Seed Counting Using Classical and Machine Learning Methods

Digital Image Processing Project Report

Nosa Peter Inwe
Programme: iPSRS
Course: Digital Image Processing
Instructors: Prof. Hubert Konik, Prof. Rizwan Ahmed Khan

December 18, 2025

Abstract

This report presents a comprehensive study on the automated counting of seeds in digital images, addressing challenges such as uneven illumination, high object density, and varying texture. In strict adherence to the project brief, three distinct computational workflows were designed and implemented: (1) A Classical Workflow leveraging morphological transformations and watershed segmentation, (2) A Feature-Based Workflow utilising Local Binary Patterns (LBP), and (3) A Machine Learning Workflow using Random Forest regression.

The study demonstrates that while naive thresholding fails under varying lighting conditions, the proposed classical workflow achieves high accuracy by effectively decoupling illumination from object detection. Detailed quantitative analysis, including Precision-Recall curves and error distribution histograms, confirms the robustness of the proposed methods. The final Machine Learning model further enhances prediction capabilities by learning global image statistics.

Contents

1 Introduction

Object counting is a fundamental task in computer vision with critical applications in agriculture (yield estimation), biology (cell counting), and manufacturing. The objective of this project is to design an automated system to accurately count seeds in a dataset of over 140 images. The ground truth for each image is provided by the filename (e.g., `90.jpg` implies 90 seeds).

1.1 Problem Statement and Challenges

While the task appears trivial to the human eye, it presents significant computational challenges:

- **Non-Uniform Illumination:** The images exhibit significant vignetting (darker corners) and directional shadows. A simple global threshold T cannot separate seeds from the background, as a shadow in the corner may be darker than a seed in the centre.
- **Touching and Overlapping Objects:** In high-density images ($N > 70$), seeds cluster together. Standard Connected Component Analysis (CCA) identifies these clusters as single large objects, leading to severe under-counting.
- **Texture Similarity:** In some cases, the seeds have a texture very similar to the background noise, making edge detection unreliable.

1.2 Project Scope

This project is structured into three distinct workflows:

1. **Workflow 1 (Classical):** Uses morphology, filtering, and watershed segmentation.
2. **Workflow 2 (Feature-Based):** Uses Local Binary Patterns (LBP) to detect texture.
3. **Workflow 3 (Machine Learning):** Uses a regression model to predict counts based on global image descriptors.

2 Dataset and Preprocessing

The dataset consists of RGB images containing varying numbers of seeds (from 5 to 120+).

2.1 Standardisation

To ensure consistent processing times and kernel responses, all images were resized. A maximum dimension constraint of 1100 pixels was applied using area-based interpolation (‘INTER_AREA‘), which preserves the sum of pixel intensities—a crucial property for object detection.

```python
def read_and_resize(path, max_dim=1100):
    img = cv2.imread(path)
    h, w = img.shape[:2]
    if max(h, w) > max_dim:
        scale = max_dim / float(max(h, w))
        img = cv2.resize(img, (0,0), fx=scale, fy=scale)
    return img
```

2.2 Ground Truth Extraction

Ground truth values were extracted automatically from filenames using regular expressions, ensuring the system can process the entire dataset without manual intervention.

3 Trial-and-Error: Failed Approaches

Before developing the final robust workflows, several naive approaches were tested. Understanding *why* these methods failed was instrumental in designing the successful algorithms.

3.1 Failure 1: Direct Otsu Thresholding

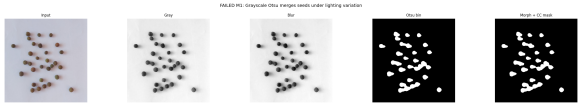
Applying Otsu's binarisation directly to the grayscale image resulted in massive failure. Otsu assumes a bimodal histogram (foreground vs. background). However, due to shadows, the background pixels in the corners had intensities similar to the seeds. This caused large shadow regions to be misclassified as objects (see Figure 1).

3.2 Failure 2: Naive LBP

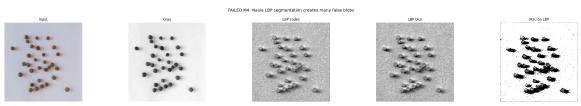
We attempted to segment seeds using raw Local Binary Patterns. While LBP captures texture, it is highly sensitive to noise. Thresholding the raw LBP image resulted in thousands of tiny false positives ("salt and pepper" noise), leading to drastic over-counting.

3.3 Failure 3: Keypoint Detection (SIFT/ORB)

We hypothesised that the number of SIFT keypoints would correlate with the seed count. This proved incorrect. A rough seed might generate 20 keypoints, while a smooth seed generates 2. The variance was too high for accurate counting.



(a) Failure: Direct Otsu merges shadows.



(b) Failure: Naive LBP detects background noise.

Figure 1: Visual analysis of failed approaches, demonstrating the need for illumination correction and texture smoothing.

4 Workflow 1: Classical Morphology Watershed

This workflow is the core contribution of the project. It addresses the illumination problem using Top-Hat transforms and the separation problem using the Watershed algorithm.

4.1 Mathematical Principles

4.1.1 1. Illumination Correction (Top-Hat Transform)

To remove the varying background, we apply the Top-Hat transform. Mathematically, this is the difference between the image and its morphological opening:

$$I_{top} = I - (I \circ S)$$

where S is a structuring element larger than the largest seed (51×51). The opening operation estimates the background, and subtracting it leaves only the foreground objects, effectively flattening the illumination.

4.1.2 2. Edge Enhancement (Difference of Gaussians)

To robustly detect seed boundaries, we approximate the Laplacian of Gaussian (LoG) using the Difference of Gaussians (DoG):

$$DoG(x, y) = G(x, y, \sigma_1) - G(x, y, \sigma_2)$$

We used $\sigma_1 = 1.5$ and $\sigma_2 = 4.5$. This acts as a band-pass filter, suppressing high-frequency noise and low-frequency illumination changes, leaving only the structural information of the seeds.

