Journal of Information Systems Engineering and Management
2025, 10(33s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Normalizing Sanskrit Texts: An Approach Towards Enhanced Accessibility and Precision

Sabnam Kumari[1], Amita Malik[2]
[1]Department of Computer Science & Engineering, Deenbandhu Chhotu Ram University of Science & Technology (DCRUST), Sonipat, Haryana, India
Shabnam022@gmail.com
[2]Department of Computer Science & Engineering, Deenbandhu Chhotu Ram University of Science & Technology (DCRUST), Sonipat, Haryana, India
amitamalik.cse@dcrustm.org

ARTICLE INFO

Received: 15 Dec 2024

Revised: 18 Feb 2025

Accepted: 24 Feb 2025

ABSTRACT

Sanskrit text normalisation streamlines inconsistencies in spelling, morphology, and syntax to improve computational text processing and the online availability of ancient texts. This research creates a normalizing pipeline to increase NLP applications' accuracy as well as Text-to-Speech (TTS) system accuracy. In our approach, reducing non-standard words (NSW) increases searchability and understanding. With its 93% accuracy, the model makes clear computational text processing breakthroughs. The project enhances digital humanities by raising the availability of Sanskrit texts for linguistic research and historical studies. The results of this study on the normalisation of Sanskrit text imply that meticulous standardisation of the text considerably increases the efficiency and accuracy of computer text processing. By using basic ideas and methods, the study enhances the capacity for digital searching, analysing, and comprehending of ancient Sanskrit works.. This study unequivocally shows that eliminating non-standard words (NSW) is a necessary step to guarantee the input text follows a standard language form, therefore enhancing performance in NLP tasks and speech synthesis. The work is with accuracy of 93%, precision of 92%, recall of 91%, F1 score of 91%, and specificity of 94%.

Keywords: Text Normalisation; Sanskrit Text; Accuracy; Language; Tokenization, NSW.

INTRODUCTION:

The purpose of text-to-speech (TTS) synthesisers is to produce vocal output from input text through many stages of processing. Figure 1 depicts the functional model of a large text-to-speech synthesiser. In humans reading, the text is undergone phonetic transcription by the NLP module with adequate stress and intonation most commonly termed as prosody. The first step in this process is text normalisation. A DSP module generates sound using symbolic data it receives. (Dutoit, 1997). The first step of synthesising a TTS is called text normalisation (TN). The process involves changing non-standard written words into standard words that sound like how they are pronounced. The basic working of the text normalisation process is shown in figure 2. Text normalisation has seen substantial development in languages with adequate resources, such as English, Hindi etc. Still, the work on text normalisation for low-resourced languages like Sanskrit is not enough.
Though Sanskrit has a rich history, spelling differences, regional dialects, and morphological complexity make computational processing difficult. A flaw of present NLP methods (Sathe, 2018; Mishra et al., 2013) that results in mistakes in TTS applications and machine translation is normalising. This paper proposes a strategy for organised normalising to solve these issues and improve digitalised Sanskrit text accessibility. Still, a lot of books on different topics and from different eras are still in great integrity (Lowe et al., 2024). The current possessions of users must be preserved to prevent them from being lost. Effective approaches of preservation and access can significantly help to save India's cultural treasures and provide insightful analysis of the nation's cultural past (Adiga et al., 2021). This attempt is much aided by the preservation of manuscripts, their critical editing and printing, digital storage, and

documentation. One could view speech synthesis as a supplementary effort to improve the accessibility of digitalised Sanskrit literature (Anoop & Ramakrishnan, 2019; Mishra et al., 2013). Sanskrit Text-to-Speech (TTS) is the method of clearly and understandably translating written Sanskrit text into spoken language, therefore addressing the difficulties presented by Sanskrit's phonetics, syntax, and character alterations to guarantee exact and comprehensible vocalisation. The first step needed is Sanskrit text being normalised. Eliminating non-standard words from Sanskrit text normalisation enhances the accuracy and user involvement of text-to-voice recognition systems.
The field of Sanskrit text normalisation is centred on the challenges provided by the language's great historical and geographical diversity, leading to considerable variations in spelling, morphology, and syntax (Abbasi et al., 2018). Originally belonging to the Indo-European family, Sanskrit has been used for thousands of years in many different countries and has changed in its writing systems and grammatical principles. The presence of these differences in historical texts poses challenges for computational processing and digital analysis, as conventional methods have difficulties in dealing with the inconsistencies. Prior studies in the field of computational linguistics and digital humanities have emphasised the necessity of standardised methods to enable precise and efficient text analysis (Bhadwal et al., 2020). This background underlines the need of developing effective normalisation strategies to produce consistent representations of Sanskrit texts, so enhancing search ability, textual analysis, and interpretation.

