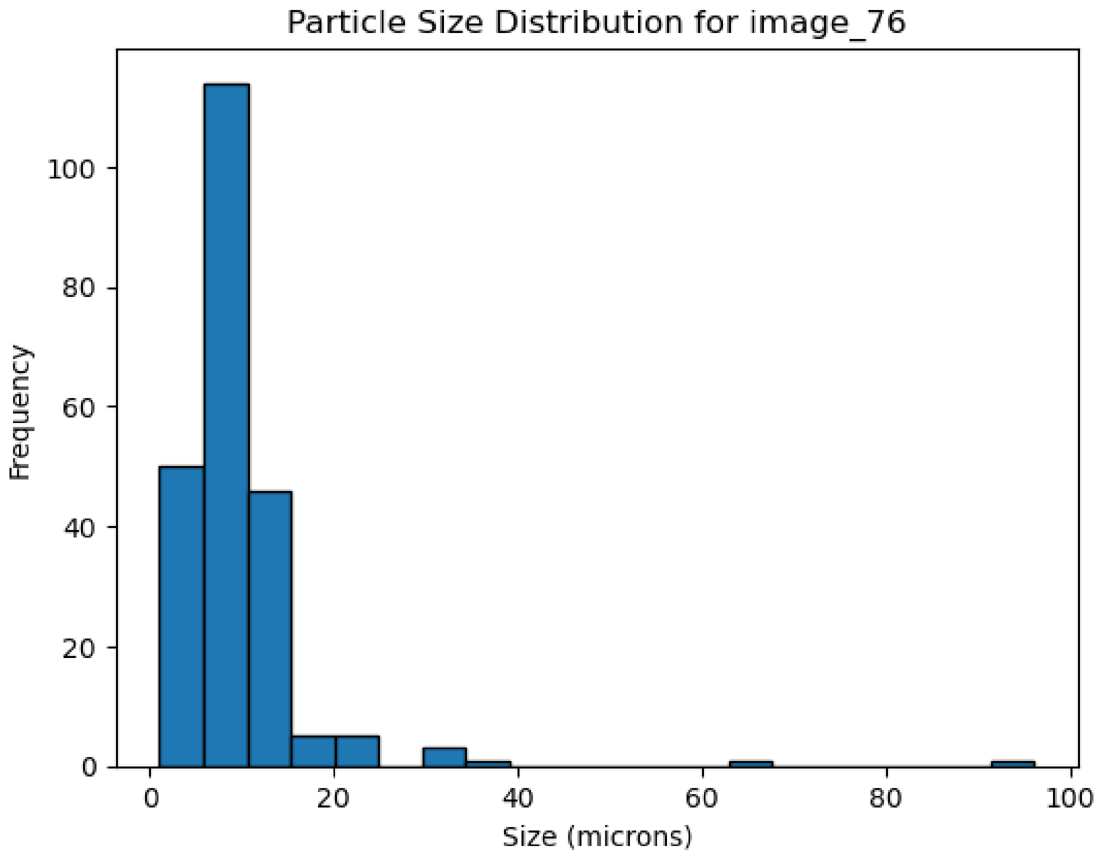

```
Particle Size Stats for image_73 - Mean: 14.659550635989376, Median:
14.339741506551949, Std Dev: 10.437157855432115
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



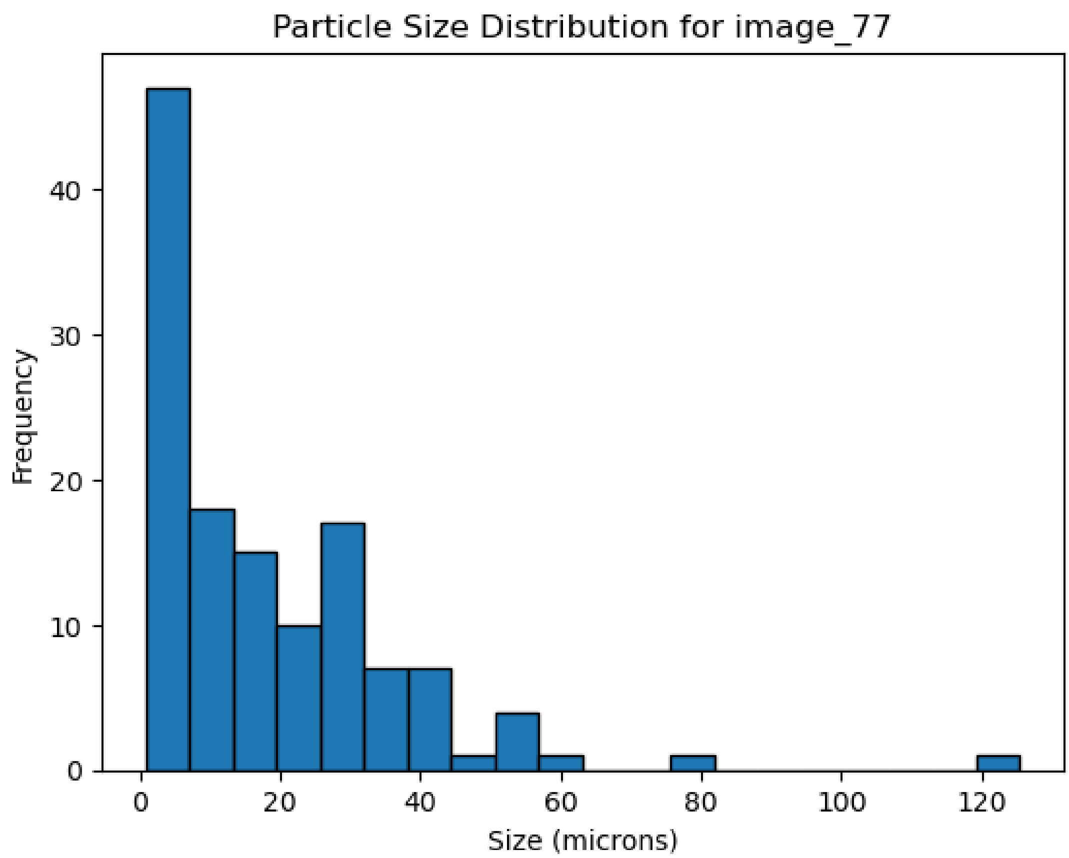

```
Particle Size Stats for image_74 - Mean: 12.492768351402104, Median:
11.658329387303402, Std Dev: 9.756303832831327
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



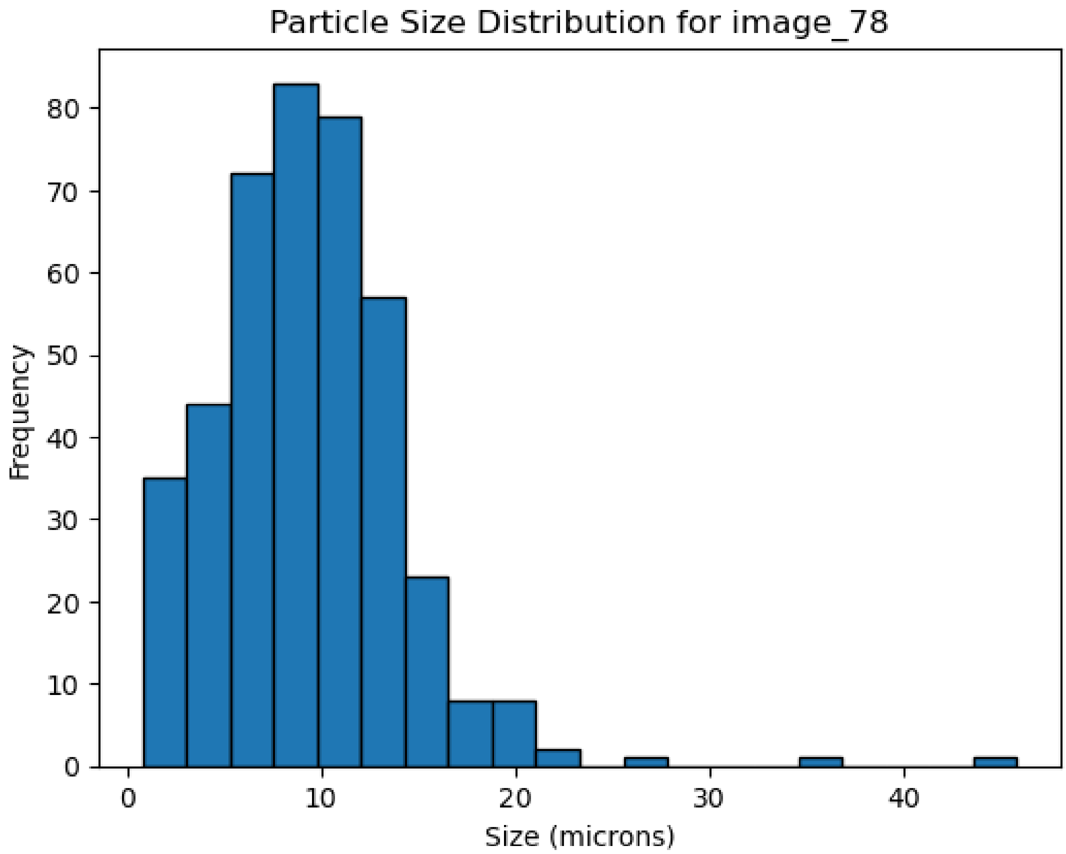

```
Particle Size Stats for image_75 - Mean: 18.775019960158232, Median:
17.131267731869812, Std Dev: 13.715380734162489
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



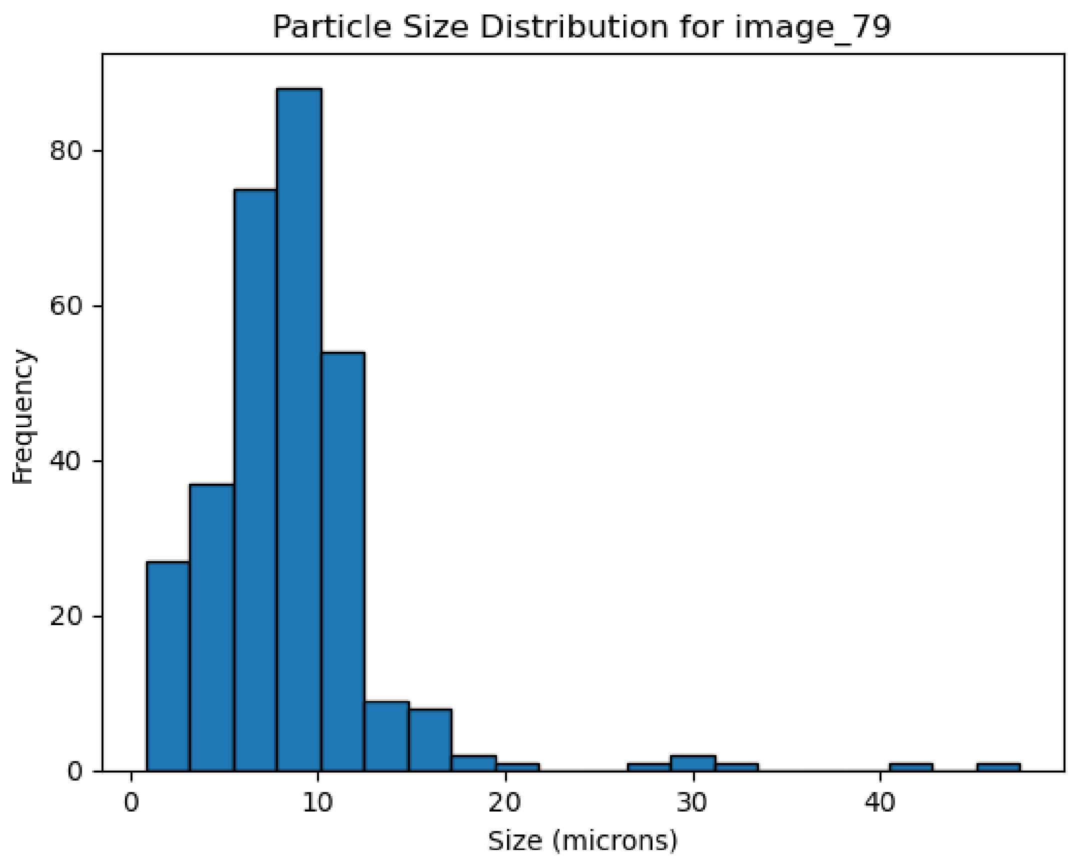

