EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

Spellbound Kannada: Harnessing Conditional Generative Adversarial Networks for Transformative Word Suggestion Systems in Kannada Language Processing

Prathibha R J[1], Anupama N[2*], Sudarshan A S[3]

[1,3]Department of Information Science and Engineering, JSS Science and Technology University, Mysore, Karnataka, India

[2]Department of Electronics and Communication Engineering, Vidyavardhaka College of Engineering, Mysuru, Karnataka, India

Abstract

INTRODUCTION: The advancement of a word suggestion system model is driven by the need to enhance user interaction and efficiency in digital communication. Hence, the word suggestion system helps minimize typographical errors and spelling mistakes. Therefore, various traditional methods are used to suggest words to sentences; however, these traditional models are extremely time consuming, prone to errors and tedious. METHODS: Owing to these factors, the present paper focuses on developing a Kannada word suggestion system using cGAN (Conditional Generative Adversarial Networks), as this system is designed to significantly enhance user interaction by offering predictive text suggestions in the Kannada language. RESULTS: The training dataset, which resides on AWS S3, comprises a comprehensive collection of Kannada texts utilized for both training and validation purposes. Furthermore, the implementation of the model leverages the TensorFlow and keras framework, specifically employing long short-term memory (LSTM) networks for effective sequence prediction and generation. LSTMs are particularly advantageous in NLP processing because they can capture long-term dependencies within sequential data. To facilitate user interaction, a web-based interface has been developed using Flask, enabling users to input initial characters and receive dynamically generated Kannada word suggestions. CONCLUSION: This paper not only delves into the application of cGANs within the realm of NLP but also illustrates practical deployment strategies utilizing cloud services and modern web technologies. Overall, the proposed approaches demonstrate the potential of the cGAN in enhancing the user experience through intelligent text prediction systems tailored for the Kannada language.

Keywords: Natural Language Processing, Word Suggestion System, Conditional Generative Adversarial Network, Kannada, LSTM

Received on 11 November 2024, accepted on 20 February 2025, published on 01 April 2025

doi: 10.4108/eetiot.7792

1. Introduction

Natural language is the way in which people communicate with each other about their knowledge [1], feelings, and emotions [2, 3]. Different native languages with unique alphabets, signs, and grammar can be found all over the world. India is the country that has the most ancient and morphologically diverse regional language variants [4]. Compared with other natural languages, computers can process data represented in English

*Corresponding author. Email: anupama.n@vvce.ac.in

via conventional ASCII codes rather easily [5]. However, developing a machine's ability to comprehend other natural languages is difficult and requires a variety of approaches [6]. NLP provides a wealth of techniques for understanding and manipulating languages such as Kannada [7]. Hence, different AI methods are used for supervised DL approaches for correcting spelling errors in multilingual texts, specifically those in the Persian and Arabic languages. This method employs a CRF-based RNN model [8] for identifying and rectifying spelling mistakes. The research generated a dataset containing 220,000 sentences with intentionally introduced errors, emphasizing the design of effective contextual features. Similarly, the author concentrated on creating a model for detecting misspelled Assamese words [9]. This highlights how misspellings can negatively affect readability and understanding of the language. By incorporating contextual information from phrases, the model uses BiLSTM and LSTM to enhance the detection of spelling errors. Experiments were conducted on a dataset of 2,000 sentences and revealed that the BiLSTM model outperformed the LSTM model.

While NLP excels at sentiment analysis and speech recognition [10], predicting full Kannada words via a single alphabet is a difficult issue. Kannada characters can have different pronunciations depending on context [11, 12]. A single consonant might have many vowel sounds connected with it [13]. There are alternatives to full-word prediction. The system may generate a list of possible Kannada letters that follow the supplied alphabet, making it easier for users to type in Kannada. Furthermore, if a Kannada dictionary is provided, it may suggest whole words on the basis of the single alphabet entered, even though this would not be considered an NLP prediction. In essence, predicting complete Kannada words with high accuracy on the basis of a single alphabet is currently not possible via NLP algorithms. The proposed method uses the cGAN to train the model and envisages the Kannada words. In a GAN, the generator and discriminator engage in a viable learning process. The discriminator role is to differentiate between real data and the synthetic data produced by the generator. Moreover, the generator aims to create artificial data such as Kannada words or phrases that loosely resemble authentic samples from a training dataset. Through this adversarial training, the generator gradually improves its ability to produce more realistic outputs. Additionally, cGANs enhance the basic framework by incorporating extra information, such as class labels and context vectors, to better guide the generation process. In NLP tasks, such as spelling suggestion and correction, cGANs can utilize contextual information derived from surrounding text. This allows them to produce realistic corrections for misspelled words to sentences. By adopting this conditional strategy, the relevance and accuracy of the generated outputs are significantly improved, making cGANs particularly effective for applications that demand context-sensitive language generation and error correction.

1.1 Research Objectives

The major objectives of the research paper are as follows:

- To employ the cGAN, the generator model learns to produce plausible Kannada words on the basis of initial character inputs, whereas the discriminator model is prepared to recognize genuine and created words.
- To leverage long short-term memory networks for sequence prediction and generation.
- To use Flask to allow users to input initial characters and receive dynamically generated Kannada word suggestions.

1.2 Paper Organisation

The paper is structured as follows. Section 2 addresses existing works performed by various researchers. Section 3 focuses on employing and implementing the proposed work for word suggestion systems. Section 4 focuses on the results obtained via the proposed model, and finally, Section 5 summarizes the entire research work along with future directions.

2. Literature Review

Various methodologies adopted by existing works for spelling error detection are reviewed in the following section.

