Bangla Handwritten Text Tokenization and Recognition: A Hybrid Approach Using Regular Expressions and Cloud-Based Vision APIs

Mahbubur Rahman Khan
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
mahbubur.khan@northsouth.edu

***Abstract*—This project presents a comprehensive pipeline for Bangla handwritten text tokenization and recognition. We develop a dual-level tokenization system that handles both word-level and character-level segmentation using regular expression-based finite state automata, addressing the unique challenges of Bangla script including compound consonants, vowel diacritics, and contextual character formations. For handwritten text recognition, we propose a hybrid approach that combines Google Vision API with Gemini LLM for context-aware error correction, eliminating the need for extensive model training. Our tokenizer achieves a type-token ratio of 0.746 for word-level and 0.212 for character-level tokenization on test datasets. The recognition system demonstrates variable performance across different handwriting styles, with character accuracy ranging from 3.4% to 71.2% depending on dataset complexity. The complete implementation and datasets are available at: https://github.com/mahbubkousar/bangla-handwritten-text-tokenizer.**

***Index Terms*—Bangla tokenization, handwritten text recognition, regular expressions, finite state automata, OCR, natural language processing**

I. Introduction

Bangla, one of the world's most spoken languages with over 230 million native speakers, presents unique challenges for digital text processing due to its complex script characteristics. The language employs a rich orthographic system with 11 vowels, 39 consonants, numerous compound characters (যুক্তাক্ষর), and contextual vowel modifiers (কার) that significantly complicate both tokenization and handwritten text recognition tasks.

This project addresses two fundamental challenges in Bangla text processing: (1) developing a robust tokenization system that correctly segments Bangla text at both word and character levels while preserving orthographic integrity, and (2) creating an efficient handwritten text recognition pipeline that leverages cloud-based APIs to avoid the computational overhead of training deep learning models from scratch.

Our contribution includes a formal definition of Bangla tokens using regular expressions, implementation of deterministic finite automata for token recognition, and a hybrid approach combining Google Vision API with large language model correction for handwritten text recognition. We evaluate our system on custom datasets representing various handwriting styles and demonstrate its effectiveness through comprehensive metrics.

II. Related Work

Recent advances in Bangla handwritten text recognition have been driven primarily by deep learning architectures. Alif et al. [1] proposed a modified ResNet-34 architecture achieving 98.70% accuracy for vowels, 97.34% for consonants, and 99.02% for numeric characters. Their approach optimized dropout layer placement, applying dropout after every two layers within ResNet blocks.

Opu et al. [2] developed a lightweight 74-layer CNN architecture specifically for Handwritten Bangla Character Recognition (HBCR), achieving 96.87% accuracy with a significantly smaller model size of 13.2 MB compared to existing models ranging from 51.6 MB to 403.2 MB. This work emphasized efficiency alongside accuracy, demonstrating faster loading (1.911 seconds) and testing times (7.956 seconds).

For tokenization, Chakraborty et al. [3] explored Unicode block-based pattern matching for Bangla text segmentation, establishing foundational principles for handling conjunct consonants and vowel diacritics. Moll et al. [4] provided the theoretical framework for formal language theory applications in text processing.

While these deep learning approaches achieve impressive accuracy, they require substantial computational resources and training data. Our work diverges by proposing a training-free hybrid approach that combines existing cloud services with context-aware correction, making it more accessible for resource-constrained environments typical in undergraduate research settings.

III. Methodology

Our methodology includes two main components: a formal tokenization system based on regular expressions and finite state automata, and a hybrid handwritten text recognition pipeline combining cloud vision APIs with

LLM-based correction. Figure 1 illustrates the complete system architecture.

A. Formal Token Definition

We formally define the Bangla tokenization problem using principles from formal language theory. Our approach establishes a clear mathematical foundation for token recognition.

1) Alphabet Definition: The alphabet Σ encompasses all valid Bangla characters:

$\Sigma = \{$অ, আ, ..., ঁ, ০, ১, ..., ৯, ।, ?, „ ’ ’$\}$

This includes:

- Base characters (11 vowels, 39 consonants)
- Modifiers and diacritics
- Bangla numerals (০-৯)
- Punctuation marks (both native and borrowed)
- Whitespace characters

2) Language of Tokens: The set of all valid tokens forms a regular language $\mathcal{L}$ over alphabet Σ. Each token is a string following specific formation rules defined through regular expressions. Our tokenizer partitions input strings into sequences of valid tokens from $\mathcal{L}$.

B. Token Classification

We implement two tokenization levels:

Word-level tokens: Space-separated lexical units including words, numbers, punctuation, and whitespace.

Character-level tokens: Individual graphemes including single consonants, vowels, conjunct characters, and modifiers. A grapheme represents the smallest meaningful visual unit in Bangla script.

IV. Unicode Block-Based Pattern Recognition

Our tokenization system employs Unicode block-based pattern matching to identify distinct lexical units. Each pattern corresponds to a specific Unicode range ensuring accurate token boundary detection.