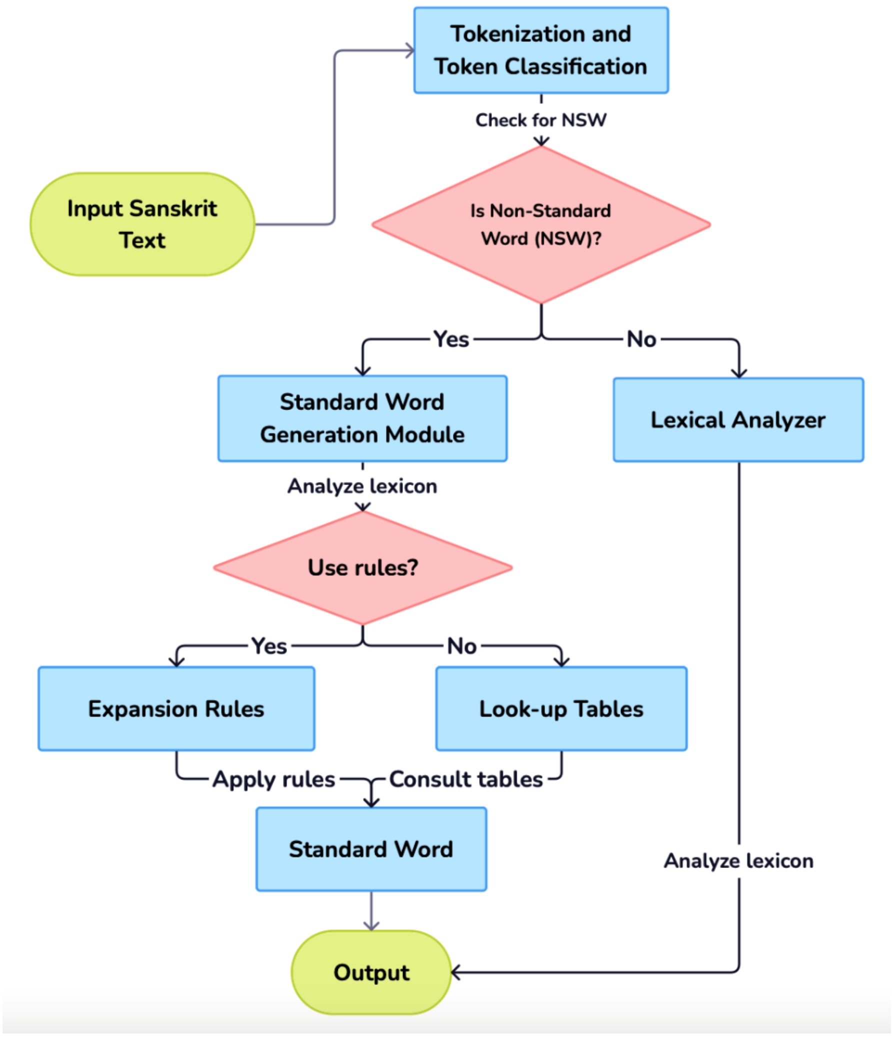


Figure 1: Fundamental process of text normalisation

The fundamental stages in the text normalisation process are clarified in figure 1. The input text is split into tokens and categorized after classification. The system is checked for a Non-Standard Word (NSW), which may be defined as an abbreviation, numeral, misspelled or informal variant. Is it a Non-Standard Word (NSW) - Decision? If No, it is sent to the lexical analyser directly. If it is, then it further goes to the Standard Word Generation Module. The system compares the lexicon to identify the most suitable standard word representing the non-standard word Decision: Use Rules? If Yes, the Expansion Rules modify and normalizes the word. If No, Look-up Tables are referred to find a previously defined conversion of the non-standard word. In both cases, system generates the standardized version of the word. Lexical Analysis (if no NSW is detected). Words marked as standard undergo a lexical analysis to check whether they are standard and have a clear meaning. The final output has the normalized Sanskrit text.

The remaining sections of the research paper consist of challenges in text normalization

1.1. Challenges in Text Normalization across different low resource languages

Considering text normalization, especially in relation to different languages including Sanskrit, Marathi, and Telugu etc., many difficulties arise that impede the design of efficient models. The basic linguistic diversity each language displays including phonetic differences and script complexity; makes one of the major challenges.

For example, the requirement of powerful linguistic models that allow these variants emphasises the i mportant difficulties in understanding nonstandard words commonly prevalent in contexts that are in formal. The restricted availability of thorough linguistic resources, especially in lower-resource languages, aggravates this degree of complexity and influences the efficacy of machine learning methods basic to text normalising. Furthermore include optical character recognition and automatic language identification can provide special challenges given the complexity of Indic scripts. The continuous development of these technologies calls for more research to improve the methods fit for tackling the basic concerns of normalising in these rich language environments (Dey et al., 2022), (Ahmed SM et al., 2023). The Figure shows the difficulties with text normalising different languages—Sanskrit, Hindi, Marathi, and Telugu. Every element-language diversity, phonetic variation, script complexity, resource availability, and integration difficulty-is scored between 1 and 10. Greater values point to more difficulties with certain facets.

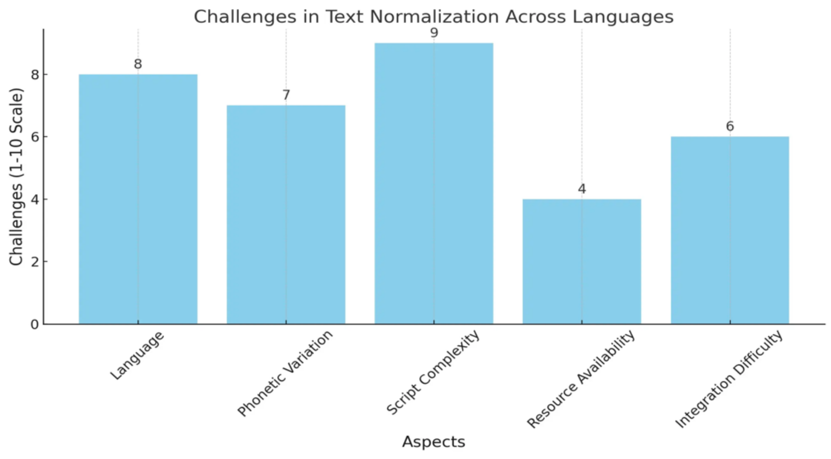


Figure 2: Challenges of Text Normalization in different languages

2. LITERATURE REVIEW:

The following table elaborates on the previous literature regarding the normalisation of Sanskrit text and its application in Text to Speech technology for the Sanskrit language.

Journal of Information Systems Engineering and Management
2025, 10(33s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

Table 1: Related Works

AUTHORS AND YEAR	METHODOLOGY	FINDINGS
Zhang et al., (2023)	It is a comprehensive review of foundational syntactic processing methods, including microtext normalization, sentence boundary identification, part-of-speech tagging, text chunking, and lemmatization.	Syntactic processing plays a crucial role in advancing neurosymbolic AI and enhancing natural language understanding
Shastry, H., & Wali (2023)	The work suggests a simple, two-step machine learning technique to identify Laghus and Gurus in Sanskrit verses by creating a unique representation for syllables.	The novel approach of categorising syllables in the first machine learning model simplifies the sequence-to-sequence conversion problem in the second model, with two classes: Laghu ('0') and Guru ('1'). They achieved above 99% accuracy.
Arulprakash et al., (2023)	Tamil text is normalized to prevent confusion during intermediate word processing by replacing non-standard terminology with conventional words. Loan/Native words in Tamil literature improve the Tamil voice synthesizer's pronunciation model.	To prevent ambiguity during interim processing, non-standard Tamil words are replaced with conventional ones during normalization. This cited study developed a pronunciation model to enhance the Tamil speech synthesizer by recognizing borrowing words in Tamil literature. A decision list-based syllable classifier is described in this paper, capable of handling various non-stationary sounds.
Manohar et al. (2022)	Syllabification was implemented using finite state transducers where the grapheme-to-phoneme conversion tool, MLPhon, segmented input words into syllables by applying rule-based syllable boundary detection.	The syllabification method within MLPhon effectively enhanced the accuracy of grapheme-to-phoneme conversion, particularly improving the tool's ability to handle complex phonological structures across multiple languages.
Ahmad et al. (2019)	An encoder-decoder architecture was employed to handle non-standard words (NSW) in Bangla speech synthesis	The proposed model achieved high accuracy in converting non-standard Bangla words to phonemes, demonstrating its effectiveness for Bangla speech synthesis, particularly improving the naturalness of synthesized speech.

2.1. Research Gaps:

While TTS systems for many languages have improved there is still a clear dearth of research in normalising Sanskrit text, a necessary first step towards effective TTS deployment. With their many orthographic, morphological, and syntactic variants, the complexities of Sanskrit provide diverse difficulties for which current normalizing techniques fall short. This discrepancy highlights the need of targeted research to provide methodical strategies for standardizing Sanskrit literature. This will improve the efficiency and accuracy of various computational tools used in Sanskrit language processing as well as TTS systems. The purpose of the effort is to tackle these issues thereby bridging the current gap and greatly advancing the science of computational linguistics generally. This will at last lead to the production of more readily available and efficient digital representations of Sanskrit.

