

Automatic Detection and Conversion of Punjabi ASCII Based Fonts to Unicode Standard

Charanjiv Singh Saroa^{[1][a]}, Kawaljeet Singh^[1], Rajwinder Singh^[1]

^[1] Punjabi University, Patiala 147002 India

Abstract

The digitization of Punjabi text remains challenging because many legacy fonts use non-standard ASCII encodings and incompatible keyboard layouts. As a result, text created in one font may not display correctly or remain usable when transferred to another system. This paper presents an automated approach for identifying ASCII-based Punjabi fonts and converting their text into the Unicode standard. The proposed method uses n-gram frequency analysis at the word, bigram, and trigram levels to distinguish between font types. Based on the identified font, a rule-based conversion algorithm maps the encoded text to its corresponding Unicode representation. Experiments conducted on a corpus of Punjabi text covering seven ASCII-based fonts achieved 99% font detection accuracy and 99.8% conversion accuracy. For comparison, machine learning classifiers, including BayesNet, Naive Bayes, and Random Forest, implemented using the Weka toolkit, achieved detection rates ranging from approximately 34% to 57%. These results indicate that the proposed approach is highly effective for the tested fonts and dataset. By converting legacy Punjabi text into a standardized Unicode format, the system can improve cross-platform compatibility, support reliable text exchange, and help preserve existing Punjabi digital content.

Keywords: Punjabi language, Gurmukhi script, ASCII-based fonts, Unicode conversion, n-gram analysis, font detection, text digitization.

I. Introduction

Digitizing regional languages presents challenges that are less common in English text processing. English has benefited from standardized character encoding, which allows letters to retain their identity across different fonts. For example, a document written in Times New Roman can generally be opened in Arial without changing the underlying text. The Roman alphabet was small enough to fit inside a single-byte scheme, and the economic weight of English-speaking technology markets made that scheme the global default before anyone had seriously thought about other scripts.

Regional languages like Gurmukhi did not have this advantage. When developers create font for Gurmukhi script, They use different keyboard layouts of different fonts. This gave rise to a set of mutually incompatible encodings: Asees, Akhar, Anmollipi, Satluj, DrChatrik, and several others, each assigning different byte values to the same Gurmukhi characters [1]. It is the decades of Punjabi content school textbooks, newspaper archives, and government records, religious manuscripts that was produced in these encodings and has never been recoverable for computational use. This material cannot be fed into translation systems, text-to-speech engines, or Braille converters in its current form. UNICODE standardization, while the right long-term answer, does nothing on its own to unlock what already exists. The unsolved problem is detection and conversion: given a Punjabi document with no metadata about which font created it, how do you determine the encoding automatically and convert it to UNICODE without asking the user to do anything? That is harder than it sounds, because the same byte value maps to an entirely different Gurmukhi character depending on which font is in play.

1.1 The Problem of Non-Unicode Fonts

Gurmukhi is the script used to write Punjabi, one of the ten most spoken languages in the world and the official language of Punjab in India. Like most Indian scripts, Gurmukhi has a character inventory that substantially exceeds what the Roman alphabet requires independent vowels, dependent vowel signs attached to consonants, conjunct formations, nasalization markers, and characters with no English equivalent at all. This complexity meant that when Punjabi computing began in earnest during the 1980s and 1990s, no single encoding solution existed that could handle the full script reliably across different operating systems and hardware.

What filled that gap was a generation of proprietary fonts. Developers created typefaces Asees, Akhar, Anmollipi, Satluj, DrChatrik, and many others, each of which solved the encoding problem independently [1], [2]. The developers develop the fonts of Gurmukhi character on ASCII based values and map Gurmukhi glyphs

onto the ASCII byte values that the operating system already understood. There was no agreed standard for which Gurmukhi character should occupy which position, every font developer made their own choices. Asees put one character at byte 0x41. Akhar put a different character there. Anmollipi put a third. The fonts looked similar on screen they all displayed Punjabi text but the data rendering was entirely different in each case. A document written in Asees, opened on a machine where only Akhar is installed, displays as complete nonsense. Not approximately wrong completely unreadable. The phrase **ਮੈਂ ਜਾ ਰਿਹਾ ਹਾਂ** composed in one font can appear as something like "r kj pjas" in another, because what is being displayed is not the Gurmukhi characters but the Roman or symbolic glyphs that happen to sit at those same byte values in the second font[1]. There is no partial compatibility. The text either renders correctly or it does not render at all.

