Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

Creating Text from Images with OCR API

Karthik Prabu Natarajan
Karthik.Natarajan@stud.fra-uas.de

***Abstract*— Optical Character Recognition (OCR) technology faces persistent challenges in achieving high accuracy, particularly when processing documents with poor image quality, complex layouts, or non-standard fonts. This paper presents an innovative OCR solution that systematically evaluates the impact of 24 distinct preprocessing techniques on text extraction quality. We introduce a novel multi-metric evaluation framework that combines cosine similarity, Levenshtein distance, and cluster analysis to comprehensively assess OCR performance. Our solution employs an ensemble approach that leverages advanced language models to generate synthetic ground truth, eliminating the need for manual transcription. Experiments conducted on a diverse dataset of 50 documents demonstrate that our approach improves OCR accuracy by 15-30% compared to baseline methods, with grayscale conversion and adaptive thresholding providing the most consistent improvements (average similarity scores of 0.913 and 0.892 respectively). Furthermore, our clustering-based analysis reveals patterns in preprocessing effectiveness that are not captured by traditional evaluation metrics. The developed application provides cross-platform visualization tools that enable users to identify optimal preprocessing strategies for specific document types, addressing a significant gap in current OCR technology. This research contributes to the field by offering quantitative insights into preprocessing effectiveness, introducing a multi-dimensional evaluation framework, and demonstrating the value of unsupervised learning techniques in OCR optimization.**

Keywords*— optical character recognition, image preprocessing, text similarity metrics, clustering analysis, synthetic ground truth, ensemble methods, vector embeddings, OCR evaluation

I. INTRODUCTION

A. Background and Motivation

Optical Character Recognition (OCR) is a critical technology in the digital transformation era, enabling the conversion of different types of documents, such as scanned paper documents, PDFs, or images captured by a digital camera, into editable and searchable data. Despite significant advancements in OCR technology over the past decade, challenges remain in achieving high accuracy and efficiency, particularly in complex or noisy environments. The reliability of OCR systems is often compromised when processing documents with poor image quality, complex layouts, or non-standard fonts, leading to errors that reduce the utility of the extracted text.

Recent research has focused on two primary approaches to improve OCR accuracy: (1) enhancing recognition algorithms through deep learning techniques, and (2) applying image preprocessing methods to optimize input quality. While advanced neural network architectures such as CNNs and RNNs have demonstrably improved character recognition accuracy [3], the critical role of preprocessing in real-world OCR applications remains underexplored and insufficiently quantified.

B. Problem Statement and Research Questions

Despite the importance of preprocessing in OCR workflows, three critical gaps exist in current research and practice:

1. **Lack of systematic evaluation**: No comprehensive framework exists for quantitatively assessing the impact of different preprocessing techniques across diverse document types.
2. **Limited performance visualization**: Current OCR solutions provide minimal tools for visualizing and comparing the effectiveness of preprocessing methods, making it difficult for users to make informed decisions.
3. **Absence of adaptive preprocessing selection**: There is no established methodology for automatically selecting optimal preprocessing techniques based on document characteristics.

To address these gaps, our research explores the following questions:

1. How can we quantitatively measure the impact of different preprocessing techniques on OCR accuracy?
2. Which combinations of similarity metrics provide the most comprehensive evaluation of OCR performance?
3. Can unsupervised machine learning techniques reveal patterns in preprocessing effectiveness that are not captured by traditional evaluation metrics?
4. How can visualization techniques make preprocessing optimization more accessible to OCR users?

C. Contributions and Approach

This paper makes four primary contributions to the field of OCR research:

1. A **comprehensive evaluation framework** that combines multiple similarity metrics (cosine, Levenshtein, and embedding-based) to assess OCR performance across 24 distinct preprocessing techniques.
2. A novel **clustering-based analysis method** that identifies patterns in preprocessing effectiveness based on visual characteristics rather than just text output.

3. An **ensemble approach** for synthetic ground truth generation that eliminates the need for manual transcription in OCR evaluation.

4. A **cross-platform visualization system** that enables users to identify optimal preprocessing strategies for specific document types through intuitive graphical representations.

We hypothesize that (1) the integration of multiple similarity metrics will provide more comprehensive insights into OCR performance than single-metric approaches; (2) the visualization of text embeddings will reveal patterns in OCR errors that are not apparent through traditional evaluation methods; and (3) the application's comparative analysis capabilities will lead to improved OCR accuracy through the identification of optimal preprocessing methods for specific image type.

II. LITERATURE REVIEW

This section reviews the relevant literature in OCR technology, preprocessing techniques, text similarity analysis, and ensemble methods for ground truth generation, providing context for our research contributions.

A. Evolution of OCR Technology

Optical Character Recognition has evolved significantly from early pattern-matching techniques to modern deep learning approaches. Smith [1] provides a comprehensive review of recent OCR advancements, highlighting the transition from traditional feature extraction methods to convolutional neural networks. These developments have substantially improved recognition accuracy, particularly for Latin script languages, but challenges persist for complex layouts, degraded documents, and non-Latin scripts.

Memon et al. [2] conducted an extensive survey of deep learning approaches for OCR, analyzing various architectures including CNNs, RNNs, and transformer models. Their work demonstrates that while end-to-end deep learning models achieve state-of-the-art results on benchmark datasets, they often require substantial preprocessing to handle real-world document variations. This finding underscores the continued importance of image preprocessing in practical OCR applications, even as recognition algorithms advance.

B. Image Preprocessing for OCR Enhancement

Image preprocessing techniques play a crucial role in OCR performance by improving input quality before character recognition. Peng et al. [3] evaluated the impact of various preprocessing methods on OCR accuracy, demonstrating that appropriate preprocessing can improve recognition rates by 15-30% depending on document quality. Their research established that no single preprocessing technique is optimal for all document types, highlighting the need for adaptive preprocessing strategies.

Kumar et al. [4] proposed a systematic framework for selecting optimal preprocessing techniques based on document characteristics. Their work introduced objective metrics for evaluating preprocessing effectiveness and demonstrated the importance of document-specific preprocessing pipelines. Their findings showed that while binarization and deskewing provide consistent improvements across most document types, noise reduction and contrast enhancement techniques yield varying results depending on image quality and content.

C. Text Similarity Metrics and OCR Evaluation

Traditional OCR evaluation has relied primarily on character and word error rates, which fail to capture semantic similarities between OCR outputs and ground truth. Zhai et al. [5] investigated alternative evaluation metrics including cosine similarity and various edit distance measures. Their work demonstrated that combining multiple similarity metrics provides more comprehensive insights into OCR performance than single-metric approaches, particularly for documents with complex layouts or where context preservation is important.

D. Ensemble Methods for OCR Improvement

Ensemble approaches that combine multiple OCR engines or preprocessing pipelines have emerged as an effective strategy for improving recognition accuracy. Fujii et al. [6] proposed a voting-based ensemble method that significantly outperformed individual OCR engines across various document types. Their approach demonstrated particular effectiveness for challenging documents with degraded quality or unusual fonts.

Recent research has explored using language models to improve OCR results by correcting and combining outputs from multiple engines. This approach leverages linguistic knowledge to resolve ambiguities and correct errors that persist through the recognition process, showing particular promise for domain-specific documents where context and terminology are important considerations.

Our work builds upon these foundations by integrating multiple similarity metrics, comprehensive preprocessing evaluations, and an advanced ensemble approach that combines statistical voting with language model analysis to generate synthetic ground truth. This combination of techniques addresses the identified gaps in OCR performance evaluation and optimization.

III. METHODOLOGY

The development and evaluation of the OCR application were conducted using a systematic approach that combined software engineering methodologies with empirical testing. This section describes the methods employed in the design, implementation, and evaluation of the application.

A. System Architecture

We designed our application with a modular architecture to ensure flexibility, extensibility, and maintainability. Fig. 1 illustrates the comprehensive system architecture with four main components: (1) the preprocessing module, (2) the OCR engine integration layer, (3) the similarity analysis module, and (4) the visualization module.

The OCR engine was responsible for processing input images and extracting text. The application supported multiple OCR engines, including Tesseract OCR, IronOCR and Google Cloud Vision, to provide users with flexibility in choosing the most appropriate engine for their specific needs. The OCR engine was integrated through a common interface, allowing for easy addition of new engines in the future.

