TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Major Project Mid-Term Report

On

Nepali Context-Aware Spelling Tool

Submitted By:

Anish Raj Manandhar	[THA077BCT010]
Nabin Shrestha	[THA077BCT026]
Prayush Bhattarai	[THA077BCT035]

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

January, 2025

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Major Project Mid-Term Report

On

Nepali Context-Aware Spelling Tool

Submitted By:

Anish Raj Manandhar	[THA077BCT010]
Nabin Shrestha	[THA077BCT026]
Prayush Bhattarai	[THA077BCT035]

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of a Bachelor's degree in Computer Engineering

Under the Supervision of

Er. Shanta Maharjan

January, 2025

ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards the Institute of Engineering, Tribhuvan University for the inclusion of the major during bachelor's in computer engineering. We are especially thankful to the Department of Electronics and Computer Engineering, Thapathali Campus for providing us with such an auspicious opportunity to work on this project. We would also like to express our respect towards the authors and various developers whose works we have referenced in building this proposal. The articles served as a great source of knowledge and inspiration for all of us. The entire proposal wouldn't have been possible without their help.

Finally, we would like to express our thankfulness to our supervisor **Er. Shanta Maharjan**, co-supervisor **Er. Prabin Acharya** and our friends who have been involved with us in the report in one way or the other. We are extremely grateful for their support, cooperation, help, guidance, and encouragement in making this report.

Anish Raj Manandhar [THA077BCT010]
Nabin Shrestha [THA077BCT026]
Prayush Bhattarai [THA077BCT035]

ABSTRACT

This project aims to develop a context-based spelling correction system for the Nepali language. In Part A, we employed the Word2Vec model, as limited work had been done for this task using this approach. However, Word2Vec's static embeddings and lack of positional encoding resulted in poor performance. To address this, we collected around 18,000 similar-sounding words (e.g., homophones) from Nepali dictionaries using transliteration and manual search, organizing them into a confusion set. Additionally, we expanded our dataset to around 0.8 billion tokens for fine-tuning. In Part B, we transitioned to a transformer-based encoder model, BERT, achieving a baseline accuracy of 60.67%. Building on this, we fine-tuned NepBERTa and observed impressive loss curves. To evaluate its performance, we created a validation set of 1,000 sentences, with 909 sentences containing frequently repeated confusion words. For these, the validation accuracy was 82.06%, while for less frequently occurring confusion words, the accuracy was 63.63%. This highlights NepBERTa's effectiveness in context-aware spelling correction, particularly for commonly repeated confusion words in the training set.

Keywords: *BERT, Homophones, Positional Encoding,Semantic Errors, Word2Vec*

TABLE OF CONTENTS

List of Figures

List of Tables

List of abbreviations

BERT	Bidirectional Encoder Representations
FFNN	Feed Forward Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
ML	Machine Learning
NLP	Natural Language Processing
POS	Part-of-Speech
RNN	Recurrent Neural Networks
TPU	Tensor Processing Unit
UI	User Interface

1 INTRODUCTION

1.1 Background

Semantically correct contextual Nepali writing has now become a common challenge. Linguistic errors made by officials who hold high positions and need to work daily in the Nepali language are neither forgivable nor ignorable. In the current era, where the world has become confined to social networks, such errors made at high levels quickly spread on social media. People then form their opinions about these individuals based on their linguistic proficiency, and in some cases, these individuals become subjects of criticism due to their linguistic weaknesses.

With Nepali being the official language of Nepal and widely spoken in neighboring regions, there's a significant demand for computational tools that cater to its unique linguistic features. However, building such tools presents challenges due to Nepali's complex morphology, syntactic structures, and script. Traditional approaches often fall short in handling Nepali's rich vocabulary, compound words, and inflections. Thus, an interdisciplinary approach that combines linguistic expertise with computational methods becomes essential to develop effective language processing solutions for Nepali. By leveraging advancements in NLP, ML, and linguistic analysis, these tools can bridge the gap in Nepali language technology, enabling better communication, content creation, and information retrieval for Nepali speakers across various domains.