Spelling errors often arise from typing mistakes in text documents. Currently available spell checkers are quite basic and are primarily developed via conventional approaches such as role-based methods and statistical approaches. Therefore, HINDIA [14] was used to address spelling errors in the Hindi language. To do so, the study has two distinct phases: the 1st stage involves detecting incorrect works in the input text, while the 2nd stage replaces these erroneous words with the most likely correct alternatives. The HINDIA model is built by employing an attention-based encoder-decoder BiRNN (bidirection Recurrent Neural Network), which incorporates LSTM cells. Therefore, various enhancements have been made to BiRNN, and the network has been fine-tuned to address spelling errors efficiently in the Hindi language. Moreover,

the BiRNN model was compared with the Malayalam language spell checker system; however, the findings revealed that the better HINDIA model delivered better performance than existing spell checkers did. Similarly, an approach for correcting various context-sensitive spelling errors via the DL approach is emphasized in the suggested work, where a correction model termed the autoregressive (AR) language technique is used for prediction in the next work via a unidirectional context, and an autoencoder (AE) language model is used for restoring words via bidirectional context information.

LSTM- and BiLSTM-based algorithms [9] have been used in existing works for precisely detecting misspelled words by assessing the perspective of each word in a sentence. Here, more than 2677 Assamese sentences are used, which consists of both correctly spelled and incorrectly spelled words. Similarly, LSTM [15] has been used for a potential suggestion system, in which the model has used error detection techniques to improve the performance and enhance the accuracy of the spell checker technique for the Punjabi language. Similarly, the BiLSTM model has been used for correcting spelling mistakes by means of a transformed input and stochastic error injection technique for Arabic text. Here, the model uses 2 BiLSTM [16] layers and dropout regularization and then trains with an error injection rate. In addition, future work will focus on handling Arabic spelling correction along with letter diacritization in one problem space. Similarly, two different techniques [17] were used to correct spelling errors in Persian text, where the first method relied on using the rule technique and the other method relied on employing the DNN-based LSTM approach. A total of 112 rules were established to address spelling mistakes resulting from phonetic similarities and typographical errors, offering recommended works for those not recognized in the dataset. For evaluation purposes, a set of 2500 word sentences containing spelling errors was used, and the word with the smallest Levenshtein distance was selected as the correct option in this approach. In the DL framework, an LSTM unit was implemented, incorporating word embeddings in the input layer along with the capsule layer, convolution and, eventually, max pooling. An artificial dataset comprising 1.2 million entries was created to terrain the DNN; out of this, 800,000 sentences were used for training and 200,000 for testing, while the remaining sentences were reserved for evaluation purposes. The findings of this study demonstrated that the DL-based LSTM model has better outcomes than the traditional rule-based approach.

Language is considered the most fundamental means of communication, in which grammar plays an important role in the excellence of a language. Therefore, [18] used a DL-based LSTM model for the grammar validation mechanism, as this model addresses these problems and proposes the development of a grammar-checking mechanism specifically for the Kannada language. In this context, the model aims to accomplish the required categorization by leveraging context-based data retention via Word2VEc, along with the use of TensorFlow and Keras libraries. Spelling mistakes can occur in text for various reasons, such as typing errors or a lack of knowledge about correct phonemes. In languages that feature intricate works, such as Sandhi, where multiple morphemes are combined according to specific rules, spell checking can become quite challenging. In these cases, a spell checker equipped with a Sandhi splitter [19] is particularly beneficial, as it can notify users of errors and offer helpful suggestions for correction. Therefore, this study uses a spell checker with a Sandhi splitter for improved results for better suggestions. This mechanism generates a comprehensive list of suggestions for considering both the meaning and context of the word in question. Likewise, error generation algorithm [20] has used in the study for fine tuning the task of spell checking for Persian language. Moreover, Burmese language has explored in the study for spelling training using Symmetric Delete Spelling Correction Algorithm. The result investigated that the performance of each error type and studied the importance of the dictionary depending on the average term length and maximum edit distance for Burmese spell checker based on SymSpell. Therefore, Table-1 showcases the summary of the existing models.

Table 1 Directory tree structure of the Mask RCNN system

Ref	Method	Language	Findings
[21]	LSTM and BiLSTM	Assamese	The findings of the work has stated that LSTM model attained accuracy of 92% and BiLSTM of 93%.
[22]	An LSTM model that encodes input word at character level, that also	Indonesian	The accuracy obtained by the Char-LSTM for Indonesian language is 90%. However, it was

	uses word and POS tag contexts as features.		identified that, for training dataset accuracy obtained is 83.76%.
[23]	The Study has used SpellBERT	Chinese	SpellBERT can be directly used without confusion set in the fine-tuning and inference phase, which is more convenient to use and easier to deal with the errors uncovered by the existing confusion sets
[24]	Hybrid Model which consisted of N-gram, LCS, Shapex and Soundex ranking algorithm	Urdu	Outcome of the model has proved that, F1 score gained is 88.29%.
[25]	LSTM with word embedding layer	Persian	The accuracy obtained by the LSTM network is 87%.
[26]	LSTM along with rule based approach and minimum edit distance	Punjabi	The outcome of the work revealed that LSTM predicted best possible suggestions by computing strong relationship of word with previous sequence and makes possible words available for correction
[27]	Sequence to sequence (Seq2Seq) model with attention mechanism is used to develop spelling correction for Azerbaijani	Azerbaijani	It is exposed that, when the distance is 0 and 1, the accuracy value obtained is 75% and 90%.

2.1 Gaps Identified

A review of existing works revealed that very few studies have covered the use of the Kannada language for word suggestions and spell-check approaches because Kannada is a unique semantic and syntactic properties, which can make it challenging to develop NLP tools compared to more widely studied languages. Additionally, its complex grammar and morphology require tailored approaches that are often not available or are less developed. Therefore, the proposed work focuses on the word suggestion system using the cGAN model for the Kannada language, as Kannada is considered one of the important languages in India and has complex scripts and grammar. Moreover, with the increasing use of Kannada in various digital platforms, generating a word suggestion and spelling correction model for enhancing the user experience and communication quality is important. Hence, the subsequent section addresses the implementation of the proposed model in detail.