```
Particle Size Stats for image_76 - Mean: 9.760338467101398, Median:
8.175883811466258, Std Dev: 8.605306522290103
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



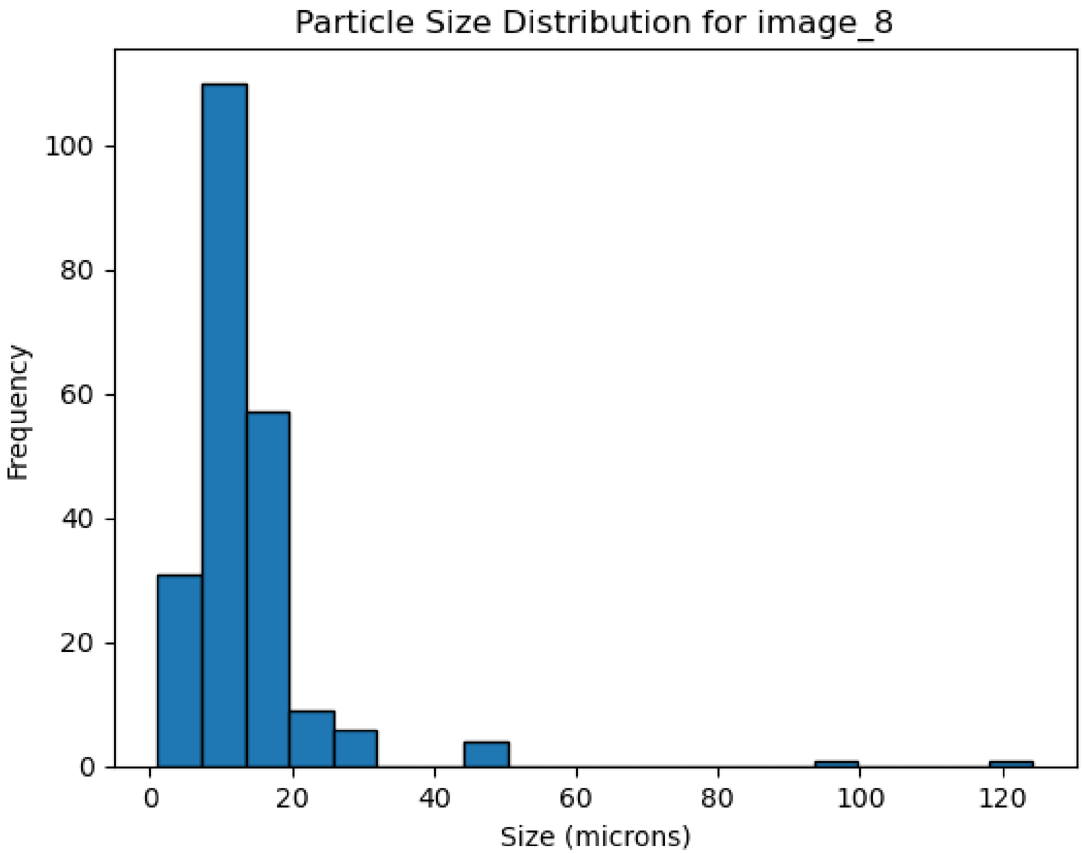

```
Particle Size Stats for image_77 - Mean: 18.287124287697175, Median:
12.766152972845846, Std Dev: 18.17649179387759
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



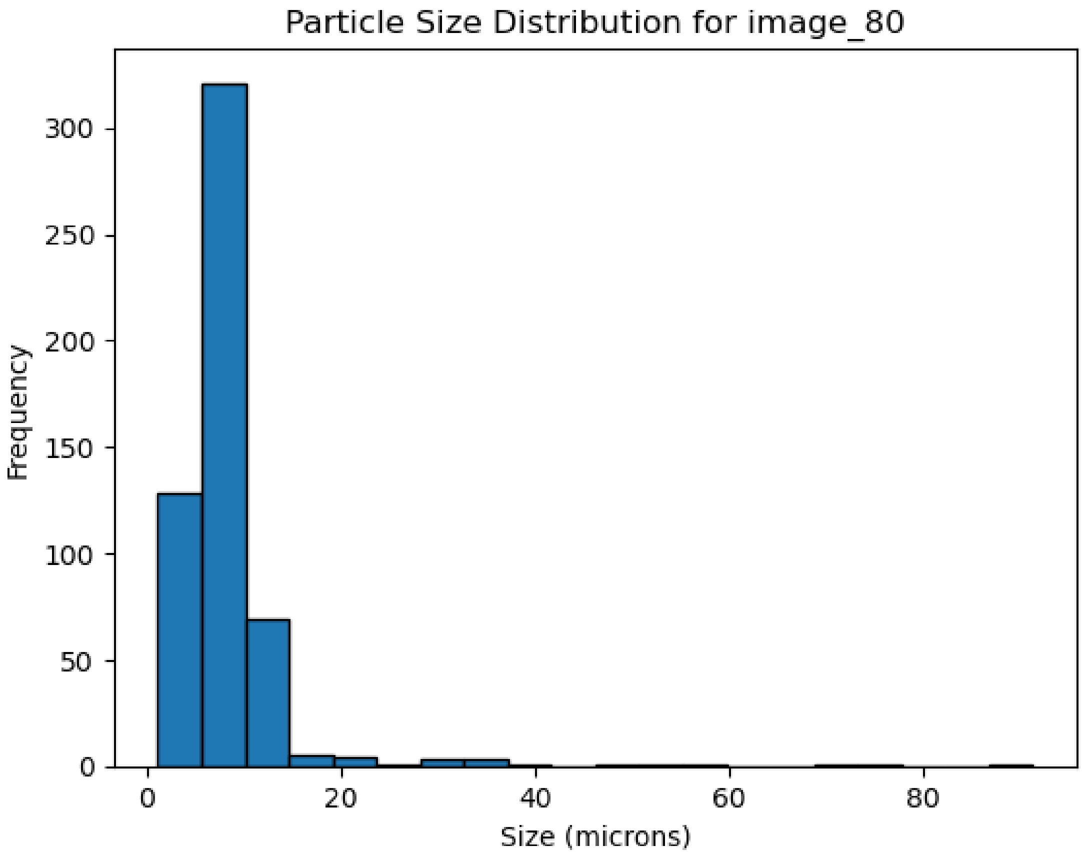

```
Particle Size Stats for image_78 - Mean: 9.310671884055314, Median:
8.973967164404486, Std Dev: 4.860554108497289
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



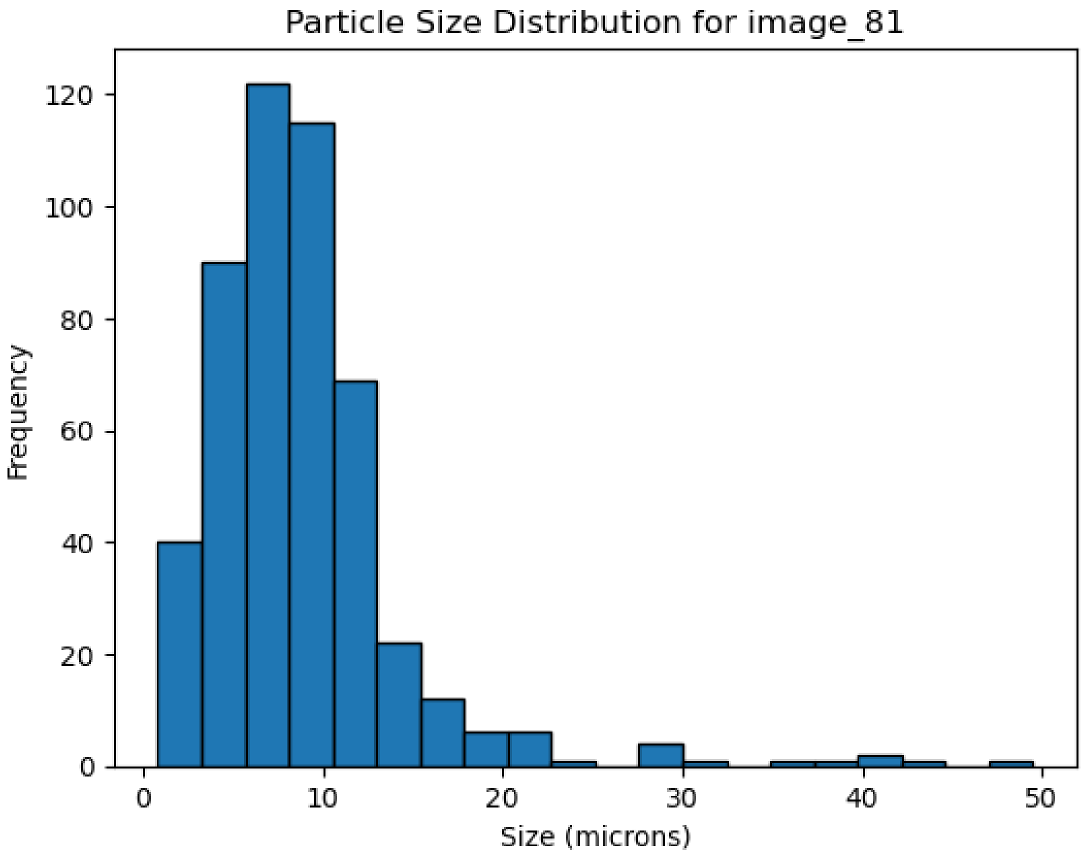

```
Particle Size Stats for image_79 - Mean: 8.546242710426318, Median:
8.368283871884005, Std Dev: 5.069517792913709
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



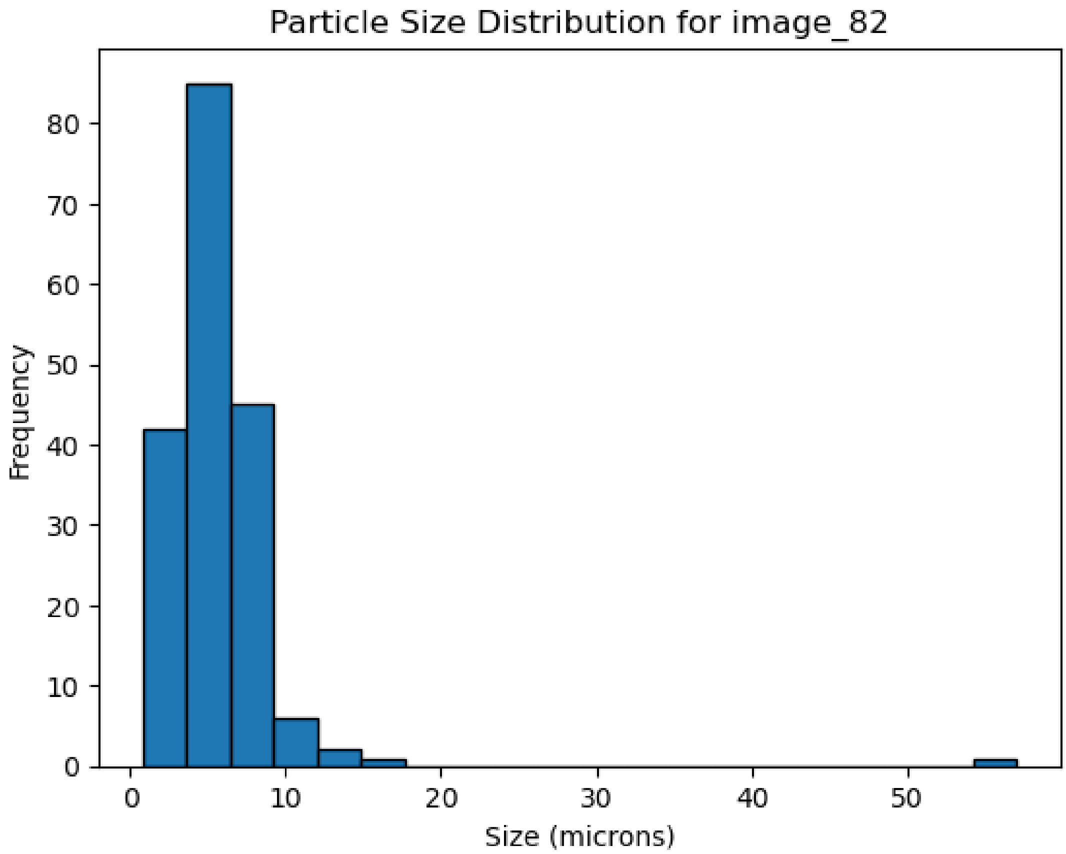