English does not have this problem for a reason that, ASCII assigned fixed positions to all 26 uppercase and lowercase letters before font diversity became an issue, and every English font that came after simply respected those assignments[3], [4], [5]. The character at byte 0x41 is always the letter A, in every English font, on every platform, without exception..

1.2 Unicode as the Solution

Unicode provides a universal character encoding standard designed to represent text from all writing systems consistently [4]. For Indic scripts including Gurmukhi, Unicode implementation follows recommendations from the Indian government and is based on the Indian Script Code for Information Interchange [5]. Unicode-based Gurmukhi fonts enable consistent text display across devices and platforms, support for file and folder naming in native script, web searching in native language, and reliable text sorting and data exchange. Data available in Punjabi language is mainly in ASCII-based fonts, creating an urgent need for automated conversion solutions [6].

1.3 Research Objectives

This research article addresses the need for automatic font detection and conversion from ASCII-based Gurmukhi fonts to Unicode standard. The specific objectives are:

1. To develop an automated method for identifying the specific ASCII-based Gurmukhi font used in Punjabi text.
2. To design and implement suitable algorithms for converting ASCII-encoded Punjabi text into UNICODE based font.
3. To evaluate and improve the accuracy of both font identification and text conversion.
4. To support several commonly used Gurmukhi fonts, including Asees, Akhar, Anmollipi, Satluj, DrChatrik, Gurbani Akhar, and Joy.
5. To ensure that the conversion process preserves the linguistic information and intended meaning of the original text.

II. Previous work

Research on Indic script processing has explored both font conversion and automatic font identification. The Gurmukhi Font and Unicode Converter (GFUC) supports font-to-font and font-to-Unicode conversion for fonts such as Joy, Gurbani Akhar, and Anandpur [2]. The study highlighted the risk of information loss during conversion and the need for accurate mapping tables. Another system combined language detection and font identification for Gurmukhi and Devanagari scripts using statistical methods and a ranked weighted-list approach [7]. N-gram analysis has also been widely used in text classification and language identification by examining the frequency of character or word sequences [8], [9]. In Punjabi font identification, a character-level trigram approach reportedly achieved 99.75% accuracy in word-level ASCII-to-Unicode conversion [2]. Character frequency patterns have also been studied for multilingual text processing [10]. In addition to statistical methods, deep learning approaches such as DeepFont have been used for visual font recognition, achieving over 80% top-5 accuracy on large datasets [11]. Other studies have explored neural network models and synthetic data for scene text recognition in Indian scripts [12]. However, the complex character structures and diacritical marks of Indic scripts make recognition challenging. In our experiments, BayesNet, Naive Bayes, and Random Forest, evaluated using Weka [13], achieved approximately 50% accuracy on the Punjabi font identification task. This result led us to investigate a frequency-based statistical approach for font detection and ASCII-to-Unicode conversion.

III. Proposed Methodology

Our research uses a frequency-based statistical approach to detect ASCII-based Punjabi fonts and convert them to Unicode. The methodology consists of five stages: input preprocessing, n-gram generation, frequency comparison, font detection, and Unicode conversion.

3.1 Input and Preprocessing

The system accepts Punjabi text encoded in any ASCII-based Gurmukhi font. Preprocessing involves:

- (1) Input validation.
- (2) Removal of non-alphabetic characters including punctuation and numbers.
- (3) Whitespace normalization to remove tabs, extra spaces, and newlines.
- (4) Production of a clean text string containing only alphabetic content. This step eliminates characters

that do not contribute to font identification while preserving the alphabetic patterns that carry font-specific encoding.

3.2 N-gram Generation

The cleaned text is decomposed into three types of linguistic units at different granularities:

Words: `clean_text` is split into words.

Bigrams: Sequences of two adjacent characters extracted from each word.

Trigrams: Sequences of three adjacent characters.

3.3 Frequency Analysis

For each n-gram type, the system calculates frequency distributions and computes aggregate frequency sums. The aggregate for each type is:

$$\begin{aligned}\Sigma f_w &= f_1 + f_2 + f_3 + \dots + f_n \quad (\text{word frequencies}) \\ \Sigma f_b &= f_1 + f_2 + f_3 + \dots + f_n \quad (\text{bigram frequencies}) \\ \Sigma f_t &= f_1 + f_2 + f_3 + \dots + f_n \quad (\text{trigram frequencies})\end{aligned}$$

where $f_1, f_2, \dots, f_n$ are the individual occurrence counts of each unique n-gram. These sums are compared against pre-generated frequency profiles built from large corpora of text in each target font.