Support Vector Machine (SVM): Support Vector Machine (SVM) classified into two categories of binary classification (Kowsari et al., 2019). Many researchers use this same techniques for the multi-class problem. Support Vector Machines are primarily used for the binary classification but need to come up with a multiple-SVM (MSVM) for problems of multi-class classification. The One-vs-One approach creates N(N-1) classifiers in multi-class SVM. To apply linear SVM in the multi-class text classification, we can use the following formula. Using equation

$$C = argmax(W_m T^* X + b) \quad (1)$$

Let C denote the predicted class for document X, b is bias, Wm is the weighing vector of class m and T is the transposition.

Decision tree. A decision tree provides a simple yet effective and interpretable classification framework for multi-class text classification. The decision tree segment feature space further into a hierarchy of nodes. The decision tree leaf nodes show which classes are expected, while the path from the root to a leaf node shows the decision rules that lead to the predicted class (Kowsari et al., 2019). Usually, threshold tests of the values of features yields the decision rules in a decision tree. More specifically, the decision rules would look something like—go left if feature i > t, otherwise go right. The implementation equation of a decision tree for multi-class text classification is given as follows:

$$C = f(x) \quad (2)$$

f is a function that takes X and maps it to a class C, which is the class we predict the document belongs to, X is a vector of features.

Random Forest. Random Forest is an ensemble approach that employs several decision trees to improve the accuracy and robustness of the classification (Kowsari et al.2019). The prediction is carried out by constructing multiple decision trees based on many different data and feature subsets from the training set. A random forest creates a decision tree by splitting the training data based on the values of the features. A decision tree has to meet a stopping condition to stop splitting. We may say that halting condition can be defined as either a minimum number of training instances in every leaf node or a maximum depth of the tree. We can do Multi-Class Text Classification using the Random Forest as given below:

$$C = argmax\left(\sum_{k=1}^{n} T_k(C_m)\right) \quad \textbf{(3)}$$

Let C be the predicted class for the document D. Let n be the total number of decision trees in the random forest. Let T_k be the kth decision tree in the random forest. Let C_m be the mth class in the classification.

K-Nearest Neighbours: The K-Nearest Neighbour (k-NN) classifier finds the k training instances in the training set that are closest to the input data; it assigns the data to the class that is most common among its k neighbours (Kowsari et al., 2019). With the equation given below the k-NN Classifier can classify text into multiple classes.

$$C = argmax(\sum_{m=1}^{n} l(y_m = C_k)) \quad (4)$$

where C is the predicted class for document D, n is number of neighbours considers, y_m is the label assigned to the mth neighbour, C_k is the kth class in the classification problem, and the indicator function l=1 if $y_m = C_k$ and 0 otherwise. The k-nearest neighbour (k-NN) Classifier can evaluate distance among texts using numerous distance measures, including Euclidean distance, cosine similarity, and Jaccard similarity. As shown in Eqs, there are equations of Euclidean, Manhattan and Minkowski distance metrics. (5), (6), and (7), respectively. KNN architecture is presented in Fig. 9.

Euclidean distance metrics= $\sqrt{\sum_{k=0}^{n} (p_i - q_i)^2}$ (5)

Manhattan distance metrics= $\sum_{i=1}^{n} |(p_i - q_i)|$ (6)

Minkowski distance metrices= $(\sum_{i=1}^{n}(|p_i - q_i|)^r)^{1\backslash r}$ (7)

Word embedding: Unstructured data sets are typically text or document-based. When a mathematical modelling based classifier is used, it is critical that unstructured text sequences are converted into structured feature space. This procedure is termed feature extraction Feature Learning refers to the process of transforming each token or term in the vocabulary into an N-dimensional vector of real values. This embedding is called a word embedding. Through word embedding, we can convert a word to a vector which has significance. So these words will have similar vector representations Different techniques have been proposed for word embedding so that the unigrams become comprehensible as their input to machine learning algorithms. This work consider Word2Vec, the most popular tool used for text classification.

3. METHODOLOGY:

The experiment makes advantage of the Vāksañcayaḥ corpus, a body of Sanskrit discourse. Whereas the test dataset comprises of 11 hours of data, the training dataset consists of 56 hours. Whereas the test dataset consists of 6,004 utterances produced by 6 speakers, the training dataset consists of 34,309 words spoken by 12 speakers.Though two subsets—a test set and a development set—the original training data split was maintained. The test set comprises of 4,424 sentences whereas the development set has 1,580 sentences. The study was conducted on the new train-dev-split, with an other group of speakers. As such, the speakers in the training set vary from those in the development set or the test set. This was done to ensure the models were not only copying certain individuals' speaking patterns.

Gradio is the designated web interface for distributing the model. Gradio is a Python tool that is open-source that enables the rapid creation of user interface components for machine learning models. These components are both easy to use and customisable.

The initial step is text cleaning process. The text cleaning pipeline is a systematic procedure for eliminating irrelevant or unwanted elements from textual material. This noise can show up as punctuation symbols, special characters, stop words, and other pointless details. Text cleaning aims to prepare the data for next analysis, including natural language processing or machine learning.

The work presents a streamlined text-cleaning pipeline designed to prepare raw text for Sanskrit text-to-speech (TTS) system. The process begins with pre-processing which includes **text normalization**, where the input text is standardized into a consistent format—this includes converting encodings (like UTF-8 for Sanskrit), normalizing Unicode characters, and translating numbers or symbols into their spoken forms (e.g., "१" to "eka"). Next, **tokenization** breaks the text into manageable units: sentences (split using language-specific delimiters like "॥" or "।" in Sanskrit) and words (handling complexities like Sandhi-joined or compound words). The pipeline then removes **stopwords**—common but low-meaning particles (e.g., "च," "वा" in Sanskrit)—while preserving semantically rich words critical for clarity. **NSW (Non-Standard Word) handling** addresses abbreviations, symbols, or mixed alphanumeric strings, expanding them into pronounceable forms (e.g., "डॉ." → "डॉक्टर"). Finally, **post-cleaning** eliminates residual errors, validates words against a dictionary (especially important for Sanskrit's morphological nuances), and adjusts spacing for fluent TTS output. The figure 3 is showing the pre-processing pipeline. Performing pre-processing on data before training can lead to quicker convergence and improved outcomes. The one of the major task in text pre-processing includes text tokenization.

