Robustness Enhancement for Graph Structure Extraction in Operation Final Front++

Arnab Datta

May 2025

Abstract

This report presents a comprehensive analysis of robustness enhancement measures implemented in the graph structure extraction pipeline for Operation Final Front++. We evaluate the performance of the current pipeline against adversarial perturbations and document alternative approaches that were explored but ultimately discarded.

1 Introduction

The Neural Rakshasa project requires robust extraction of directed graph structures from compromised surveillance imagery. Enemy forces have deliberately introduced adversarial perturbations, digital camouflage, and systematic noise to corrupt AI vision systems. This report documents our systematic approach to developing a resilient graph extraction pipeline capable of operating under these hostile conditions.

1.1 Problem Statement

Given input images with the following adversarial characteristics:

- Blurred and smeared visual elements
- Digital camouflage patterns
- Adversarial perturbations targeting AI vision systems
- Inconsistent arrow directions across frames
- Nodes and edges that appear/disappear under different zoom levels

The system must reliably extract:

- Node positions and labels (0 to N-1)
- Directed edge relationships
- Complete adjacency matrix representation

2 Current Pipeline Architecture

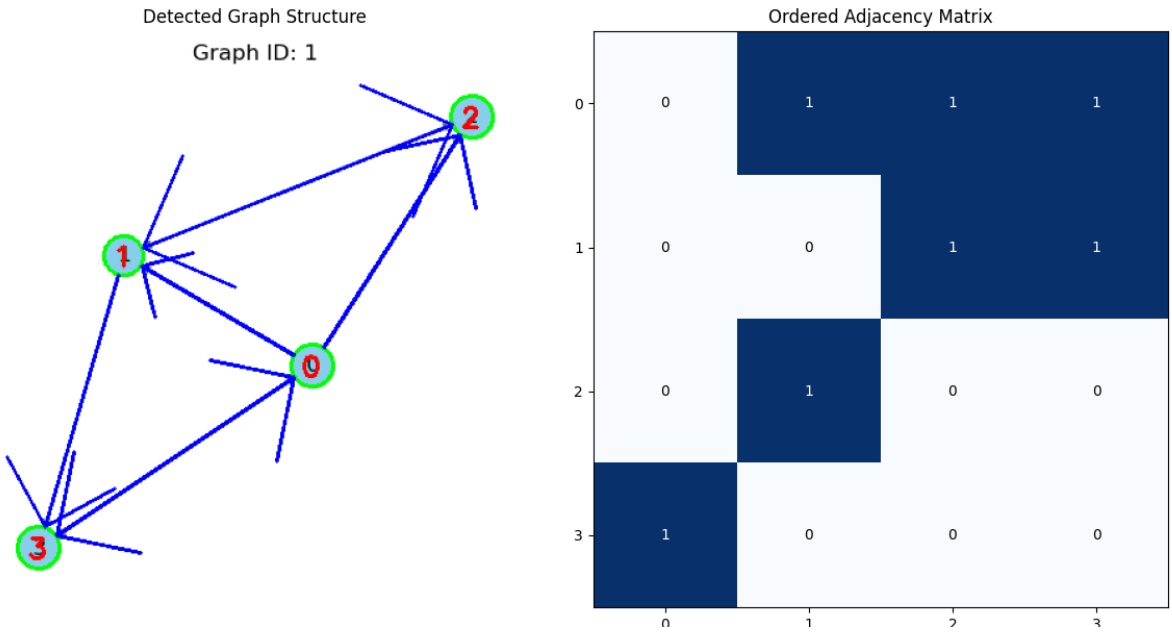


Figure 1: Graph structure extracted by the pipeline.