3. Research Methodology

The proposed work for the Kannada word suggestion system is discussed in the subsequent

section. Thus, Figure 1 shows the overall mechanism of the model.

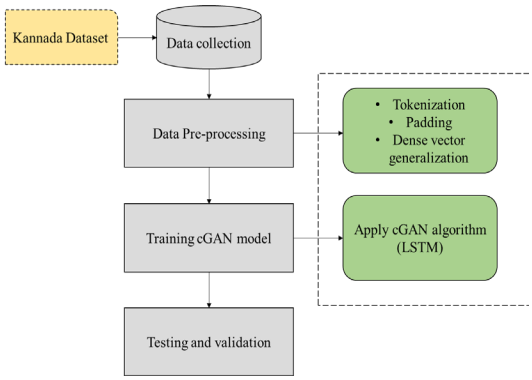

Figure 1. Overall Flow

Figure 1 depicts the process involved in the proposed work. When the model is carried out by loading the dataset, the data presented in the dataset are preprocessed via various techniques. After the preprocessing technique, the cGAN model is trained, and the LSTM algorithm is further applied. The model is subsequently tested and validated via different techniques.

3.1 Data Collection

The raw textual data for the proposed work encompass a comprehensive dataset featuring Kannada words and phrases. This dataset consists of approximately 10,000 sentences in Kannada, serving as a foundational resource for training the model for effectively predicting and generating text in the Kannada language.

The data are organized as a collection of text files, each containing sentences or paragraphs written in Kannada. These text files undergo a tokenization process, wherein the content is segmented into individual words or sequences of words. Hence, the subsequent section focuses on employing various preprocessing techniques to enhance the performance of the model by using different approaches.

3.2 Preprocessing Approaches

3.2.1 Tokenization

Tokenization is a crucial preprocessing technique that aids in transforming a sequence of text into smaller, manageable units known as tokens. These tokens can represent various linguistic elements, including words, phrases or even individual characters. The tokenization process typically involves segmenting the text into discrete components, such as individual words or subword units. This method not only facilitates the model's comprehension of the structure and segmentation of the Kannada language but also enhances its ability to process and generate text effectively. By converting words into unique integer representations, the model can effectively manage and analyse linguistic data, paving the way for improved performance in NLP tasks. Hence, Figure 2 depicts the Kannada words tokenized into unique integers.

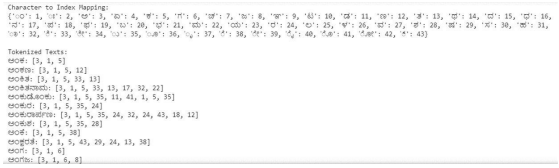

Figure 2. Tokenzied Text

As illustrated in Figure 2, the Kannada words have been tokenized and mapped to unique integers, and the process involved is discussed.

- Creation of vocabulary: The tokenizer systematically analyses the entire text corpus to construct a comprehensive vocabulary, which comprises all unique tokens present within the dataset. Each distinct token in this vocabulary is consequently allocated a unique integer identifier. This process is instrumental in standardizing textual data into a numerical format that can be efficiently interpreted by the model.
- Word-to-integer mapping: Once the vocabulary is established, each term within the textual data is assigned an associated identifier through a systematic mapping process. This entails creating a dictionary structure where each unique word is linked to a specific integer value.
- Tokenization: The text is then converted into sequences of integer identifiers.

Through the conversion of textual words into numerical representations, tokenization enables the model to effectively process and learn from the underlying text data. In the context of the proposed methodology utilizing cGANs, tokenization plays a crucial role by providing both the generator and discriminator with a uniform numerical format of the Kannada text. This standardization enhances the training process and improves predictive accuracy, as it allows the models to operate on a consistent and structural input format, thereby facilitating more effective learning and generation of text.

3.2.2 Input padding sequence

Once tokenization is completed, the text data are transformed into numerical sequences. However, these sequences can vary in length, and NNs typically need input data to be of uniform length. The pad sequence function from Keras is typically used for padding. This function takes a list of sequences and pads them to the desired length. The input sequences are split into predictors (all but the last word in each sequence) and labels (the last word), and the labels are then converted into a one-hot encoded format for training a neural network to predict the next word in the Kannada text sequences. After padding and splitting the input sequence into predictors (sequences excluding the last word) and labels (the last word), the uniform-length input sequences are prepared and fed into the neural network for training. Thus, Figure 3 shows the input-padding sequences.

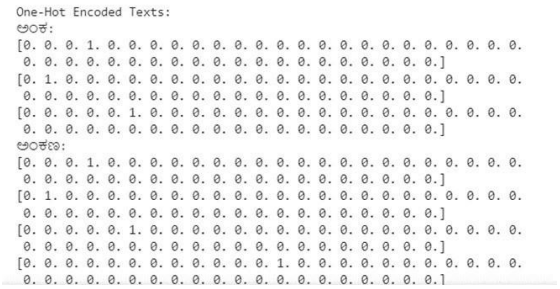

Figure 3. Input and padding sequences

3.3 Conditional generative adversarial network

A GAN encompasses 2 major parts, which include a generator and a discriminator. The generator produces the data that are likely, which seem to be from the training data; however, the discriminator attempts to differentiate the data generated by the generator as real or fake data. The discriminator castigates the generator to create improbable outcomes. The generative model aids in capturing the distribution of data and is trained in such a way that it generates a new sample, which attempts to exploit the probability of the discriminator to make a mistake. However, the discriminator performs well in classifying the sample it receives from the training data and further minimizes the accuracy of the GAN. Therefore, the GAN network is expressed as min–max, in which the discriminator focuses on minimizing the rewards and the generator helps in maximizing the discriminator loss.