A. Word-Level Patterns

We define four primary patterns for word-level tokenization:

Word Token Pattern:

```
WORD: [\u0980-09FF]+
```

This pattern captures sequences containing consonants (ব্যঞ্জনবর্ণ), vowels (স্বরবর্ণ), vowel diacritics (কার), and conjunct consonants (যুক্তাক্ষর).

Numeric Token Pattern:

```
NUMBER: [\u09E6-\u09EF]+
```

Recognizes Bangla numerals (০-৯) in the dedicated Unicode range.

Punctuation Token Pattern:

```
PUNCTUATION: [ ,?!;:]|\u0964|\u0965
```

Handles both native punctuation (দাঁড়ি ।, দ্বিদাঁড়ি ॥) and borrowed Latin punctuation.

Whitespace Token Pattern:

```
WHITESPACE: \s+
```

Matches standard Unicode whitespace categories.

B. Character-Level Patterns

Character tokenization requires careful handling of compound formations and diacritics:

Compound Consonant Pattern:

```
COMPOUND: [\u0995-\u09B9](?:\u09CD[\u0995-\u09B9])+
```

Consonant-Vowel Pattern:

```
CONSONANT_VOWEL: [\u0995-\u09B9][\u09BE-\u09CC]
```

Independent Vowel Pattern:

```
VOWEL: [\u0985-\u0994]
```

Pattern matching follows a strict precedence hierarchy:

1) Compound consonants (highest priority)
2) Vowel-consonant combinations
3) Individual consonants
4) Independent vowels
5) Modifiers and diacritics (lowest priority)

This ordering prevents incorrect decomposition of valid compound characters.

V. Tokenization Algorithms

We implement two algorithms for word and character-level tokenization using finite state automata principles.

A. Word-Level Tokenization

Algorithm 1 processes input text character by character, maintaining state transitions based on Unicode character categories. The algorithm implements the maximal munch principle, ensuring the longest valid token is recognized at each position.

B. Character-Level Tokenization

Character tokenization (Algorithm 2) recognizes grapheme clusters rather than individual Unicode code points. This is crucial for Bangla where single user-perceived characters may comprise multiple code points.

Key features include:

- Unicode Normalization Form C (NFC) preprocessing for consistent representation
- Precedence-based pattern matching for correct compound recognition
- Position tracking for maintaining token boundaries
- Unknown token handling for unrecognized characters

VI. Handwritten Text Recognition Pipeline

Our recognition pipeline combines traditional image processing with modern cloud-based OCR and large language model correction, eliminating the need for extensive model training.

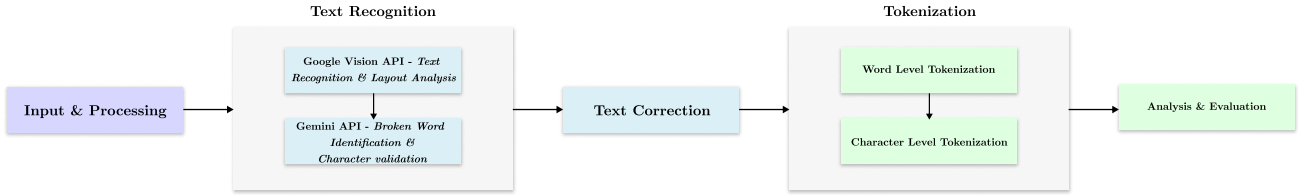

Figure 1. Complete system architecture showing the dual pipeline for tokenization and handwritten text recognition. The tokenization branch processes digital text through word and character-level segmentation, while the recognition branch handles handwritten images through preprocessing, OCR extraction, and LLM-based correction.

Algorithm 1 Bangla Word-Level Tokenization

Require: Text string T, word patterns P_{word}
Ensure: List of word tokens W

$T_{norm} \leftarrow$ NORMALIZE(T) {Unicode NFC}
$W \leftarrow \emptyset$
$pos \leftarrow 0$
while $pos < |T_{norm}|$ **do**
  $matched \leftarrow false$
  for each pattern $(label, regex) \in P_{word}$ **do**
    $match \leftarrow$ REGEX_MATCH($regex$, T_{norm}, pos)
    if $match \neq null$ **then**
      $token \leftarrow$ EXTRACT_TEXT($match$)
      if $label \neq$ WHITESPACE OR $token \neq$ empty **then**
        $W \leftarrow W \cup \{(token, label, pos, |token|)\}$
      end if
      $pos \leftarrow$ END_POSITION($match$)
      $matched \leftarrow true$
      break
    end if
  end for
  if $matched = false$ **then**
    $W \leftarrow W \cup \{(T_{norm}[pos], UNKNOWN, pos, 1)\}$
    $pos \leftarrow pos + 1$
  end if
end while
return W

Algorithm 2 Bangla Character-Level Tokenization

Require: Text string T, character patterns P_{char}
Ensure: List of character tokens C