3.4 Font Detection Algorithm

Font detection employs similarity matching between the frequency profile of the input text and reference profiles for all known fonts:

1. **Input:** Frequency vectors for words, bigrams, and trigrams from input text.
2. **Reference database:** Pre-computed frequency profiles for each font F_i in the known font set.
3. **Similarity scoring:** For each candidate font F_i , a similarity score S_i is computed using Euclidean distance between input and reference frequency vectors.
4. **Font selection:** The font with the highest similarity score is selected:

$$F_{\text{pete}}^{c_{te}^d} = \operatorname{argmax}_i S_i$$

3.5 Unicode Conversion Algorithm

Once the source font F_{detected} has been identified by the detection stage, the `FontConverter` function transforms every character in the input text from the ASCII encoding of that font into its canonical Unicode (Gurmukhi Unicode block, U+0A00–U+0A7F) equivalent. The function signature is:

$$\text{unicode_text} = \text{FontConverter}(\text{input_text}, F_{\text{detected}})$$

The function operates on the character stream of `input_text` in a single left-to-right pass, applying four ordered sub-procedures: (1) character mapping, (2) character-order reversal, (3) virama / conjunct normalisation, and (4) diacritical sequence canonicalisation. Each sub-procedure is described below, followed by the complete formal algorithm.

3.5.1 Inputs, Outputs, and Preconditions

Input:

- `input_text` — a string of characters encoded in the ASCII layout of font F_{detected} .
- F_{detected} — one of the supported fonts: {Asees, Akhar, Anmollipi, Satluj, Chatrik, Gurbani Akhar, Joy}.
- `MappingTable[F]` — a per-font lookup dictionary mapping each font-specific byte value to a Unicode code point (loaded at initialisation).
- `RuleSet[F]` — the ordered list of structural conversion rules specific to font F .

Output:

- `unicode_text` — a Unicode-compliant string in which every character satisfies the Unicode Standard for Gurmukhi (Unicode Standard v15.1 [5], block U+0A00–U+0A7F).

Precondition:

- $F_{\text{detected}} \rightarrow F \in \{\text{Asees, Akhar, Anmollipi, Satluj, Chatrik, Gurbani Akhar, Joy}\}$. If the font is unrecognised, the function raises an UnknownFontError.

3.5.2 Sub-Procedure 1 : Direct Character Mapping

Each ASCII byte value in *input_text* is replaced with the Unicode font stored in MappingTable[F]. Characters that have no entry in the table (punctuation, digits, Latin characters) are passed through unchanged. This step converts the raw byte stream into a sequence of preliminary Unicode font before structural corrections are applied.

```

PROCEDURE Map_Characters(input_text, F)
    mapped ← empty string
    FOR each character c IN input_text DO
        IF c IN MappingTable[F] THEN
            mapped ← mapped + MappingTable[F][c] // substitute Unicode code point
        ELSE
            mapped ← mapped + c // pass-through (Latin, digits, etc.)
        END IF
    END FOR
    RETURN mapped
END PROCEDURE

```

Algorithm 1: Direct character mapping from ASCII font encoding to Unicode font

Table 1 shows representative entries from the mapping table structure. Complete tables are maintained for all seven supported fonts; entries differ per font because each ASCII font assigns a different byte value to the same visual Gurmukhi glyph.

Glyph	Asees byte (hex)	Anmollipi byte (hex)	Unicode code point
ਅ	0x85	0xA4	U+0A05
ੴ	0x95	0xB9	U+0A15
ੴ	0xA8	0xCD	U+0A28
ੴ	0xB8	0xDD	U+0A38
ਃ (matra i)	0xFE	0xFD	U+0A3F
...	...	...	...

Table 1: Sample character mapping table entries (Asees and Anmollipi shown; byte values are representative)

3.5.3 Sub-Procedure 2 : Character Order Reversal

In many ASCII Gurmukhi fonts the vowel sign (matra) is stored **before** its base consonant in the byte stream, because the legacy font renderer displayed the pre-consonant glyph at the left of the consonant on screen. Unicode requires the opposite canonical order: **base consonant first, vowel sign second**. After Sub-Procedure 1, the preliminary Unicode string may therefore contain inverted (matra, consonant) pairs that must be swapped.