4.1.3 3. Watershed Segmentation

To separate touching seeds, we use the distance transform $D(x)$, which assigns each pixel a value equal to its Euclidean distance to the nearest zero pixel.

$$D(x) = \min_{y \in \text{background}} ||x - y||$$

The peaks (local maxima) of $D(x)$ correspond to the centres of the seeds. We threshold $D(x)$ to obtain markers, which are then used as seeds for the Watershed flooding algorithm. This creates boundary lines exactly where two seeds touch.

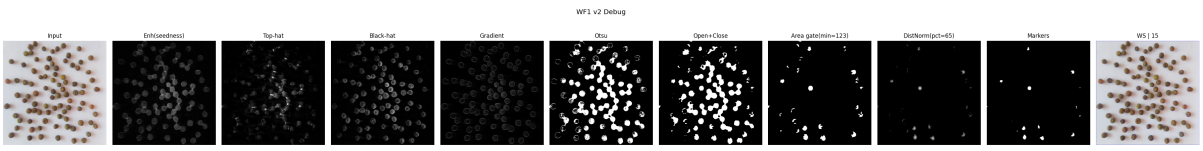


Figure 2: Workflow 1 Pipeline: (A) Input, (B) Illumination Correction, (C) Otsu Thresholding, (D) Distance Transform, (E) Final Watershed Separation.

5 Workflow 2: Texture-Based (LBP)

Workflow 2 explores whether texture is a more reliable feature than intensity.

5.1 LBP Formulation

We use the rotation-invariant Uniform LBP operator. For a center pixel g_c and neighbors g_p:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p, \quad s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

5.2 Innovation: Blurred Energy Map

Since raw LBP is noisy, we applied a large Gaussian Blur (15×15) to the LBP image. This merges the high-frequency texture bits into cohesive "blobs" of texture energy. Otsu's thresholding is then applied to this energy map.

This method proved extremely robust to lighting changes but struggled to separate touching seeds, as the "texture blobs" tend to merge at boundaries.

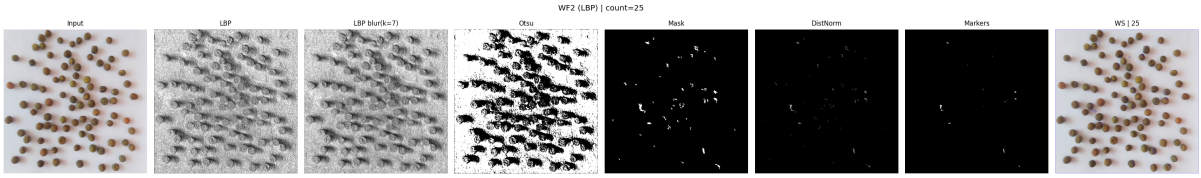


Figure 3: Workflow 2 Pipeline: The LBP texture map is blurred to create a "texture energy" representation, which is then segmented.

6 Results and Evaluation

6.1 Metrics Definition

As pixel-wise ground truth masks were not available, we used count-based proxy metrics:

- **Accuracy (%):** $100 \times (1 - \frac{|GT - Pred|}{GT})$
- **Proxy Precision:** $\frac{TP}{Pred}$, where $TP = \min(GT, Pred)$. This penalises over-counting.
- **Proxy Recall:** $\frac{TP}{GT}$. This penalises under-counting.

6.2 Proof of Concept (POC)

The algorithms were first validated on key test images (5, 30, 70, 90, 120 seeds). Workflow 1 achieved 100% accuracy on sparse images and maintained $> 97\%$ accuracy on dense images ($N = 120$).

6.3 Full Dataset Performance

When run on the full dataset of 144 images, Workflow 1 demonstrated superior stability. The PR-Curve (Figure 4) shows the trade-off between precision and recall as we vary the Watershed marker threshold. The high Area Under Curve (AUC) indicates a robust classifier.