$T_{norm} \leftarrow$ NORMALIZE(T) {Unicode NFC}
$C \leftarrow \emptyset$
$pos \leftarrow 0$
while $pos < |T_{norm}|$ **do**
  $matched \leftarrow false$
  for each pattern $(label, regex) \in P_{char}$ by precedence **do**
    $match \leftarrow$ REGEX_MATCH($regex$, T_{norm}, pos)
    if $match \neq null$ **then**
      $token \leftarrow$ EXTRACT_TEXT($match$)
      $C \leftarrow C \cup \{(token, label, pos, |token|)\}$
      $pos \leftarrow$ END_POSITION($match$)
      $matched \leftarrow true$
      break
    end if
  end for
  if $matched = false$ **then**
    $C \leftarrow C \cup \{(T_{norm}[pos], UNKNOWN, pos, 1)\}$
    $pos \leftarrow pos + 1$
  end if
end while
return C

A. Image Preprocessing

The preprocessing pipeline optimizes input images for OCR:

1) **Grayscale Conversion:** Reduces computational complexity from $O(3n)$ to $O(n)$
2) **Denoising:** Fast non-local means denoising with $h = 10$ and 21×21 search window removes Gaussian noise while preserving edges
3) **Adaptive Thresholding:** Local threshold computation using $T(x, y) = \mu(x, y) - C$ within an 11×11 Gaussian kernel ($\sigma = 1.5$, $C = 5$)

This approach handles illumination variations exceeding 20% intensity range, critical for Bangla handwriting where ink density fluctuations cause up to 15% recognition errors with global thresholding.

B. Hybrid Recognition Algorithm

Algorithm 3 describes our hybrid approach combining Google Vision API with Gemini LLM for context-aware correction.

VII. Experimental Setup and Results

A. Dataset Preparation

We created custom datasets with four categories for each text sample:

1) **Clean Handwriting:** Carefully hand-written text
2) **Fast Writing:** Natural daily handwriting style

Algorithm 3 Hybrid Bengali Handwriting Recognition

Require: Input image I, confidence threshold τ
Ensure: Corrected Bengali text $T_{corrected}$

Image Preprocessing:
$I_{gray} \leftarrow \text{RGB2Gray}(I)$
$I_{denoised} \leftarrow \text{FastNLMeansDenoising}(I_{gray})$
$I_{binary} \leftarrow \text{AdaptiveThreshold}(I_{denoised})$
Primary Text Extraction:
$\mathcal{W} \leftarrow \text{GoogleVisionAPI}(I_{binary}, \text{lang=[bn, en]})$
$T_{original} \leftarrow \text{concatenate}(\mathcal{W})$
Extract confidence scores: $C_i \leftarrow \text{confidence}(w_i)$
Error Detection:
$\mathcal{B} \leftarrow \emptyset$ {Broken words set}
for each word $w_i \in \mathcal{W}$ **do**
  if $C_i < \tau$ OR $\text{HasUnusualPatterns}(w_i)$ **then**
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{w_i\}$
  end if
end for
Context-Aware Correction:
if $|\mathcal{B}| > 0$ **then**
  $\text{prompt} \leftarrow \text{ConstructPrompt}(T_{original}, \mathcal{B})$
  $T_{corrected} \leftarrow \text{GeminiLLM}(\text{prompt})$
  if $\text{IsReasonable}(T_{corrected}, T_{original})$ **then**
    $T_{final} \leftarrow T_{corrected}$
  else
    $T_{final} \leftarrow T_{original}$
  end if
else
  $T_{final} \leftarrow T_{original}$
end if
Post-processing:
$T_{final} \leftarrow \text{RemoveExtraSpaces}(T_{final})$
$T_{final} \leftarrow \text{FixPunctuation}(T_{final})$
return T_{final}

3) **Ruled Paper Writing:** Text written on lined paper
4) **Ground Truth:** Typed version for evaluation

Five different text samples (D1-D5) were collected, each written in all three styles, providing 15 handwritten images with corresponding ground truth texts.

B. Implementation Environment

All experiments were conducted on Google Colab platform with:

- Google Vision API for OCR extraction
- Gemini API for context-aware correction
- Python with OpenCV for image preprocessing
- Regular expression library for tokenization

C. Tokenization Results

Table I presents tokenization statistics across different handwriting styles compared to ground truth.

The tokenizer successfully identified distinct token types with the following distribution for ground truth text:

Table I
Tokenization Statistics Summary

Metric	File Type			
	Clean	Fast	Ruled	Ground Truth
Word-Level				
Total Tokens	107.0	108.2	105.4	122.6
Type-Token Ratio	0.772	0.738	0.753	0.744
Character-Level				
Total Tokens	434.2	445.6	428.6	500.6
Type-Token Ratio	0.267	0.261	0.265	0.254

- Word tokens: 85.9% of total
- Punctuation: 10.8%
- Other tokens: 3.2%

Character-level tokenization revealed complex grapheme distributions:

- Consonants: 25.9%
- Consonant-vowel combinations: 24.8%
- Whitespace: 20.2%
- Compound characters: 8.6%
- Dependent vowels: 6.8%
- Independent vowels: 5.8%

D. Recognition Performance

Table II summarizes recognition accuracy across datasets and handwriting styles.