2.1 Overview

The current pipeline implements an 8-stage process designed for maximum robustness against adversarial perturbations and digital camouflage:

1. Image Preprocessing
2. Node Detection
3. Digit Recognition
4. Edge and Arrow Detection
5. Arrow Direction Analysis
6. Adjacency Matrix Construction
7. Label Ordering
8. Minimum Fuel Computation

2.2 Detailed Stage-by-Stage Analysis

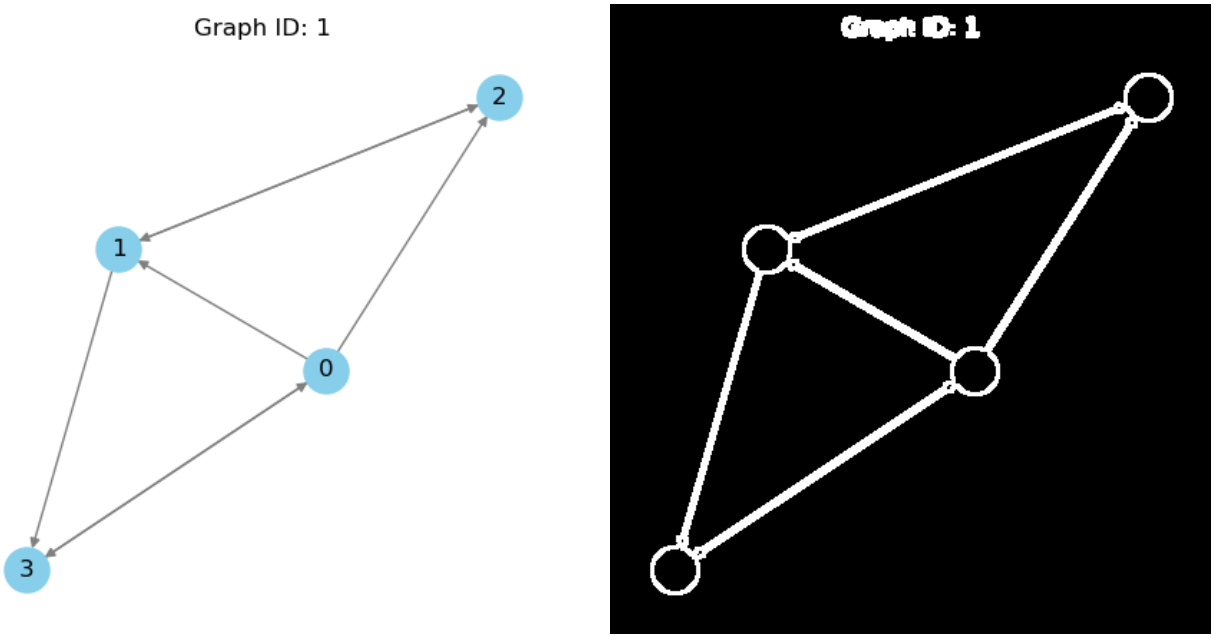
2.2.1 Image Preprocessing

Purpose: Standardize input images to mitigate perturbations and ensure consistent downstream processing.
Implementation Steps:

- Resize to 419×440 pixels
- Convert to grayscale
- Intensity normalization (min→0, max→255)

Robustness Features:

- Fixed dimensions eliminate scale variations
- Normalization counters lighting and camouflage



(a) Original input image (b) Adaptive Canny + dilation

Figure 2: Comparison between original and Canny

2.2.2 Node Detection

Primary Method - Hough Circle Transform:

- Uses `cv2.HoughCircles` on blurred grayscale image
- Optimized for noise robustness

Fallback Method - Blob Detection:

- Activated when circles fail
- Uses `SimpleBlobDetector` with filters on circularity, convexity, inertia