1.2 Motivation

The Nepali language, with its rich linguistic heritage and significant cultural importance, faces challenges in the digital age due to limited resources and tools for accurate language processing. Existing tools often fail to account for contextual nuances, leading to errors and misunderstandings in written communication. This gap underscores the need for a sophisticated, context-aware solution that can accurately process and correct Nepali text.

Developing a context-aware language processing tool for Nepali is motivated by the desire to enhance digital communication, support education, and preserve the

language in a rapidly evolving technological landscape. By incorporating contextual understanding, this tool will not only improve text accuracy but also contribute to the broader goals of language preservation and digital inclusivity.

1.3 Problem Definition

There is not much work in the context of Natural Nepali Language Processing. Most state-of-the-art research has been done for English Language. In the context of modern communication, writing text in a certain language digitally is one of the most basic tasks that any user can perform. In the case of Nepali, there are various semantic errors. Moreover there is not much work done in to resolve the problem of contextual spelling correction or suggestion. Although all possible semantic errors cannot be partitioned neatly into specific categories, the following are some of the most common spelling errors: Confusion between long (dirgha)(example: फुल) and short vowels (hrasva)(फूल). Vowels in Nepali are characterized by their duration, that is long vowels are spoken for approximately twice as long as short vowels. So, the confusion between long and short vowels often leads to errors.

1.4 Objectives

The main objective of the project is listed below:

- To develop a sophisticated spelling checker that can detect and correct spelling errors based on the context of the entire sentence.

1.5 Scope of Project

The project focuses on developing a context-based spelling checker specifically for the Nepali language. The primary objective is to create a system that can accurately detect and correct spelling errors by understanding the context within sentences. Unlike traditional spell checkers that only identify isolated errors, this project aims to enhance the accuracy of corrections by considering the surrounding words and phrases. For example, the system will be able to distinguish between homophones like ”फूल” (phool - flower) and ”फल” (phal - fruit) based on contextual usage, ensuring that sentences such as ”मैले फल खाएँ” (Maile phal khāẽ) are correctly interpreted and corrected. The

scope involves collecting a diverse corpus of Nepali text, preprocessing the data to handle special characters and normalization, and developing advanced machine learning models, potentially leveraging BERT for its superior context understanding capabilities. The project will culminate in a user-friendly application that provides real-time spelling corrections, significantly improving the quality and accuracy of written Nepali. This tool will be particularly valuable for educational purposes, professional writing, and digital content creation, addressing a critical need for precise language tools in the Nepali-speaking community.

1.6 Applications

Context based Spelling correction model can be implemented as a standalone application or integrated with other systems to improve their performance. The standalone use may be limited but can be combination with the exiting systems can be significant.

Media and Publishing

Nepali context based spell checker and similar word generator can enhance the quality and professionalism of written content in newspapers, magazines, websites, and other media outlets

Education

Enhancing writing skills and aiding language learning in Nepali classrooms.

Search Engine

Correcting the context of the query may result in more relevant results in the searching process.

Text Processor Systems

Spelling Checker is a useful part of Text Processing Systems. When writing any text in a text processor, errors naturally arise which hinder comprehension. To solve this in the case of Nepali, the spell checker project proposed can be used in the text processor to highlight the incorrectly spelled words and suggest their correct spelling to the user.

OCR Projects

Optical Character Recognition(OCR) is a software that converts texts written on physical things(paper, whiteboard, etc) to digital text. In doing so, they can often identify the characters incorrectly and give words with incorrect spellings as the output. To solve this, spell checker and correcter can be used to correct the optically identified characters. So, Spell Checker can be used as a downstream layer for OCR applications to increase their accuracy.

Voice Recognition Systems

Integration of the context based spelling correction model into speech-to-text systems enables transcription accuracy enhancement through automatic correction of contextual spelling errors in the transcribed test ensuring more precise and error-free output. This enhancement reduces the need for manual proofreading and enhances the overall user experience by providing more reliable transcripts.