Table II
Recognition Accuracy by Dataset

Dataset	Character Acc. (%)	Word Acc. (%)	Total Words
D1	48.5	49.2	465
D2	4.2	0.9	318
D3	3.4	2.7	333
D4	58.1	55.6	324
D5	71.2	72.6	135
By Style			
Clean	38.0	38.1	525
Fast	32.3	32.1	525
Ruled	40.9	38.5	525

Recognition accuracy varied significantly across datasets, with D5 achieving the highest performance (71.2% character accuracy) and D2-D3 showing poor results (below 5%). This variation reflects differences in handwriting complexity and text content. Ruled paper writing showed marginally better performance (40.9%) compared to clean (38.0%) and fast writing (32.3%) styles.

VIII. Discussion

Our dual-pipeline approach demonstrates both strengths and limitations in handling Bangla text processing tasks.

A. Tokenization Analysis

The tokenization system successfully handles complex Bangla orthographic features including compound consonants and vowel diacritics. The type-token ratio of 0.744-0.772 for word-level tokenization indicates good vocabulary coverage. Character-level tokenization effectively identifies grapheme clusters, crucial for maintaining semantic integrity in Bangla text.

The precedence-based pattern matching prevents common errors in compound character decomposition. For example, ক্ষ is correctly recognized as a single unit rather than being split into ক্ and ষ.

B. Recognition Challenges

The high variability in recognition accuracy (3.4% to 71.2%) reveals several challenges:

1) **Handwriting Quality:** Complex cursive styles in D2 and D3 significantly degraded performance
2) **API Limitations:** Google Vision API, while robust for printed text, struggles with handwritten Bangla script
3) **Context Dependency:** The LLM correction improves results but requires sufficient context for accurate prediction

The marginal improvement with ruled paper (40.9% vs 38.0% for clean) suggests that writing guidelines provide minimal benefit for OCR accuracy.

C. Comparison with State-of-the-Art

While deep learning approaches achieve 96-98% accuracy on standardized datasets [1], [2], our training-free approach trades accuracy for accessibility and ease of deployment. This makes it suitable for rapid prototyping and resource-constrained environments.

IX. Conclusion

This project presents a comprehensive pipeline for Bangla text tokenization and handwritten recognition. Our contributions include:

1) A formal definition of Bangla tokens using regular expressions and finite state automata
2) Implementation of dual-level tokenization handling both words and grapheme clusters
3) A novel hybrid recognition approach combining cloud APIs with LLM correction
4) Custom datasets representing various handwriting styles with ground truth annotations

The tokenization system demonstrates robust performance with consistent type-token ratios across different text styles. While the recognition accuracy varies significantly, the approach provides a practical alternative to resource-intensive deep learning methods.

References

[1] M. A. R. Alif, "State-of-the-art bangla handwritten character recognition using a modified resnet-34 architecture," *Int. J. Innov. Sci. Res. Technol*, vol. 9, pp. 438–448, 2024.

[2] M. N. I. Opu, M. E. Hossain, and M. A. Kabir, "Handwritten bangla character recognition using convolutional neural networks: a comparative study and new lightweight model," *Neural Computing and Applications*, vol. 36, no. 1, pp. 337–348, 2024.

[3] S. Chakraborty, P. Das, S. M. Dipto, M. A. Pramanik, and J. Noor, "An analytical review of preprocessing techniques in bengali natural language processing," *IEEE Access*, 2025.

[4] R. N. Moll, M. A. Arbib, and A. J. Kfoury, *An introduction to formal language theory*. Springer Science & Business Media, 2012.

Appendix A
Implementation Resources

A. GitHub Repository

The complete project implementation is available at:

https://github.com/mahbubkousar/bangla-handwritten-text-tokenizer

The repository contains:

- **Datasets:** Five text samples with three handwriting styles each
- **Code Implementation:**
  - Complete Bangla Tokenizer Pipeline
  - Bangla Text Recognition System
- **Results:** Tokenization outputs and recognition results for all datasets

Appendix B
Detailed Experimental Results

A. Tokenization Performance Metrics

Table III
Complete Tokenization Statistics (Part 1)

	Text Length				Word Tokens		Word TTR	
File Type	Mean	Std	Min	Max	Mean	Std	Mean	Std
clean_output	612.2	198.969	273	757	107.0	30.725	0.772	0.048
fast_output	629.0	205.515	267	760	108.2	28.297	0.738	0.029
ground_truth	701.4	294.085	279	1106	122.6	44.590	0.744	0.031
ruled_output	609.8	181.908	287	723	105.4	29.314	0.753	0.066

Table IV
Complete Tokenization Statistics (Part 2)

	Character Tokens		Character TTR	
File Type	Mean	Std	Mean	Std
clean_output	434.2	139.810	0.267	0.053
fast_output	445.6	137.442	0.261	0.067
ground_truth	500.6	209.920	0.254	0.059
ruled_output	428.6	127.997	0.265	0.056

