PHASE 2 SUBMISSION

Github link: https://github.com/seiftamerr/image-puzzle-solver.git

Ahmed Walid Elsayed Zahran Ali Mostafa 23P0038
Seif-Eldin Tamer Safwat Abdelsalam 23P0240
Abdelrahman Khaled Gaber Ibrahim 23P0065
Omar Fouad 23P0146
Tarek Hadef Mostafa 21P0199

2×2 Jigsaw Puzzle Solver — Methodology, Failure Analysis, and Demonstration

1. Pipeline Overview (Methodology)

This solver reconstructs a **2×2 jigsaw puzzle** using **edge-based image analysis** without any machine learning.
The pipeline is fully deterministic and consists of the following stages:

Step 1: Input and Preprocessing

Each scrambled puzzle image is loaded and divided into **four equal quadrants** using a fixed grid split.

```
Input image → 4 equal tiles (top-left, top-right, bottom-left, bottom-right)
```

This guarantees consistent tile dimensions and avoids shape ambiguity.

Step 2: Edge Extraction

For each tile, the four borders (**top, bottom, left, right**) are extracted as narrow strips of fixed thickness (`EDGE_THICKNESS = 3`).

These strips represent the **only regions used for matching**, which reflects the real jigsaw constraint that pieces connect via edges only.

Step 3: Edge Feature Encoding

Each extracted edge is converted into a **multi-channel descriptor** consisting of:

1. **LAB color channels** — robust to illumination changes
2. **Gradient magnitude** — captures texture continuity
3. **Laplacian response** — highlights edge sharpness and boundaries

These features are concatenated and **standardized per channel** to ensure fair comparison across different scales.

Step 4: Edge Similarity Computation

Edges are compared pairwise using an **absolute difference metric** with weighted contributions:

Feature Type	Weight
LAB Color	0.5
Gradient	0.3
Laplacian	0.2

Lower scores indicate **higher edge compatibility**.

All valid edge-to-edge comparisons are stored in **cost matrices** (right, left, top, bottom).

Step 5: Global Optimization (Brute Force)

Since a 2×2 puzzle has only **4! = 24 possible arrangements**, the solver evaluates **all permutations**.

For each candidate layout, it computes the **total edge cost** of the four internal adjacencies:

- Top-left → Top-right
- Top-left → Bottom-left
- Bottom-left → Bottom-right
- Top-right → Bottom-right

The arrangement with the **minimum total cost** is selected as the solution.

A **confidence margin** is also computed as the difference between the best and second-best scores, indicating solution certainty.

Step 6: Reconstruction

The selected tile order is used to reassemble the image into a single 2×2 grid.

The reconstructed image is saved for inspection.

Step 7: Evaluation

Each reconstructed puzzle is compared against its ground-truth image using **Mean Squared Error (MSE)**.

```
MSE < ERROR_LIMIT  → Correct
MSE ≥ ERROR_LIMIT → Incorrect
```

Final accuracy is reported as:

Accuracy=Correct PuzzlesTotal Puzzles×100\text{Accuracy} = \frac{\text{Correct Puzzles}}{\text{Total Puzzles}} \times 100Accuracy=Total PuzzlesCorrect Puzzles×100

---

2. Why This Is Not Machine Learning

This solver uses:

- No training data
- No learned parameters
- No model fitting
- No optimization through gradient descent

All decisions are based on **explicit rules and handcrafted features**.

✔ This is a **classical computer vision + combinatorial optimization approach**, not machine learning.

---

3. Points of Failure (Limitations)

Despite high accuracy, the method can fail in specific scenarios:

---

1. Low-Texture or Uniform Regions

If neighboring tiles have:

- Flat colors
- Little gradient variation
- Smooth textures

Then edge descriptors become ambiguous, leading to incorrect matches.

---

2. Symmetric Edge Patterns

Edges with visually similar patterns on multiple sides can cause:

- Multiple permutations to have similar scores
- Reduced confidence margin
- Occasional incorrect placement

---

3. Color Bleeding or Compression Artifacts

Strong JPEG compression or color bleeding across tile boundaries can distort:

- LAB color values
- Gradient continuity

This increases edge mismatch error.

---

4. Fixed Threshold Dependency

The correctness decision relies on a manually chosen MSE threshold (`ERROR_LIMIT = 300`).

- A stricter threshold increases false negatives
- A looser threshold increases false positives

---

4. Demonstration Explanation

What Is Demonstrated

For each puzzle:

1. The scrambled input is processed automatically
2. All 24 possible layouts are evaluated
3. The best-scoring configuration is selected
4. The reconstructed image is saved and evaluated

Confidence Margin

The solver reports a **confidence margin**, defined as:

```
(second-best cost) − (best cost)
```

- **Large margin** → clear correct solution
- **Small margin** → ambiguous edge matching

This provides interpretability and debugging insight.