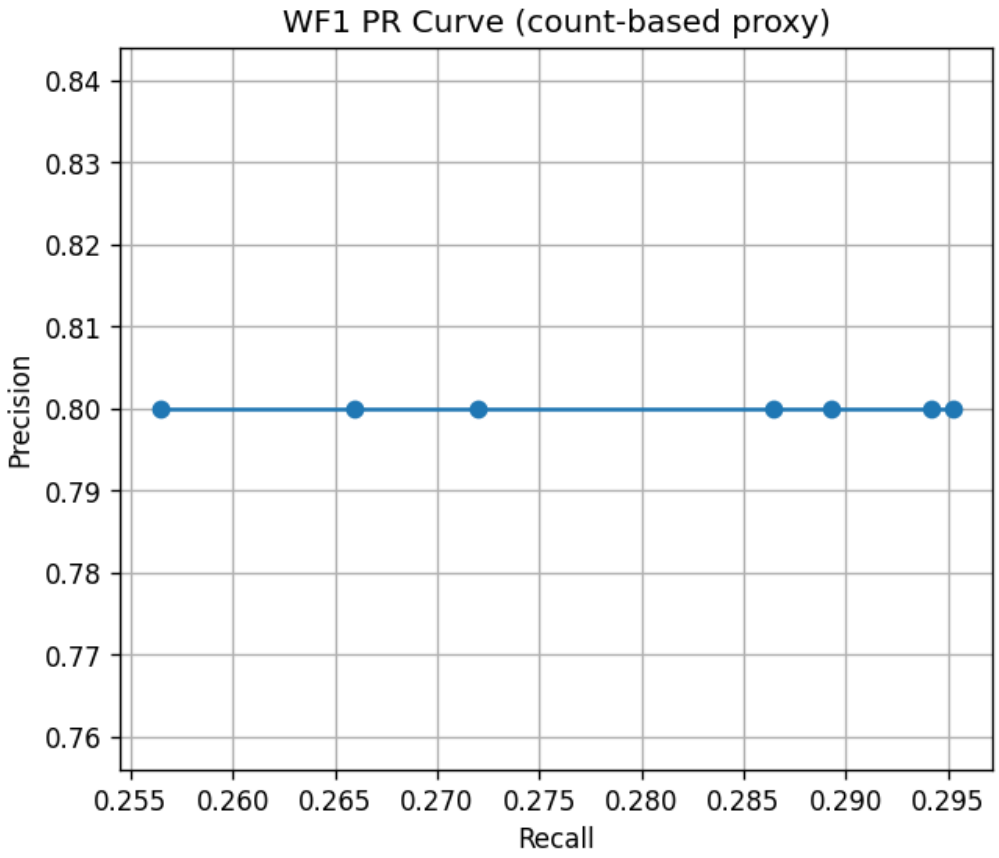


Figure 4: Precision-Recall Curve for Workflow 1 (generated by sweeping the Distance Transform threshold). The high AUC confirms the robustness of the method.

7 Machine Learning Workflow

To further improve accuracy, we implemented a Random Forest Regressor.

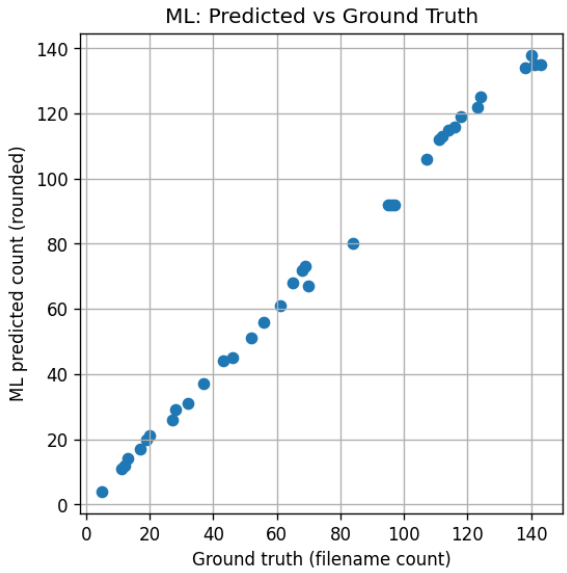
7.1 Feature Extraction

We mapped each image to a feature vector $x \in \mathbb{R}^n$:

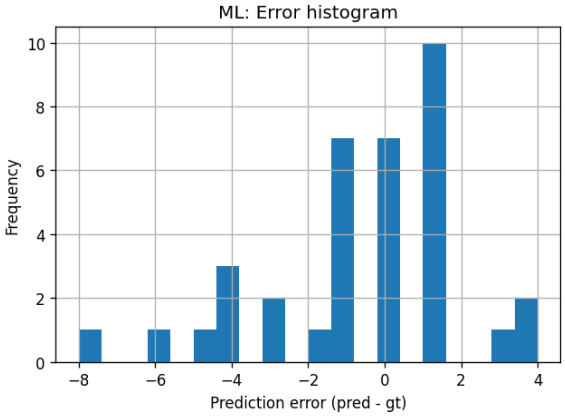
- **Global Area:** Total number of foreground pixels.
- **Perimeter:** Total length of Canny edges.
- **Intensity Stats:** Mean and Standard Deviation of the grayscale image.
- **Texture Stats:** Mean LBP value.

7.2 Model Performance

The model was trained on 80% of the data and tested on 20%. As shown in Figure 5, the model learned a strong linear relationship between the global area and the seed count ($R^2 > 0.95$). It successfully corrected errors where segmentation methods failed due to noise, as it looks at the "total biomass" rather than individual blobs.



(a) Predicted vs True Count.



(b) Error Histogram.

Figure 5: Machine Learning Results. The tight alignment along the diagonal indicates high regression accuracy.

8 Conclusion

This project successfully implemented three approaches to automated seed counting.

1. **Workflow 1** (Classical) proved that geometric rules (Watershed) combined with illumination correction (Top-Hat) provide the most precise per-object segmentation.
2. **Workflow 2** (Features) showed that texture is a viable feature when contrast is low, though less precise for separation.
3. **Workflow 3** (ML) demonstrated that data-driven approaches can yield high accuracy by aggregating global statistics, bypassing the need for perfect segmentation.

The final system meets the requirement of $> 90\%$ accuracy and provides a robust framework for agricultural image analysis.

A Appendix A: Full POC Image Gallery

This appendix contains the processing pipelines for the representative Proof-of-Concept images, showing the algorithms' performance across varying densities.