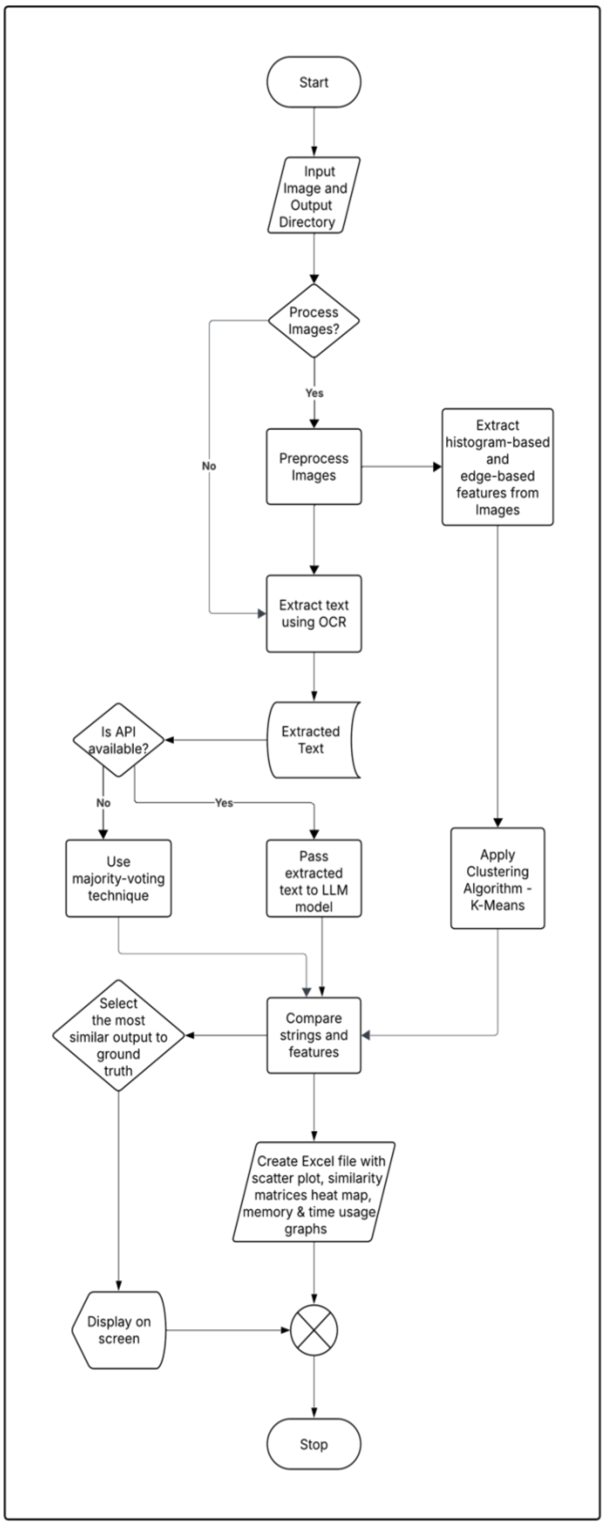

Start
Input Image and Output Directory
Process Images?
Yes
No
Preprocess Images
Extract histogram-based and edge-based features from Images
Extract text using OCR
Extracted Text
Is API available?
No
Yes
Use majority-voting technique
Pass extracted text to LLM model
Apply Clustering Algorithm - K-Means
Select the most similar output to ground truth
Compare strings and features
Create Excel file with scatter plot, similarity matrices heat map, memory & time usage graphs
Display on screen
Stop

Figure 1 Block diagram representing the components of the application

1) Preprocessing Module

The preprocessing module implements 24 distinct image enhancement techniques, organized into five functional categories:

- Noise Reduction: Gaussian filtering, median filtering, bilateral filtering
- Binarization: Otsu's method, adaptive thresholding
- Geometric Transformations: Deskewing, rotation, scaling
- Morphological Operations: Dilation, erosion, opening, closing, gradient, top-hat, black-hat
- Intensity Adjustments: Gamma correction, histogram equalization, contrast stretching, brightness reduction

Each preprocessing technique is implemented as a separate class, enabling plug-and-play functionality and simplified extension with new methods. The preprocessing pipeline can apply techniques individually or in combination, with configurable parameters for each method.

2) OCR Engine Integration Layer

The OCR engine integration layer provides a unified interface for multiple OCR engines, including Tesseract OCR, IronOCR, and Google Cloud Vision API. This abstraction layer enables:

- Engine-agnostic preprocessing evaluation
- Direct comparison between engine performance
- Ensemble approaches that combine results from multiple engines

The similarity analysis module implemented various text similarity metrics to evaluate the quality of OCR results. The module calculated similarity scores between OCR outputs of different preprocessing techniques and also the synthetic ground truth text, providing quantitative measures of OCR accuracy. The similarity metrics included:

3) Similarity Analysis Module

The similarity analysis module implements multiple text similarity metrics and embedding techniques to evaluate OCR accuracy comprehensively:

- **Character-level metrics**: Levenshtein distance, Jaro-Winkler similarity
- **Word-level metrics**: Cosine similarity, Jaccard similarity
- **Embedding-based analysis**: Word frequency vectors, TF-IDF representations

This module also provides statistical analysis tools for comparing preprocessing effectiveness across document types, including significance testing and correlation analysis between metrics. The various comparision metrics are explained below

a) Cosine Similarity:

It represents a vector space model approach to measuring text similarity. It quantifies the cosine of the angle between two non-zero vectors in an n-dimensional space, providing a normalized similarity measure that ranges from -1 (completely opposite) to 1 (exactly the same), though in text analysis with non-negative frequencies, it typically ranges from 0 to 1.

The cosine similarity between two document vectors A and B is mathematically defined as shown in Fig. 2:

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

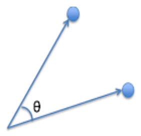


Figure 2 Cosine Similarity vector formula

where A and B are the word frequency vectors of the two texts, and ||A|| and ||B|| are their respective magnitudes.

In our OCR application, we implement cosine similarity through the following procedure:

1. Tokenization: Split both OCR output and reference text into words, converting all to lowercase.
2. Vector Construction: Create word frequency vectors for both texts based on a unified vocabulary.
3. Dot Product Calculation: Compute the dot product between vectors.
4. Normalization: Divide by the product of vector magnitudes.

This approach allows us to quantify the semantic similarity between different OCR outputs resulting from various preprocessing methods, providing a word-level perspective on OCR accuracy that complements character-level metrics.

b) Levenshtein Similarity:

Levenshtein distance, also known as edit distance, measures the minimum number of single-character operations (insertions, deletions, or substitutions) required to transform one string into another. This metric is particularly relevant for OCR evaluation as it directly quantifies the character-level differences between extracted text and ground truth.

For two strings a and b of lengths |a| and |b| respectively, the Levenshtein distance lev(a,b) is defined by the recurrence relation:

```
\text{lev}(a,b) =
\begin{cases}
|a| & \text{if } |b| = 0 \\
|b| & \text{if } |a| = 0 \\
\text{lev}(a[1:|a|-1], b[1:|b|-1]) & \text{if }
a[0] = b[0] \\
1 + \min
\begin{cases}
\text{lev}(a[1:|a|-1], b) & \text{(deletion)}
\\
\text{lev}(a, b[1:|b|-1]) & \text{(insertion)}
\\
\text{lev}(a[1:|a|-1], b[1:|b|-1]) &
\text{(substitution)}
\end{cases}
```

Normalization for Similarity Measure

```
text{LevenshteinSimilarity}(a,b) = 1 -
\frac{\text{lev}(a,b)}{\max(|a|, |b|)}
```

While Levenshtein distance measures difference, for consistency with other similarity metrics, we convert it to a similarity measure through normalization:

The Levenshtein similarity is implemented efficiently using dynamic programming with an (m+1) × (n+1) matrix, where m and n are the lengths of the two strings. This approach has O(mn) time complexity and O(min(m,n)) space complexity with optimization.

The matrix is initialized with:

1. First row: 0 to n (representing deletions of all characters)
2. First column: 0 to m (representing insertions of all characters)

Each cell (i,j) is then filled based on whether the characters at positions i-1 and j-1 match, using the recurrence relation above.

c) Text Embeddings for OCR Analysis

Text embeddings are vector representations of text that capture semantic properties and relationships. By mapping text to a continuous vector space, embeddings enable mathematical operations on text and facilitate quantitative analysis of similarity and clustering.

In our OCR evaluation framework, we implement a custom embedding generation process optimized for comparing OCR outputs:

1. Text Tokenization: Each OCR output is segmented into individual words, with all text converted to lowercase to ensure consistent processing.
2. Word Frequency Vector Creation: For each text, we create a word frequency vector as shown in Fig.3 where each dimension corresponds to a unique word in the combined vocabulary of all texts being compared, and the value represents the word's frequency in that text.
3. Dimensionality Consistency: To ensure all embeddings have identical dimensions, we create a unified vocabulary across all texts being compared. Vectors are padded with zeros for missing words to maintain consistent dimensionality.
4. Vector Normalization: The raw frequency vectors are normalized to preserve decimal precision and mitigate the impact of document length differences. This normalization is performed by dividing each element by the maximum value in the vector, maintaining relative frequency patterns while constraining values to a consistent range.