```
Particle Size Stats for image_8 - Mean: 13.666299036123592, Median:
11.507254783503184, Std Dev: 11.642255182241726
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



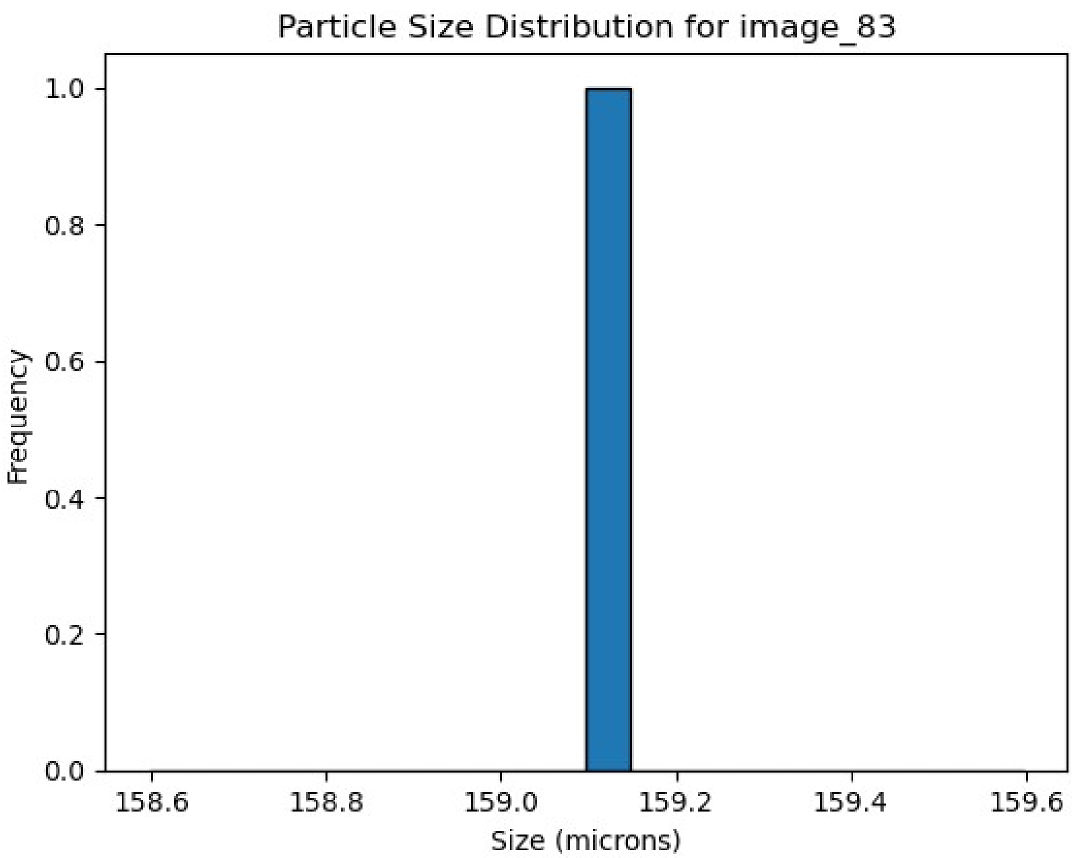

```
Particle Size Stats for image_80 - Mean: 8.487853270381235, Median:
7.48482063701911, Std Dev: 7.604701664683703
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



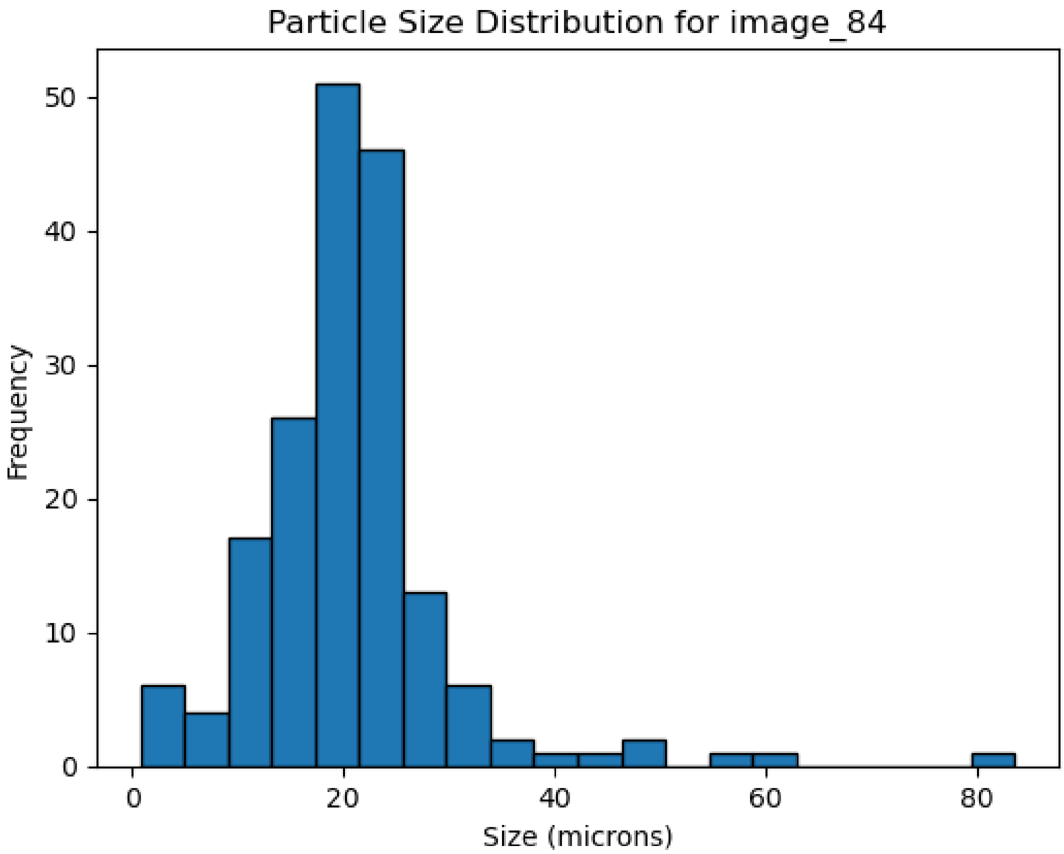

```
Particle Size Stats for image_81 - Mean: 8.899201604686182, Median:
8.018640596081465, Std Dev: 5.753871004321751
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



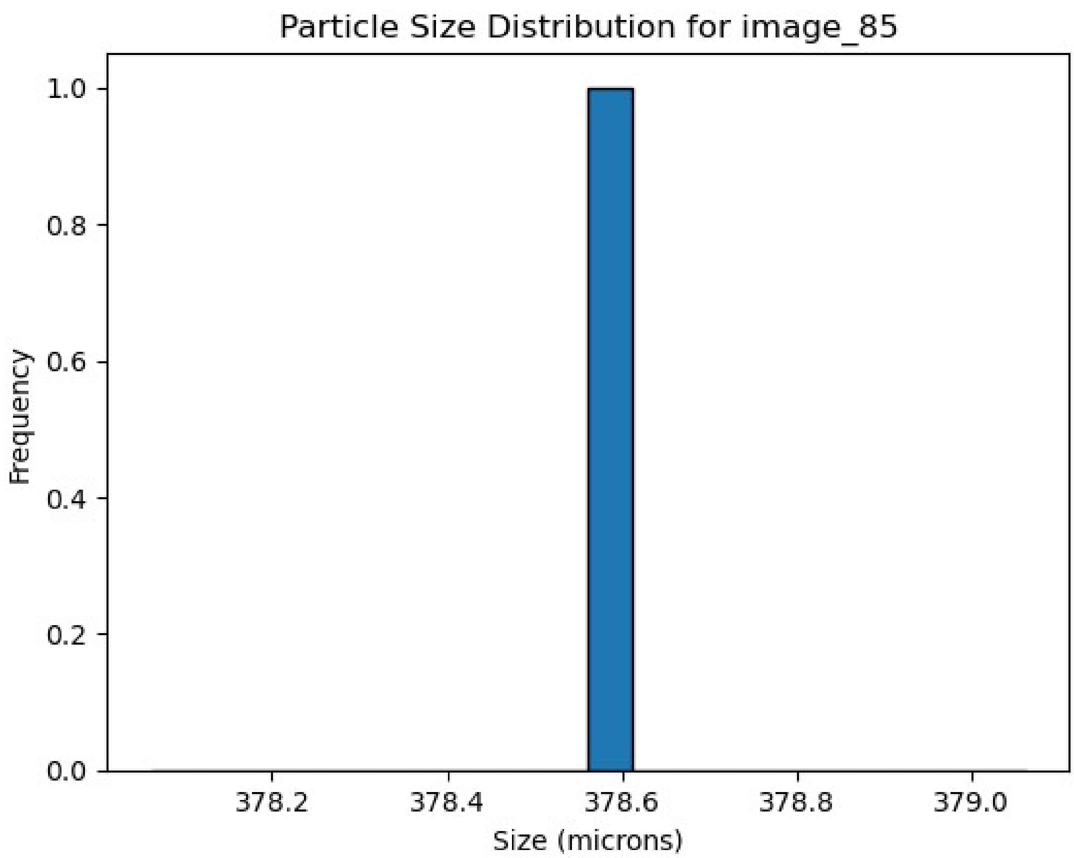

```
Particle Size Stats for image_82 - Mean: 5.580203292842521, Median:
5.170882945826411, Std Dev: 4.521230409487258
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



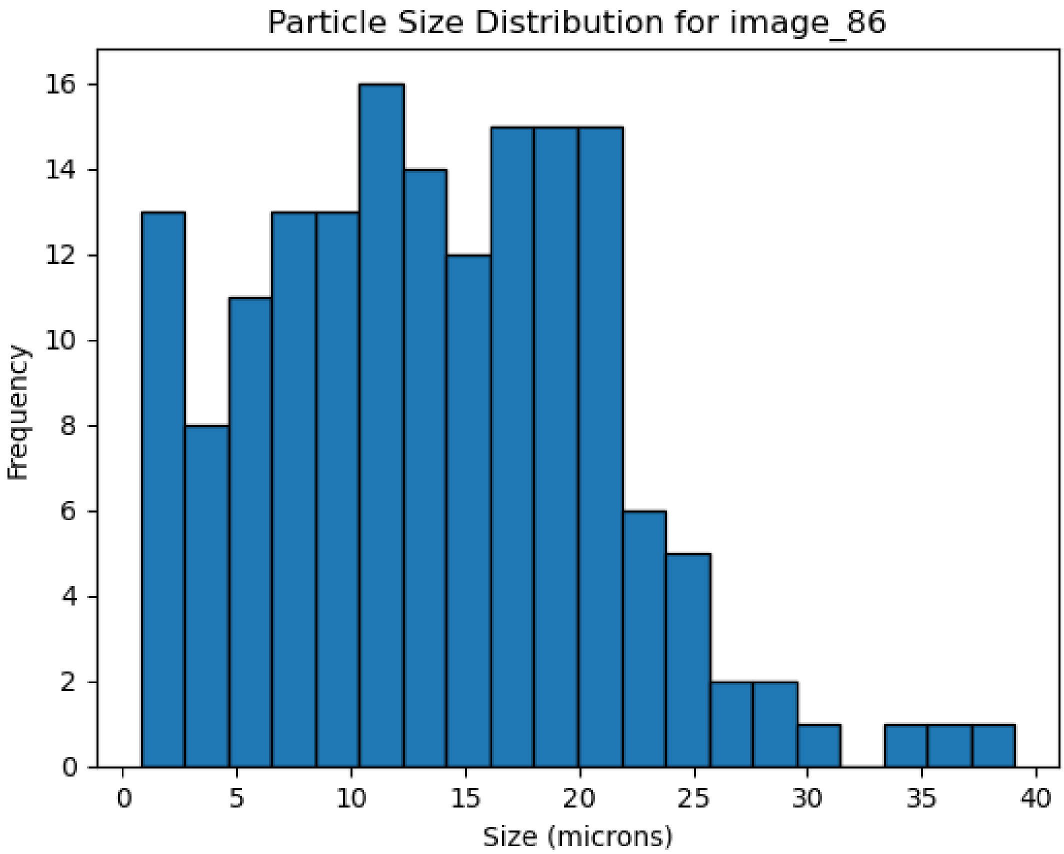

```
Particle Size Stats for image_83 - Mean: 159.09746116558566, Median:
159.09746116558566, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