Although the GAN has different advantages, there are major disadvantages of employing a GAN, such as unstable performance and sensitivity to hyperparameters, making it quite daunting to detect the right settings for optimal performance. This instability can lead to challenges in word sentence suggestion systems. Therefore, the proposed research opts for the cGAN to suggest the right words, as the cGAN can generate higher-quality samples than can the traditional GAN by leveraging the additional conditioning information. In addition, cGANs can adapt various types of input data and conditions, making them extremely versatile for different spelling corrections and word suggestion scenarios, irrespective of language. Furthermore, cGAN models offer better stability during training by leveraging paired distributions, reducing the volatility seen in standard GAN models. Owing to these factors, the proposed research work opts for the cGAN model.

A cGAN is an extension of the traditional GAN and is designed for enhancing the control over the generated outputs by incorporating additional conditioning information. This allows for more targeted data generation depending on the specific attributes or labels. The architecture of cGANs mirrors that of conventional GANs but introduces a crucial modification: both the generator and discriminator networks, which receive additional input in the form of conditioning information.

3.3.1 Generator Network

The generator takes 2 inputs, which include arbitrary noise sampled from a predefined distribution and conditioning information. It processes these inputs to produce synthetic data samples that not only look like real data but also conform to the characteristics specified by conditioning information y. This allows for the generation of data that meet the particular criteria on the basis of the provided label.

3.3.2 Discriminator network

The discriminator receives both real data samples and produced samples a from the generator, along with the same conditioning information y. The functionality of the discriminator is to evaluate whether the input data generated are real data or generated data while considering the conditioning information. This dual input allows it to assess both the authenticity of the data and its consistency with the specified conditions.

3.3.3 Conditioning

The incorporation of additional input into both the generator and discriminator ensures that the generated outputs are not only coherent but also aligned with specific characteristics and requirements described by the conditioning parameters. This mechanism enhances the model's ability to yield targeted and contextually relevant

outcomes, thereby enhancing the overall quality and applicability of the generated data. Hence, the overall working mechanism of the cGAN is shown in Figure 4.

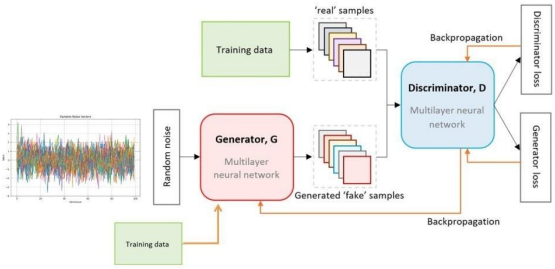

Figure 4. cGAN mechanism

Figure 4 shows the process involved in the cGAN. Here, the cGAN applied to the Kannada character suggestion uses conditional inputs to generate sequences of Kannada characters that meet specific criteria. During the training phase, the generator component of the cGAN receives two types of inputs: a random noise vector sampled from a latent space and conditional information that specifies the desired attributes or characteristics of the Kannada character sequences. The main objective of the generator is to generate character sequences that are not only realistic but also consistent with the given conditions, such as starting with a particular letter or adhering to a specific pattern. This approach enhances the relevance and accuracy of the generated character suggestions, making them more useful for application in text processing and input correction. On the other hand, the discriminator in the cGAN is responsible for distinguishing between real Kannada character sequences and those generated by generators. It also receives the same conditional input as the generator does, enabling it to assess the authenticity and adherence of generated sequences to the specified conditions. Through adversarial training, where the generator plans to trick the discriminator and discriminator actions to accurately order genuine from produced successions, the cGAN improved iteratively. Similarly, the figure shows the internal modules of the generator and discriminator.

3.4 Long short-term memory

To overcome the weaknesses of ordinary recurrent neural networks in learning and recalling long-haul conditions in consecutive information, LSTM networks were constructed. Unlike a conventional RNN, which tussles with vanishing gradient delinquency, an LSTM utilizes a distinctive style that includes a memory cell and a gating mechanism. Each LSTM typically consists of a cell state, which acts as a memory that carries information across time steps and 3 gates, such as the forget gate, input gate and output gate.

- **Forget Gate:** An LSTM unit's forget gate is in charge of deciding how much of the prior cell state to keep or discard when fresh data enter the network. Concatenation of the current input with the preceding hidden state is what it accepts as input. The information has $sigmoid$ enactment capability, which yields a numeral anywhere in the range of 0 to 1 for each number in the cell state. A total of 0 methods fail to remember these data, whereas a total of 1 method results in these data.
- **Input Gate**: An LSTM unit's input gate regulates how much different detail is incorporated into the cell state. It consists of two essential components: an $tanh$ layer and an $sigmoid$ layer, known as the input gate layer. The $tanh$ layer generates a vector of potential new values that can be incorporated into the state, whereas the sigmoid layer regulates which of these values needs to be altered.
- **Output Gate:** An LSTM unit's result entryway determines the secret expression that is shipped off the result of the model and the resultant time step. In view of the earlier covered state and the ongoing information, it determines what segments of the cell state ought to be yielded. The outcome entryway includes an $tanh$ layer that makes a vector of new outcome values and an $sigmoid$ layer that picks which parts of the cell state are yielded.

The operations within an LSTM network are defined by several key gates and state variables. The forget gate, input gate, and output gate regulate the flow of information in the cell. Below is the description of each operation with its corresponding equation.