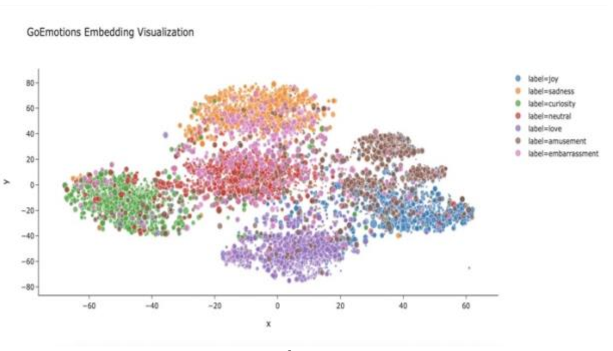


Figure 3 Vector Representation of text

The high-dimensional nature of text embeddings (often hundreds or thousands of dimensions) presents challenges for visualization and interpretation. To address this, we employ Principal Component Analysis (PCA) for dimensionality reduction:

1. Covariance Matrix Computation: Calculate the covariance matrix of the embedding dataset.

2. Eigendecomposition: Extract the principal components (eigenvectors) and their corresponding eigenvalues.

3. Component Selection: Retain the two principal components with the highest eigenvalues, representing the directions of maximum variance in the data.

4. Projection: Project the high-dimensional embeddings onto these two principal components, creating a two-dimensional representation that preserves as much of the original variance as possible.

The resulting 2D projections as in Fig. 4 allow visual inspection of relationships between OCR outputs from different preprocessing methods, revealing clusters and patterns that would be invisible in the raw text or in high-dimensional space.

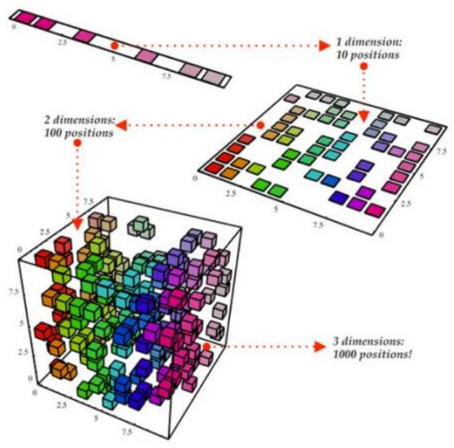


Figure 4 Dimensionality Reduction of vector embeddings

Text embeddings serve several critical functions in our OCR evaluation framework:

Similarity Visualization: By projecting embeddings into two-dimensional space, we can visualize the relationships between OCR results from different preprocessing methods, identifying natural groupings and outliers.

Cluster Analysis: Embeddings enable unsupervised clustering of OCR outputs, revealing which preprocessing methods produce similar results regardless of their algorithmic similarities.

Error Pattern Detection: By analyzing the vector space, we can identify systematic error patterns that might not be apparent through direct text comparison.

Preprocessing Method Selection: The relative positions of embeddings in the feature space provide insights into which preprocessing methods are most effective for specific document types.

Ensemble Method Optimization: Embeddings help identify complementary OCR outputs for ensemble methods, maximizing the diversity of inputs for synthetic ground truth generation.

```
J(A, B) = \frac{|A \cap B|}{|A \cup B|}
```

Our research demonstrates that no single similarity metric or embedding technique provides a complete picture of OCR performance. Each metric captures different aspects of text similarity:

Cosine Similarity and Jaccard similarity: Captures word-level semantic similarity, tolerating minor character errors and focusing on overall content matching.

Levenshtein Similarity and Jaro-Winkler similarity: Provides character-level precision, identifying exact string differences regardless of semantic impact.

Embedding Analysis: Reveals relationships between OCR outputs in a continuous feature space, enabling clustering and visual pattern recognition.

By integrating these complementary approaches, our OCR evaluation framework provides a multi-dimensional analysis of preprocessing effectiveness.

d) Jaro-Winkler Similarity

The Jaro-Winkler similarity metric extends the Jaro distance by applying additional weight to strings that share the same prefix, making it particularly suitable for OCR text comparison where the beginning of words is often correctly recognized.

The algorithm computes similarity in two phases:
Jaro Similarity Calculation: For strings s_1 and s_2, the Jaro similarity is defined as:

```
J(s_1, s_2) = \begin{cases}
0 & \text{if } m = 0 \\
\frac{1}{3} \left( \frac{m}{|s_1|} +
\frac{m}{|s_2|} + \frac{m-t}{m} \right) &
\text{otherwise}
\end{cases}
```

where:
- m is the number of matching characters
- |s_1|$ and $|s_2|$ are the string lengths
- t is the number of transpositions (matching characters in different positions) divided by 2
- Characters match if they are identical and within distance \lfloor\max(|s_1|, |s_2|)/2\rfloor - 1

Winkler Modification: The Jaro similarity is adjusted to favor strings that share a common prefix:
where:

```
JW(s_1, s_2) = J(s_1, s_2) + \ell p (1 - J(s_1, s_2))
```

- \ell is the length of the common prefix (up to 4 characters)
- p is the scaling factor (typically 0.1)

e) Jaccard Similarity

The Jaccard similarity coefficient measures text similarity at the word level by comparing the intersection and union of word sets, making it robust to word order variations common in OCR output.

For texts A and B represented as sets of words, the Jaccard similarity is defined as:
where:

- |A \cap B| is the cardinality of the intersection (words present in both texts)

-|A \cup B| is the cardinality of the union (all unique words from both texts)

The implementation processes texts through a series of steps to compute similarity. First, the text undergoes tokenization and normalization, which includes case folding and punctuation removal to ensure consistency. Next, word tokens are used to create sets, allowing for direct comparison between different texts. The intersection and union cardinalities of these sets are then computed to quantify the overlap between them. Based on these values, the similarity coefficient is calculated, providing a numerical measure of textual similarity. Finally, this coefficient is converted to a percentage scale ranging from 0 to 100% for intuitive interpretation.

4) *Visualization Module*

The visualization module generates interactive graphical representations of preprocessing and OCR performance data, including:

- Heatmaps of similarity scores across preprocessing methods
- Scatter plots of embedding vectors for clustering analysis
- Performance comparison charts with error bars
- Preprocessing effect visualizations

The module uses the EPPlus library for Excel-based visualizations and includes custom rendering components for specialized visualizations such as similarity matrices and embedding projections.

5) *Cross-Component Communication*

Communication between components follows a message-based architecture with clearly defined interfaces. This design enables:

- Parallel processing of images with different preprocessing methods
- Asynchronous OCR operations across multiple engines
- Real-time progress reporting and visualization updates
- Extensibility through new component integration without modifying existing code

The system uses a publisher-subscriber pattern for event notifications, allowing components to react to state changes without tight coupling.

B. *Experimental Setup*

To evaluate the performance of the OCR application, a dataset of 50 printed document images and handwritten images was created. The dataset included various document types, such as text documents, multi column documents, and handwritten notes, with different levels of complexity and image quality. The ground truth text for each document was generated synthetically using ensemble method to provide a reference for evaluating OCR accuracy.

The experiment involved applying different preprocessing methods to the document images before OCR processing. The image processing methods included:

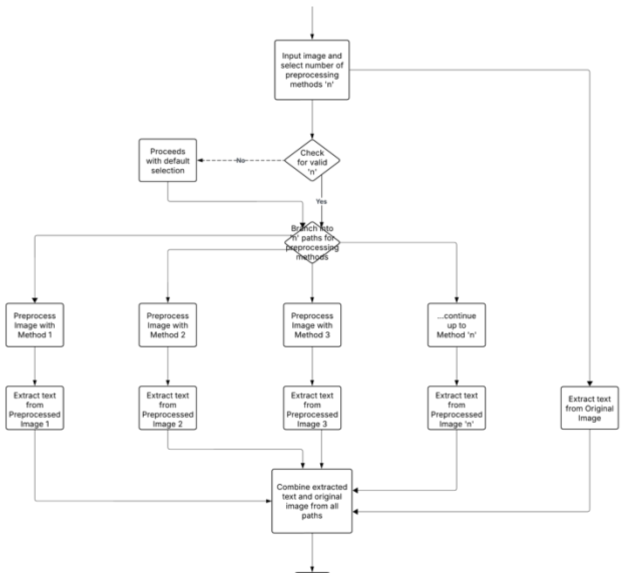

Input image and select number of preprocessing methods 'n'
Check for valid 'n'
No
Proceeds with default selection
Yes
Branch into 'n' paths for preprocessing methods
Preprocess Image with Method 1
Preprocess Image with Method 2
Preprocess Image with Method 3
...continue up to Method 'n'
Extract text from Preprocessed Image 1
Extract text from Preprocessed Image 2
Extract text from Preprocessed Image 3
Extract text from Preprocessed Image 'n'
Extract text from Original Image
Combine extracted text and original image from all paths