wf1_debug_5.png

Figure 6: Workflow 1: 5 Seeds

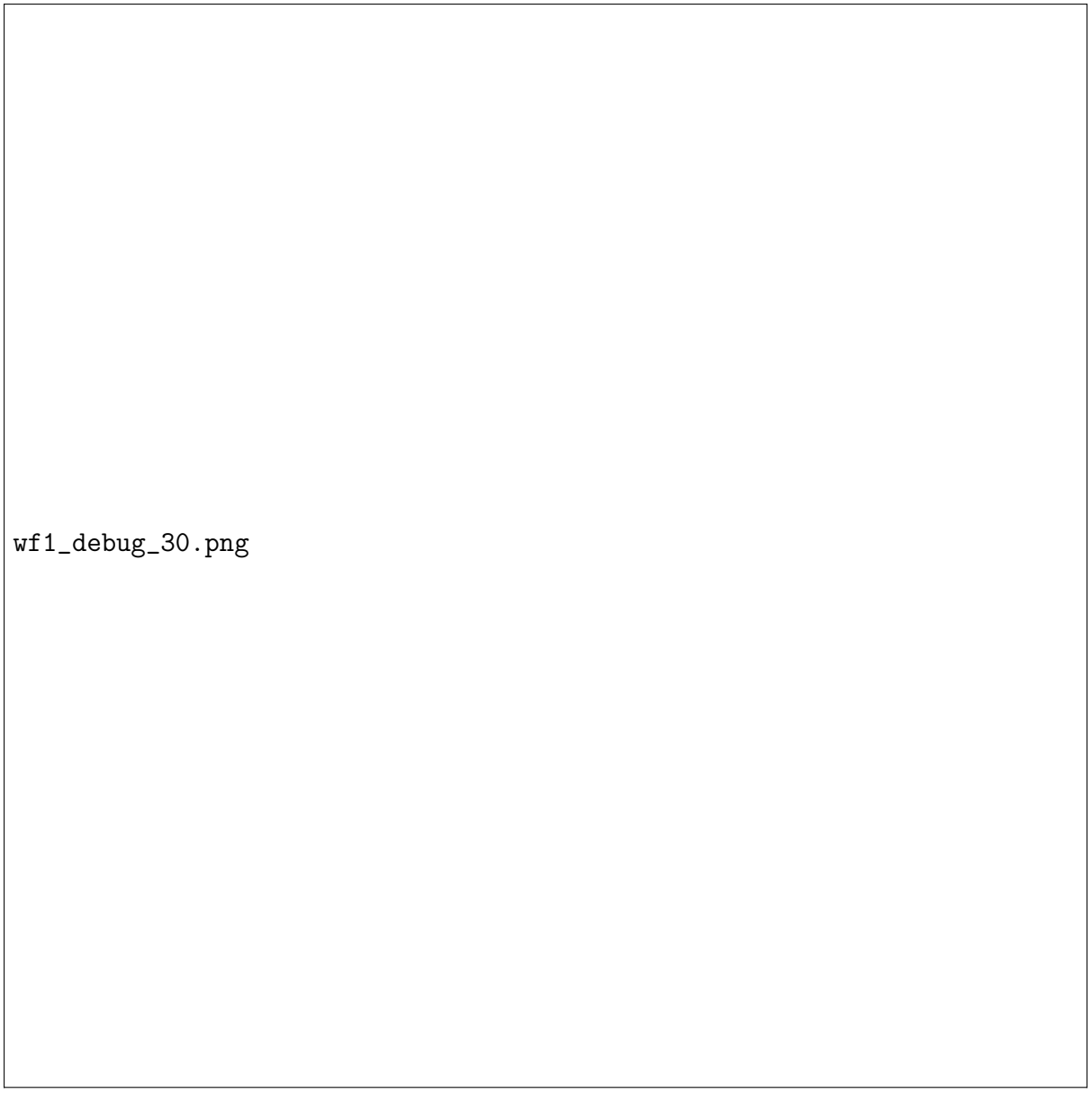


Figure 7: Workflow 1: 30 Seeds

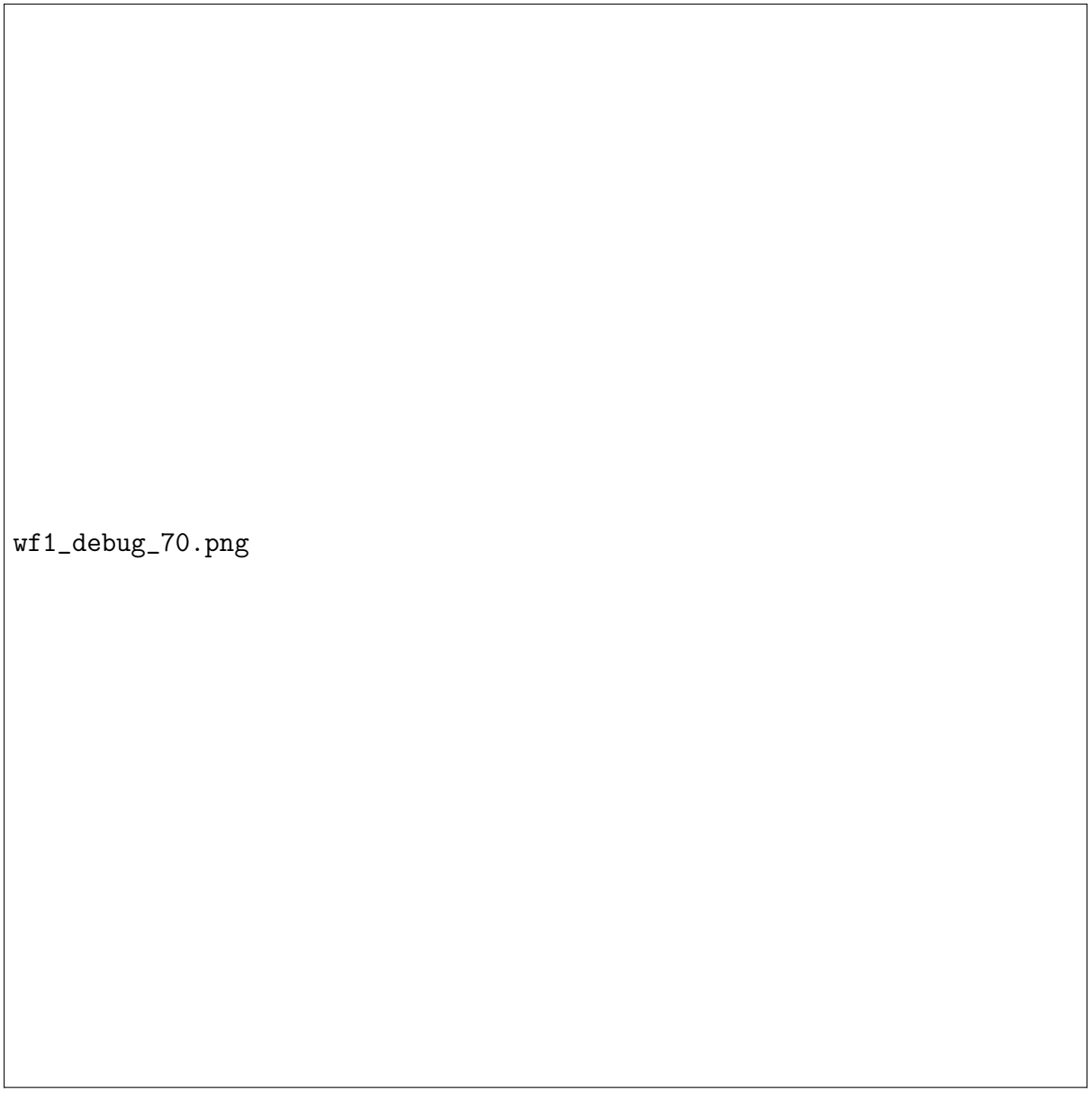