Forget Gate Activation: The forget gate controls the portion of the previous cell state that should be retained or discarded. Here, the decision-making process is achieved via a sigmoid function, which processes both the current input and the hidden state from the previous timestamp, allowing the proposed model to dynamically adjust its memory on the basis of the current context. By effectively controlling what information is retained or discarded, the forget gate enables LSTMs to maintain long-term dependencies and adaptively respond to new inputs, thereby facilitating performance. This is mathematically represented by:

$$for_{gate} = \sigma_{gate}(Wei_{for} \cdot m_t + U_{for} \cdot h_t - 1 + bi_{for}) \quad (1)$$

where h_{t-1} represents the hidden state from the preceding time step, for_{get} represents the forget gate, σ_g represents the sigmoid activation function, Wei_f and U_f represent the weight matrices, bi_{for} represents the bias of the forget gate, m_t represents the input at time t, and b_f represents the bias term.

Input Gate Activation: The input gate regulates the amount of new detail that is incorporated into the cell state. The corresponding equation 2 is as follows:

$$Inp_{gate} = \sigma g(Wei_{inp} \cdot m_t + U_{inp} \cdot h_t - 1 + bi_{inp}) \quad (2)$$

In the equation, Inp_{gate} is denoted as the input gate, Wei_{inp} signifies the weight matrix of the input gate, and bi_{inp} signifies the bias term of the input gate.

Cell input activation: The cell input activation phase estimates candidate values for updating the cell state, given by

$$\tilde{c}_t = \sigma_{cell}(Wei_{cell} \cdot m_t + U_{cell} \cdot ht - 1 + bi_{cell}) \quad (3)$$

where σ_{cell} is the activation function used for generating candidate values, Wei_{cell} and U_{cell} are the weight matrices for the cell input, and bi_{cell} is represented as a bias term.

Update the cell state: the updated cell state is an amalgamation of the preceding cell state and the candidate values. It is calculated via equation (4).

$$cell_t = for_t . can_{t-1} + inp_t . can_t \quad (4)$$

where can_t signifies the updated cell state and where can_{t-1} is the preceding cell state.

Updated Hidden State: Eventually, the hidden state output is estimated depending on the updated cell state, as depicted in equation 5:

$$h_t = out_t . \sigma_c(can_t) \quad (5)$$

Here, the hidden state h_t is derived by applying the output gate activation, can_t is transformed by the activation function σ_c, and out_t is represented as the cell state.

Finally, training and testing take place. During the training phase, a dataset curated by users, which includes correctly spelled words, is utilized to teach the model. This process enables the LSTM model to learn significant patterns and characteristics, allowing it to make predictions. Furthermore, by analysing the relationships within the data, the proposed model becomes apt at recognizing linguistic structures, thereby enhancing the performance in tasks such as text generation or error correction. Thus, the internal mechanism carried out is shown in Figure 5.

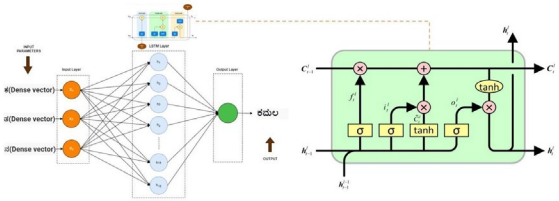

Figure 5. Internal mechanism

Figure 5 shows that the input parameters such as Kannada letters are passed to the model, where the input layer is responsible for accepting sequences of letters and characters. Later, it enters the LSTM layer, where the LSTM layer effectively handles the sequential data and maintains the context across longer sentences, enabling more precise predictions of the next words depending on the previous words in a sequence. Eventually, the output layer shows the word suggested on the basis of the input given to the model.

3.5 Hyperparameter tuning

Hyperparameter tuning refers to systematically adjusting parameters in an AI model that are not learned during training but have an important influence on its performance and behavior. It includes aspects such as the learning rate, batch size, number of epochs, and architectural elements of the model itself, such as layer sizes and activation functions. Unlike model parameters (weights and biases), which are updated during training on the basis of the data, hyperparameters are set before the training process begins and typically require manual intervention and experimentation for optimization. The objective of hyperparameter tuning is to treasure the mix of values that grow the model's display estimations, for instance, accuracy or disaster, on an endorsement set or during cross-endorsement. Therefore, Table 2 shows the hyperparameters used in the proposed work for the word suggestion system.

Table 2 Hyperparameter tuning

Parameters	Description
Learning Rate	➢ The learning rate of 0.002 for the LSTM in the generator of cGAN indicates a careful balance between stability and efficiency in training.

	➢ In the context of LSTMs, which are used for sequence prediction tasks like generating Kannada words, selecting the right learning rate is vital to ensure the model can learn intricate patterns over time.
Epochs	➢ Increasing the number of epochs to 30 means that the model will go through the complete training dataset 30 times. ➢ This decision suggests a more thorough training process, allowing the model to learn deeper patterns from the data.
Batch Size	➢ Increasing the number of epochs to 30 means that the model will pass through the complete training dataset 30 times. This decision suggests a more thorough training process, allowing the model to study deeper patterns from the data.
Loss Function (Binary Cross Entropy)	➢ The loss function evaluates the accuracy of the model predictions in relation to the actual target values. ➢ For the cGAN setup, binary cross entropy (BCE) is used as the loss function, suitable for distinguishing between real and generated data.
Optimizer (Adam)	➢ The optimizer refreshes the model loads to restrict the mishap capacity. ➢ Adam (Flexible Second Evaluation) is picked for its adaptable learning rate limits and powerful treatment of pitiful tendencies.

4. Results and discussion

The results obtained via cGAN for word suggestion with the aim of correcting spelling mistakes are depicted below. In addition, a web-based application output is also demonstrated.