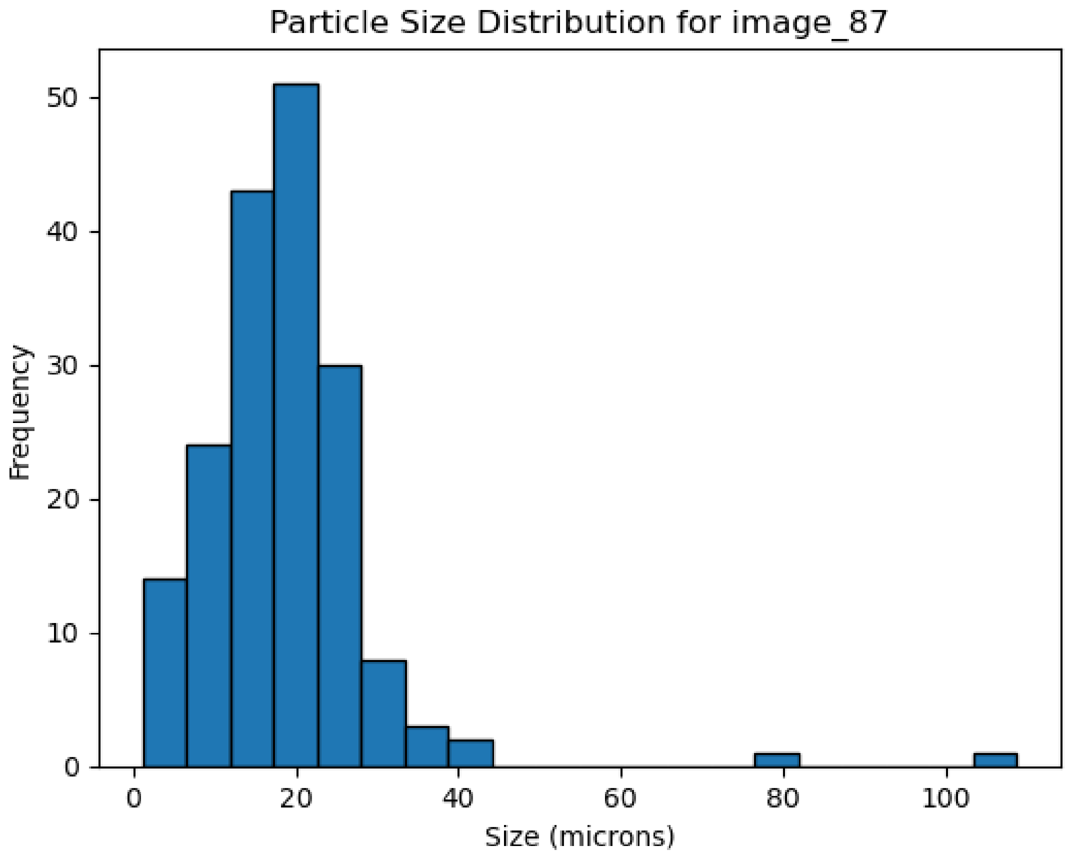

```
Particle Size Stats for image_84 - Mean: 20.88382961227589, Median:
20.49801962674961, Std Dev: 9.619163708901812
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



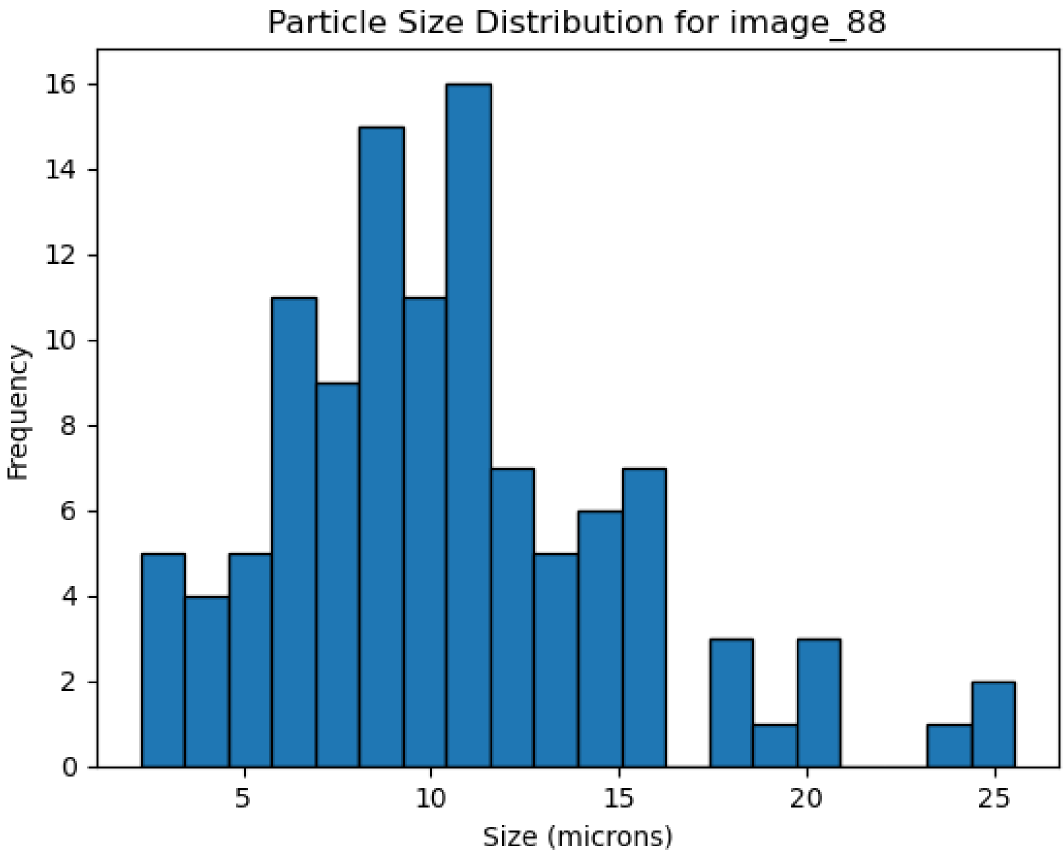

```
Particle Size Stats for image_85 - Mean: 378.56154101796164, Median:
378.56154101796164, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



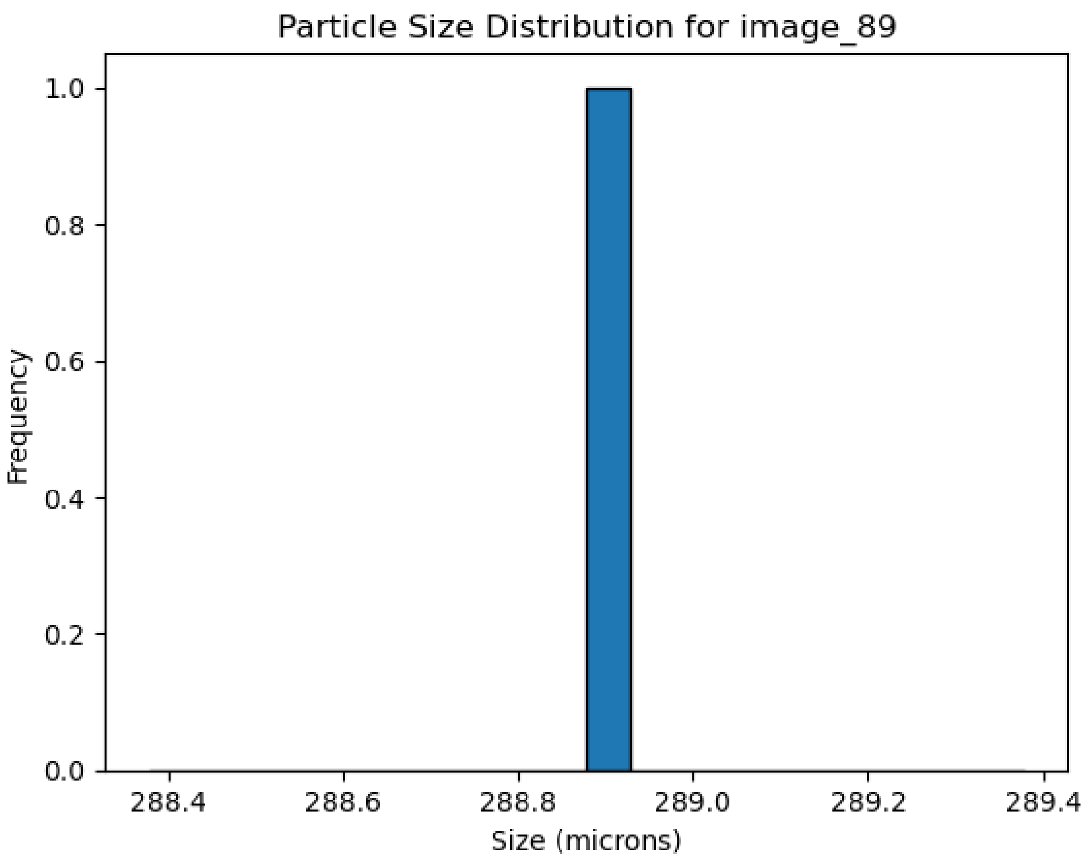

```
Particle Size Stats for image_86 - Mean: 13.76588586058154, Median:
13.374791901212825, Std Dev: 7.579875497772949
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



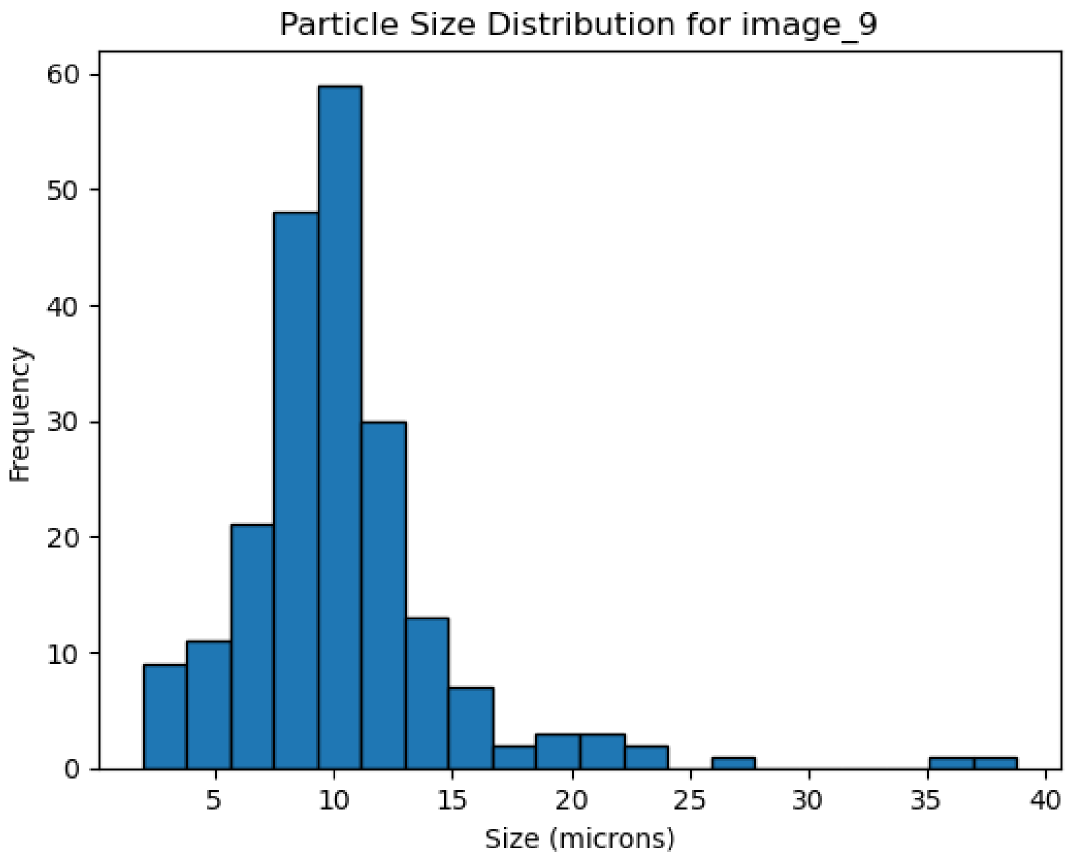

```
Particle Size Stats for image_87 - Mean: 18.484886352683038, Median:
18.29923496393627, Std Dev: 11.31058598785376
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