4×4 Jigsaw Puzzle Solver

Edge-Based Greedy Assembly with Tile-Level Accuracy Evaluation

---

1. Pipeline Explanation (Methodology)

This system reconstructs a **4×4 jigsaw puzzle (16 tiles)** using **classical edge matching** and a **greedy placement strategy**.
No machine learning or training data is used.

Step 1: Input and Tile Extraction

Each puzzle image is loaded and divided into **16 equal tiles** using a fixed 4×4 grid.

```
Input image → 16 tiles (row-major order)
```

This ensures:

- Identical tile sizes
- No geometric distortion
- Deterministic tile boundaries

Step 2: Edge Similarity Metric

Tile compatibility is measured using **Sum of Squared Differences (SSD)** between neighboring edges.

For two edges:

SSD(A,B)=1N∑(A−B)2\text{SSD}(A,B) = \frac{1}{N} \sum (A - B)^2SSD(A,B)=N1∑(A−B)2

Edges compared:

- Right edge of left tile ↔ Left edge of candidate tile
- Bottom edge of top tile ↔ Top edge of candidate tile

Lower SSD ⇒ better visual continuity.

Step 3: Greedy Assembly with Anchors

The solver uses a **greedy anchor-based strategy**:

1. Each tile is **temporarily treated as the top-left anchor**
2. The grid is filled **left-to-right, top-to-bottom**
3. At each position:
    - The unused tile with the **lowest combined edge cost** is selected
4. The total grid cost is computed
5. The anchor that yields the **lowest total cost** is selected

This avoids committing to a single arbitrary starting point.

Step 4: Global Cost Evaluation

For each candidate grid, total cost is computed by summing:

- All horizontal edge SSDs
- All vertical edge SSDs

The grid with the **minimum total cost** is chosen as the final solution.

Step 5: Image Reconstruction

The solved grid is reassembled into a full image by placing tiles according to their grid positions.

---

2. Tile-Level Accuracy Evaluation (Key Contribution)

Instead of relying only on image-level error, this implementation evaluates **tile correctness individually**, which is more informative.

Ground Truth Comparison

- Each reconstructed tile is compared to the corresponding ground-truth tile
- **Structural Similarity Index (SSIM)** is used instead of raw pixel error

```
SSIM > 0.90 → Tile considered correct
```

This allows tolerance to:

- Minor brightness changes
- Small color shifts
- Compression artifacts

Metrics Reported

The system computes:

- **Tile Accuracy per Puzzle**
- **Average Tile Accuracy**
- **Total Tile Accuracy**
- **Image Accuracy** (all 16 tiles correct)
- **Min / Max / Std Deviation**

This provides both **local** and **global** performance insight.

---

3. Why This Is NOT Machine Learning

This solver:

- Uses no training data
- Has no learned parameters
- Makes decisions using explicit rules
- Relies on fixed similarity metrics (SSD, SSIM)

✔ This is a **classical computer vision + greedy optimization approach**

---

4. Points of Failure (Limitations & Error Sources)

This method is intentionally simple and therefore has predictable failure cases.

---

1. Greedy Commitment Errors

Once a tile is placed, it is **never reconsidered**.

If an early placement is incorrect:

- The error propagates
- Later tiles are forced into suboptimal positions

This is the main limitation of greedy assembly.

---

2. Low-Texture or Uniform Regions

Tiles containing:

- Sky
- Flat backgrounds
- Repetitive patterns

Produce very similar edge SSD values, making correct matching ambiguous.

---

3. SSD Sensitivity

SSD is:

- Sensitive to lighting changes
- Sensitive to color shifts
- Not invariant to noise

This can mis-rank visually correct edges.

---

4. No Global Consistency Enforcement

The algorithm does not check:

- Loop consistency

- Best-buddy symmetry
- Global arrangement validity

Thus, a locally good match may still be globally incorrect.

---

5. Demonstration Explanation (For Report & Presentation)

What Is Visualized

Only **one puzzle** is visualized to keep the report concise:

- Top: Scrambled tiles (input)
- Bottom: Reconstructed image
- Title shows tile accuracy

This demonstrates:

- Input disorder
- Reconstruction result
- Quantitative performance

Why Tile-Level Accuracy Is Used

Image-level metrics alone are misleading:

- One misplaced tile fails the entire image
- Partial correctness is ignored

Tile-level accuracy provides:

- Finer evaluation
- Better algorithm comparison
- Clear progress tracking

---

6. Summary

Strengths

- Simple and explainable

- Deterministic and reproducible
- Tile-level evaluation is robust and informative
- Good baseline for comparison

Limitations

- Greedy, not optimal
- Sensitive to low-texture regions
- No global optimization

---

7. How This Fits the Project

This solver serves as:

- A **baseline method**
- A reference for improvement
- A contrast to more advanced solvers (Hungarian, SA, beam search)

It justifies why more sophisticated strategies are needed for higher accuracy.