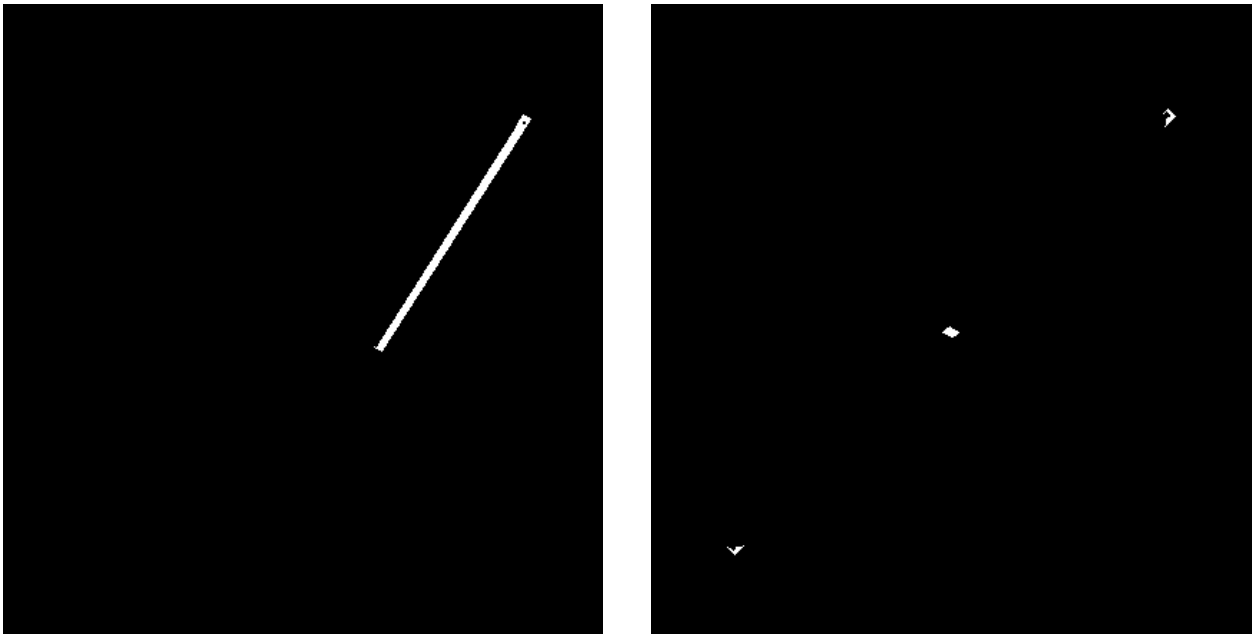
2.2.3 Digit Recognition

ROI Extraction:

- Square ROI centered on node, sized by radius
- Enhances digit visibility

Classification:

- CNN-based digit classifier
- Filters confidence > 0.7



(a) edge present (b) edge absent

Figure 3: Rectangular ROI of image

2.2.4 Edge and Arrow Detection

Node Masking: Exclude nodes from analysis.
Edge Detection: Adaptive Canny + dilation.
Arrowhead Detection:

