Tamper Detection in Academic Documents — Technical Report

# 1. Introduction

The goal of this project is to design a system that detects tampering in academic documents such as transcripts, certificates, or mark sheets. These documents are often targets for forgery, and automated tools can assist human evaluators in verifying authenticity.  
  
This system compares a known clean (template) image with a potentially tampered version using both image-based and OCR-based techniques. It flags suspicious modifications and generates a CSV report summarizing each comparison.

# 2. Approach and Methodology

## 2.1 Data Preparation

- A Roboflow-hosted `.jsonl` file provided URLs to clean academic document images.  
- Images were downloaded and saved in the `data/templates/` folder.  
- Annotations in the file specified bounding boxes representing tamperable fields such as name, grade, or ID.

## 2.2 Tampering Simulation

- Tampered images were synthetically generated using the annotations.  
- A script (`samples\_create.py`) applied white rectangles over labeled fields and overlaid fake text (e.g., 'Fake Name').  
- These simulated tamperings mimic real-world alterations like name changes, grade inflation, or forged seals.

## 2.3 Tamper Detection Pipeline

Implemented in `main.py`, the detection pipeline follows 3 phases:  
  
1. SSIM Image Comparison: Structural Similarity Index Measure (SSIM) is used to compare template vs tampered image.  
2. OCR Text Comparison: Tesseract OCR extracts text from both images and text similarity is computed using `difflib.SequenceMatcher`.  
3. Reporting: Results are saved in `outputs/reports/exploration\_results.csv` with SSIM scores, OCR similarity, and a suspicion flag.

# 3. Assumptions and Design Decisions

- The system assumes access to clean reference templates for comparison.  
- White-box tampering was used for initial testing to ensure controlled experiments.  
- OCR differences were used to capture content-based tampering undetectable by SSIM alone.  
- Fixed thresholds were chosen based on qualitative testing but can be tuned.

# 4. Challenges and Trade-offs

|  |  |
| --- | --- |
| Challenge | Approach Taken |
| OCR noise & errors | Applied fuzzy matching (`difflib`) to tolerate variations |
| Subtle tampering detection | Combined SSIM with OCR to cover both image & text |
| Template-tampered misalignment | Assumed same dimensions; future work may use alignment |
| Lack of real tampered data | Simulated forgery using annotations for training/testing |

# 5. Suggestions for Improvement and Scaling

1. Use Real Tampered Documents: Incorporating real-world forged certificates can improve robustness.  
2. Enhance Tampering Simulation: Use realistic fonts and erasure techniques.  
3. Multimodal Detection: Combine SSIM, OCR, and models like YOLO or LayoutLM.  
4. Automated Highlighting: Show SSIM heatmaps or OCR token-level diffs.  
5. Deploy as an API or Tool: Web interface or Flask app for usability.  
6. Model Training: Fine-tune deep models for forgery classification without needing clean templates.

# 6. Conclusion

This project demonstrates a modular pipeline for tamper detection using traditional CV and OCR techniques. While effective for controlled tampering, it can be significantly improved with deeper learning models, better data, and a more robust UI for field deployment.