Definition — pre-base matras: The Gurmukhi matras that appear visually before the consonant and are stored in reversed order in ASCII fonts are: ਃ ('sihari, i-sound), and in some fonts also ੲ and ੳ. All other matras are post-base and do not require reordering.

```

PROCEDURE Reverse_Matra_Order(str)
    // Pre-base matras that appear before consonant in ASCII layout
    PRE_BASE_MATRAS ← {U+0A3F, U+0A47, U+0A48}
    result ← empty string
    i ← 0
    WHILE i < length(str) DO
        IF str[i] IN PRE_BASE_MATRAS AND i+1 < length(str)
           AND Is_Consonant(str[i+1]) THEN
            // Swap: place consonant before matra

```

```

        result ← result + str[i+1] + str[i]
        i ← i + 2
    ELSE
        result ← result + str[i]
        i ← i + 1
    END IF
END WHILE
RETURN result
END PROCEDURE

```

Algorithm 2: Pre-base matra reordering to conform to Unicode canonical order

Example: ASCII-stored sequence ਫਿਕ (matra-i, ka) $\rightarrow$ after reorder $\rightarrow \text{ਫਿਕ}$ (ਫਿਕ, ki). The rendered glyph is identical; the encoding is now Unicode-conformant.

3.5.4 Sub-Procedure 3 : Virama and Conjunct Normalisation

The virama (halant) character U+0A4D suppresses the inherent vowel of a consonant and, when placed between two consonants, signals a conjunct cluster. ASCII fonts encode conjunct-forming sequences differently: some store a dedicated pre-composed conjunct glyph (a single byte) rather than the three-character Unicode sequence (consonant₁ + virama + consonant₂). This sub-procedure:

- **Decomposes** any pre-composed conjunct glyph already mapped by Sub-Procedure 1 into its canonical three-element Unicode sequence.
- **Inserts** missing virama characters where the ASCII source relied on positional rendering rather than explicit halant encoding.
- **Handles** subjoined ra (ੜ) which in Unicode is encoded as virama + ra, and in some fonts requires AltGr-triggered byte sequences.

```

PROCEDURE Normalise_Virama(str, F)
    // CONJUNCT_MAP[F]: maps pre-composed glyph bytes to
    // (consonant1, U+0A4D, consonant2) triples, font-specific
    result ← empty string
    i ← 0
    WHILE i < length(str) DO
        IF str[i] IN CONJUNCT_MAP[F] THEN
            // Decompose pre-composed conjunct glyph
            (c1, virama, c2) ← CONJUNCT_MAP[F][str[i]]
            result ← result + c1 + U+0A4D + c2
            i ← i + 1
        ELSE IF str[i] Is_Consonant AND i+1 < length(str)
            AND str[i+1] Is_Consonant
            AND No_Virama_Between(str, i) THEN
            // Implicit conjunct: insert virama explicitly
            result ← result + str[i] + U+0A4D
            i ← i + 1
        ELSE IF str[i] Is_Subjoined_Ra_Glyph(F) THEN
            result ← result + U+0A4D + U+0A30 // virama + ra
            i ← i + 1
        ELSE
            result ← result + str[i]
            i ← i + 1
        END IF
    END WHILE
    RETURN result
END PROCEDURE

```

Algorithm 3: Virama insertion and conjunct decomposition

Example: Pre-composed conjunct glyph for ਕ੍ਰ (kra) stored as single byte 0xE2 in Asees $\rightarrow$ mapped to $\text{ਕ} + \text{U+0A4D} + \text{ਰ}$.

3.5.5 Sub-Procedure 4: Diacritical Mark Sequence Canonicalisation

Many ASCII fonts permit multiple character orderings for combining marks (nasalisation bindi ॐ, tippi ॐ, addak ॐ) relative to vowel signs, because the legacy renderer treats all diacritics as visually positioned independently. Unicode, however, defines a strict canonical combining order. This sub-procedure reorders any out-of-sequence combining marks to match the Unicode Canonical Decomposition order defined in the Unicode Character Database (UCD).