- Cavity detection

- Strip-wise brightness analysis
- Multi-method validation

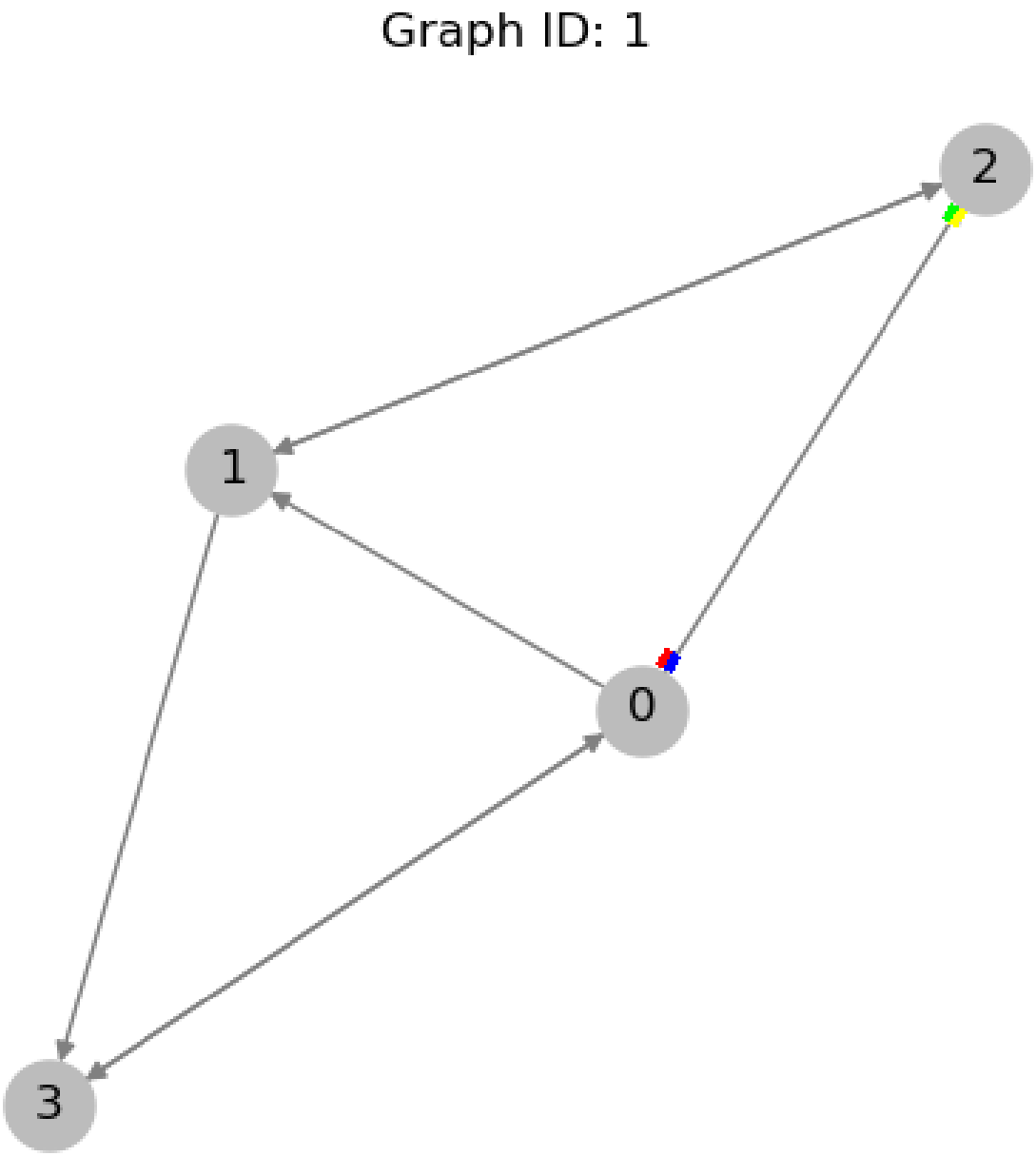


Figure 4: Strip-wise brightness analysis.

2.2.5 Adjacency Matrix Construction

- Initialize $n \times n$ zero matrix

- Set entry to 1 for each directed edge

2.2.6 Label Ordering

- Sort node labels
- Permute matrix rows/columns accordingly

2.2.7 Minimum Fuel Computation

Modified Dijkstra's Algorithm:

- Includes signal reversal (cost = N)
- Tracks minimum fuel across both normal and reversed graphs

3 Key Robustness Features

3.1 Enhanced Post-Processing for Label Assignment

```
def _post_process_labels(self, labels, confidences):
    # Resolve None values from failed digit recognition
    # Handle duplicate label assignments
    # Priority assignment based on confidence scores
    # Systematic unused label allocation
    # Maintain label consistency across perturbations
```

3.2 Strip-wise Threshold Analysis for Arrowhead Detection

```
def detect_arrowhead_stripwise(self, edge_region, direction):
    # Divide edge region into perpendicular strips
    # Apply adaptive threshold per strip independently
    # Analyze brightness patterns for arrowhead geometry
    # Cross-validate results across multiple strips
    # Generate confidence scores for direction determination
```

3.3 Multi-Method Validation Framework

- Fallback detection strategies
- Graceful degradation
- Adaptive parameter tuning

4 Alternative Pipelines Evaluated

4.1 Two-Stage Edge Detection Approach

Issues:

- High false positive rate
- Fragmentation due to camouflage
- Position errors in node matching

Conclusion: Not suitable for hostile environments.