4.1 Environmental Configuration

Table-3 Implementation Detail

Hardware-Configuration	Software-Configuration
CPU-Intel Core i7-7700@2.80 GHz	Windows 11
GPU - GTX 1050	Python-3.7
RAM: 16 GB	Anaconda-Spyder

Table 3 shows the environmental setup used by the proposed model for obtaining better outcome, where hardware configuration of CPU with Intel core i7, GPU of GTX 1050 and RAM of 16 GB model. For software configuration, Windows 11 OS, python 3.7 and Anaconda Spyder is used.

4.2 Experimental Outcomes

The experimental outcome obtained via the cGAN is demonstrated in the subsequent section. Figure 6 shows D loss and G loss for the cGAN model.

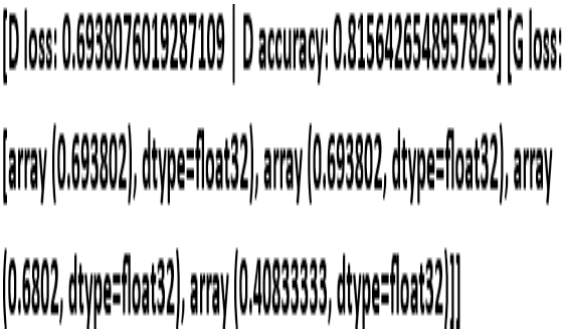

Figure 6. cGAN generator and discriminator loss

In a cGAN, the goal is for the generator to deliver information that is unclear from genuine information to the discriminator. This adversarial process leads

both networks to improve iteratively. The stability of the metrics across the epochs suggests that the networks are in equilibrium, where neither the generator nor the discriminator is gaining a significant advantage over the other.

4.2.1 Discriminator Metrics

- D loss: This measures how effective the discriminator is at recognizing genuine and created information. A lower D misfortune shows that the discriminator is better at its undertaking.
- D accuracy: This measurement reflects the accuracy of the discriminator in accurately characterizing genuine and produced information tests. Higher precision corresponds to better execution.

4.2.2 Generator metrics

- G loss: This evaluates how well the generator is fooling the discriminator. A decreasing G loss suggests that the generator is improving at generating more realistic data.

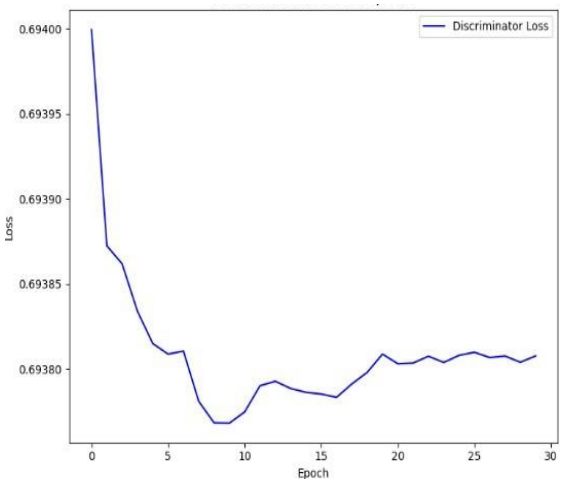

Figure 7. Discriminator Loss

Figure 7 shows the discriminator loss for an epoch of 30. The discriminator loss (D loss) is a crucial metric in cGANs, reflecting how well the discriminator model distinguishes between real data and generated data. Figure 6 shows the discriminator loss for an epoch of 30. The cGAN was trained via TensorFlow, and the D loss was monitored throughout the training process. The cGAN consists of a generator model that learns to generate synthetic data and a discriminator model that learns to distinguish between real and generated data. The models have trained adversarially, where the generator seeks to reduce its loss by fooling the discriminator, while the discriminator aims to maximize its accuracy in distinguishing between real and fake data. Initially, the D loss values were relatively high, averaging approximately 0.693. This high starting point indicates that the discriminator faced challenges in accurately classifying real and generated data, possibly because the generator produced outputs that were easily distinguishable as fake. As training progressed, the D loss exhibited a trend on the graph. The gradual decline in D loss values over epochs indicates that the discriminator improved its ability to differentiate between real and generated data. This trend is indicative of the discriminator becoming more apt at differentiating realistic data from synthetic outputs.

During mid-training, fluctuations in D loss were observed. These fluctuations are characteristic of the combative training process, where both the generator and discriminator continuously adapt and refine their strategies. The graph displays these fluctuations as oscillations or minor peaks and valleys. Towards the later epochs, the D loss stabilized or showed consistent behavior. This stabilization phase on the graph signifies that the discriminator reached a level of proficiency where it consistently classified real and generated data with higher accuracy.

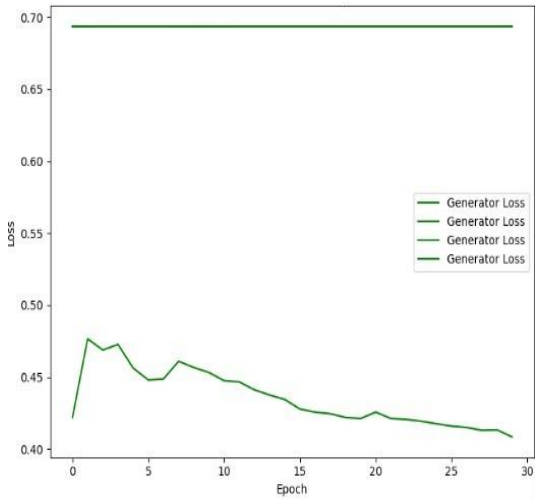

Figure 8. Generator loss

Figure 8 shows the generator loss. The generator loss (G loss) is a critical metric in cGANs, reflecting how well the generator model learns to produce outputs that mimic real data. cGAN was trained via TensorFlow and monitored the G loss throughout the training process. The cGAN comprises a generator model that yields artificial data and trains a discriminator model to discern between artificial and genuine data. The models were trained adversarially, with the discriminator trying to increase its accuracy in identifying real data from fake data and the generator trying to minimize its

loss by tricking the discriminator. In the initial epochs, the G loss values were relatively high, averaging approximately 0.693. This high starting point indicates that the generator initially produced outputs that were easily discernible as fake by the discriminator. As training progresses, the G loss gradually decreases on the graph. This decline signifies that the generator improved its ability to produce outputs that more closely resembled real data, thereby challenging the discriminator's capacity to recognize genuine and produced information.