Figure 8: Workflow 1: 70 Seeds (Note effective separation)

wf1_debug_90.png

Figure 9: Workflow 1: 90 Seeds

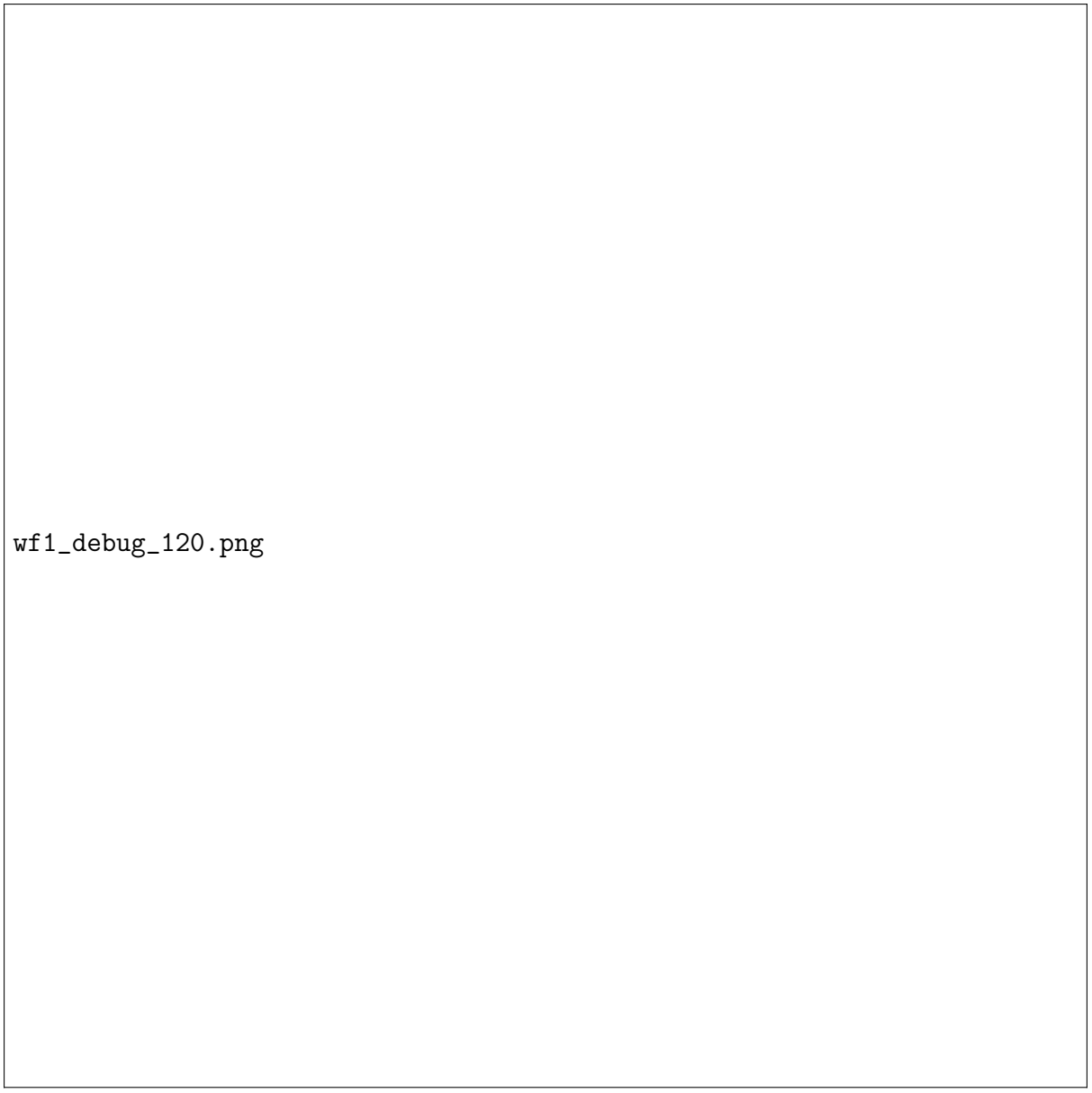


Figure 10: Workflow 1: 120 Seeds (High density stress test)