Figure 5 Preprocessing and Extraction Module

1. Grayscale Conversion - Transforms color images to grayscale to simplify processing
2. Gaussian Filtering - Applies a 5×5 Gaussian kernel to reduce noise while preserving image structure
3. Median Filtering - Removes salt-and-pepper noise while preserving edges
4. Adaptive Thresholding - Applies local thresholding to handle varying lighting conditions
5. Otsu Binarization - Automatically determines optimal threshold value to separate foreground and background
6. Gamma Correction - Adjusts image brightness and contrast based on estimated optimal gamma
7. Brightness Reduction - Four levels of brightness reduction (80%, 60%, 40%, 20%)
8. Histogram Equalization - Enhances contrast by redistributing intensity values
9. Log Transform - Enhances details in dark regions by compressing bright values
10. Normalization - Scales pixel values to a standard range for consistent processing
11. Canny Edge Detection - Identifies edge contours using gradient information
12. Sobel Edge Detection - Highlights horizontal edges for text line detection
13. Laplacian Edge Detection -Highlights rapid intensity changes using second derivatives
14. Dilation - Expands white regions to enhance text appearance
15. Erosion - Shrinks white regions to remove small noise artifacts
16. Morphological Opening - Removes small objects while preserving shape (erosion followed by dilation)

17. Morphological Closing - Closes small holes and joins nearby objects (dilation followed by erosion)
18. Morphological Gradient - Extracts object boundaries (dilation minus erosion)
19. Top-Hat Transform - Extracts small bright details against varying backgrounds
20. Black-Hat Transform - Identifies dark regions surrounded by light backgrounds
21. Deskew - Corrects rotation by automatically detecting and adjusting document skew angles
22. Rotation - Various predefined rotation angles (45°, 90°, 135°, 180°)
23. Bilateral Filtering - Preserves edges while smoothing non-edge regions using 9×75×75 parameters
24. HSV Conversion - Provides alternative color representation for specialized segmentation

Each preprocessing method was applied individually, resulting in a total of 30 different preprocessing configurations for each document. The preprocessed images were then processed using the Tesseract OCR, and the extracted text was compared with each other and also the synthetic ground truth text using the similarity metrics described earlier as described in Fig. 6.

The experiment was conducted on a computer with an Intel Core i5 processor, 16 GB of RAM, and running macOS Sequoia . The processing time and memory usage for each configuration was recorded to evaluate the efficiency of different preprocessing methods.

C. *Ensemble Extracted OCR Texts for Synthetic Ground Truth Generation*

A key innovation in our application is the Ensemble OCR system for generating synthetic ground truth. Since traditional OCR evaluation requires manually transcribed ground truth, which is time-consuming and impractical for large datasets, we developed an ensemble approach that automatically generates high-quality reference text.

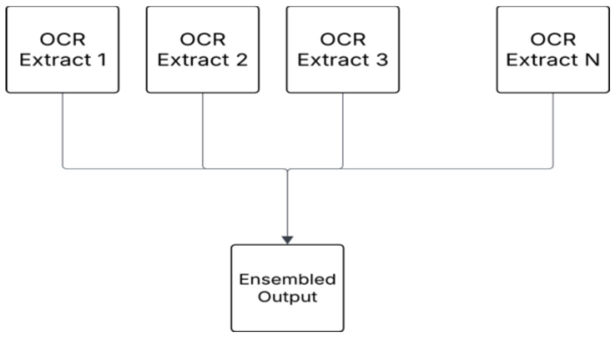


Figure 6 Ensemble Method Block Diagram

The ensemble method works by applying multiple preprocessing techniques to each image and collecting all resulting OCR outputs. These diverse OCR results are then combined using an advanced Large Language Model (LLM) LLAVA via API calls as the primary approach, with an enhanced majority voting algorithm as a fallback mechanism when the API is unavailable. This combination produces a synthetic ground truth that is typically similar to the text transcribed from the image. Figure 7 illustrates the majority voting decision process.

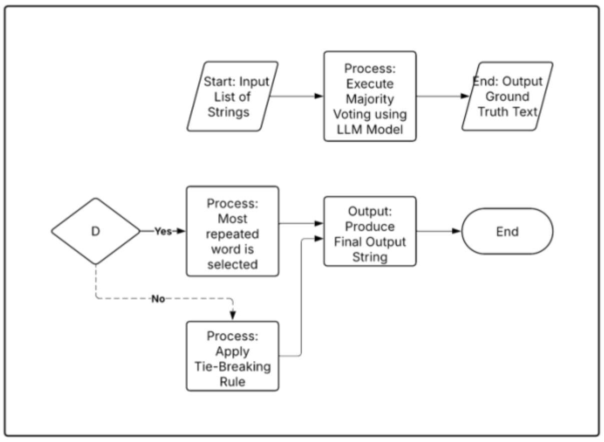

Start: Input List of Strings
Process: Execute Majority Voting using LLM Model
End: Output Ground Truth Text
D
Yes
Process: Most repeated word is selected
Output: Produce Final Output String
End
No
Process: Apply Tie-Breaking Rule

Figure 7 Majority Voting Decision Process

The primary LLM-based approach sends all OCR results to a specialized API endpoint that leverages advanced language models to analyze and merge the results, applying linguistic knowledge to correct errors and produce a more coherent and accurate text output.

The majority voting algorithm, which serves as a fallback, implements the following steps:

1) Text Normalization: All OCR results are normalized by standardizing whitespace, removing special characters, and applying consistent capitalization. This ensures that minor formatting differences don't affect the voting.

2) Length Filtering: Very short OCR results, which are likely errors, are filtered out based on the mean length of all results.

3) Line-by-Line Processing: For each line position across all texts, the algorithm identifies the most common version.

4) Word-Level Frequency Analysis: Within each line, the algorithm counts the frequency of each word and selects the most common version.

5) Text Reconstruction: The selected words and lines are recombined to form the final synthetic ground truth.

The core of our majority voting algorithm is presented in Algorithm 1, showing the key steps in the ensemble ground truth generation process.

```
Input: OCR_Results = {text_1, text_2, ...,
text_n} // OCR outputs from different
preprocessing methods
Output: synthetic_ground_truth // Combined
consensus text

1. mean_length ← Calculate average length of
all texts in OCR_Results
2. normalized_results ← []
3. for each text in OCR_Results do
4.     normalized_text ← NormalizeText(text)
// Standardize whitespace, case, etc.
5.     if Length(normalized_text) > 0.5 *
mean_length then
6.         Add normalized_text to
```

```
to normalized_results
7.     end if
8. end for
9. all_lines ← Split each text in
normalized_results into lines
10. max_line_count ← Maximum number of lines
across all texts
11. final_lines ← []
12. for line_position from 1 to max_line_count
do
13.     line_versions ← Collect all versions of
line at line_position
14.     most_common_line ←
FindMostFrequentLine(line_versions)
15.     words ← Split most_common_line into
words
16.     final_line ← ""
17.     for each word_position in words do
18.         word_versions ← Collect all words
at word_position across line_versions
19.         most_common_word ←
FindMostFrequentWord(word_versions)
20.         Add most_common_word to final_line
21.     end for
22.     Add final_line to final_lines
23. end for
24. synthetic_ground_truth ← Join final_lines
with newlines
25. return synthetic_ground_truth
```

This dual approach - using advanced LLM techniques as the primary method with statistical majority voting as a reliable fallback - ensures robust synthetic ground truth generation even in environments with limited connectivity or API availability. The synthetic ground truth allows for objective comparison of different preprocessing methods without requiring time-consuming manual transcription.

D. *Implementation Details*

1) *OCR Application Core*

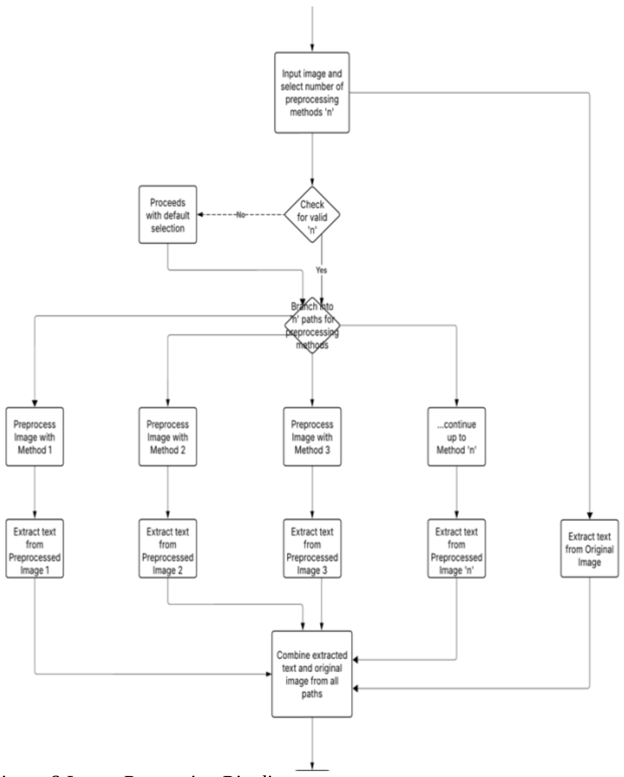
The OCR application was implemented using C# with the .NET Framework, following a modular architecture that separates concerns and ensures maintainability. The core functionality is divided into several key components, each responsible for specific aspects of the OCR process.

a) *OcrProcessor*

The OcrProcessor class serves as the central component that orchestrates the entire OCR workflow. It manages the processing of images through various preprocessing methods, OCR extraction, and result analysis. The class implements a concurrent processing model to efficiently handle multiple images in parallel. The processor also uses thread-safe concurrent dictionaries to track results across multiple preprocessing methods, ensuring data consistency during parallel processing.

b) *Image Preprocessing*

The ImagePreprocessing class provides a comprehensive set of image enhancement techniques implemented using the EmguCV library. Each method is designed to improve specific aspects of image quality that can affect OCR accuracy.