8×8 Jigsaw Puzzle Solver

Edge-Based Agglomerative Reconstruction with Tile-Level Accuracy

---

1. Pipeline Explanation (Methodology)

This approach reconstructs **8×8 jigsaw puzzles (64 tiles)** using a **classical computer vision pipeline** based on **edge similarity** and **incremental group merging**.
No machine learning or training data is used.

Step 1: Input and Tile Extraction

Each puzzle image is divided into **64 equal-sized tiles** using a fixed 8×8 grid.
All tiles have identical dimensions, ensuring consistent edge comparison.

Step 2: Feature Space for Edge Matching

To compare tiles, the algorithm operates in **LAB color space**, which separates luminance from chromatic information and improves perceptual matching.

For every tile:

- Right edge
- Left edge
- Top edge
- Bottom edge

are extracted and compared using **Mean Squared Error (MSE)**.

Step 3: Edge Cost Matrix Construction

Two cost matrices are computed:

- **Horizontal cost matrix (H)**:
  Right edge of tile *i* vs left edge of tile *j*
- **Vertical cost matrix (V)**:
  Bottom edge of tile *i* vs top edge of tile *j*

Lower cost indicates better edge continuity.

All possible tile pairs are evaluated except self-matches.

Step 4: Global Edge Ranking

All possible adjacency candidates are collected as:

- (cost, tile_i, tile_j, direction)

These candidates are **sorted globally** from lowest to highest cost.
This ensures that the most confident matches are considered first.

Step 5: Agglomerative Group Merging

Initially, each tile is treated as an independent group.

The algorithm iterates over sorted edge candidates and:

1. Checks whether two tiles belong to different groups
2. Verifies that merging them does not cause overlap
3. Merges the groups if the spatial constraints are satisfied

This forms progressively larger connected components, similar to **hierarchical clustering**.

Step 6: Final Reconstruction

Once no further valid merges are possible:

- The largest connected group is selected
- All tiles are shifted to align within the 8×8 grid
- The final reconstructed image is rendered

---

2. Tile-Level Accuracy Evaluation

Unlike image-level metrics, this solver evaluates **correctness at the tile level**, which is more informative for large puzzles.

Tile Accuracy Definition

A reconstructed tile is considered **correct** if:

MSE(tilepred,tilegt)<5\text{MSE(tile}_{pred}, \text{tile}_{gt}) < 5MSE(tilepred,tilegt)<5

This strict threshold ensures:

- Correct spatial placement
- Correct visual content
- No tolerance for wrong neighbors

Reported Metric

Final accuracy is computed as:

Tile Accuracy=Correct TilesTotal Tiles×100\text{Tile Accuracy} = \frac{\text{Correct Tiles}}{\text{Total Tiles}} \times 100Tile Accuracy=Total TilesCorrect Tiles×100

This reflects **partial correctness**, not just perfect reconstructions.

---

3. Demonstration (Visualization Strategy)

To keep results interpretable:

- **All 110 puzzles** are processed for evaluation
- **Only the first 5 puzzles** are visualized

Each visualization shows:

1. Scrambled input (random tile order)
2. Final reconstructed image
3. Number of correctly placed tiles (out of 64)

This balances **clarity** and **computational completeness**.

---

4. Points of Failure (Limitations & Error Analysis)

Despite being effective as a baseline, the algorithm has several known limitations.

---

1. Greedy Early Commitment

Once two tiles are merged, the decision is **never revisited**.

If an incorrect edge is selected early:

- The error propagates
- Entire regions may shift incorrectly

This is the main source of failure in complex scenes.

---

2. Ambiguous Edges in Low-Texture Regions

Tiles with:

- Sky

- Uniform backgrounds
- Repetitive textures

Produce similar edge costs, making correct adjacency ambiguous.

---

3. No Global Optimization

The algorithm:

- Does not enforce loop consistency
- Does not check best-buddy symmetry
- Does not solve a global assignment problem

Thus, it may converge to a **locally optimal but globally incorrect** arrangement.

---

4. Sensitivity to Color Noise

Edge matching relies on raw pixel differences in LAB space.

Illumination changes or compression artifacts can increase edge error even when adjacency is correct.

---

5. Why This Is Not Machine Learning

This method:

- Uses no training data
- Has no learned parameters
- Makes decisions using explicit mathematical rules
- Is fully deterministic

It is a **classical vision + greedy optimization approach**, suitable as a **baseline**.

---

6. Strengths of the Approach

- Scales to large puzzles (8×8)
- Simple and explainable

- Deterministic and reproducible
- Tile-level accuracy gives meaningful evaluation
- Strong baseline for comparison with advanced methods

---

7. Positioning in the Project

This solver serves as:

- A **baseline reference**
- A motivation for global optimization (Hungarian, SA, beam search)
- A justification for more advanced feature descriptors

It clearly demonstrates **why local edge matching alone is insufficient** for large puzzles.