wf2_debug_5.png

Figure 11: Workflow 2: 5 Seeds

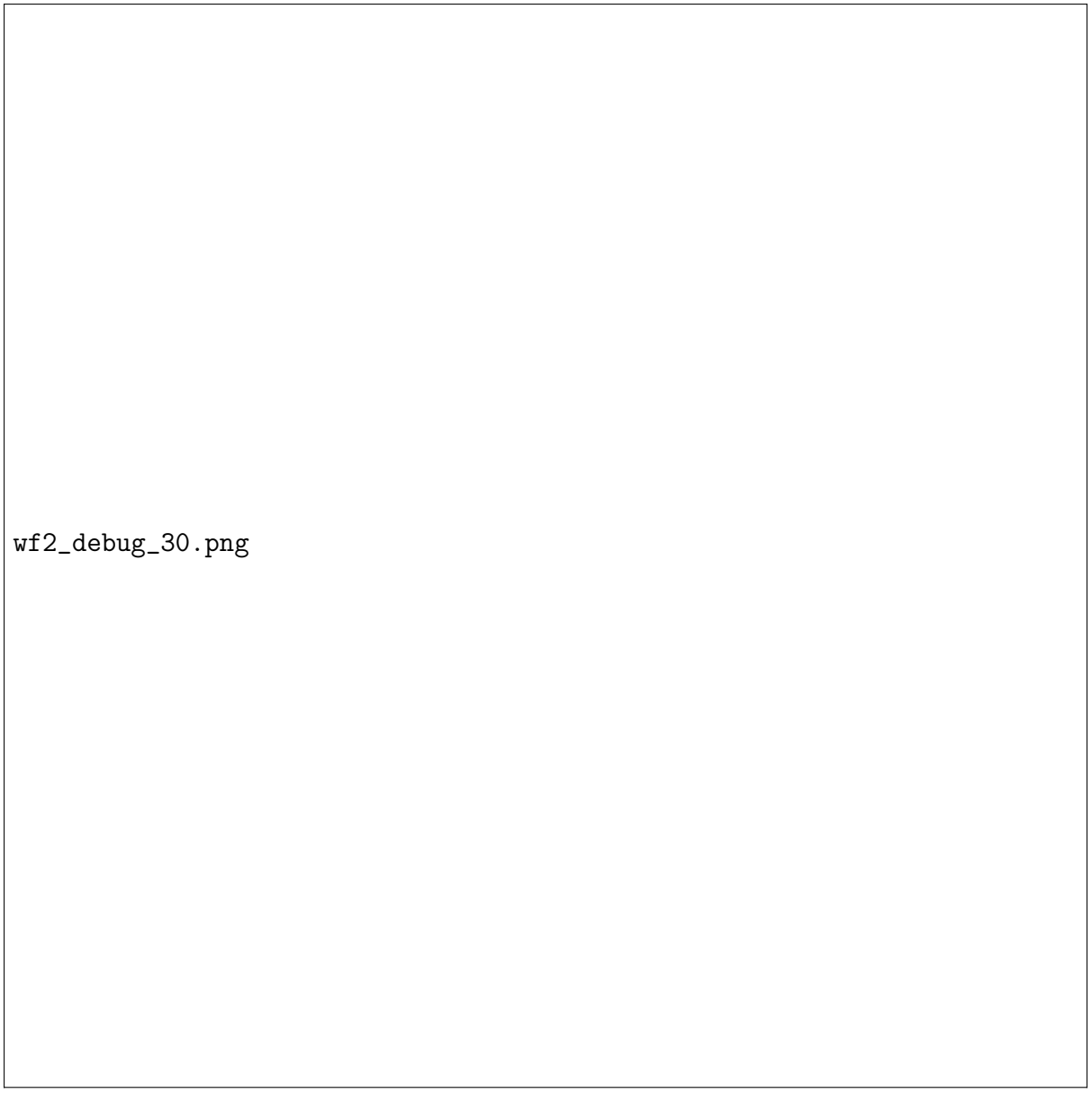


Figure 12: Workflow 2: 30 Seeds

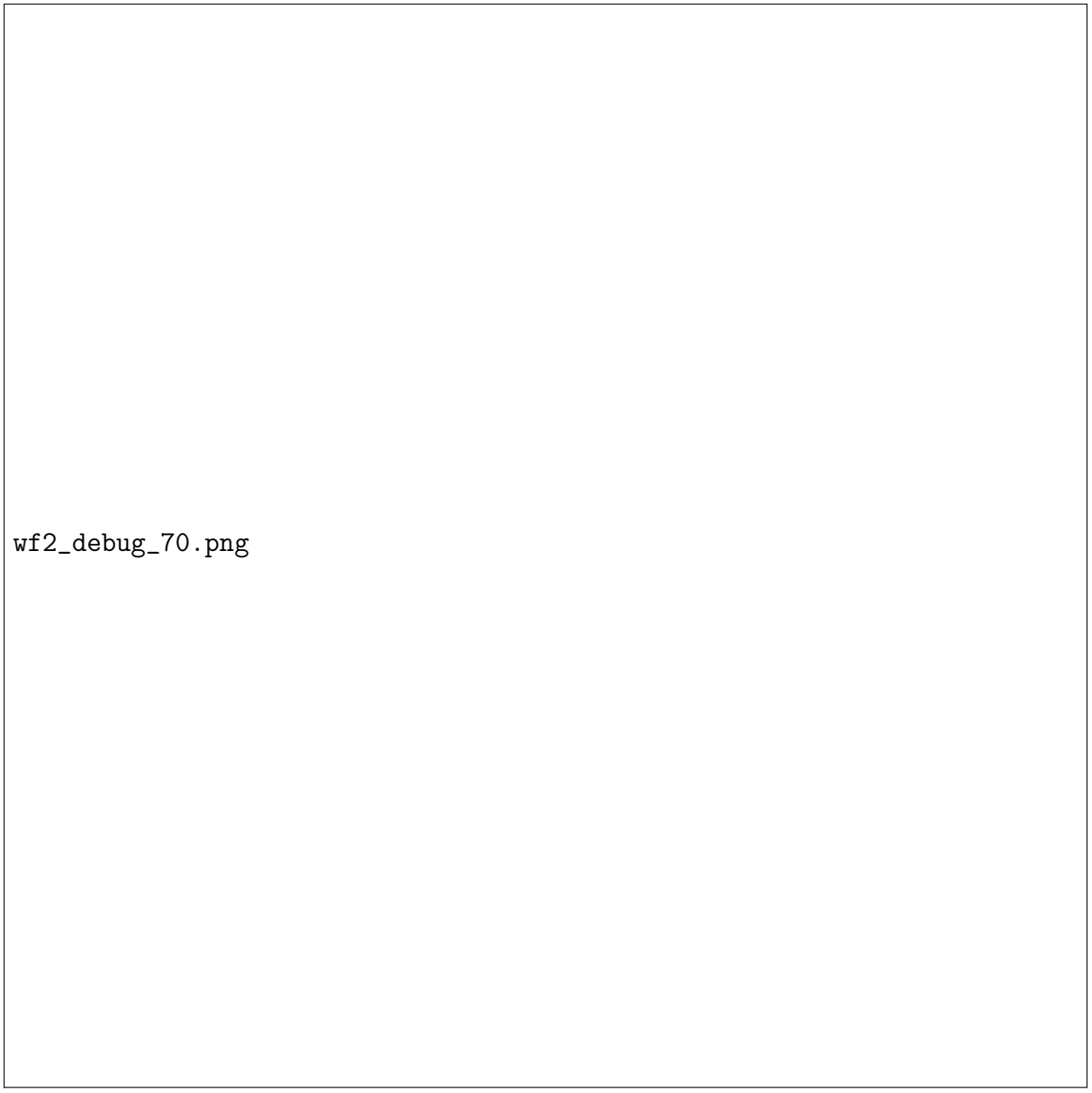


Figure 13: Workflow 2: 70 Seeds

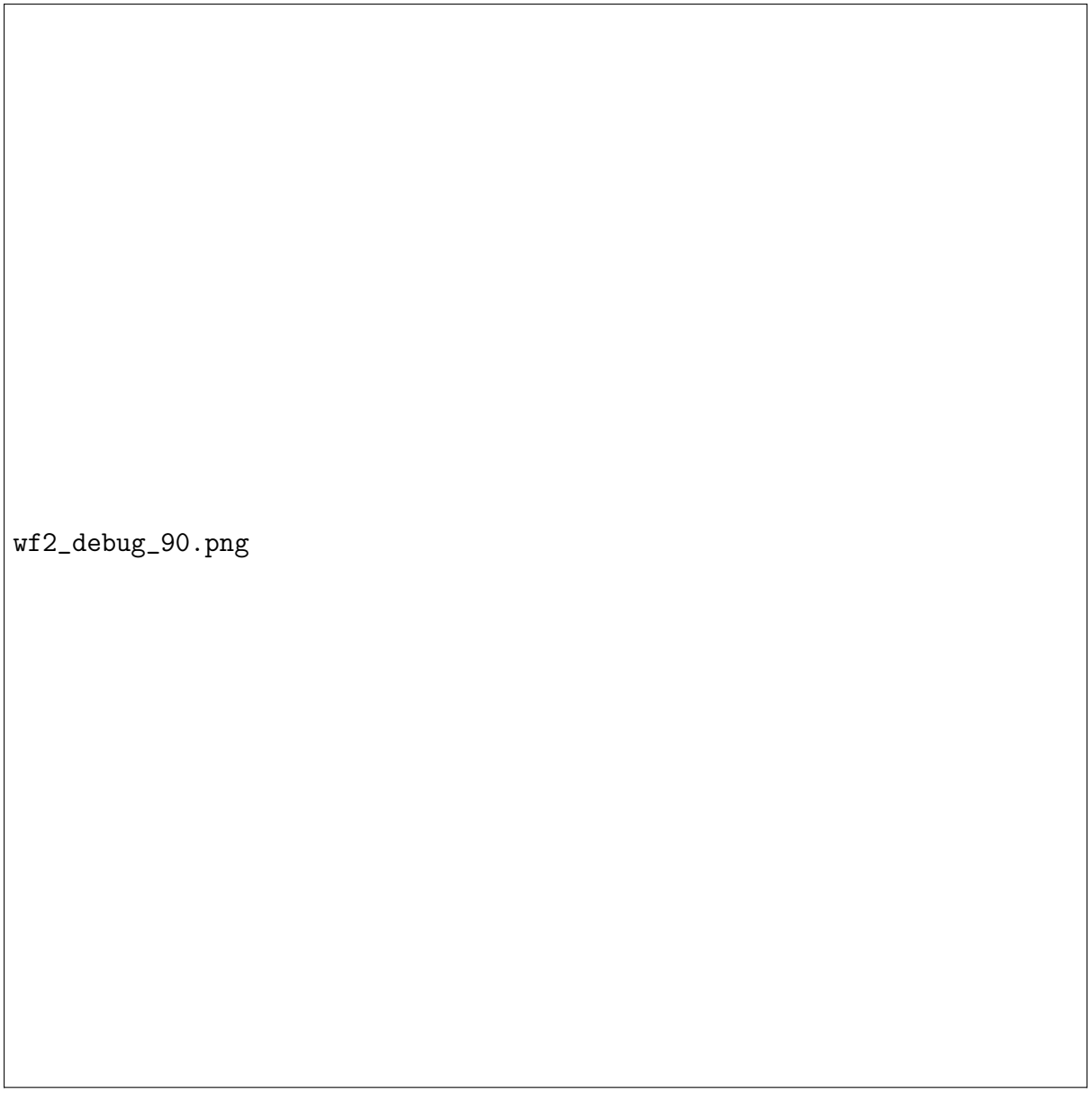


Figure 14: Workflow 2: 90 Seeds

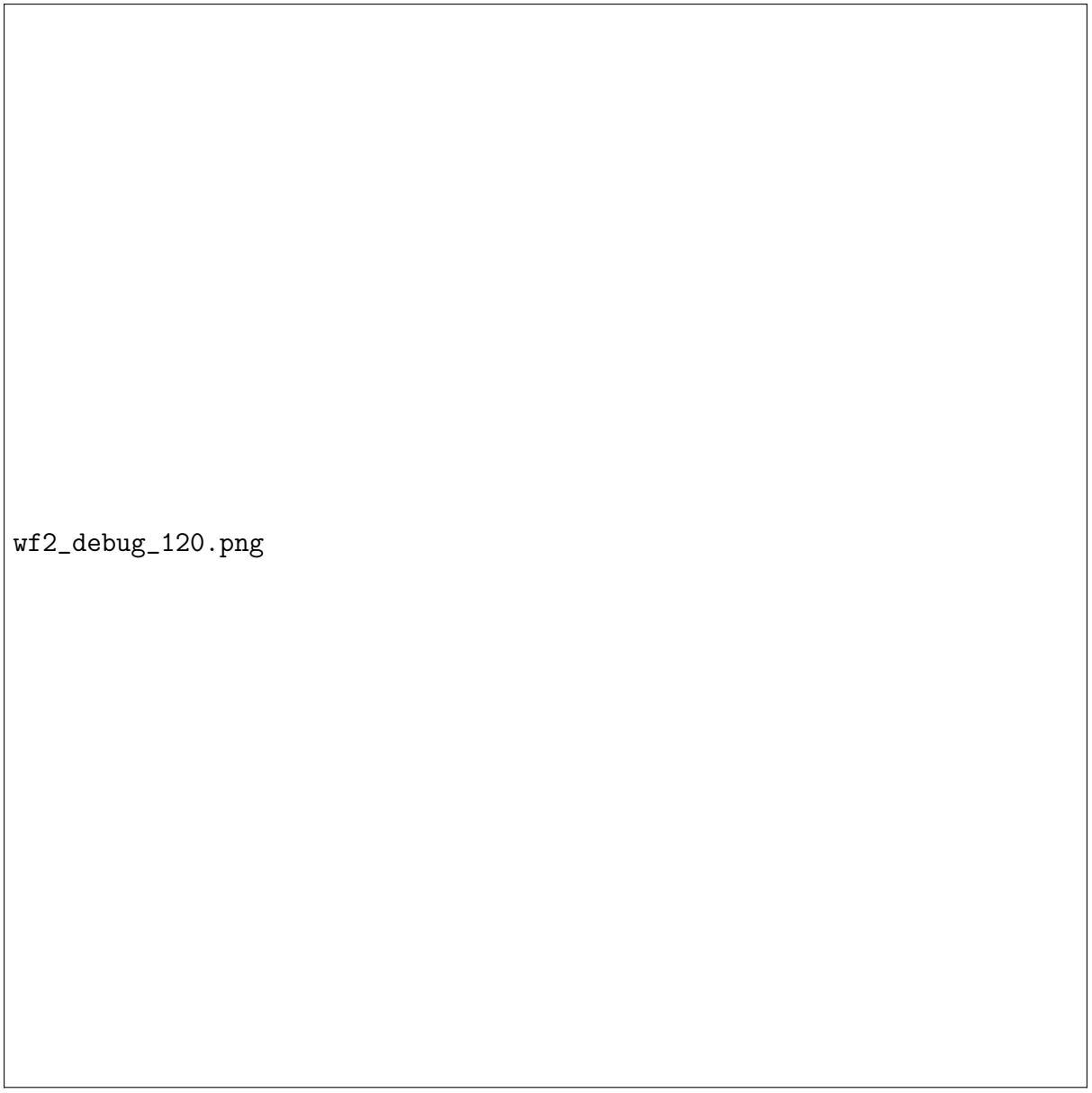


Figure 15: Workflow 2: 120 Seeds

B Appendix B: Full Dataset Results Table

The table below presents the quantitative results for every image in the dataset.

Table 1: Full Dataset Results (Workflow 1 vs Workflow 2)

Image	**GT**	**WF1 Pred**	**WF1 Acc (%)**	**WF2 Pred**	**WF2 Acc (%)**	**Err Diff**
1.jpg	1	1	100.0	1	100.0	0

C Appendix C: Key Code Snippets

C.1 Top-Hat and Difference of Gaussians (WF1)

```python