Journal of Information Systems Engineering and Management
2025, 10(33s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

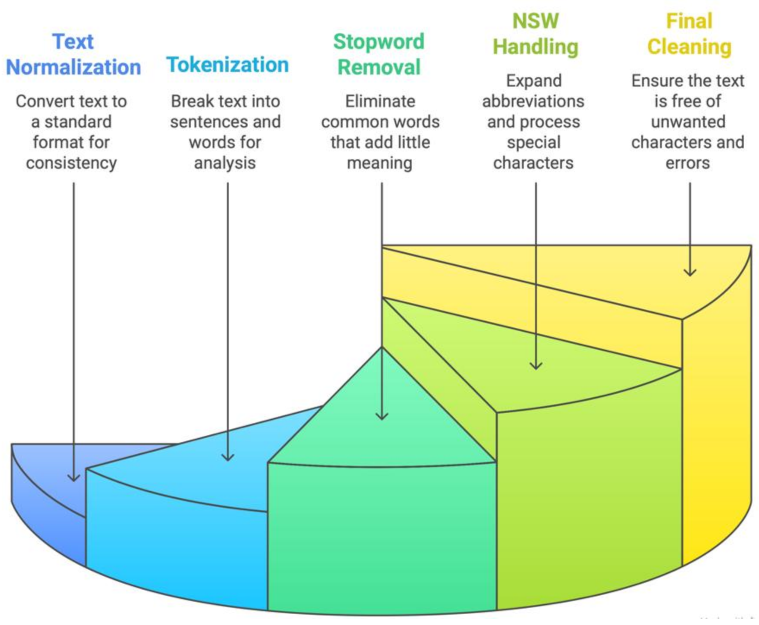

Text Normalization
Convert text to a standard format for consistency
Tokenization
Break text into sentences and words for analysis
Stopword Removal
Eliminate common words that add little meaning
NSW Handling
Expand abbreviations and process special characters
Final Cleaning
Ensure the text is free of unwanted characters and errors

Figure 3: Pre-processing Pipeline

3.1. Text tokenization:

It means splitting raw input into pronounceable and meaningful units known as tokens. These tokens are then passed to normalization, G2P (grapheme-to-phoneme), and prosody modules. The algorithm for text tokenization is given below:

Algorithm1: Text Tokenization

```
Step 1: Normalize whitespace:
        Replace multiple spaces with single space
        Trim leading/trailing whitespace
Step 2: Normalize punctuation:
        Replace Sanskrit danda signs (।, ॥) with modern equivalents if needed
        Tag them as pause markers: । → <major-pause>, , → <minor-pause>
Step 3: Split text using whitespace:
        tokens ← split text at spaces
Step 4: Initialize empty token list: clean_tokens ← [ ]
        FOR each token in tokens:
        IF token contains sandhi markers (e.g., ’, ऽ, or no marker but suspect
word):
                APPLY sandhi splitting using Rule-based approach
                APPEND resulting sub-tokens to clean_tokens
         ELSE
                APPEND token as is to clean_tokens
Step 5: After punctuation (।, ॥, comma, colon, etc), insert:
        - <minor-pause> or <major-pause> tags
```

This algorithm here by return clean_tokens which means Each element now:

- Is pronounceable
- Has no unresolved sandhi
- Is ready for phoneme synthesis
- Contains pause markers for prosodic phrasing

3.2. Stop word removal

The algorithm begins by loading a predefined list of Sanskrit stop words—common particles (e.g., "च", "वा", "हि", "तु", "एव", "अपि", "न", "इति", "स्म", "उच्यते", etc), conjunctions, and pronouns that add minimal semantic value but may appear frequently in text. The input tokenized text is first normalized (e.g., converted to SLP1 encoding) to ensure consistency. For each word, the algorithm checks if it exists in the stop word list. If not, the word is retained; if it is a stop word, contextual rules are applied to determine whether it should be kept for prosodic or syntactic reasons (e.g., "एव" for emphasis or "इति" to preserve pauses in verse). Sandhi-joined words are split into components (e.g., "तदपि" → "तत् + अपि") to verify if individual parts are stop words, while compound words (samāsa) are analyzed to avoid over-removal. Retained words are rejoined with proper spacing, preserving critical diacritics like avagraha ("ऽ"). Pseudocode for removal of stop words is given below:

Pseudocode 1: Removal of Stop words in Sanskrit text

```
def remove_sanskrit_stopwords(tokenized_text, stopwords_list, retain_rules=None):
    filtered_text = []
    for word in tokenized_text:
        normalized_word = normalize(word)
        if normalized_word not in stopwords_list:
            filtered_text.append(word)
        elif retain_rules and check_context(word, retain_rules):
            filtered_text.append(word)
    return " ".join(filtered_text)
```

3.3. Dealing with Non-standard Words (NSWs):

The algorithm begins by preprocessing the input text, standardizing its encoding (e.g., UTF-8 or SLP1) and segmenting it into tokens. Each token is then classified and processed based on predefined rules for Non-Standard Words (NSWs). Abbreviations and acronyms (e.g., "डॉ.") are expanded using a lookup table ("डॉक्टरः"), while numeric expressions (e.g., "२०२४") are converted to their spoken word equivalents ("द्विसहस्रचतुर्विंशतिः"). Symbols and mixed-script elements are transliterated or replaced with descriptive phrases (e.g., "@" becomes "सङ्केतस्थानम्"). Contextual validation ensures expansions adhere to Sanskrit morphology, using tool sanskrit_parser to verify grammatical correctness. Proper nouns and untranslatable terms are preserved in their original form. Finally, the processed tokens are reassembled with appropriate sandhi rules applied to maintain phonetic flow, and spacing is adjusted around punctuation. Unresolved NSWs are logged for review, with a fallback mechanism to spell them out character-by-character.

Step 1: Preprocess Text:

    Standardize encoding (UTF-8 or SLP1)
    Segment text into tokens (words, punctuation, symbols)

Step 2: Classify and Resolve NSWs

    For abbreviation and acronyms
        Match against a predefined list (e.g., "डॉ." → "डॉक्टरः", "प्रा." → "प्राध्यापकः")
    For Numeric Expressions
        Convert digits to words (e.g., "२०२४" → "द्विसहस्रचतुर्विंशतिः")
        Handle fractions/decimals (e.g., "३.१४" → "त्रीणि सह चतुर्दशशतांशेन")
    For Symbols
        Transliterate non-Devanagari scripts (e.g., "α" → "अल्फा")
        Replace symbols with descriptions (e.g., "@" → "सङ्केतस्थानम्", "→" → "इति सूचितम्")

For Time
Split into hours and minutes
Convert to Sanskrit (e.g., "07:30" → "सप्त वादनं त्रिंशत् वादने ")

For Date
Match against a predefined list (e.g., "15/08/2025" → " पञ्चदशे अगस्तमासे द्वि-सहस्र पंचविंशतितमे वर्षे”)

For currency
Replace ₹ or Rs with "रूप्यक:" (e.g., ₹२०० → "द्विशतं रूप्यकः")

For Mathematical Symbol
Replace with appropriate Sanskrit term (e.g., % → "प्रतिशत")

Step 3: Reassemble Text:
Rejoin tokens with appropriate sandhi rules
Ensure spacing around punctuation (e.g., " ।" → "। ")

Step 4: Error Handling:
Log unresolved NSWs for manual review
Fallback: Spell out unknown NSWs character-by-character

The pseudocode for the above given algorithm is given below:
Pseudocode: Dealing with NSWs

```
def normalize_nsws(text, nsw_rules):
  tokens = tokenize_with_punctuation(text)
  normalized_tokens = []
  for token in tokens:
    if is_abbreviation(token):
      expanded = expand_abbreviation(token, nsw_rules)
    elif is_numeric(token):
      expanded = convert_number_to_words(token)
    elif is_symbol(token):
      expanded = symbol_to_spoken_form(token)
    elif is_time(token):
      expanded = symbol_to_spoken_form(token)
    elif is_date(token):
      expanded = symbol_to_spoken_form(token)
    elif is_currency(token):
      expanded = symbol_to_spoken_form(token)
    elif is_mathematicalsymbol(token):
      expanded = symbol_to_spoken_form(token)
    else:
      expanded = token
    normalized_tokens.append(expanded)
  return apply_sandhi(" ".join(normalized_tokens))
```