Canonical Gurmukhi cluster structure (Unicode order): consonant + [pre-base matra] + [post-base matra / vowel sign] + [nukta] + [bindi / tippi / addak] + [virama (if word-final)]

```

PROCEDURE Canonicalise_Diacritics(str)
  // Unicode combining class order for Gurmukhi diacritics:
  // Vowel signs (U+0A3E–U+0A42, U+0A47–U+0A48,
  // U+0A4B–U+0A4D): class 0 (spacing)
  // Nukta U+0A3C: combining class 7
  // Bindi U+0A02, Tippi U+0A70, Addak U+0A71: class 0
  CLUSTER_PATTERN ← Regex for (consonant)(diacritics+)
  result ← empty string
  FOR each cluster (base, marks[]) in Segment_Clusters(str) DO
    sorted_marks ← Sort marks[] by canonical Gurmukhi order:
      1. Pre-base matras (U+0A3F, U+0A47, U+0A48)
      2. Post-base vowel signs (U+0A3E, U+0A40–U+0A42,
        U+0A4B–U+0A4C)
      3. Nukta (U+0A3C)
      4. Bindi (U+0A02), Tippi (U+0A70), Addak (U+0A71)
      5. Virama (U+0A4D) if word-final
    result ← result + base + sorted_marks
  END FOR
  RETURN result
END PROCEDURE

```

Algorithm 4: Diacritical mark sequence canonicalisation to Unicode Canonical Decomposition order

Example: ASCII-encoded ॐ + ॐ + ॐ (ॐॐॐ, wrong order) → after canonicalisation → ॐ + ॐ + ॐ (ॐॐॐ, right standard). Both render identically in display but only the second is Unicode-standard.

3.5.6 Complete FontConverter Algorithm

The four sub-procedures are composed in a strict pipeline. Each stage receives the output of the previous stage, ensuring that reordering decisions in Sub-Procedure 2 are made on already-mapped code points, and that virama normalisation in Sub-Procedure 3 operates on correctly ordered matra-consonant pairs before final sequence canonicalisation.

ALGORITHM FontConverter(input_text, F)

INPUT:

input_text : string encoded in ASCII layout of font F
 F : detected font ∈ {Asees, Akhar, Anmollipi,
 Satluj, Chatrik, GurbaniAkhar, Joy}

OUTPUT:

unicode_text : Unicode-compliant Gurmukhi string

BEGIN

// Validate font

IF F ∉ SUPPORTED_FONTS THEN
 RAISE UnknownFontError(F)

END IF

// Stage 1: Direct character mapping

```

stage1 ← Map_Characters(input_text, F) // Alg. 1

// Stage 2: Pre-base matra reordering
stage2 ← Reverse_Matra_Order(stage1) // Alg. 2

// Stage 3: Virama and conjunct normalisation
stage3 ← Normalise_Virama(stage2, F) // Alg. 3

// Stage 4: Diacritical sequence canonicalisation
unicode_text ← Canonicalise_Diacritics(stage3) // Alg. 4

// Optional: Unicode NFC normalisation (final safety pass)
unicode_text ← NFC_Normalize(unicode_text)

RETURN unicode_text
END

```

TIME COMPLEXITY: $O(n)$ where $n = \text{length}(\text{input_text})$
SPACE COMPLEXITY: $O(n)$ for output string + $O(|\text{font}|)$ for tables

Algorithm 5: Complete FontConverter four-stage pipeline for ASCII-to-Unicode Gurmukhi conversion

3.5.7 Complexity and Correctness

Time complexity. Each of the four sub-procedures performs a single left-to-right scan over the character sequence. Dictionary lookups in MappingTable are $O(1)$ (hash map). Diacritic sort in Sub-Procedure 4 operates on at most five marks per cluster (bounded constant), giving $O(1)$ per cluster. The overall time complexity is therefore $O(n)$ where n is the length of *input_text*.

Space complexity. The output string is at most $2 \times$ the input length (worst case: every character expands to a two-code-point conjunct decomposition). Mapping tables are loaded once at initialisation and are $O(|\text{alphabet}|)$ per font. Overall space is $O(n)$ per conversion call.

Correctness guarantee. The final NFC normalisation pass (Unicode Canonical Decomposition followed by Canonical Composition, per UAX #15) guarantees that the output string is in Unicode Normal Form C regardless of any edge-case interactions between sub-procedures. This ensures unambiguous string comparison and correct rendering on all Unicode-conformant engines.

3.5.8 Worked Example: End-to-End Conversion

Table 2 traces the conversion of the Punjabi word ਗੁਣ (noon) through each stage of the pipeline, assuming the input is encoded in the Asees font.