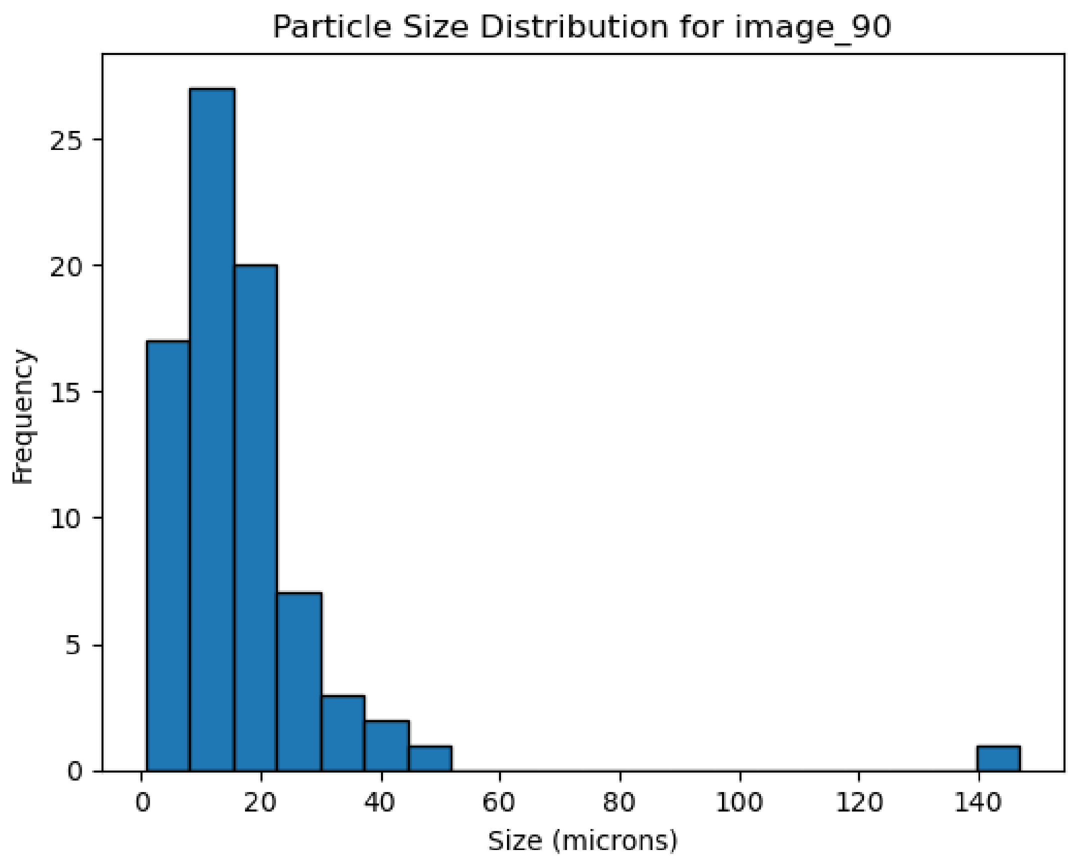

```
Particle Size Stats for image_88 - Mean: 10.525435680810737, Median:
9.965574970333758, Std Dev: 4.68114801427355
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



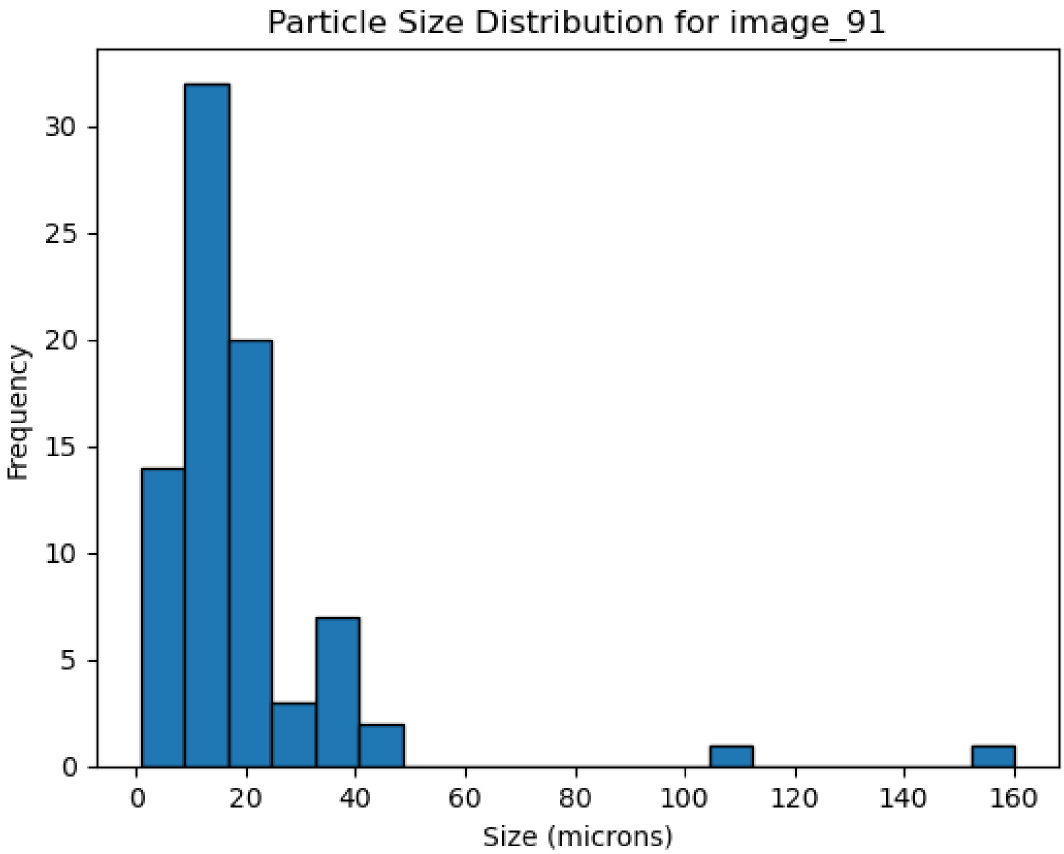

```
Particle Size Stats for image_89 - Mean: 288.8782896671746, Median:
288.8782896671746, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



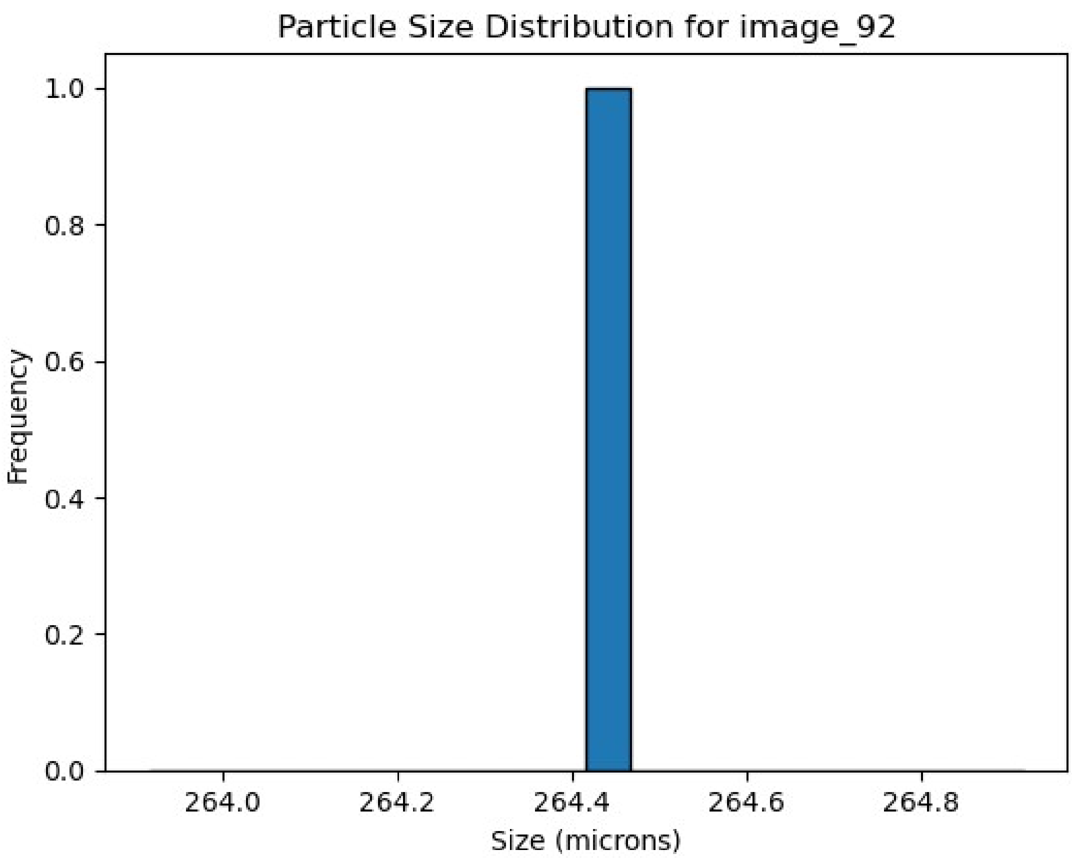

```
Particle Size Stats for image_9 - Mean: 10.272989641828971, Median:
9.739422266375435, Std Dev: 4.682779883474952
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



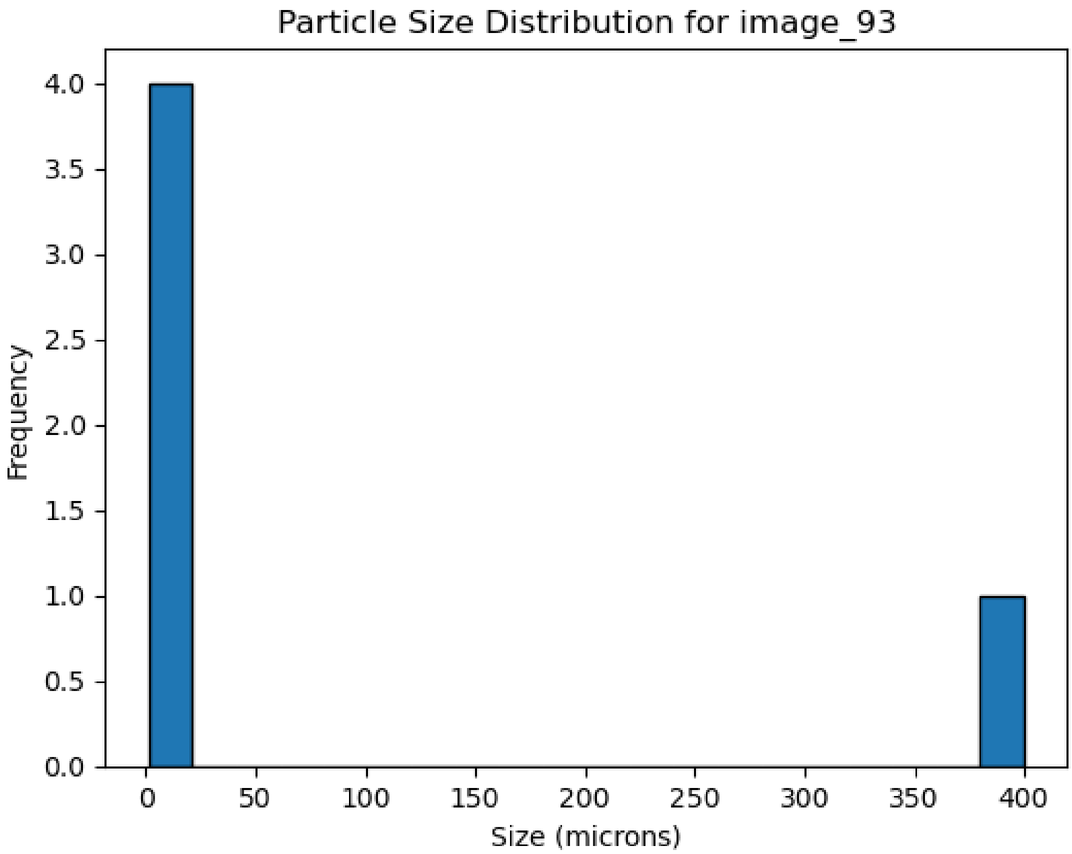

```
Particle Size Stats for image_90 - Mean: 17.04993057830179, Median:
13.747198480430622, Std Dev: 17.434444849495055
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