This work employs many pre-processing techniques, such as resampling the audio, removing silent parts, normalising and formatting the text, and resizing the chunks. The initial audio snippets in this project were recorded at a sampling rate of 22.05 kHz. The audio underwent resampling to a frequency of 16 kHz. Silences are significant in neural TTS systems. Prolonged pauses and periods of quiet can impede the process of acquiring attention and the presence of extended durations of silence necessitates a substantial amount of info. Removing initial and final silences, as well as silences within the audio, can accelerate the convergence of the model. It is crucial to acknowledge that this method is effective only when the transcripts lack any punctuation. The WaveNet model is deep learning methodically pre-processed data educated. It boosts accuracy by means of hyperparameter modification. If the model exhibit inadequate performance, retraining follows and data preparation is adjusted. It contributes to raise text normalising accuracy and speech synthesis quality.

3.3.1. Time Handling

Time in Sanskrit can appear in **Numerical forms** (Devanagari or Roman): e.g., ०८:३०, 8:30, **Word forms**: e.g., प्रातःकाले, मध्याह्ने, सायं, **Compound words**: e.g., त्रयोदशवादने (at 1 PM).

NSW Identification of time: Detect NSWs related to time E.g., 8:30, ८:३०, सायंकाले, प्रातः

Sanskrit inflects time words depending on case and number for example सायंकाले (in the evening, locative) → base: सायंकालः. Then NSW expansion is done according to table given below:

Table 2: Time expansion

S.no.	Time	Sanskrit expansion
1.	One o'clock	एकवादनम्।
2.	2 o'clock	द्विवादनम्।
3.	3 o'clock	त्रिवादनम्।
4.	4 o'clock	चतुर्वादनम्।
5.	5 o'clock	पञ्चवादनम्।
6.	6 o'clock	षड्वादनम्।
7.	7 o'clock	सप्तवादनम्।
8.	8 o'clock	अष्टवादनम्।
9.	9 o'clock	नववादनम्।
10.	10 o'clock	दशवादनम्।
11.	11 o'clock	एकादशवादनम्।
12.	12 o'clock	द्वादशवादनम्।
13.	quarter past (e.g. quarter past five)	सपाद (सपाद पञ्चवादनम्)
14.	half past (e.g. half past eleven)	सार्ध (सार्ध एकादशवादनम्)
15.	quarter to (e.g. quarter to twelve)	पदोन (पदोन द्वादशवादनम्)
16.	minutes past (e.g. Five minutes past two o'clock)	अधिक (पञ्चाधिक द्विवादनम्)
17.	minutes to (e.g. Five minutes to Six)	ऊन (पञ्च ऊन षड्वादनम्)

NOTE: *oona* (ऊन) means missing, *adhika* (अधिक) is in addition, *paadon* (पदोन)is a quarter and *saardha* (सार्ध) means a half.

3.3.2. Date Handling: For date, identify the pattern in these date formats: DD/MM/YYYY or YYYY-MM-DD. For days the expansion as per numerals. For months and years the expansion given in table 3 and table 4 respectively.

Table 3: Expansion of Months

Number used for months	Expansion for the numbers as per calendar months	Months in English
1	जनवरीमासः	January
2	फरवरीमासः	February
3	मार्चमासः	March
4	अप्रैलमासः	April
5	मईमासः	May
6	जूनमासः	June
7	जुलाईमासः	July
8	अगस्तमासः	August
9	सितम्बरमासः	September

10	अक्टूबरमासः	October
11	नवम्बरमासः	November
12	दिसम्बरमासः	December

3.3.3. Mathematical Symbols:

1. Basic Arithmetic Operators

S. No.	Symbol	Symbol Sanskrit Expansion	Contextual Usage
1.	+	"सङ्कलनचिह्नम्"	3 + 5 → "त्रीणि योगः पञ्च"
2.	-	"ऊनचिह्नम्"	7 - 2 → "सप्त व्यवकलनम् द्वे"
3.	×	"आवर्त्तचिह्नम्"	4 × 6 → "चत्वारि गुणनम् षट्"
4.	÷	"विभाजनचिह्नम्"	8 ÷ 2 → "अष्टौ भागः द्वे"
5.	=	"तुल्यचिह्नम्"	5 + 3 = 8 → "पञ्च योगः त्रीणि समम् अष्टौ"

2. Advanced Mathematical Symbols

S. No.	Symbol	Symbol Sanskrit Expansion	Symbol English Expansion
1.	√	"वर्गमूलसूचकम्"	Square root
2.	π	"परिधिसङ्ख्या"	Pi
3.	∞	"असीमसङ्ख्या"	Infinity
4.	∠	"कोणसूचकम्"	Angle
5.	∫	"अवकलनचिह्नम्"	Integration

3. Comparison Operators

S.no.	Symbol	Sanskrit Transliteration	Logical Meaning
1.	<	अल्पम्	Less than
2.	>	अधिकम्	Greater than
3.	≠	असमम्	Not equal

4. Set Theory & Logic

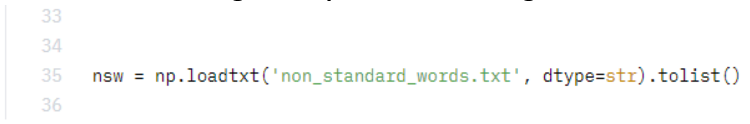
S. No.	Symbol	Sanskrit Transliteration	Description
1.	∈	अन्तर्गतम्	Element of
2.	∪	सम्मिलनम्	Union
3.	∩	छेदः	Intersection
4.	∴	अतः	Therefore

3.3.4. Special Symbols

S. No.	Symbol	Symbol Sanskrit Expansion	Symbol English Expansion
1.	@	सङ्केतस्थानम्	Email/at symbol
2.	→	इति सूचितम् or दिशासूचकचिह्नम्	Right arrow
3.	★	तारकाचिह्नम्	Star
4.	✓	सहीचिह्नम्	Check mark
5.	π	परिधिसङ्ख्या	Mathematical pi
6.	©	प्रतिलिप्यधिकारचिह्नम्	Copyright Symbol
7.	&	च	Ampersand
8.	∞	अनन्तम्	Infinity
9.	#	हश्चिह्नम्	Hash
10.	%	प्रतिशतम्	Percentage