During mid-training, changes in G misfortune were observed. These oscillations are normal for the ill-disposed preparation process, where both the generator and discriminator continuously adapt and improve their strategies. The graph displays these fluctuations as oscillations or small peaks and valleys. Towards the later epochs, the G loss stabilized or consistently decreased. This stabilization phase on the graph indicates that the generator achieved a level of performance where it consistently generated outputs that were increasingly difficult for the discriminator to classify as fake. Additionally, a lower loss component (0.408) was identified, which reflected the effects of regularization techniques applied during training.

4.3 Front-end application outcomes

Figure 9 illustrates the web interface results suggesting words using Flask. Flasks are versatile web frameworks that facilitate the development of web applications and offer various advantages for building applications because of their simplicity and flexibility. Thus, Figure 9 shows the working interface of the model, where similar words are generated on the basis of the query word.

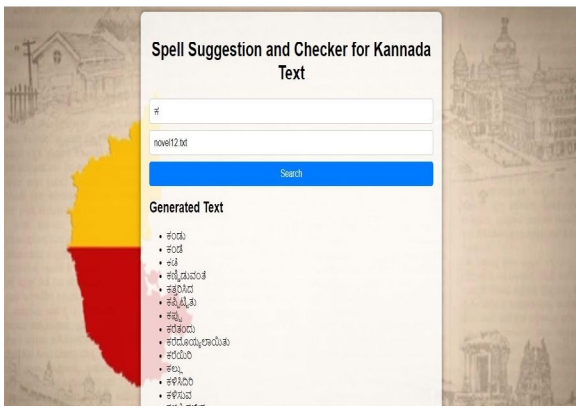

Figure 9. Web interface showing results

The front-end web interface application is designed for retrieving words starting with a particular letter in Kannada and involves several key components to ensure a seamless user experience. The interface begins with a simple yet effective user input mechanism, typically featuring a text input field where users can enter a single letter or a sequence of letters in the Kannada script. This input is complemented by a submit button that triggers the request to the backend server for data retrieval. Upon submission, the front end sends an HTTP request to the backend, transmitting the specified letter(s) as parameters or within the request body. This interaction is crucial for initiating the backend's processing logic, which is responsible for handling the user query.

4.4 Discussion

Assamese [21], Urudu [24], Punjabi [26], Azerbaijani [27], Indonesian [22] and many more using different approaches like LSTM, BiLSTM, Seq2Seq models and other, and the accuracy obtained by these models are 92% [21], 87% [25]. However, uniqueness of the proposed model lies by employing cGAN model along with LSTM as most existing works have not employed by using GAN network architecture, in cGAN, the generator model learnt to produce plausible Kannada words on the basis of initial character inputs, whereas the discriminator model is prepared to recognize genuine and created words. By doing, effective word suggestion process takes place. Moreover, the accuracy gained by the proposed model is 94.4% which is higher than the existing works. Therefore, by covering Kannada language with better numerical outcome of the present model makes it effective and efficient than state-of-the-art approaches.

The proposed cGAN-LSTM model achieves higher accuracy (94.5%) compared to previous studies, demonstrating superior spell correction performance in Kannada. The incorporation of adversarial learning and contextual embeddings enhances the model's ability to generate precise spell corrections.

5. Conclusion

The present research focuses on creating a Kannada word suggestion system utilizing a cGAN integrated with a web interface. By leveraging AWS S3 for data storage and employing TensorFlow for model development, the objective was to train cGAN models to propose Kannada words on the basis of the letters input by users, ensuring seamless integration with flasks for deploying the web application. Through careful preparation of the dataset, tokenization and training of the LSTM model, the system achieved precise word predictions. The cGAN model demonstrated efficient learning dynamics, as evidenced by the convergence of the loss for both the discriminator and generator throughout the training epochs. The

stabilization of the D loss indicated a heightened accuracy in differentiating between authentic and generated data, whereas the decrease in G loss demonstrated an improved ability to produce realistic Kannada words. The frontend web interface provides an intuitive platform for users, enabling easy input of letters and quick display of word suggestions. This seamless integration of frontend and backend elements not only highlights the application of sophisticated ML techniques in NLP but also emphasizes the importance of user-centered design in effectively implementing such solutions.

In the future, a more sophisticated DL architecture, such as the transformer model, can be used, as it has the potential to greatly improve the prediction accuracy and contextual comprehension of Kannada text. Furthermore, expanding the system to accommodate additional languages beyond the Kannada language by using transfer learning and multilingual models would enable it to address a wider range of linguistic requirements as the present work limits to Kannada language, thus enhancing its applicability and effectiveness across different linguistic environments.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
None.

Conflict of Interest
There is no conflict of interest.

Funding Support
There is no funding support for this study.

Data Availability Statement
Not applicable.

Authors' contributions
Prathibha R J – Supervising and Project Administration.
Anupama N – Conceptualization and Writing
Sudarshan A S - Conceptualization and Writing

Acknowledgements
None.

References

[1] Goswami, B., Bhavsar, N., Alzobidy, S. A., Lavanya, B., Udhayakumar, R., & Rajapandian, K. (2025). Sentiment Analysis Using Natural Language Processing. Natural Language Processing for Software Engineering, 283-294.

[2] Esfahani, M. N. (2024). Content Analysis of Textbooks via Natural Language Processing. American Journal of Education and Practice, 8(4), 36-54.