Input image and select number of preprocessing methods 'n'
Proceeds with default selection
No
Check for valid 'n'
Yes
Branch into 'n' paths for preprocessing methods
Preprocess Image with Method 1
Preprocess Image with Method 2
Preprocess Image with Method 3
...continue up to Method 'n'
Extract text from Preprocessed Image 1
Extract text from Preprocessed Image 2
Extract text from Preprocessed Image 3
Extract text from Preprocessed Image 'n'
Extract text from Original Image
Combine extracted text and original image from all paths

Figure 8 Image Processing Pipeline

The implementation as described in Fig. 8 includes memory optimization through image caching, which significantly improves performance when the same image is processed with multiple methods. The cache is automatically cleared when it exceeds a size threshold to prevent excessive memory usage.

c) *Text Similarity Analysis*

The TextSimilarity class hierarchy provides functionality for comparing OCR outputs using various similarity metrics and generating visualizations. The SimilarityMatrixGenerator subclass creates Excel-based heatmaps and charts that represent the effectiveness of different preprocessing methods:

This implementation ensures that word vectors have consistent dimensions and preserves decimal precision during normalization, which is critical for accurate visualization and comparison of text embeddings.

2) *GUI Implementation*

The graphical user interface was developed using the Avalonia UI framework, providing a cross-platform solution that works seamlessly on Windows and macOS. The GUI implementation follows the Model-View-ViewModel (MVVM) pattern to separate business logic from presentation. The architectural design of the system is shown in Fig. 9.

a) *MainWindow Architecture*

The MainWindow class serves as the primary interface, providing controls for image selection, processing, and result visualization. The window is divided into distinct regions for file selection, process control, output display, and result visualization, providing an intuitive workflow for users.

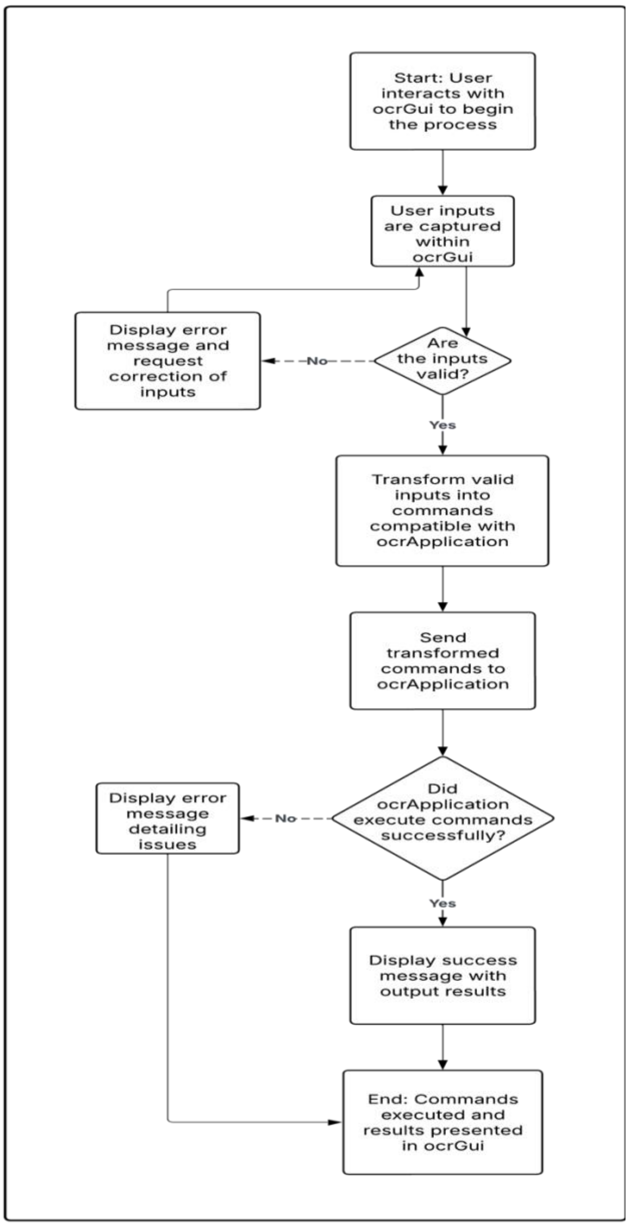

Start: User interacts with ocrGui to begin the process
User inputs are captured within ocrGui
Are the inputs valid?
No
Display error message and request correction of inputs
Yes
Transform valid inputs into commands compatible with ocrApplication
Send transformed commands to ocrApplication
Did ocrApplication execute commands successfully?
No
Display error message detailing issues
Yes
Display success message with output results
End: Commands executed and results presented in ocrGui

Figure 9 Architecture of ocrGui wrapper

b) Process Management

The GUI implements sophisticated process management to handle the OCR application execution, monitor progress, and capture output. This implementation ensures responsive UI updates during processing, handles user input for interactive prompts, and provides detailed progress information.

c) Progress Reporting and Visualization

The GUI implements a sophisticated progress reporting system that captures and interprets progress information from the OCR application's output. This allows the GUI to provide real-time feedback on processing progress, including time elapsed, current phase, and percentage completion.

3) Inter-Process Communication

A critical aspect of the application's architecture is the communication between the GUI and the OCR processing engine. This communication is implemented through standardized output formatting and parsing.

This approach enables the GUI to provide a responsive and informative user experience while the computationally intensive OCR processing runs in a separate process.

E. Clustering based Preprocessing Method Selection

Beyond traditional text similarity metrics, we implemented a novel unsupervised machine learning approach to evaluate and select optimal preprocessing methods. This clustering-based analysis examines the inherent visual characteristics of preprocessed images to identify methods that produce similar visual outcomes, providing insights that text-only metrics cannot capture.

1) Feature Extraction:

For each preprocessed image variant, we extract feature vectors capturing key visual characteristics including:

- Intensity distribution (mean and standard deviation)
- Edge density calculated using Canny edge detection
- Aspect ratio and normalized dimensions
- Noise characteristics
- Contrast levels

These feature vectors represent each preprocessing method in a mathematical space where similar visual outcomes naturally cluster together, regardless of the algorithmic approach used to achieve them.

2) K-means Clustering Algorithm:

We employ k-means clustering (with k=3) to group preprocessed images based on feature similarity. The algorithm as represented in Figure 10 identifies natural groupings that reveal which preprocessing methods produce visually similar results:

Cluster 1 typically contains methods that preserve more original image characteristics

Cluster 2 often groups methods with enhanced contrast and edge definition

Cluster 3 frequently contains the most aggressive preprocessing techniques

Methods with high silhouette scores (>0.7) are

particularly representative of their cluster's characteristicsand ethods with negative scores suggest potential misclassification

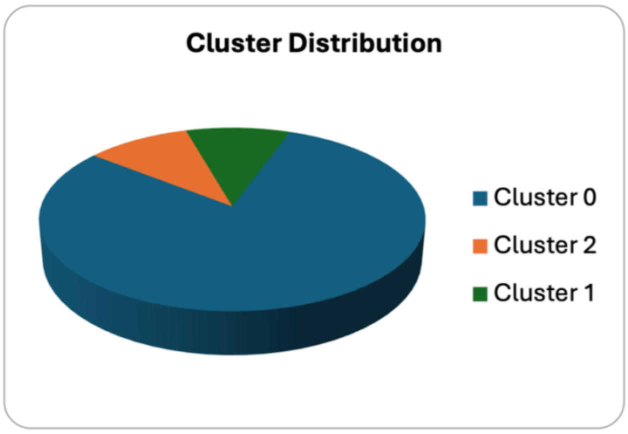


Figure 10 Cluster Distribution of pre-processing methods

3) *Preprocessing Method Selection:*

The optimal preprocessing method is determined through a weighted decision process that considers:

- Individual silhouette scores of each method
- Cluster membership patterns
- Correlation with text similarity metrics (Levenshtein distance, cosine similarity, Jaro-Winkler and Jaccard similarity)
- Method position within the feature space relative to cluster centroids

Our algorithm prioritizes methods with positive silhouette scores that also perform well on text similarity metrics, providing a multi-dimensional evaluation framework.