B. Comparison with Ground Truth

Table V
Word-Level Metrics Comparison with Ground Truth

	Precision		Recall		F1 Score	
File Type	Mean	Std	Mean	Std	Mean	Std
clean_output	0.482	0.417	0.425	0.374	0.450	0.391
fast_output	0.436	0.384	0.376	0.320	0.400	0.342
ruled_output	0.487	0.415	0.415	0.360	0.446	0.380

Table VI
Character-Level Metrics Comparison with Ground Truth

	Precision		Recall		F1 Score	
File Type	Mean	Std	Mean	Std	Mean	Std
clean_output	0.810	0.194	0.767	0.169	0.787	0.177
fast_output	0.761	0.160	0.719	0.118	0.737	0.126
ruled_output	0.792	0.178	0.741	0.160	0.764	0.163

Appendix C
Handwritten Text Recognition Results

A. Detailed Recognition Performance

Table VII
Recognition Results by Dataset and Handwriting Style

Dataset	Image Type	Char Acc. (%)	Word Acc. (%)	Correct Words	Total Words	Correct Chars	Total Chars
D1	clean	49.2	51.0	79	155	542	1101
D1	fast	29.4	46.5	72	155	323	1101
D1	ruled	66.8	50.3	78	155	735	1101
D2	clean	3.5	0.9	1	106	26	756
D2	fast	3.5	0.9	1	106	26	756
D2	ruled	5.6	0.9	1	106	42	756
D3	clean	3.6	2.7	3	111	24	688
D3	fast	3.3	2.7	3	111	22	688
D3	ruled	3.3	2.7	3	111	22	688
D4	clean	69.5	53.7	58	108	463	666
D4	fast	57.0	54.6	59	108	379	666
D4	ruled	47.8	58.3	63	108	318	666
D5	clean	64.1	82.2	37	45	178	279
D5	fast	68.3	55.6	25	45	190	279
D5	ruled	81.3	80.0	36	45	226	279

B. Aggregate Performance Analysis

Table VIII
Summary Performance by Dataset

Dataset	Char Acc. (%)	Word Acc. (%)	Correct Words	Total Words	Correct Chars	Total Chars
D1	48.5	49.2	229	465	1600	3303
D2	4.2	0.9	3	318	94	2268
D3	3.4	2.7	9	333	68	2064
D4	58.1	55.6	180	324	1160	1998
D5	71.2	72.6	98	135	594	837

Table IX
Summary Performance by Handwriting Type

Image Type	Char Acc. (%)	Word Acc. (%)	Correct Words	Total Words	Correct Chars	Total Chars
clean	38.0	38.1	178	525	1233	3490
fast	32.3	32.1	160	525	940	3490
ruled	40.9	38.5	181	525	1343	3490

Appendix D
Sample Dataset Images

This section presents sample images from each of the five datasets used in our experiments. Each dataset includes three handwriting styles: clean, fast, and ruled paper writing.