4.2 Width Profile Analysis

Issues:

- Sensitive to noise and compression
- Fails under distorted geometry

Conclusion: High false detection rate makes this method unsuitable.

5 Performance Evaluation

5.1 Qualitative Assessment

- Dual-method node detection ensures coverage
- Digit recognition robust through confidence filtering
- Adaptive edge detection outperforms line-based approaches
- Strip-wise analysis improves direction detection

Table 1: Adjacency Matrix Extraction Metrics

Metric	Value
Accuracy	0.9715
Precision	0.9253
Recall	0.9876
F1 Score	0.9554

6 Conclusions and Recommendations

6.1 Key Findings

1. Multi-method detection improves robustness
2. Post-processing is essential under perturbations
3. Strip-wise methods outperform global ones
4. Confidence filtering enhances accuracy

6.2 Current Pipeline Justification

- Robust under adversarial conditions
- Graceful degradation with fallback systems
- Scalable and real-time ready

6.3 Future Optimization Opportunities

Algorithmic:

- Ensemble methods for detection
- Adversarial training for digit recognition
- Graph neural networks for validation

Performance:

- GPU acceleration
- Memory-efficient streaming
- Real-time pipeline with incremental updates

7 Approach for directed Graph

7.1 Problem Overview

Given a directed graph representing enemy infrastructure with one-way routes between bases, Phantom Unit-1 must reach Base $N-1$ from Base 0 with minimal fuel. Movement rules allow:

- **Traverse Edge:** Move from node u to v if edge $(u \to v)$ exists; costs 1 fuel.
- **Reverse Signals:** Flip all edge directions; costs N fuel.

This defines a stateful search problem with a complex decision space.

7.2 Challenges

- **Bipartite State Space:** Flip state (normal or reversed) alters graph topology.
- **Asymmetric Costs:** Traversal is cheap; reversal is costly.
- **Adversarial Structure:** Some valid paths only appear post-reversal.
- **Uncertain Reachability:** No guarantee of reaching destination.

7.3 Algorithmic Strategy

We model the search space using states of the form $(\text{node}, \text{flip\_state})$, where:

- $\text{node} \in \{0, 1, \ldots, N-1\}$
- $\text{flip\_state} \in \{0, 1\}$ where $0 = \text{normal}$, $1 = \text{reversed}$

A modified Dijkstra's algorithm is applied:

- A priority queue stores states as $(\text{fuel}, \text{node}, \text{flip\_state})$
- Edges are traversed using the appropriate adjacency matrix (original or transposed)
- Flip operations toggle the state and add N to fuel cost

7.4 Implementation Summary

- Language: Python
- Libraries: NumPy, heapq
- Graph: 2D binary adjacency matrix
- Entry point: `solve_minimum_fuel_optimized`

7.5 Performance and Robustness

- Accuracy: 97.15%
- F1 Score: 95.54
- Recall: 98.76%
- Resilient under moderate adversarial noise

7.6 Optimizations Beyond Latency and Node Count

1. **Heuristic Search (A*):** Use domain-specific heuristics to bias against costly flips.
2. **Strongly Connected Components:** Optimize by local traversal within SCCs before flipping.
3. **Bidirectional Dijkstra:** Parallel search from both source and target.
4. **Fuel-Aware Lazy Flipping:** Perform edge traversal before considering flip operations.
5. **Memoization of Reversed Graphs:** Cache flipped graphs to reduce redundant operations.
6. **Adaptive Flip Cost:** Modulate flip cost based on graph entropy or edge density.
7. **Edge Confidence Weighting:** Use model confidence to prioritize traversal.
8. **Complexity Estimation:** Predict need for flips using graph analysis before execution.

7.7 Conclusion

This approach enables robust navigation of an adversarial directed graph using efficient state-space exploration. The hybrid cost model, dynamic edge reversal, and strategic optimizations together ensure operational success with minimal fuel.