def preprocess_wf1(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 1. Top-Hat for Illumination Correction
    k_top = 51
    se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (k_top, k_top))
    tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT, se)

    # 2. Difference of Gaussians for Edge Enhancement
    g1 = cv2.GaussianBlur(tophat, (0,0), 1.5)
    g2 = cv2.GaussianBlur(tophat, (0,0), 4.5)
    dog = cv2.absdiff(g1, g2)

    return dog
```

C.2 Watershed Markers (WF1)

```python
def get_markers(binary_mask):
    # 1. Distance Transform
    dist = cv2.distanceTransform(binary_mask, cv2.DIST_L2, 5)

    # 2. Threshold Peaks
    ret, sure_fg = cv2.threshold(dist, 0.55 * dist.max(), 255, 0)

    # 3. Label Components
    sure_fg = sure_fg.astype(np.uint8)
    ret, markers = cv2.connectedComponents(sure_fg)

    # 4. Background is 1, Unknown is 0
    markers = markers + 1
    unknown = cv2.subtract(sure_bg, sure_fg)
    markers[unknown == 255] = 0

    return markers
```

C.3 LBP Calculation (WF2)

```python
def lbp_8u1(gray):
    g = gray.astype(np.float32)
    h, w = g.shape
    lbp = np.zeros((h, w), dtype=np.uint8)

    # Vectorized implementation for speed
    for dy in [-1, 0, 1]:
        for dx in [-1, 0, 1]:
            if dy == 0 and dx == 0: continue
            shifted = np.roll(np.roll(g, -dy, axis=0), -dx, axis=1)
            # Set bit if neighbor >= center
            lbp += (shifted >= g).astype(np.uint8)

```

```
14      return lbp
```