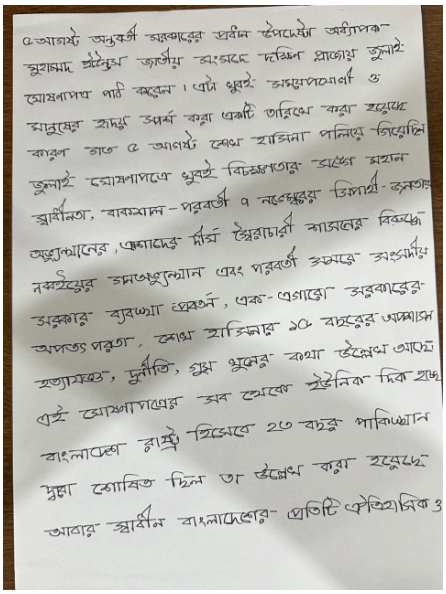

Figure 2. *
(a) Clean Handwriting

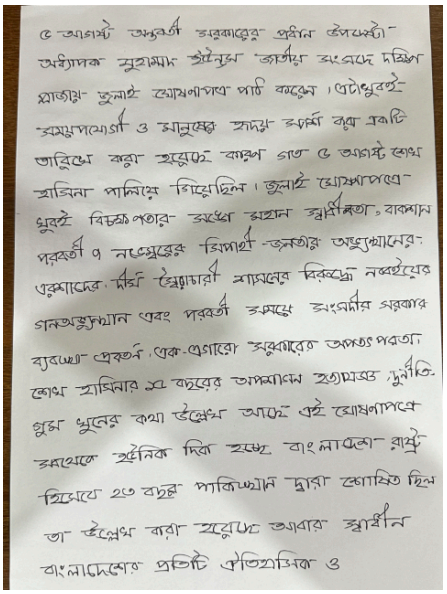

Figure 3. *
(b) Fast Writing

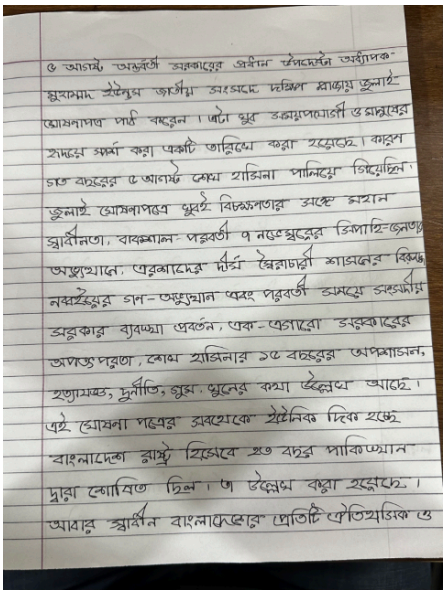

Figure 4. *
(c) Ruled Paper

Figure 5. Dataset 1 (D1): Sample handwriting styles showing variation in writing quality

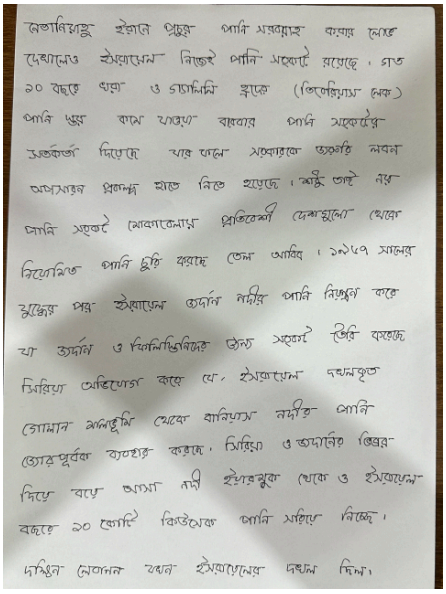

Figure 6. *
(a) Clean Handwriting

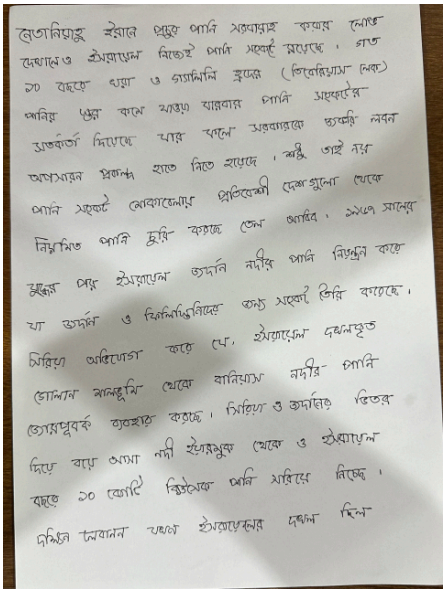

Figure 7. *
(b) Fast Writing

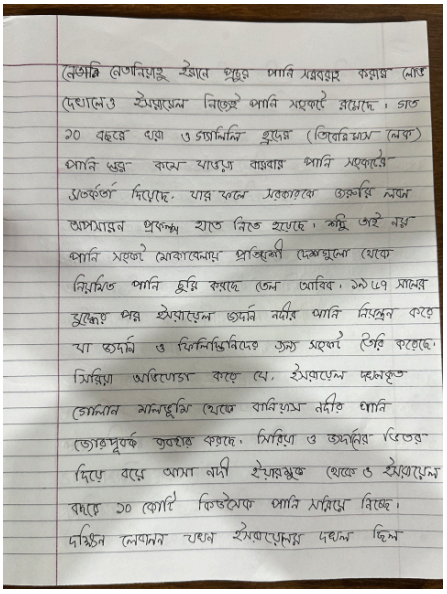

Figure 8. *
(c) Ruled Paper

Figure 9. Dataset 2 (D2): Complex cursive handwriting style with lower recognition accuracy