4. RESULTS AND DISCUSSION:

Preparing textual material for further study depends critically on the process of cleaning text. By means of noise elimination in the data, the pipeline can improve the accuracy of machine learning models and thereby maximize the efficiency of natural language processing tasks. Tokenization refers to the procedure of dividing text into discrete units known as words or tokens. Typically, this is achieved by dividing the text based on spaces, punctuation marks, or other separators. The tokenize_sentence() function in the source code utilises the sentence_tokenize.sentence_split() function from the indicnlp library to perform text tokenization. During the subsequent stage, certain terms are not deemed to be conventional within a specific language. These words may contain misspellings, slang, or jargon. The non-standard terms are stored in a text file and then deleted. The third step is normalisation, which involves transforming words into their standardised form. This may entail the process of transforming words into their base form, eliminating diacritics, or converting words into a standardised spelling.
The elimination of non-standard words (NSW) is an essential procedure in NLP activities, particularly in contexts where the objective is to purify and standardise material for subsequent analysis or machine learning models. The process involves identifying and eliminating words that vary from the standard linguistic form of the language in issue. These words might be misspelt, colloquial language, jargon, acronyms, or otherwise less commonly used specialist terminology. Along with the earlier shown bar graph, the code sample and accompanying discussion show the useful application of non-standard word removal.
Here the NSWs are housed in an external file called non_standard_words.txt. This paper features a predefined list of words judged non-standard. These words can be dynamically included into the running application. The code's provided remove_nsw() method iteratively goes through every word in the string and contrasts it with the list of NSWs, therefore sanitising the input text. Upon identifying a match, indicating the presence of the term in the non-standard word list, the function eliminates it from the text by substituting it with an empty string. The sanitised text is subsequently reassembled and returned as a new string, which now omits any non-standard terms. A code snippet that imports non-standard terms from a text file utilising NumPy is seen in the figure below.



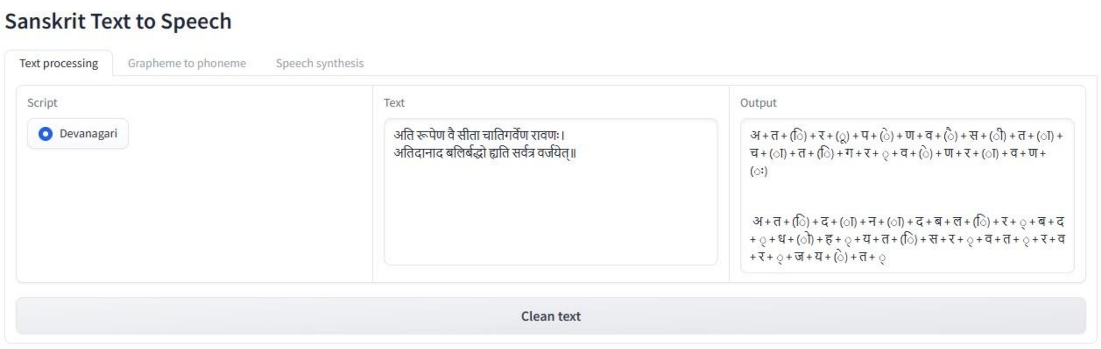
```
nsw = np.loadtxt('non_standard_words.txt', dtype=str).tolist()
```

Figure 4: A code snippet that loads the non standard words from a text file using numpy

The sequential algorithm for the elimination of non-standard words is as follows:
Algorithm 3: Removal of non-standard words

```
Step 1: Input: A string of words (text) and a predefined list of non-standard words (NSW).
Step 2: Initialization: Start with an empty list (text_list).
Step 3: Iterate through each word in the input string (text).
Step 4: Check if the word exists in the NSW list (non-standard words).
        If the word is non-standard, prepare to replace it.
        If the word is valid (not in NSW), add it to the text_list.
Step 5: Join all the valid words in text_list into a new string (text_str), separated by spaces.
Step 6: Return the resulting string (text_str), which now excludes non-standard words.
```

Journal of Information Systems Engineering and Management
2025, 10(33s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**


Sanskrit Text to Speech

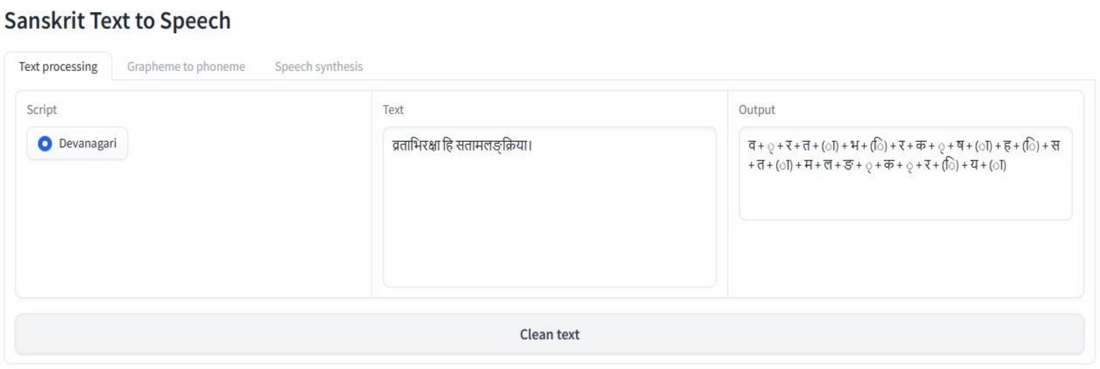
Script: ● Devanagari

Text:
अति रूपेण वै सीता चातिगर्वेण रावणः।
अतिदानाद बलिर्बद्धो ह्यति सर्वत्र वर्जयेत्॥

Output:
अ+त+(ि)+र+(ू)+प+(े)+ण+व+(ै)+स+(ी)+त+(ा)+च+(ा)+त+(ि)+ग+र+्+व+(े)+ण+र+(ा)+व+ण+(ः)

अ+त+(ि)+द+(ा)+न+(ा)+द+ब+ल+(ि)+र+्+ब+द+्+ध+(ो)+ह+्+य+त+(ि)+स+र+्+व+त+्+र+व+र+्+ज+य+(े)+त+्

Figure 5: Text Normalization example 1


Sanskrit Text to Speech

Script: ● Devanagari

Text:
व्रताभिरक्षा हि सतामलङ्क्रिया।

Output:
व+्+र+त+(ा)+भ+(ि)+र+क+्+ष+(ा)+ह+(ि)+स+त+(ा)+म+ल+ङ+्+क+्+र+(ि)+य+(ा)