Content Creation Tools

The spelling correction integration on the content creation tools can streamline the editing process, allowing automatic context based spelling correction in drafts. This feature not only saves valuable time but also helps maintain correctness and clarity in written content, ultimately improving the quality of the final product.

2 LITERATURE REVIEW

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) and linguistics that focuses on the interaction between computers and humans through natural language. As computers cannot generally understand natural languages in the same way that humans can, there are great incentives in training a computer to understand natural languages. Although some basic development in the field of NLP can be traced back all the way back to the 1950s to Alan Turing [1], who proposed the famous Turing Test, there has been rapid development only in the last few decades. Especially with the public launch of ChatGPT in 2022, there has been a lot of public attention not only in the scientific and development community but even across society as a whole.

The main intuition behind this paper is to develop a system that can be extended to work for other languages out of the box as most of the state-of-the-art(SOTA) research has been done for spelling checking in English language. Traditionally, instead of using probablistic model for word suggestions and spelling checking rule based systems were implemented. This research utilized the n-gram probability dictionaries by using various wikipedia-articles and video subtitles. Here, dataset was create by using their own techniques. They have proposed three approaches to create noisy channel datasets of real world typographic errors. They are: Randomized Characters, Character Swap and Character Bigrams. They have claimed and proved that their system outperformed in industry-wide accepted english spell checkers.The experiment was performed on 24 different language datasets. The 24 languages in which this experiment was performed on were: Bengali, Czech, Danish, Dutch, English, Finnish, French,Hindi, Italian, Indonesian, Greek, German, Marathi, Polish, Romanian, Portugurese, Russian, Swedish, Spanish, Tamil, Thai, Telugu and Turkish. This paper talks about two approaches to suggests the words in the small dataset. First, checking edit-distance of incorrect spelling words in th dictionary and second generating a list of the words with edit distance of 2 of the incorrect word. Here, for word suggestion weighted sum of the unigram contex score, bigram context score and trigram context score is calculated and used as the overall context score. [2]

The [3], was designed to detect and correct the misspelled words in the health domain. This model firstly detected the misspelled word using the seq2seq language model built using Gated Recurrent Unit (GRU) deep neural network. After that, using Levenshtein distance algorithm was applied to find the probabilities of correctly spell words among the several possible candidates. They pointed out Train set accuracy is 75.11% and test set accuracy is 73%. The vocab size used was only 1,20,000. It may encounters the Out of Vocabualry (OOV) problems. GRU can understand some context but they may struggle with words that have multiple meaning or synonyms. Also levenshtein distance works character level differences without understanding the semantic relationships between the words. If The new word is introducing. It requires extensive retraining to adapt to new data, which can be resource-intensive as this model is static in nature.

The development of Nepali Lexicon [4] , has been crucial part of enhancing language processing tools and applications. A lexicon is a structured database containing information about the words and phrases of the Nepali language, including their meanings, pronunciations, parts of speech, and other linguistic properties. This review highlights the methodologies, challenges, and outcomes associated with the development of such a lexicon, emphasizing its importance in spell checking and other NLP applications. Early efforts in Nepali lexicon development were minimal and largely manual, involving traditional methods of dictionary compilation. The lack of digital resources posed significant challenges for language researchers and practitioners. Several projects aimed at creating comprehensive lexical databases, which included morphological and syntactic information, essential for natural language processing (NLP) applications. This had utilized text corpora to automatically extract and validate lexical entries. Techniques such as frequency analysis and context examination were employed to identify common words and phrases. Also, they had engaged native speakers and linguists through digital platforms to contribute to and validate lexicon entries. This method helps in gathering diverse linguistic data and ensuring accuracy. They had aimed a minimum measure of 25,000 words with at least 400 noun headwords, one hundred and fifty verbs, one hundred adverbs and adjective and all pronouns of Nepali grammar in the entries of lexicon database but due to limited Resources and the orthographic variations had caused to only make 500 root words.