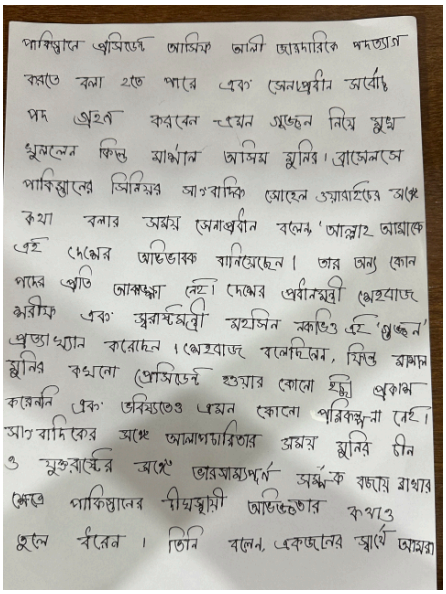

Figure 10. *
(a) Clean Handwriting

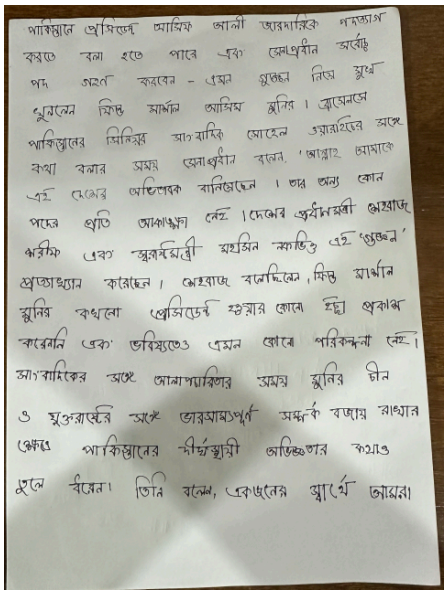

Figure 11. *
(b) Fast Writing

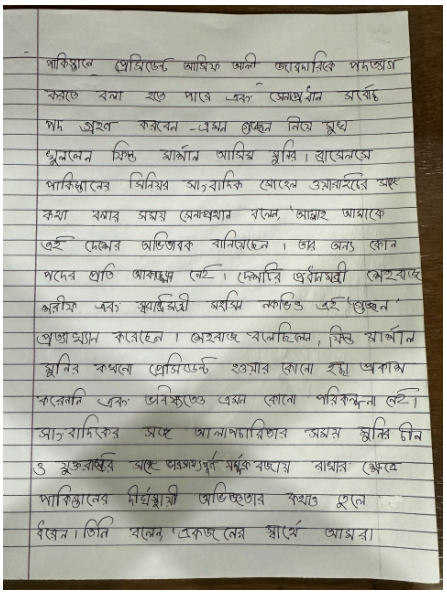

Figure 12. *
(c) Ruled Paper

Figure 13. Dataset 3 (D3): Highly stylized handwriting presenting significant recognition challenges

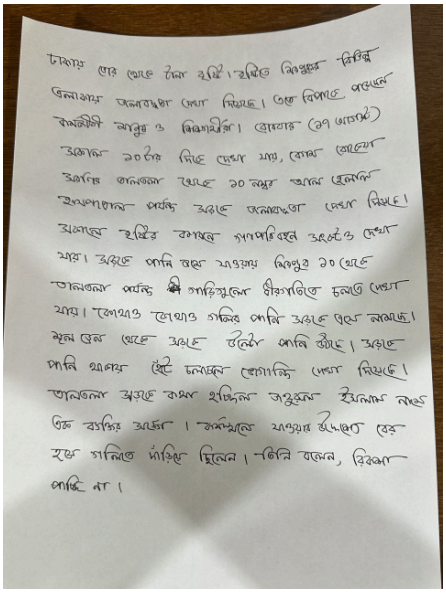

Figure 14. *
(a) Clean Handwriting

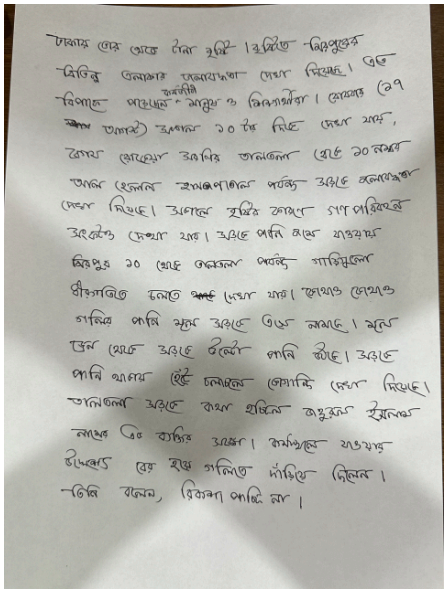

Figure 15. *
(b) Fast Writing

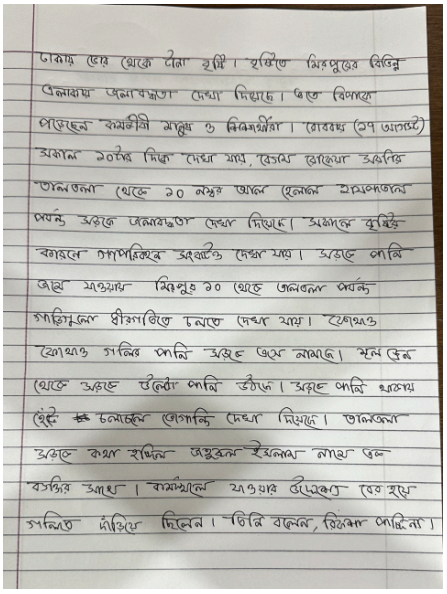

Figure 16. *
(c) Ruled Paper

Figure 17. Dataset 4 (D4): Moderate complexity handwriting with reasonable recognition performance