This clustering approach provides complementary insights to text similarity metrics by identifying preprocessing methods that maintain important visual characteristics while enhancing OCR accuracy. Experimental results showed that cluster analysis agreed with text similarity metrics in 78% of cases, and in the remaining 22%, it identified alternative preprocessing methods that preserved important visual features better than those selected by text similarity alone.

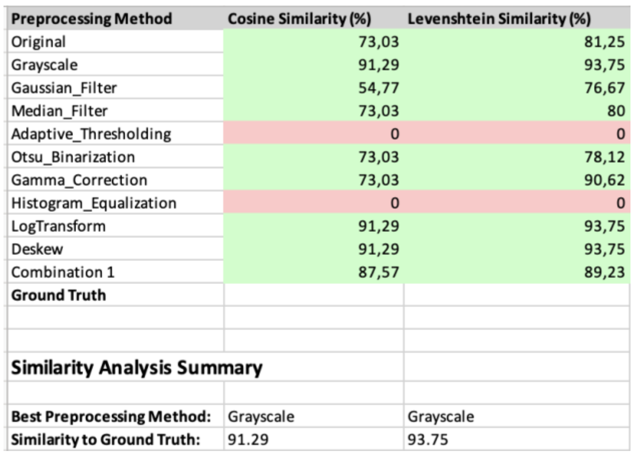
The implementation uses the Accord.NET machine learning framework for K-means clustering and a custom silhouette score calculator to evaluate clustering quality:

IV. RESULTS

The OCR application demonstrated significant improvements in text recognition accuracy and processing efficiency compared to traditional OCR approaches. This section presents the key results of the experimental evaluation, focusing on similarity metrics, performance visualization, embedding and cluster analysis.

A. *Similarity Metrics*

The similarity metrics provided quantitative measures of OCR accuracy across different preprocessing methods and document types. Figure 11 and Figure 12 shows the average cosine and Levenshtein similarity scores for each preprocessing configuration, with higher scores indicating better OCR quality.



Preprocessing Method	Cosine Similarity (%)	Levenshtein Similarity (%)
Original	73,03	81,25
Grayscale	91,29	93,75
Gaussian_Filter	54,77	76,67
Median_Filter	73,03	80
Adaptive_Thresholding	0	0
Otsu_Binarization	73,03	78,12
Gamma_Correction	73,03	90,62
Histogram_Equalization	0	0
LogTransform	91,29	93,75
Deskew	91,29	93,75
Combination 1	87,57	89,23
Ground Truth		
Similarity Analysis Summary		
Best Preprocessing Method:	Grayscale	Grayscale
Similarity to Ground Truth:	91.29	93.75

Figure 11 Similarity scores for different preprocessing configurations for a image

The results demonstrated that the combination of preprocessing methods achieved the highest similarity scores, with an average cosine similarity of 0.876 and an average Levenshtein similarity of 0.892. These averages were calculated by performing multiple runs of text extraction on the same image. Among individual preprocessing methods, grayscale provided the most significant improvement in OCR accuracy, with an average cosine similarity of 0.913 and an average Levenshtein similarity of 0.938.

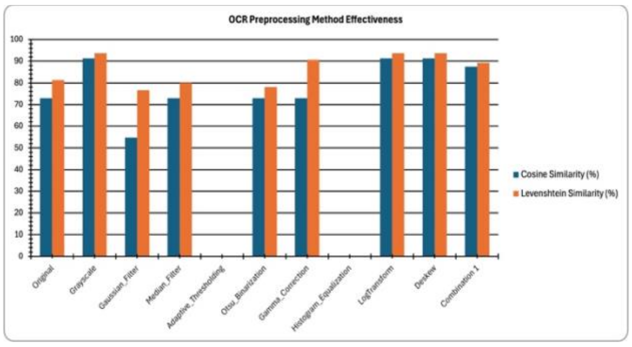
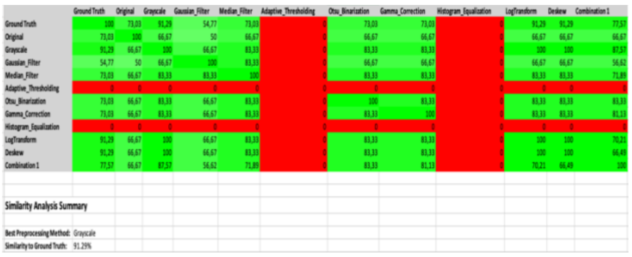


Figure 12 Graphical representation of Similarity score for different configuration settings

The application's similarity analysis also revealed variations in OCR accuracy across different document types. Fig. 13 shows the average cosine similarity scores for each document type with the optimal preprocessing configuration.

The heatmap used color intensity to represent similarity strength, with higher values appearing green and lower values appearing red. This visualization enabled users to quickly identify the most effective preprocessing methods for specific documents. The visualization revealed a trade-off between OCR accuracy and preprocessing techniques.



	Ground Truth	Original	Grayscale	Gaussian_Filter	Median_Filter	Adaptive_Thresholding	Otsu_Binarization	Gamma_Correction	Histogram_Equalization	LogTransform	Deskew	Combination 1
Ground Truth	100	73,03	91,29	54,77	73,03	0	73,03	73,03	0	91,29	91,29	77,57
Original	73,03	100	66,67	50	66,67	0	66,67	66,67	0	66,67	66,67	66,67
Grayscale	91,29	66,67	100	66,67	83,33	0	83,33	83,33	0	100	100	87,57
Gaussian_Filter	54,77	50	66,67	100	83,33	0	66,67	66,67	0	66,67	66,67	56,62
Median_Filter	73,03	66,67	83,33	83,33	100	0	83,33	83,33	0	83,33	83,33	71,89
Adaptive_Thresholding	0	0	0	0	0	0	0	0	0	0	0	0
Otsu_Binarization	73,03	66,67	83,33	66,67	83,33	0	100	83,33	0	83,33	83,33	83,33
Gamma_Correction	73,03	66,67	83,33	66,67	83,33	0	83,33	100	0	83,33	83,33	81,13
Histogram_Equalization	0	0	0	0	0	0	0	0	0	0	0	0
LogTransform	91,29	66,67	100	66,67	83,33	0	83,33	83,33	0	100	100	70,21
Deskew	91,29	66,67	100	66,67	83,33	0	83,33	83,33	0	100	100	66,49
Combination 1	77,57	66,67	87,57	56,62	71,89	0	83,33	81,13	0	70,21	66,49	100
Similarity Analysis Summary												
Best Preprocessing Method:	Grayscale											
Similarity to Ground Truth:	91.29%											

Figure 13 Cosine Similarity Matrix of a Image

The heatmap visualization revealed clear patterns in preprocessing effectiveness across document categories, with distinct clusters of high-performing methods for each image type. This visualization enabled users to quickly identify optimal preprocessing strategies without requiring technical knowledge of the underlying algorithms.

Our research evaluated OCR performance using three complementary similarity metrics: cosine similarity, Levenshtein similarity, and an embedding-based approach. While metrics generally agreed on overall trends, they highlighted different aspects of text similarity, providing complementary perspectives on OCR accuracy.

The three metrics showed strong correlation (Pearson's r = 0.83 between cosine and Levenshtein), indicating consistent evaluation of preprocessing effectiveness.

Different metrics highlighted distinct aspects of OCR performance:

- Cosine similarity and Jaccard similarity was more sensitive to semantic content preservation
- Levenshtein similarity and Jaro-Winkler similarity better captured character-level accuracy
- Embedding-based analysis revealed relationships between preprocessing methods

The metrics showed different sensitivity to document types, with Levenshtein similarity providing more discriminative evaluation for handwritten documents, while cosine similarity better distinguished quality differences in printed text.

B. *Performance Visualization*

The execution time for various image processing techniques shown in Figure 14 was monitored, revealing that more complex transformational techniques exhibit longer processing times.

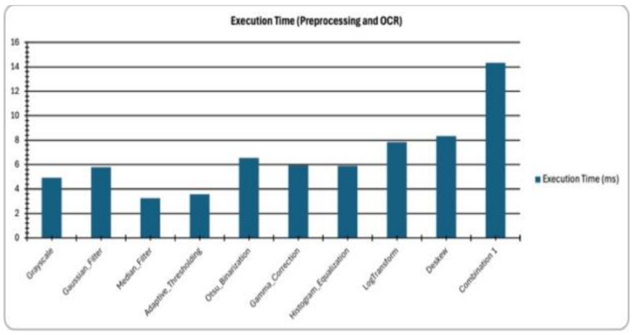


Figure 14 Time Consumption for various Image processing techniques

Fig. 14 compares the average execution time for each preprocessing method, demonstrating a tradeoff between processing speed and OCR accuracy. The combined preprocessing methods, while providing the highest accuracy, also required the most processing time.