[3] Dong, J. (2024). Natural Language Processing Pretraining Language Model for Computer Intelligent Recognition Technology. ACM Transactions on Asian and Low-Resource Language Information Processing, 23(8), 1-12.

[4] Perera, S. S., & Sumanathilaka, D. K. (2025, January). Machine Translation and Transliteration for Indo-Aryan Languages: A Systematic Review. In Proceedings of the First Workshop on Natural Language Processing for Indo-Aryan and Dravidian Languages (pp. 11-21).

[5] Shukla, K., Vashishtha, E., Sandhu, M., & Choubey, P. R. (2023). Natural Language Processing: Unlocking the Power of Text and Speech Data. Xoffencerpublication.

[6] Stanisz, T., Drożdż, S., & Kwapień, J. (2024). Complex systems approach to natural language. Physics Reports, 1053, 1-84.

[7] Anitha, G., Kumar, G. S., Swamy, B. M., Thriveni, J., & Venugopal, K. R. (2021). Kannada morphological analyser and generator using natural language processing and ML approaches. Turkish Journal of Computer and Mathematics Education, 12(11), 2826-2838.

[8] Irani, M., Elahimanesh, M. H., Ghafouri, A., & Bidgoli, B. M. (2022, December). A Supervised Deep Learning-based Approach for Bilingual Arabic and Persian Spell Correction. In 2022 8th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS) (pp. 1-7). IEEE.

[9] Phukan, R., Neog, M., Goutom, P. J., & Baruah, N. (2024). Automated Spelling Error Detection in Assamese Texts using Deep Learning Approaches. Procedia Computer Science, 235, 1684-1694.

[10] Jim, J. R., Talukder, M. A. R., Malakar, P., Kabir, M. M., Nur, K., & Mridha, M. F. (2024). Recent advancements and challenges of NLP-based sentiment analysis: A state-of-the-art review. Natural Language Processing Journal, 100059.

[11] Aithal, S. R., Sn, M., Ganiga, R., Rao B, A., & Hegde K, G. (2024). KannadaLex: A lexical database with psycholinguistic information. ACM Transactions on Asian and Low-Resource Language Information Processing, 23(7), 1-21.

[12] Prasad, C., Kallimani, J. S., Reddy, G., & Dhanashekar, K. (2025). Developing spell check and transliteration tools for Indian regional language–Kannada. In Applied Data Science and Smart Systems (pp. 152-161). CRC Press.

[13] Lubis, Y., Ramadhany, C. A., Widyana, A., Mahara, E. F., & Sarahseti, D. F. (2024). The Role of Vowels and Consonants In English Language Learning. Socius: Jurnal Penelitian Ilmu-Ilmu Sosial, 1(11).

[14] Singh, S., & Singh, S. (2021). HINDIA: a deep-learning-based model for spell-checking of Hindi language. Neural Computing and Applications, 33(8), 3825-3840.

[15] Abdullaev, I., Prodanova, N., Ahmed, M. A., Lydia, E. L., Shrestha, B., Joshi, G. P., & Cho, W. (2023). Leveraging metaheuristics with artificial intelligence for customer churn prediction in telecom industries. Electronic Research Archive, 31(8).

[16] Abandah, G. A., Suyyagh, A., & Khedher, M. Z. (2021). Correcting arabic soft spelling mistakes using bilstm-based machine learning. arXiv preprint arXiv:2108.01141.

[17] Kasmaiee, S., Kasmaiee, S., & Homayounpour, M. (2023). Correcting spelling mistakes in Persian texts with rules and deep learning methods. Scientific reports, 13(1), 19945.
[18] Caryappa, B. C., Hulipalled, V. R., & Simha, J. B. (2020, October). Kannada grammar checker using LSTM neural network. In 2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE) (pp. 332-337). IEEE.
[19] Murthy, S. R., Akshatha, A. N., Upadhyaya, C. G., & Kumar, P. R. (2017, September). Kannada spell checker with sandhi splitter. In 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 950-956). IEEE.
[20] Dashti, S. M. S., Khatibi Bardsiri, A., & Jafari Shahbazzadeh, M. (2024). PERCORE: A deep learning-based framework for persian spelling correction with phonetic analysis. International Journal of Computational Intelligence Systems, 17(1), 114.
[21] Phukan, R., Baruah, N., Sarma, S. K., & Konwar, D. (2024). Exploring Character-Level Deep Learning Models for POS Tagging in Assamese Language. Procedia Computer Science, 235, 1467-1476.
[22] Zaky, D., & Romadhony, A. (2019, September). An LSTM-based spell checker for indonesian text. In 2019 international conference of advanced informatics: concepts, theory and applications (ICAICTA) (pp. 1-6). IEEE.
[23] Ji, T., Yan, H., & Qiu, X. (2021, November). SpellBERT: A lightweight pretrained model for Chinese spelling check. In Proceedings of the 2021 conference on empirical methods in natural language processing (pp. 3544-3551).
[24] Aziz, R., Anwar, M. W., Jamal, M. H., & Bajwa, U. I. (2021). A hybrid model for spelling error detection and correction for Urdu language. Neural Computing and Applications, 33, 14707-14721.
[25] Kasmaiee, S., Kasmaiee, S., & Homayounpour, M. (2023). Correcting spelling mistakes in Persian texts with rules and deep learning methods. Scientific reports, 13(1), 19945.
[26] Kaur, G., Kaur, K., & Singh, P. (2019, March). Spell checker for Punjabi language using deep neural network. In 2019 5th international conference on advanced computing & communication systems (ICACCS) (pp. 147-151). IEEE.
[27] Ahmadzade, A., & Malekzadeh, S. (2021). Spell correction for azerbaijani language using deep neural networks. arXiv preprint arXiv:2102.03218.