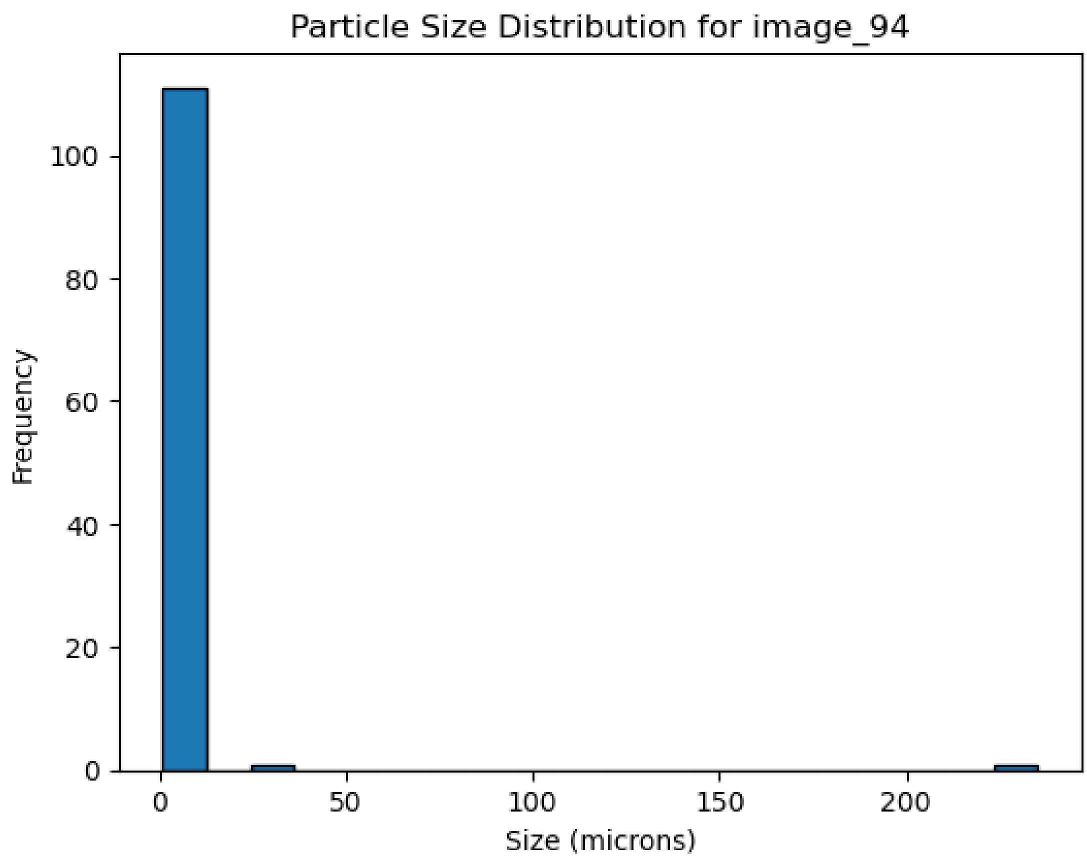

```
Particle Size Stats for image_91 - Mean: 19.45330215429201, Median:
16.11646602204536, Std Dev: 21.054736331086712
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



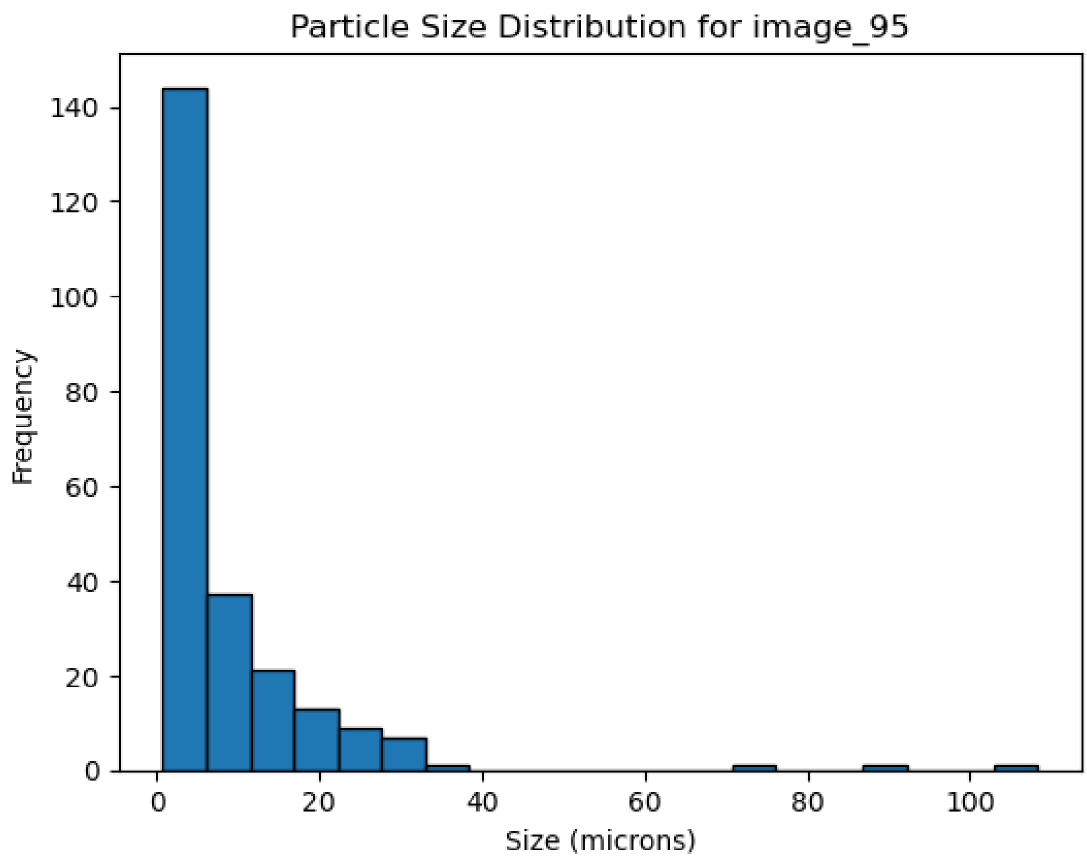

```
Particle Size Stats for image_92 - Mean: 264.4165839740338, Median:
264.4165839740338, Std Dev: 0.0
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



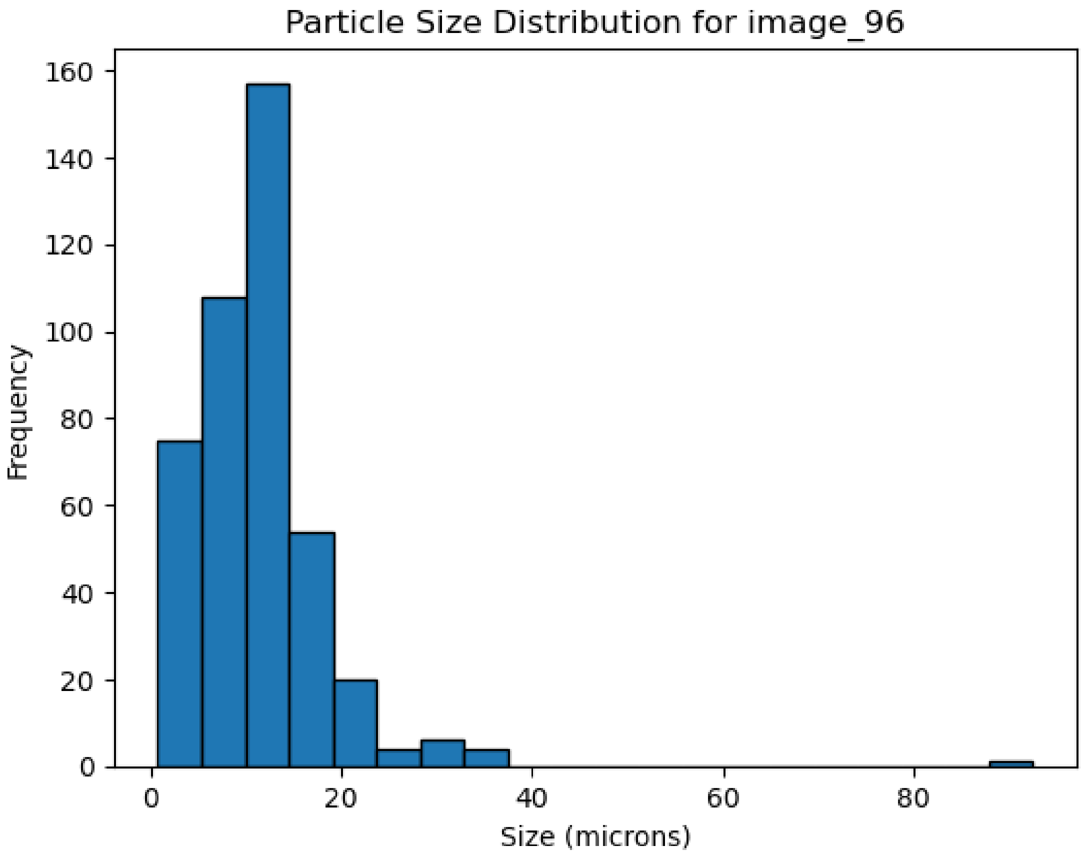

```
Particle Size Stats for image_93 - Mean: 84.42452105034639, Median:
5.4700209598571305, Std Dev: 157.74109070207263
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



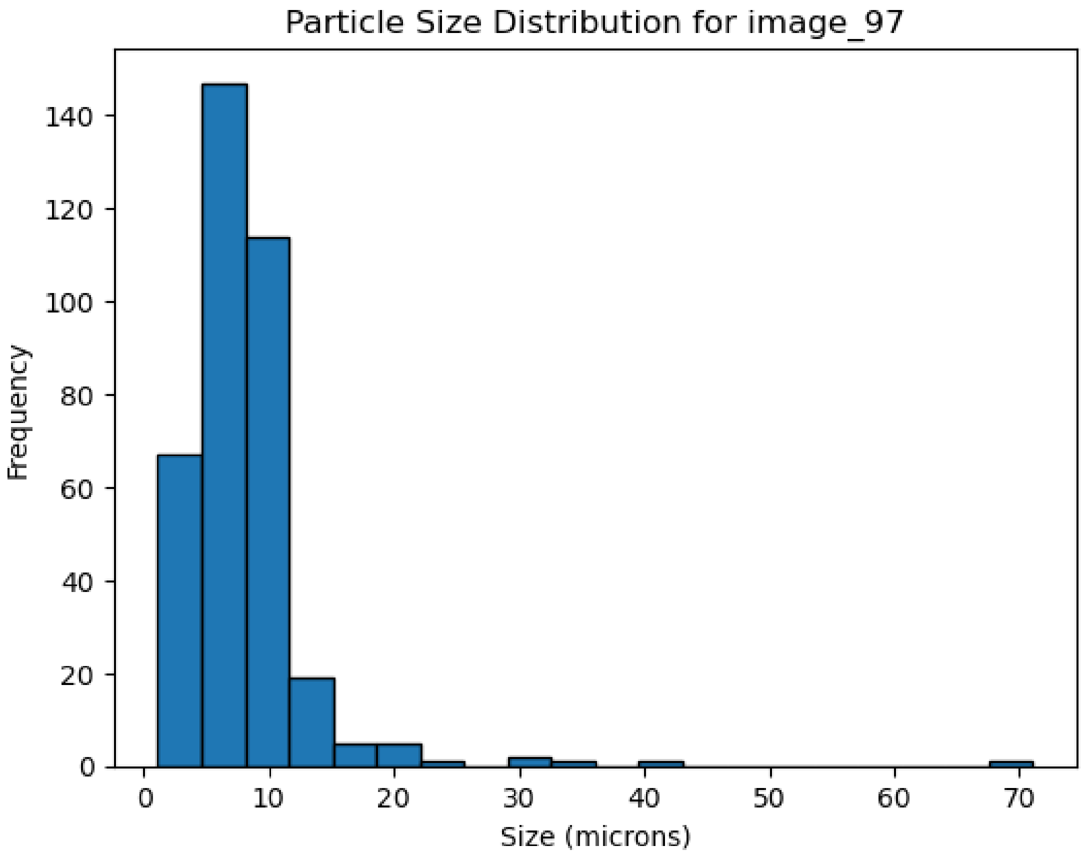

```
Particle Size Stats for image_94 - Mean: 5.640759448581952, Median:
2.876813695875796, Std Dev: 21.94080166791586
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