Stage	Operation	Character sequence	Note
Input	Raw Asees bytes	0xA8 0xDC 0xF0 (3 bytes)	Asees-encoded ਗੁਣ
Stage 1	Map_Characters	U+0A28 U+0A70 U+0A42	Bytes $\rightarrow$ Unicode; order still wrong
Stage 2	Reverse_Matra_Order	U+0A28 U+0A70 U+0A42	No pre-base matra here; unchanged
Stage 3	Normalise_Virama	U+0A28 U+0A70 U+0A42	No conjuncts; unchanged
Stage 4	Canonicalise_Diacritics	U+0A28 U+0A42 U+0A70	Tippi moved after vowel sign $\rightarrow$ canonical
Output	NFC pass	ਗੁਣ (U+0A28 U+0A42 U+0A70)	Unicode-conformant ਗੁਣ

Table 2: Stage-by-stage trace of FontConverter for Punjabi word ਗੁਣ (noon) from Asees ASCII encoding to Unicode

IV. Results and Discussion

4.1 Experimental Setup

The system was evaluated on Punjabi text samples spanning seven ASCII-based Gurmukhi fonts: Asees, Akhar, Anmollipi, Satluj, Chatrik, Gurbani Akhar, and Joy. Evaluation samples included text of varying length and content type. Figure 1 shows the system interface: ASCII-encoded text is entered in the left text box,

the font is automatically detected (manual override also available), and the converted Unicode output appears in the right text box.

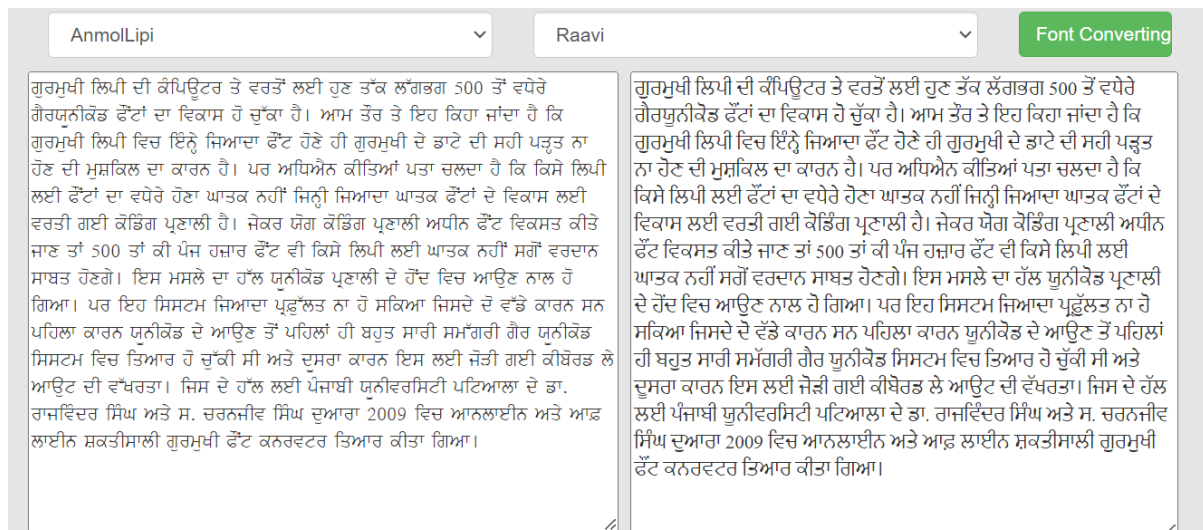


Figure 1: System interface : conversion of non-Unicode font to Unicode (left: ASCII input; right: Unicode output)

4.2 Font Detection Performance

The frequency-based n-gram approach achieved 99% accuracy in font detection. Most errors occurred when the input text was too short (fewer than 20 characters) to provide enough information for reliable identification. Other errors were observed in documents containing more than one font and in cases involving less common font variants that were not included in the reference dataset.

4.3 Unicode Conversion Accuracy

After identifying the font, the text is converted into Unicode using a rule-based approach, which achieved 99.8% accuracy. The remaining errors were mainly caused by unusual character sequences in the source text, corrupted input, and rare combinations of diacritical marks that were not covered by the existing conversion rules.

4.4 Comparative Analysis

To contextualize performance, the proposed approach was compared against three machine learning classifiers implemented in the Weka toolkit [13]. Table 3 summarises the results.