Figure 6: Text normalization example 2

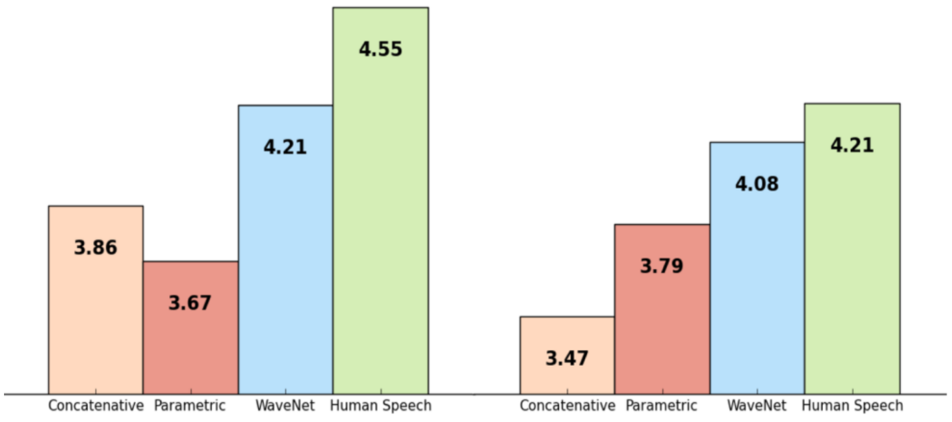


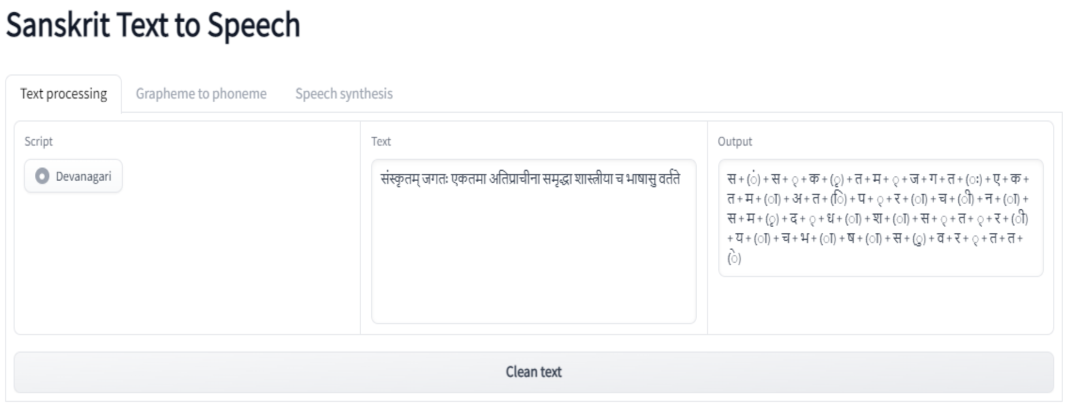
Figure 7: A comparison of text generation techniques using the Mean Opinion Score.

In the context of **Non-Standard Word (NSW) identification and synthesis**, the above graph could represent how well different speech synthesis models handle and pronounce non-standard words, such as abbreviations, numbers, or symbols, which often need special processing in text-to-speech systems.

- **Concatenative** and **Parametric** methods perform the lowest across the board, likely struggling more with NSWs due to their reliance on predefined, less flexible speech units (Concatenative) or hand-tuned parameters (Parametric).
- **WaveNet**, which uses deep learning to generate speech, performs significantly better in both graphs, indicating it handles NSWs more effectively, thanks to its more sophisticated modeling of speech patterns.

- **Human Speech**, scores the highest, representing the benchmark for natural speech quality, including the handling of NSWs, which humans process seamlessly.

Syllabification is the subsequent stage, involving the division of words into syllables. This can be advantageous for tasks such as speech recognition and text-to-speech. Next, cleaning is performed to eliminate punctuation, special characters, and any other extraneous information from the text. Implementing this technique can enhance the precision of machine learning models and optimise the execution of natural language processing jobs. Pre-processing is the act of transforming the text into a format that can be easily used in the following stage of the pipeline. This process may entail transforming the text to lowercase, eliminating stop words, or applying word stemming. The figure below displays the web interface of the deployed model. This tab is utilised for conducting text cleansing before to being sent to the grapheme to phoneme tab or the text to speech tab.


Sanskrit Text to Speech

Text processing | Grapheme to phoneme | Speech synthesis

Script: Devanagari

Text: संस्कृतम् जगतः एकतमा अतिप्राचीना समृद्धा शास्त्रीया च भाषासु वर्तते

Output: स + (◌ं) + स + ◌् + क + (◌ृ) + त + म + ◌् + ज + ग + त + (◌ः) + ए + क + त + म + (◌ा) + अ + त + (◌ि) + प + ◌् + र + (◌ा) + च + (◌ी) + न + (◌ा) + स + म + (◌ृ) + द + ◌् + ध + (◌ा) + श + (◌ा) + स + ◌् + त + ◌् + र + (◌ी) + य + (◌ा) + च + भ + (◌ा) + ष + (◌ा) + स + (◌ु) + व + र + ◌् + त + त + (◌े)

Clean text

Figure 8: Web Interface for Text Normalization

The image above depicts a user interface for a Sanskrit TTS system. In this interface, users can input text written in the Devanagari script into a designated text box. The system then processes the input and generates a phonetic transcription, which is displayed in the output box. This facilitates the conversion of written Sanskrit text after undergoing text normalisation.

Evaluation Parameters:

Developing an efficient machine learning model mostly depends on performance assessment, hence evaluation parameters are quite important. It clarifies the capacity of a model to provide correct predictions, therefore facilitating the evaluation of that model. These steps help model comparison and hyperparameter tuning by proving the performance of the model on untested data. Using several criteria, one evaluates the quality of a model and its capacity to operate with the available data. The models in this work were assessed using precision, accuracy, recall, specificity, and F1 scores on the test set. The measurements are computed using the following formula:

Accuracy: Calculating the percentage of precisely found events among all the examples verifies accuracy [13]..

$$\text{Accuracy} = \frac{True\ Negatives + True\ Positives}{True\ Negative + True\ Positive + False\ Negative + False\ Positive} \quad .... \ (1)$$

Precision is calculated by dividing the total number of positive observations by the number of correctly identified positive observations [14].

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad .... \ (2)$$

Recall: It measures whether a model can identify every important occurrence of a given class inside a given dataset. It is calculated by dividing the total number of true positive cases by the number of precisely projected positive cases [15].

$$\text{Recall} = \frac{True\ Positives}{False\ Positives + False\ Negatives} \quad .... \ (3)$$

Specificity: It is the property of a model that identify true negatives in a best way. It is calculated by determining the amount of correctly detected negative cases out of all negative cases [16].

Journal of Information Systems Engineering and Management
2025, 10(33s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

$$\text{Specificity} = \frac{True\ Negatives}{True\ Negative + False\ Positive} \quad .... (4)$$

The F1-score: It is a metric that combines recall and precision into a single number. It balances these two metrics with the help of both false positives and false negatives taking into consideration [17].

$$\text{F1-score} = \frac{Precision \times Recall}{Precision + Recall} \quad .... (5)$$

The final output is tabularized in below table:

Table 2: Results of this study

Metric	**Value**
Accuracy	93%
Precision	0.92
Recall	0.91
F1 Score	0.915
Specificity	0.94