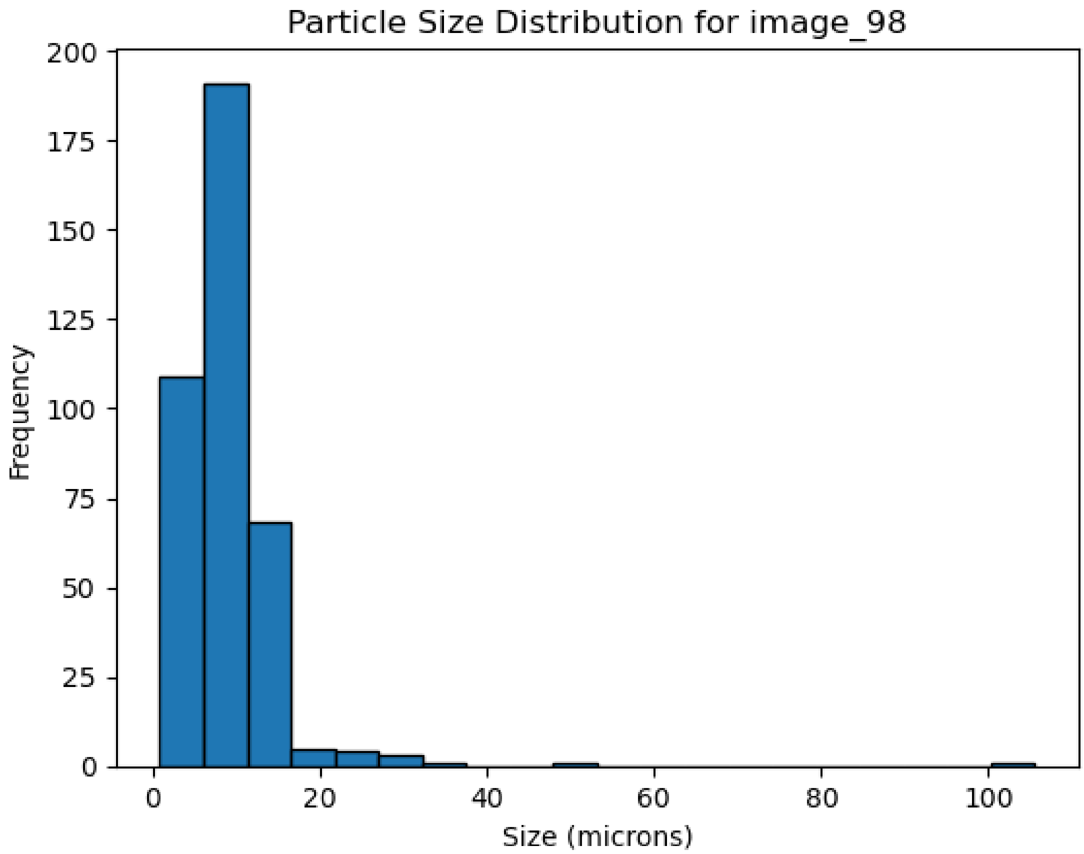

```
Particle Size Stats for image_95 - Mean: 8.47184462241942, Median:
4.145929793656026, Std Dev: 12.14244880542467
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```



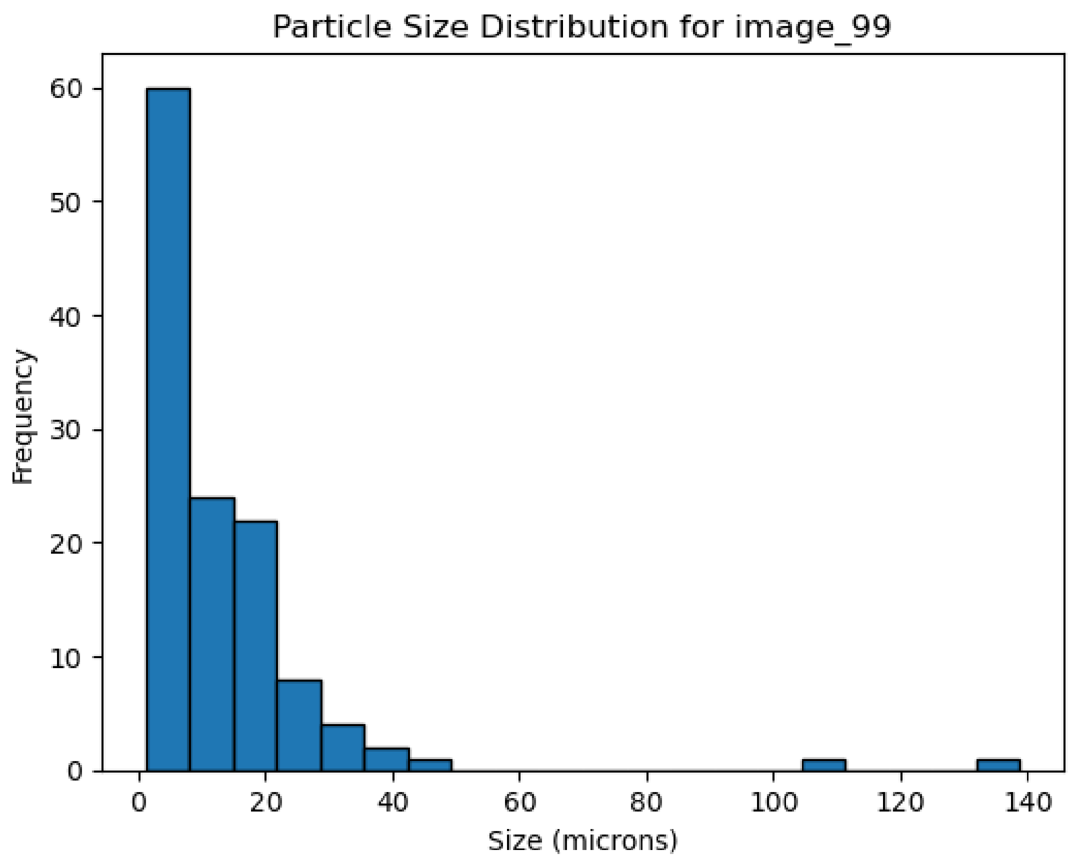

```
Particle Size Stats for image_96 - Mean: 11.211822737507205, Median:
10.704744696916627, Std Dev: 7.145742583380789
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```



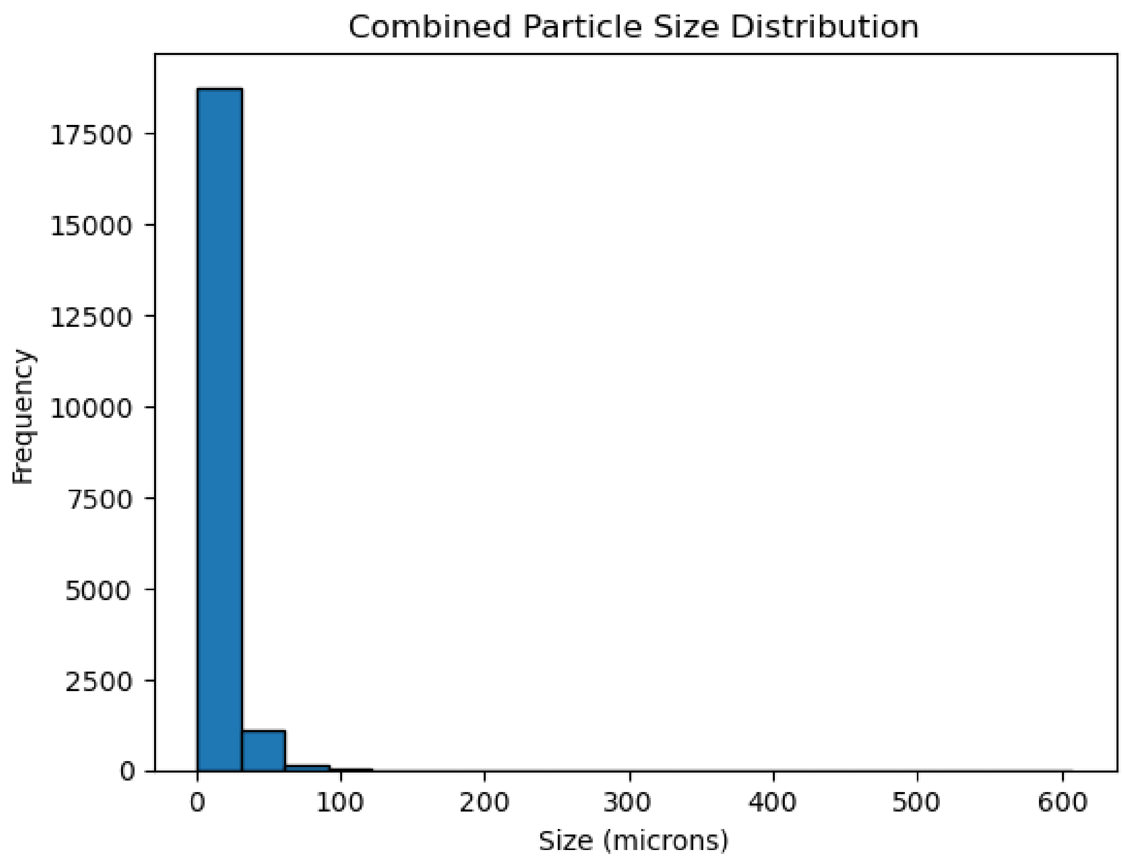

```
Particle Size Stats for image_97 - Mean: 8.038682093430333, Median:
7.569397566060481, Std Dev: 5.5343275601028346
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```

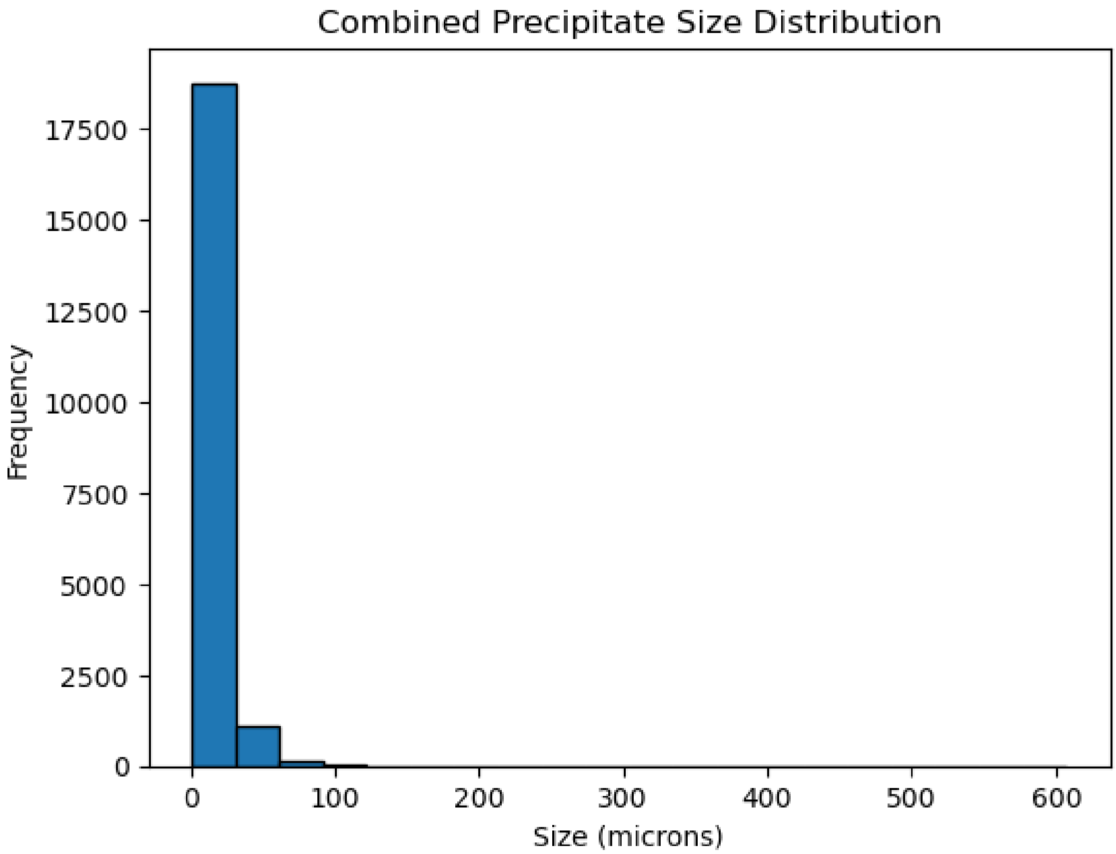