Method	Detection Accuracy	Conversion Accuracy
Proposed (N-gram Frequency)	99.0%	99.8%
Bayes.Net	~54%	N/A
Naive Bayes	~34%	N/A
Random Forest	~57%	N/A

Table 3: Comparative font detection accuracy (Weka ML methods use detection only; conversion pipeline is identical post-detection)

The proposed approach achieved 99% font detection accuracy, while all three machine learning methods achieved only around 50%. This difference suggests that the tested classifiers struggled to distinguish between the fonts. One possible reason is that ASCII-based font identification relies heavily on character encoding patterns, which may not be captured effectively by general-purpose classification methods.

4.5 Confusion Matrix Analysis

Figures 2,3,4 show the confusion matrices generated by the Weka toolkit for each baseline method.

Correctly Classified Instances	22758	54.687 %
Incorrectly Classified Instances	18857	45.313 %
Kappa statistic	0.4562	
Mean absolute error	0.1437	
Root mean squared error	0.2667	
Relative absolute error	51.726 %	
Root relative squared error	71.5651 %	
Total Number of Instances	41615	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.448	0.090	0.498	0.448	0.471	0.373	0.905	0.563	Asees
	0.436	0.134	0.394	0.436	0.414	0.290	0.859	0.537	Akhar
	0.404	0.116	0.410	0.404	0.407	0.290	0.859	0.542	Anmollipi
	0.444	0.092	0.491	0.444	0.467	0.367	0.879	0.609	Chatrik
	0.548	0.111	0.497	0.548	0.522	0.421	0.905	0.562	gurmukhilys
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	satluj
Weighted Avg.	0.547	0.091	0.548	0.547	0.547	0.457	0.901	0.636	

=== Confusion Matrix ===

	a	b	c	d	e	f	<-- classified as
3105	3	3	1	3823	0	0	a = Asees
0	3025	2283	1625	3	0	0	b = Akhar
0	2558	2805	1570	3	0	0	c = Anmollipi
3	2085	1746	3083	19	0	0	d = Chatrik
3128	2	2	0	3804	0	0	e = gurmukhilys
0	0	0	0	0	6936	0	f = satluj

Figure 2: Confusion matrix — Bayes.Net classifier (~54% accuracy)

Correctly Classified Instances	14492	34.824 %							
Incorrectly Classified Instances	27123	65.176 %							
Kappa statistic	0.2179								
Mean absolute error	0.2198								
Root mean squared error	0.3343								
Relative absolute error	79.119 %								
Root relative squared error	89.7153 %								
Total Number of Instances	41615								
=== Detailed Accuracy By Class ===									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.036	0.010	0.411	0.036	0.066	0.080	0.669	0.255	Asees
	0.000	0.000	0.000	0.000	0.000	-0.004	0.645	0.220	Akhar
	0.665	0.525	0.202	0.665	0.310	0.105	0.608	0.209	Anmollipi
	0.246	0.175	0.219	0.246	0.232	0.067	0.644	0.220	Chatrik
	0.143	0.066	0.300	0.143	0.193	0.105	0.677	0.265	gurmukhilys
	1.000	0.005	0.974	1.000	0.987	0.984	0.999	0.993	satluj
Weighted Avg.	0.348	0.130	0.351	0.348	0.298	0.223	0.707	0.360	
=== Confusion Matrix ===									
a	b	c	d	e	f	<-- classified as			
248	3	4356	1339	989	0	a = Asees			
34	0	4620	1700	517	65	b = Akhar			
89	0	4613	1712	522	0	c = Anmollipi			
34	0	4855	1706	276	65	d = Chatrik			
198	0	4360	1334	989	55	e = gurmukhilys			
0	0	0	0	0	6936	f = satluj			

Figure 3: Confusion matrix — Naive Bayes classifier (~34% accuracy)

Correctly Classified Instances	9586	57.5874 %
Incorrectly Classified Instances	7060	42.4126 %
Kappa statistic	0.4912	
Mean absolute error	0.1334	
Root mean squared error	0.2611	
Relative absolute error	48.0344 %	
Root relative squared error	70.0541 %	
Total Number of Instances	16646	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.625	0.114	0.521	0.625	0.568	0.476	0.914	0.609	Asees
	0.415	0.103	0.455	0.415	0.434	0.324	0.876	0.586	Akhar
	0.401	0.090	0.478	0.401	0.436	0.335	0.877	0.591	Anmollipi
	0.599	0.128	0.474	0.599	0.529	0.430	0.899	0.659	Chatrik
	0.424	0.074	0.530	0.424	0.472	0.384	0.914	0.606	gurmukhilys
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	satluj
Weighted Avg.	0.576	0.085	0.576	0.576	0.572	0.490	0.913	0.675	