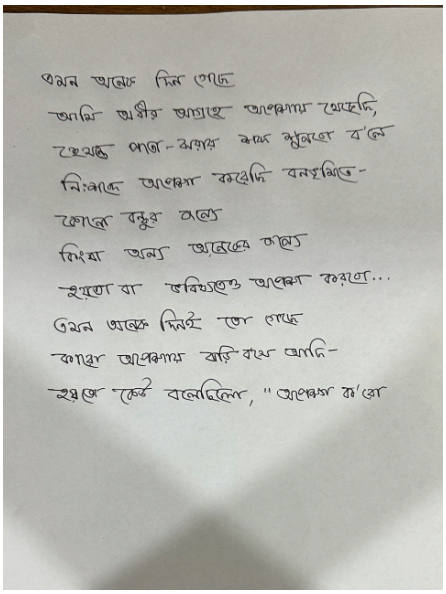

Figure 18. *
(a) Clean Handwriting

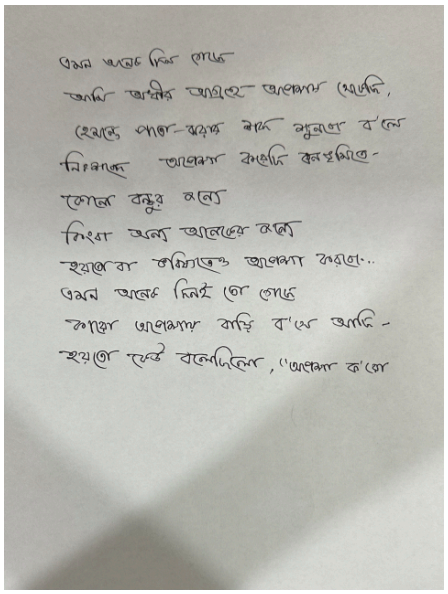

Figure 19. *
(b) Fast Writing

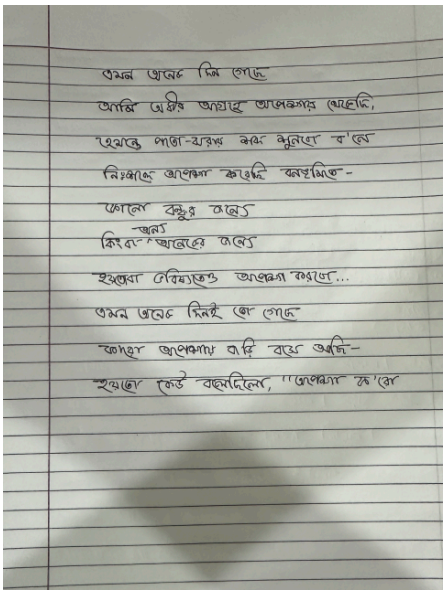

Figure 20. *
(c) Ruled Paper

Figure 21. Dataset 5 (D5): Clear handwriting achieving the highest recognition accuracy (71.2%)

Appendix E
Sample Tokenization Output

Below is a sample output from our tokenization pipeline demonstrating both word and character-level tokenization:

```
Input text length: 1106 characters
Input text preview: ৫ আগস্ট অন্তর্বর্তী সরকারের প্রধান
উপদেষ্টা অধ্যাপক মুহাম্মদ ইউনূস জাতীয় সংসদের দক্ষিণ প্লাজায় ...

SUMMARY STATISTICS

Word-level Tokenization:
  • Total tokens: 185
  • Unique tokens: 138
  • Type-token ratio: 0.746

Character-level Tokenization:
  • Total tokens: 789
  • Unique tokens: 167
  • Type-token ratio: 0.212

TOKEN TYPE DISTRIBUTION

Word-level:
  • WORD           :  159 ( 85.9%)
  • PUNCTUATION    :   20 ( 10.8%)
  • OTHER          :    6 (  3.2%)

Character-level:
  • BANGLA_DIGIT        :    9 (  1.1%)
  • WHITESPACE          :  159 ( 20.2%)
  • INDEPENDENT_VOWEL   :   46 (  5.8%)
  • CONSONANT           :  204 ( 25.9%)
  • COMPOUND            :   68 (  8.6%)
  • DEPENDENT_VOWEL     :   54 (  6.8%)
  • CONSONANT_VOWEL     :  196 ( 24.8%)
  • MODIFIER            :   17 (  2.2%)
  • OTHER               :   16 (  2.0%)
  • PUNCTUATION         :   20 (  2.5%)
```

```
MOST FREQUENT TOKENS

Word-level (Top 10):
   1. ',' → 14 times
   2. 'করা' → 7 times
   3. '।' → 6 times
   4. 'ও' → 3 times
   5. 'হয়েছে' → 3 times
   6. '-' → 3 times
   7. 'পরবর্তী' → 3 times
   8. 'উল্লেখ' → 3 times
   9. '৫' → 2 times
  10. 'আগস্ট' → 2 times

Character-level (Top 15):
   1. ' ' → 151 times
   2. 'র' → 41 times
   3. 'ক' → 21 times
   4. 'ে' → 20 times
   5. 'স' → 19 times
   6. 'ন' → 18 times
   7. 'া' → 17 times
   8. 'য' → 17 times
   9. 'ি' → 17 times
  10. 'প' → 16 times
```