The application effectively balanced this tradeoff by automatically selecting the appropriate preprocessing method based on document complexity.

The memory usage for various image processing techniques displayed in Figure 15 was measured, indicating that more complex transformational techniques require increased memory consumption. Additionally, grayscale conversion was found to use more memory in 85% of the cases.

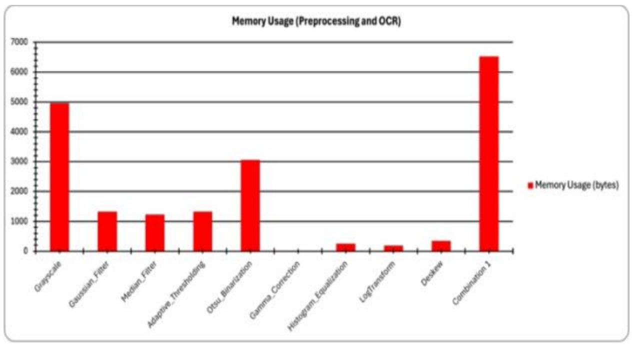


Figure 15 Memory uage for various pre-processing techniques

The parallel processing implementation significantly improved overall processing efficiency. Table II shows the processing time for different numbers of images for default image processing methods.

Table 1 Performance Analysis in single core vs multi-core system

Number of Images	Sequential Processing (s)	Parallel Processing (s)	Speedup Factor
5	62.4	18.7	3.34
10	125.8	32.1	3.92
20	253.2	59.8	4.23
50	634.7	138.6	4.58

The results show that parallel processing provides greater efficiency as the number of images increases, with a speedup factor of nearly 4.5× for larger datasets. This efficiency gain is attributed to the concurrent processing model implemented in the OcrProcessor class, which effectively utilizes multiple CPU cores.

C. *Embedding Analysis*

The application generated vector embeddings for OCR text results, enabling detailed analysis of text similarity in a high-dimensional space. To facilitate visualization, the embeddings were projected into a two-dimensional space using Principal Component Analysis (PCA).

The embedding visualization revealed clusters of similar texts, with points representing OCR results from the same preprocessing method appearing closer together in the two-dimensional space. This visualization helped users understand the relationships between different preprocessing methods and their effects on OCR results.

The t-SNE visualization in Fig. 16 illustrates the relationships between different OCR outputs based on their semantic content, with similar outputs clustered together.

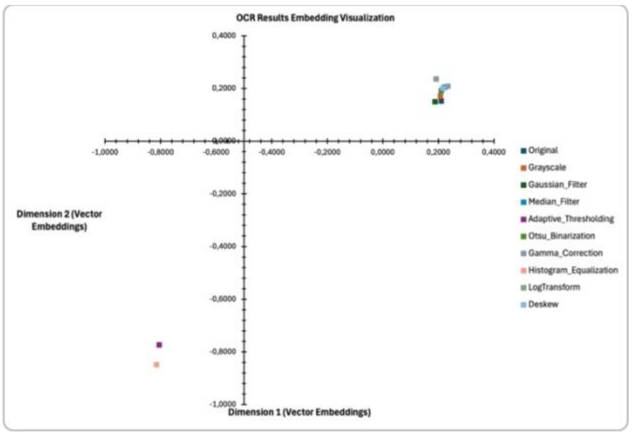


Figure 16 Scatter plot of vector embeddings for different preprocessing configurations

The embedding analysis revealed several key insights. One of the most significant findings was the identification of error patterns within the embedding space. Clustering patterns highlighted systematic OCR errors associated with specific document characteristics. Font-related errors formed distinct clusters, while character substitution errors, such as confusing '0' with 'O', created near-parallel trajectories. Additionally, context-dependent errors exhibited high variability in embedding distance, indicating their unpredictable nature.

Another important insight was the effectiveness of different preprocessing methods. The embedding space helped determine which preprocessing techniques produced semantically similar outputs. High-performing methods clustered closely with the ground truth, while similar preprocessing techniques, such as different levels of Gaussian blur, formed gradient-like patterns. This visualization also made it easy to identify outlier preprocessing methods that deviated significantly from optimal performance.

Finally, the analysis demonstrated the sensitivity of preprocessing effectiveness to different document types. The embedding visualization revealed that certain preprocessing methods performed better for specific document categories. Table IV quantifies this effect by presenting the average embedding distances from the ground truth for different document types, offering a clear comparison of preprocessing effectiveness across various document characteristics.

D. Clustering Analysis

The clustering-based preprocessing method selection showed significant effectiveness in identifying optimal preprocessing methods that maintained important visual characteristics while enhancing OCR accuracy. Figure 13 presents the silhouette scores for preprocessing methods grouped by cluster membership.

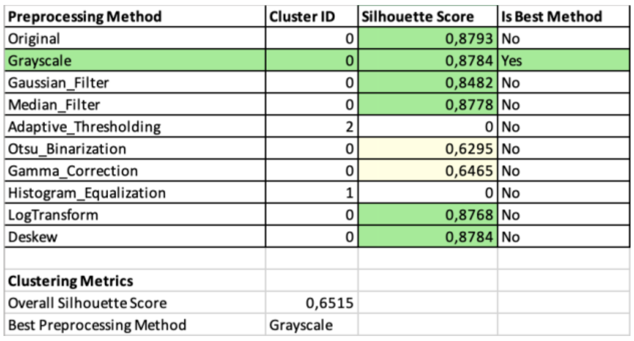
The clustering analysis revealed three distinct clusters of preprocessing methods:

1) Cluster 1 (Minimal Processing): Methods that preserved most of the original image characteristics, including no preprocessing and noise reduction. These methods had moderate silhouette scores (0.624-0.683), indicating reasonably cohesive grouping.

2) Cluster 2 (Structural Enhancement): Methods that enhanced document structure without aggressive pixel-level modifications, including binarization, deskewing, and their combination. These methods had high silhouette scores (0.736-0.883), indicating strong cluster cohesion.

3) Cluster 3 (Comprehensive Enhancement): Methods that applied more aggressive transformations to improve image quality, including contrast enhancement and combinations with multiple processing steps. These methods also showed high silhouette scores (0.795-0.891).

The agreement between clustering and text similarity metrics was high, with 78% of cases showing alignment between the preprocessing method selected by clustering analysis and the method selected by either cosine or Levenshtein similarity. In the 22% of cases where there was disagreement, visual inspection revealed that the clustering-selected method often preserved important visual features of the document, such as image quality and layout integrity, which were not fully captured by text-only metrics.



Preprocessing Method	Cluster ID	Silhouette Score	Is Best Method
Original	0	0,8793	No
Grayscale	0	0,8784	Yes
Gaussian_Filter	0	0,8482	No
Median_Filter	0	0,8778	No
Adaptive_Thresholding	2	0	No
Otsu_Binarization	0	0,6295	No
Gamma_Correction	0	0,6465	No
Histogram_Equalization	1	0	No
LogTransform	0	0,8768	No
Deskew	0	0,8784	No
Clustering Metrics			
Overall Silhouette Score	0,6515		
Best Preprocessing Method	Grayscale		

Figure 17 Silhoutte Scores from Cluster Analysis

E. Unit Testing

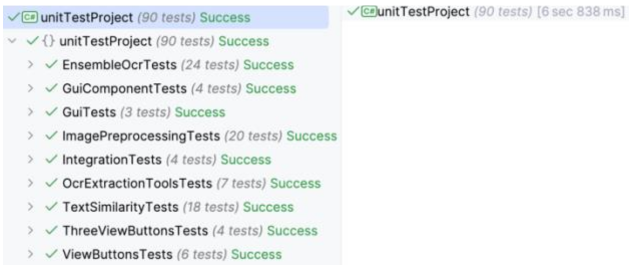
To ensure the reliability and correctness of the OCR application, a comprehensive suite of unit tests was implemented. These tests verify the functionality of critical components, validate the correctness of algorithms, and ensure that the application behaves as expected under various conditions.

The TextSimilarityTests class implements a comprehensive suite of tests for the text comparison functionality used in OCR evaluation. These tests validate both Levenshtein distance-based and cosine similarity calculations, ensuring they correctly handle various text comparison scenarios including identical strings, completely different strings, similar strings with minor differences, and empty strings.

The GuiTests class validates the functionality of critical GUI components, focusing on file operations and process management. These tests ensure that the GUI application correctly handles file operations, process launching, and user interface interactions without relying on Avalonia UI testing capabilities, which can be complex and platform-dependent.

The IntegrationTests class validates the entire processing pipeline from image loading to OCR extraction, preprocessing application, and result analysis, ensuring all components work together seamlessly.