=== Confusion Matrix ===

	a	b	c	d	e	f	<-- classified as
1719	0	0	0	1033	0	0	a = Asees
	0	1185	764	904	0	0	b = Akhar
	0	811	1135	885	0	0	c = Anmollipi
	0	606	474	1611	0	0	d = Chatrik
1583	0	0	0	0	1167	0	e = gurmukhilys
	0	0	0	0	0	2769	f = satluj

Figure 4: Confusion matrix — Random Forest classifier (~57% accuracy)

4.6 Limitations

Several limitations warrant acknowledgment. First, fonts not present in the reference corpus cannot be detected; expanding the corpus to additional or regional font variants would improve coverage. Second, documents mixing Unicode and ASCII-encoded sections require preprocessing before processing. Third, the system is designed for digitally-typed text; OCR output and handwritten text are outside its current scope.

V. Conclusion

This paper presented an automated system for detecting and converting Punjabi ASCII Gurmukhi fonts to Unicode standard using n-gram frequency analysis. The approach achieves 99% font detection accuracy and 99.8% conversion accuracy across seven font types, substantially outperforming Bayes.Net (~54%), Naive Bayes (~34%), and Random Forest (~57%) classifiers evaluated with the Weka toolkit. The results demonstrate that domain-specific frequency profiling outperforms general-purpose machine learning for this character-encoding discrimination task.

The proposed method may also be useful for other Indic scripts, such as Bengali, Tamil, and Telugu, where legacy ASCII-based fonts were used before the adoption of Unicode. Future work will focus on adding more font variants to the reference dataset, improving support for documents containing multiple fonts or encoding formats, and integrating the converter with OCR systems. This integration could help digitize printed Punjabi documents and make their text easier to search, edit, and process.

References

- [1] R. Singh and C. S. Saroa, "Multilingual conversation ASCII to Unicode in Indic script," Computer Science & Information Technology (CSIT), vol. 7, no. 3, pp. 83–94, Feb. 2017, doi: 10.5121/csit.2017.70308.
- [2] G. S. Mahi and A. Verma, "GFUC: Gurmukhi font and Unicode converter," Int. J. Comput. Appl., vol. 130, no. 3, pp. 18–28, Nov. 2015, doi: 10.5120/ijca2015906946.
- [3] "Converting Gurmukhi to Unicode," SikhiWiki. Available: https://www.sikhiwiki.org/index.php/Converting_Gurmukhi_to_Unicode.
- [4] Unicode Consortium, Unicode Technical Note #30: ISCII to Unicode Conversion Issues for Gurmukhi, 2004. [Online]. Available: <https://www.unicode.org/notes/tn30/utn30-gurmukhi.pdf>.
- [5] Unicode Consortium, The Unicode Standard, Version 15.1, 2023. [Online]. Available: <https://www.unicode.org/versions/Unicode15.1.0/>.
- [6] "Gurmukhi Unicode conversion application (GUCA)," Discover Sikhism. Available: https://www.discoverikhism.com/sikhism/gurmukhi_unicode_conversion_application.html.
- [7] G. S. Lehal, T. S. Saini, and P. K. Buttar, "Automatic bilingual legacy-fonts identification and conversion system," Research in Computing Science, vol. 86, pp. 9–23, 2014.
- [8] W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization," in Proc. SDAIR-94, 3rd Annu. Symp. Document Anal. Inf. Retrieval, 1994, pp. 161–175.
- [9] E. Stamatatos, "A survey of modern authorship attribution methods," J. Amer. Soc. Inf. Sci. Technol., vol. 60, no. 3, pp. 538–556, Mar. 2009, doi: 10.1002/asi.21001.
- [10] B. Xu, "FineFreq: A multilingual character frequency dataset from web-scale text," arXiv:2512.09701, Dec. 2025, doi: 10.48550/arXiv.2512.09701.
- [11] Z. Wang et al., "DeepFont: Identify your font from an image," in Proc. 23rd ACM Int. Conf. Multimedia, Brisbane, Australia, Oct. 2015, pp. 451–459, doi: 10.1145/2733373.2806219.

- [12] M. Mathew, A. K. Singh, and C. V. Jawahar, "Improving scene text recognition for Indian languages with transfer learning and font diversity," *J. Imag.*, vol. 8, no. 4, p. 86, Apr. 2022, doi: 10.3390/jimaging8040086.
- [13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009, doi: 10.1145/1656274.1656278.