Traditional word-based models struggle with misspelled words since they rely on a fixed vocabulary and might not recognize “visitted” word in below example.

For example: “I visitted Tokyo on Nov 2003.:”

This [5] , process input at the character level, which allows it to understand and correct misspellings by considering sequence of characters. During training, the model learns to map common misspellings to their correct forms based on context. For the non-idiomatic prepositions, this uses attention mechanism which allows the model to focus on relevant parts of the sentence when generating each character of the corrected output. This helps in understanding the context better and making appropriate corrections. For example, the model can learn from training data that ”on” should be ”in” when the context is a month and year. This approach inherently processes every character, including punctuation and special characters, making it well-suited to handle emoticons and other non-standard text elements. By following similar steps, the character-based neural network approach can be effectively adapted to correct grammatical and orthographic errors in the Nepali language, providing significant assistance to Nepali language learners and speakers but character-level models can be slower during inference due to the granularity of processing each character sequentially.

Contextual Spelling Correction with Language Model for Low-resource Setting proposed by Nishant Luitel et al (2023) presents unique challenges that require specialized approaches leveraging state-of-the-art transformer models. This paper introduces an innovative method employing models such as KnLM-2, NepaliBert, deBerta, Trans, and Trans-word to enhance correction accuracy. The approach begins with heuristic-based candidate generation, reducing candidates based on minimal edit distances for efficiency. For error modeling, a probabilistic framework approximates the likelihood that observed misspellings derive from intended words using partitioning and frequency-based metrics from large corpora. Training a neural LM, including variants like NepaliBert and deBerta, improves predictions by contextualizing candidate probabilities. By integrating these models via Bayesian inference, the method effectively ranks candidate words and sentences, demonstrating robust performance in generating and correcting errors across Nepali and potentially other Indic languages. Evaluation on a dataset of Nepali sentences underscores the approach’s efficacy, paving

the way for broader applications in linguistic correction and enhancement. [6]

Vartani Spellcheck by Aditya pal et al, introduces a novel approach to address spelling error correction in a low-resource Indic language using the BERT transformer model. The study demonstrates promising results, indicating potential enhancements through further fine-tuning of BERT and exploration of advanced transformer models like ALBERT and RoBERTa. Future plans include adapting Indic-BERT, pretrained on extensive Hindi text, to improve candidate selection for masked tokens. While initial focus has been on Out of Vocabulary (OOV) words, ongoing work aims to extend capabilities to detect real-word errors across various Indic languages such as Bengali, Tamil, Telugu, Marathi, and Gujarati. The development also includes creating datasets for classifying grammatically correct and incorrect sentences, paving the way for broader language coverage in subsequent iterations of Vartani Spellcheck. Additionally, efforts are underway to fine-tune BERT at the character level, potentially enhancing performance in real-world applications like on-the-fly spell correction within text editors, where contextual analysis can anticipate corrections in a user-friendly manner.[7]

Context-sensitive Spelling Correction by Youssef Bassil et al (2012) proposes a novel context-sensitive spelling correction method that leverages statistical insights from the Google Web 1T 5-gram dataset, containing extensive n-gram sequences extracted from the internet. The method integrates an error detector, utilizing statistical models to identify misspellings, a candidate spellings generator based on character 2-gram models for correction suggestions, and an error corrector that applies contextual understanding to enhance correction accuracy. Experimental results across various domains demonstrate significant improvements in both non-word and real-word error correction rates. Future work aims to parallelize the algorithm to optimize computational efficiency in error detection and correction processes.[8]