```
Particle Size Stats for image_98 - Mean: 9.151090443404211, Median:
8.51907589177983, Std Dev: 6.983732935436805
Fine-particle microstructure: Likely to have higher strength.
High number of defects detected: Material integrity may be
compromised.
```




```
Particle Size Stats for image_99 - Mean: 12.626095962681, Median:
8.51907589177983, Std Dev: 17.16588403306092
Coarse-particle microstructure: Likely to have lower strength.
High number of defects detected: Material integrity may be
compromised.
```


Combined Particle Size Distribution
Frequency
17500
15000
12500
10000
7500
5000
2500
0
0
100
200
300
400
500
600
Size (microns)



Images processing to extract particle size statistics, performs statistical analysis,

and saves the results to a CSV file

```
# Create a directory to save the CSV file if it doesn't exist
output_folder = Path(r"C:\Users\isaac\Downloads\coco - Copy6\
output_stats")
output_folder.mkdir(parents=True, exist_ok=True)

# Define the CSV file path
csv_file_path = output_folder / "particle_size_stats.csv"

# Initialize a list to store the stats for each image
particle_stats = []

# Process all images in the directory
image_paths = list(image_folder.glob("*.png"))  # images are .png
format

for image_path in image_paths:
    # Step 1: Preprocess the image
    preprocessed_image = preprocess_image(image_path)
```

```
    if preprocessed_image is None:
        continue  # Skip image if it couldn't be processed

    # Step 2: Segment the image
    segmented_image = segment_image(preprocessed_image)

    # Step 3: Extract geometrical features (particle sizes,
precipitate sizes)
    particle_areas, particle_diameters =
calculate_area_and_diameter(segmented_image)

    # Step 4: Perform statistical analysis
    mean, median, std_dev = compute_statistics(particle_diameters)

    # Store the stats in the list
    particle_stats.append({
        "Image Name": image_path.stem,
        "Mean Particle Size (microns)": mean,
        "Median Particle Size (microns)": median,
        "Standard Deviation (microns)": std_dev
    })

# Create a DataFrame from the stats list
df = pd.DataFrame(particle_stats)

# Save the DataFrame to a CSV file
df.to_csv(csv_file_path, index=False)

print(f"Particle size stats saved to {csv_file_path}")
```

```
Particle size stats saved to C:\Users\isaac\Downloads\coco - Copy6\
output_stats\particle_size_stats.csv
```

Splits image and mask datasets into training and testing sets, builds and trains a U-Net model

for image segmentation, evaluates its performance, and

visualizes the predicted segmentation mask on a test image.

```
from sklearn.model_selection import train_test_split

# Load the dataset from your image and mask directories
image_folder = r"C:\Users\isaac\Downloads\coco - Copy6\images\default"

output_folder = r"C:\Users\isaac\Downloads\coco - Copy6\mask$$$"

# Create directories for train/test splits
train_image_dir = 'train_images'
test_image_dir = 'test_images'
```

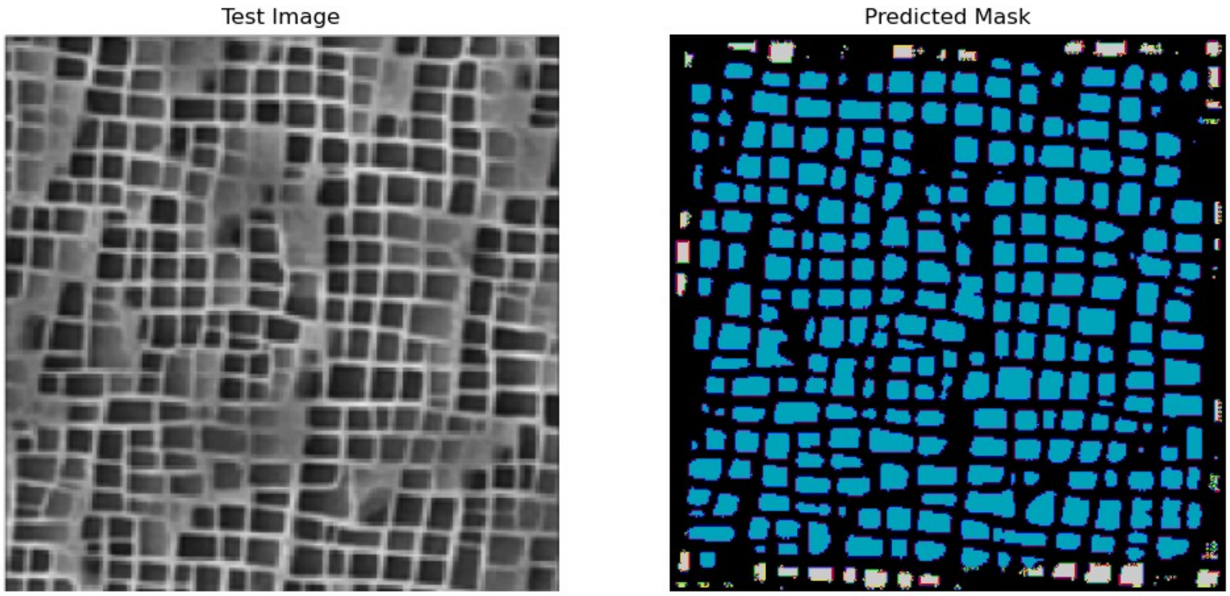
```
train_mask_dir = 'train_masks'
test_mask_dir = 'test_masks'

# Create the directories if they do not exist
os.makedirs(train_image_dir, exist_ok=True)
os.makedirs(test_image_dir, exist_ok=True)
os.makedirs(train_mask_dir, exist_ok=True)
os.makedirs(test_mask_dir, exist_ok=True)

# Load images
image_files = sorted(os.listdir(image_folder))
mask_files = sorted(os.listdir(output_folder))

train_images = []
train_masks = []
test_images = []
test_masks = []

for img_file, mask_file in zip(image_files, mask_files):
    img = load_img(os.path.join(image_folder, img_file),
target_size=(256, 256))
    mask = load_img(os.path.join(output_folder, mask_file),
target_size=(256, 256), color_mode="grayscale")

    img = img_to_array(img) / 255.0  # Normalize the images
    mask = img_to_array(mask)  # Convert the mask to array (grayscale)

    # Ensure the mask values are within the range [0, 3]
    mask = np.clip(mask, 0, 3)

    train_images.append(img)
    train_masks.append(mask)

    # Copy the image and mask to the corresponding directories
    shutil.copy(os.path.join(image_folder, img_file), train_image_dir)
    shutil.copy(os.path.join(output_folder, mask_file),
train_mask_dir)

# Convert to numpy arrays
train_images = np.array(train_images)
train_masks = np.array(train_masks)

# Convert masks to one-hot encoding
train_masks_one_hot = tf.keras.utils.to_categorical(train_masks,
num_classes=4)

# Split the data into training and testing
test_size = 0.2
train_images, test_images, train_masks_one_hot, test_masks_one_hot =
train_test_split(
```

```
    train_images, train_masks_one_hot, test_size=test_size,
random_state=42
)

# Save test images and masks to the corresponding directories
for i in range(len(test_images)):
    # Assuming the filenames are similar in both image and mask
directories
    img_filename = f"test_image_{i+1}.png"
    mask_filename = f"test_mask_{i+1}.png"

    shutil.copy(os.path.join(image_folder, image_files[i]),
test_image_dir)
    shutil.copy(os.path.join(output_folder, mask_files[i]),
test_mask_dir)

# Define the U-Net model
def unet_model(input_size=(256, 256, 3)):
    inputs = layers.Input(input_size)

    # Contracting path
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')
(inputs)
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')
(c1)
    p1 = layers.MaxPooling2D((2, 2))(c1)

    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')
(p1)
    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')
(c2)
    p2 = layers.MaxPooling2D((2, 2))(c2)

    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')
(p2)
    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')
(c3)
    p3 = layers.MaxPooling2D((2, 2))(c3)

    # Bottleneck
    c4 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')
(p3)
    c4 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')
(c4)

    # Expansive path
    u5 = layers.Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(c4)
    u5 = layers.concatenate([u5, c3])
    c5 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')
```

```
(u5)
    c5 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')
(c5)

    u6 = layers.Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(c5)
    u6 = layers.concatenate([u6, c2])
    c6 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')
(u6)
    c6 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')
(c6)

    u7 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(c6)
    u7 = layers.concatenate([u7, c1])
    c7 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')
(u7)
    c7 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')
(c7)

    outputs = layers.Conv2D(4, (1, 1), activation='softmax')(c7)

    model = tf.keras.models.Model(inputs, outputs)
    return model

# Instantiate and compile the U-Net model
model = unet_model()
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(
    train_images, train_masks_one_hot,
    validation_data=(test_images, test_masks_one_hot),
    epochs=10,
    batch_size=8
)

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_images, test_masks_one_hot)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_acc}")
```

```
Epoch 1/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 72s 7s/step - accuracy: 0.5772 - loss:
1.1607 - val_accuracy: 0.7171 - val_loss: 0.8736
Epoch 2/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 64s 6s/step - accuracy: 0.6758 - loss:
0.9115 - val_accuracy: 0.7249 - val_loss: 0.7445
Epoch 3/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 64s 6s/step - accuracy: 0.6687 - loss:
```

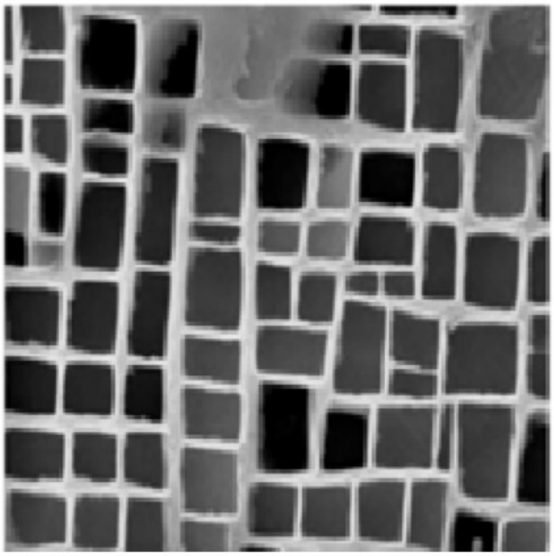
```
0.8674 - val_accuracy: 0.7155 - val_loss: 0.8839
Epoch 4/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 65s 7s/step - accuracy: 0.6668 - loss:
0.8902 - val_accuracy: 0.7352 - val_loss: 0.7234
Epoch 5/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 132s 14s/step - accuracy: 0.6903 - loss:
0.8074 - val_accuracy: 0.7375 - val_loss: 0.7035
Epoch 6/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 67s 7s/step - accuracy: 0.7221 - loss:
0.7485 - val_accuracy: 0.7439 - val_loss: 0.7053
Epoch 7/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 55s 5s/step - accuracy: 0.7093 - loss:
0.7872 - val_accuracy: 0.7497 - val_loss: 0.6797
Epoch 8/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 53s 5s/step - accuracy: 0.7187 - loss:
0.7510 - val_accuracy: 0.7541 - val_loss: 0.7074
Epoch 9/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 54s 5s/step - accuracy: 0.6929 - loss:
0.7946 - val_accuracy: 0.7612 - val_loss: 0.6411
Epoch 10/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 52s 5s/step - accuracy: 0.7312 - loss:
0.7256 - val_accuracy: 0.7622 - val_loss: 0.6406
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.7622 - loss: 0.6406
Test Loss: 0.6406146287918091, Test Accuracy: 0.7621574401855469

# Make predictions on the test image
test_image = test_images[0]  # Use the first test image
predicted_mask = np.argmax(model.predict(np.expand_dims(test_image,
axis=0)), axis=-1)[0]

# Visualize the test image and predicted mask
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(test_image)
plt.title("Test Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(predicted_mask, cmap='nipy_spectral')
plt.title("Predicted Mask")
plt.axis("off")

plt.show()

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 985ms/step
```




```
# Make predictions on the test image
test_image = test_images[17]  # Use the first test image
predicted_mask = np.argmax(model.predict(np.expand_dims(test_image,
axis=0)), axis=-1)[0]

# Visualize the test image and predicted mask
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(test_image)
plt.title("Test Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(predicted_mask, cmap='nipy_spectral')
plt.title("Predicted Mask")
plt.axis("off")

plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```

Test Image



Predicted Mask