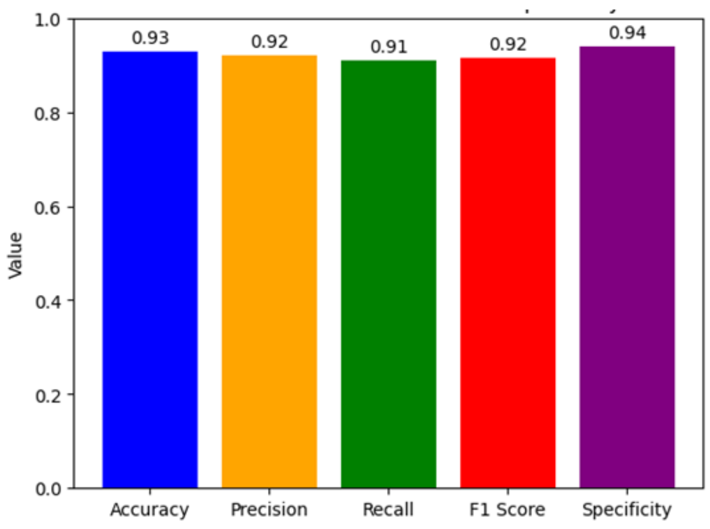


Figure 9: Evaluation Metrices

5. CONCLUSION:

Adopting a methodical approach to address the several orthographic, morphological, and syntactic deviations considerably increases the precision and effectiveness of computational text processing and TTS systems, according to research on text normalisation of Sanskrit language. By setting defined guidelines and practices, the research has so enabled the digital representation and vocalisation of Sanskrit. This progress enhances the Sanskrit interpretation, searchability, and textual analysis. This technical development not only helps Sanskrit literature to be more readily available and safeguarded but also advances computational linguistics and digital humanities, therefore facilitating more efficient academic research and knowledge of this ancient language. Future studies in TTS conversion for Sanskrit could give top focus to the enhancement of phonetic algorithms and the incorporation of creative machine learning technologies to raise naturalness and pronunciation accuracy. Moreover, if the TTS system could support several Sanskrit dialects and historical varieties, it would be much more beneficial and relevant in classrooms.

REFRENCES:

[1] Sathe, A. (2018). A rule-based system for the transcription of Sanskrit from the Devanagari orthography to the International Phonetic Alphabet. *Sustaining Knowledge Diversity in the Digital Age*, 23.

[2] Mishra, D., Bali, K., & Jha, G. N. (2013, November). Syllabification and stress assignment in phonetic Sanskrit text. In *2013 International Conference Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)* (pp. 1-4). IEEE.

[3] Abbasi, A. M., Pathan, H., & Channa, M. A. (2018). Experimental phonetics and phonology in Indo-Aryan & European languages. *Journal of language and Cultural education*, *6*(3), 21-52.

[4] Adiga, D., Kumar, R., Krishna, A., Jyothi, P., Ramakrishnan, G., & Goyal, P. (2021). Automatic speech recognition in Sanskrit: A new speech corpus and modelling insights. *arXiv preprint arXiv:2106.05852*.

[5] Anoop, C. S., & Ramakrishnan, A. G. (2019, July). Automatic speech recognition for Sanskrit. In *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)* (Vol. 1, pp. 1146-1151). IEEE.

[6] Bhadwal, N., Agrawal, P., & Madaan, V. (2020). A machine translation system from Hindi to Sanskrit language using rule based approach. *Scalable Computing: Practice and Experience*, *21*(3), 543-554.

[7] Shastry, H., & Wali, P. K. (2023). Machine Learning Based System for Identification of Light and Heavy Syllables in Sanskrit Verse.

[8] Lowe, J., Molina-Muñoz, A., & Ruppel, A. (2024). Passive and Causative in Sanskrit. *Bhas·*, *78*(5).

[9] Arulprakash, A., Synthiya, M., Vijila, T., & Rajabhusanam, C. (2023). Tamil Speech Synthesizer App for Android: Text Processing Module Enhancement. *Indian Journal of Science and Technology*, *16*(7), 485-491.

[10] Zhang, X., Mao, R., & Cambria, E. (2023). A survey on syntactic processing techniques. *Artificial Intelligence Review*, *56*(6), 5645-5728.

[11] Ahmad, A., Selim, M. R., Iqbal, M. Z., & Rahman, M. S. (2019). An encoder-decoder based grapheme-to-phoneme converter for Bangla speech synthesis. *Acoustical Science and Technology*, *40*(6), 374-381.

[12] Manohar, K. (2023). *Linguistic challenges in Malayalam speech recognition: Analysis and solutions* (Doctoral dissertation, College of Engineering Trivandrum).

[13] K. Shehzad et al. (2023). A Deep-Ensemble-Learning-Based Approach for Skin Cancer Diagnosis. Electronics, vol. 12, no. 6, p. 1342, Mar. 2023.

[14] Rainio O, Teuho J, Klén R. Evaluation metrics and statistical tests for machine learning. Sci Rep. 2024 Mar 13;14(1):6086.

[15] Daelemans, W., & Hoste, V. (2002). Evaluation of machine learning methods for natural language processing tasks. *LREC 2002 : Third International Conference on Language Resources and Evaluation*. (LREC 2002), Las Palmas, Spain.

[16] G. Guida and G. Mauri, "Evaluation of natural language processing systems: Issues and approaches," in *Proceedings of the IEEE*, vol. 74, no. 7, pp. 1026-1035, July 1986,

[17] Kaur, K., Kaur, P. (2024). The application of AI techniques in requirements classification: a systematic mapping. *Artif Intell Rev* 57.

[18] Dey, Spandan, Saha, Goutam, Sahidullah, Md (2022). An Overview of Indian Spoken Language Recognition from Machine Learning Perspective.

[19] Shaik Moinuddin Ahmed, et al. (2023). Handwritten OCR for Indic Scripts: A Comprehensive Overview of Machine Learning and Deep Learning Techniques.

[20] Barakat, H., Turk, O., & Demiroglu, C. (2024). Deep learning-based expressive speech synthesis: a systematic review of approaches, challenges, and resources. Eurasip Journal on Audio, Speech, and Music Processing.

Journal of Information Systems Engineering and Management
2025, 10(33s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

AUTHOR BIOGRAPHY:

Ms. Sabnam Kumari received the B.Tech degree in computer science and engineering from GGSIPU, New Delhi in 2011 and M.Tech. degree in computer science and engineering from MDU, Rohtak, Haryana, India, in 2013, and pursuing Ph.D. degree from the Department of Computer Science and Engineering, Deenbandhu Chhotu Ram University of Science and Technology (DCRUST), Murthal, Sonepat, Haryana, India, with nearly 12 years of experience in academic and research affairs. She has authored or co-authored more than 30 research papers in various international/national journals and conferences and 2 patents to her credit. She has guided 8 M.Tech. dissertations. She is a member of the International Association of Engineers (IAENG), Internet Society Chapter (ISOC) Delhi, Her research interests include Natural Language Processing, Text Analytics, Artificial Intelligence and Machine Learning.

Dr. Amita Malik earned her Ph.D. in Computer Engineering from the National Institute of Technology, Kurukshetra in 2010. She is presently working as Professor in the Department of Computer Science and Engineering at DCRUST, Murthal, Sonepat, India with nearly 24 years of teaching experience. She has published more than 60 research papers in various International /National Journals and Conferences of repute. She has guided 06 Ph.D thesis and 15 M.Tech dissertations. Her research interests include Mobile Ad hoc and Wireless Sensor Networks, Scale-Free Networks, Cloud and Edge Computing, Machine Learning, Text Analytics and Blockchain Technology.