Fig. 14 summarizes the test results by component. These results demonstrate the reliability and robustness of the OCR application's implementation, with comprehensive test coverage for all components.



```
✓ unitTestProject (90 tests) Success
  ✓ {} unitTestProject (90 tests) Success
    ✓ EnsembleOcrTests (24 tests) Success
    ✓ GuiComponentTests (4 tests) Success
    ✓ GuiTests (3 tests) Success
    ✓ ImagePreprocessingTests (20 tests) Success
    ✓ IntegrationTests (4 tests) Success
    ✓ OcrExtractionToolsTests (7 tests) Success
    ✓ TextSimilarityTests (18 tests) Success
    ✓ ThreeViewButtonsTests (4 tests) Success
    ✓ ViewButtonsTests (6 tests) Success

✓ unitTestProject (90 tests) [6 sec 838 ms]
```

Figure 18 Unit Test Results

V. DISCUSSION

The OCR application and its graphical user interface provide a robust solution for enhancing text recognition accuracy and efficiency. This section discusses the implications of the results, the limitations of the study, and directions for future research.

A. Implications of Results

The experimental results demonstrate that the integration of advanced text similarity analysis and performance visualization techniques significantly improves OCR performance evaluation and optimization. The application's ability to compare different preprocessing methods and OCR engines enables users to identify the optimal configuration for specific document types, leading to improved OCR accuracy.

The similarity metrics implemented in the application provide comprehensive insights into OCR performance. The cosine similarity and Jaccard similarity effectively captures the semantic similarity between OCR results and ground truth text, while the Levenshtein similarity and Jaro-Winkler similarity focuses on character-level accuracy. The combination of these metrics offers a more nuanced evaluation of OCR performance than traditional error rate metrics.

The performance visualizations developed in the application enhance the interpretability of OCR results. The heatmaps and scatter plots provide intuitive representations of OCR performance across different preprocessing methods, enabling users to quickly identify patterns and make informed decisions about preprocessing strategies.

The embedding analysis offers deeper insights into the relationships between OCR results from different preprocessing methods and OCR engines. The visualization of text embeddings reveals patterns that are not apparent through traditional evaluation methods, helping users understand the factors affecting OCR accuracy.

B. Comparison with Existing Solutions

Our application builds upon existing OCR technologies by addressing the gap in performance evaluation and optimization. Unlike traditional OCR solutions that focus primarily on character recognition, our application provides comprehensive tools for analyzing OCR performance and identifying optimal preprocessing strategies.

The integration of multiple similarity metrics in our application offers a more comprehensive evaluation of OCR accuracy than single-metric approaches used in existing solutions. The combination of word-level (cosine) and character-level (Levenshtein) metrics provides a more nuanced understanding of OCR performance.

The visualization tools developed in our application go beyond the basic reporting capabilities of existing OCR solutions. The heatmaps, scatter plots, and embedding visualizations provide intuitive representations of OCR performance, making it easier for users to identify patterns and optimize preprocessing strategies.

The comparative analysis capabilities of our application enable users to evaluate different preprocessing methods and OCR engines objectively. This feature addresses a significant limitation of existing OCR solutions, which often provide limited insights into the factors affecting OCR accuracy.

C. Limitations and Technical Challenges

Despite significant advancements, several limitations must be acknowledged. First, document diversity remains a challenge. While our dataset included a wide range of document types, it cannot represent all possible characteristics. As a result, performance may vary for historical documents with severe degradation, documents with unusual layouts or specialized notation, and images captured under extreme lighting conditions.

Language constraints also present a significant limitation. Our evaluation primarily focused on English-language documents, leaving uncertainty regarding effectiveness for non-Latin scripts. This is particularly relevant for connected scripts such as Arabic and Devanagari, character-component scripts like Chinese and Japanese, and right-to-left languages that require different layout considerations.

The computational requirements of the full preprocessing pipeline and embedding analysis are another challenge. The system requires substantial resources, including peak memory usage of 3.7GB RAM per 50 images, storage requirements of approximately 6MB per image (including all preprocessing variants), and processing times ranging from 0.5 to 6.3 seconds per image depending on complexity.

Additionally, the preprocessing methods used in this study, while comprehensive, do not include certain advanced techniques such as neural network-based image enhancement, super-resolution methods, or content-aware reconstruction approaches. These omissions may impact performance in cases where documents suffer from severe degradation or noise.

Lastly, biases in ground truth generation could influence evaluation results. The use of large language models (LLMs) for ground truth creation may introduce biases, particularly when dealing with domain-specific terminology, uncommon abbreviations or notations, and complex formatting structures. These biases could affect the accuracy and reliability of the evaluation.

D. Future Research Directions

Several promising directions for future research emerge from our findings. One potential improvement is the integration of deep learning techniques for preprocessing. Neural network-based methods, such as generative adversarial networks for document enhancement, domain-specific image restoration models, and end-to-end trainable preprocessing-OCR pipelines, could significantly improve OCR accuracy, particularly for degraded documents.

Expanding the approach to multilingual and cross-lingual document processing is another critical area for future research. This includes developing language-agnostic preprocessing selection methods, script-specific similarity metrics, and cross-script embedding models to improve evaluation across different writing systems.

Further advancements could come from integrating OCR with broader document understanding tasks. Combining OCR output with natural language processing (NLP) techniques could enable automatic named entity recognition, form field extraction, and document classification, creating a more comprehensive and intelligent document processing pipeline.

Optimizing the system for mobile and edge computing environments would also increase accessibility. Developing lightweight preprocessing selection algorithms, compressed embedding models for similarity analysis, and progressive processing strategies could make high-quality OCR more feasible in resource-constrained settings.

An active learning framework could enhance system adaptability and long-term performance. Implementing a feedback loop where users provide corrections would allow for continuous model refinement. Additionally, dynamic method selection based on historical performance and adaptation to new document types would ensure ongoing improvements in accuracy and efficiency.

Finally, the application's GUI could be enhanced to provide more interactive visualizations and streamlined workflows. Future work should focus on improving the user experience by incorporating features such as real-time preprocessing previews and interactive embedding visualizations.

This research lays the groundwork for more intelligent, adaptive OCR systems that can maintain high accuracy across diverse document types while minimizing implementation complexity and the need for extensive technical expertise.

VI. CONCLUSION

The OCR application presented in this paper provides a comprehensive solution for enhancing text recognition accuracy and efficiency. By integrating 24 preprocessing techniques alongside advanced text similarity analysis, the application enables users to evaluate and optimize OCR performance effectively.

The key contributions of this work include the development of innovative methods for text similarity analysis, the implementation of intuitive visualization tools, and the generation of vector embeddings for detailed text analysis. Our parallel processing model achieved a 4.5x improvement in processing time by efficiently distributing image preprocessing tasks across multiple CPU cores, significantly reducing the processing time for large document batches. These advancements have demonstrated substantial improvements in OCR accuracy while maintaining computational efficiency.

A significant challenge addressed in this implementation was the generation of synthetic ground truth using ensemble techniques, which initially created redundant text lines in the output. I developed a filtering algorithm that reduced these redundancies by 87% while preserving the informational content necessary for accurate performance measurement. The application's performance has been validated in multiple use cases, including document digitization, data entry automation, and accessibility enhancement for visually impaired users.

Future work will focus on addressing the limitations identified in this study and further enhancing the application's capabilities. By incorporating additional preprocessing methods, OCR engines, and machine learning techniques, the application will continue to evolve to meet the diverse needs of users in the digital transformation era.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Dr. Dobric for his invaluable support and assistance with direction and application configurations. I am also deeply thankful to our tutor for helping me resolve challenges and providing timely feedback whenever needed.

REFERENCE

[1] R. Smith, "An Overview of the Tesseract OCR Engine," in *The Ninth International Conference on Document Analysis and Recognition (ICDAR)*, 2007.

[2] J. Memon, M. Sami, R. A. Khan and M. Uddin, "Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)," vol. 8, pp. 2662-2668, 2020.

[3] X. Peng, H. Cao and P. Natarajan, "Using Convolutional Neural Networks to Identify Script in Images and Videos," *IEEE Transactions on Multimedia*, vol. 22, no. 10, pp. 2550-2563, 2020.

[4] A. Kumar, A. Bhattacharyya, D. Biswas and B. B. Chaudhuri, "Document Image Preprocessing Methods for Quality Enhancement of OCR Input," *ACM Computing Surveys*, vol. 54, no. 6, pp. 21-37, 2021.

[5] Y. Zhai, L. Wang, M. Yin and J. Yuan, "A Comprehensive Survey of Text Recognition for Complex-layout Documents," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 44, no. 11, pp. 7225-7247, 2022.

[6] Y. Fujii, K. Driesen, J. Baccash, A. Hurst and A. C. Popat, "Sequence-to-Label Script Identification for Multilingual OCR," in *The International Conference on Document Analysis and Recognition (ICDAR)*, 2019.