Rijal, Birodh, et al. used the n-gram model to suggest the correct word in place of the incorrect word as mentioned in the paper ’Vector distance based spelling checking systemin Nepali with language-dependent’.[9] The work in this article focuses on spell checking for non-word errors in Nepali. The system utilizes two types of ranking

measures. The first measure involves language-specific heuristics that identify errors in special symbols, as well as in the independent and dependent forms of characters commonly made during Nepali text writing. The second measure uses vector distances between n-grams to generate a list of potential replacements for the misspelled words. These candidate replacements are then ranked based on the Levenshtein distance measure. Finally, the top-ranked words are provided as suggestions for correcting the misspelling. In this paper it is mentioned that the spelling correction method can be decomposed into two components that are: First one is the edit distance function and the second one relies on the bayesian network or probability theory. The model built here maily focuses on the non-word errors for Nepali. The model proposed suffers from multi word spelling words errors (split of words errors). The model utilizes the handcrafted heuristics to make suggestion. The baseline model was able to give the accuracy of 51.74% and after the addition of the heuristics the accuracy increased to 92.32%. Here, it has been claimed that this is the first Nepali spell checker to use count based statistics of character n-grams and nepali language heuristic.

R. Birodh, A.Sushil, et al has proposed a method than could be used to cleaning the noisy data in the nepali corpus.'Preprocessing of Nepali News Corpus for Downstream Tasks' provides a preprocessing pipeline which could be used in preprosessing the Nepali news corpus for training NLP models. As the news available in the web are very erroneous and the performance of the NlP model without preprocessing these corpus leads to poor performance of the model. First, the text files are merged into a single large file and feed it into the pipeline. Then, text is tokenized in word level. The special characters not belonging to the nepali language are filtered out. The glyph and typo sanitizer in the pipeline are used to tokenize a given word into Unicode characters then thetokens which donot belong to any glyph groups are filtered out. This system utilizes the vector distance based spelling checking system which is the state-of-the-art for Nepali spelling checking.[10]

The paper "NepaliBERT: Pre-training of Masked Language Model in Nepali Corpus" by Pudasaini et al. explores the development of a BERT-based model tailored for the Nepali language, addressing the lack of NLP resources for low-resource languages. Using over 4.6 GB of Nepali text sourced from news articles, the authors successfully trained a

masked language model leveraging the BERT architecture. The study demonstrates the model's effectiveness in intrinsic evaluations, achieving a perplexity of 8.56, and extrinsic evaluations, outperforming existing models in tasks such as sentiment analysis of Nepali tweets. By overcoming challenges like limited data and computational constraints through modern GPU infrastructure, this work highlights the potential of language-specific models in improving NLP applications for low-resource languages. [11]

The paper NepBERTa: Nepali Language Model Trained in a Large Corpus focuses on developing a monolingual BERT-based model specifically for the Nepali language, which has traditionally been underrepresented in Natural Language Processing (NLP). NepBERTa was trained on a massive dataset of 0.8 billion words collected from 36 Nepali news sites, making the dataset three times larger than previously available corpora. The model outperformed multilingual models like Multilingual BERT, XLM, and XLM-RoBERTa in Nepali-specific NLP tasks such as Named Entity Recognition (NER), POS Tagging, and Sequence Pair Similarity. The paper also introduces the Nepali General Language Understanding Evaluation (Nep-gLUE), a benchmark for evaluating Nepali NLP models, and offers pre-trained models and datasets to the research community. This work represents a significant advancement in Nepali language processing by focusing on its rich grammatical and semantic structures, which existing multilingual models have struggled to handle effectively. [12]

The paper "Structure of Nepali Grammar" explores the intricacies of the Nepali language's structure, focusing on its phonetics, morphology, syntax, and lexicon. It provides an in-depth analysis of the Devanagari script, used to write Nepali, and explains features such as vowels, consonants, and conjunct formations. Special emphasis is placed on the roles of viram (halanta), vowel signs, and conjunct consonants, which are critical in representing Nepali sounds and words. The grammar framework categorizes Nepali words into open and closed classes, detailing how inflections operate for nouns (number and case) and verbs (tense and mood). This structural breakdown sheds light on how Nepali sentences are constructed, the interplay between form classes, and the usage of particles for meaning and nuance. Additionally, it highlights the language's Sanskrit influence, evident in some loanwords and grammatical conventions. This study

contributes significantly to computational linguistics and natural language processing for Nepali by providing foundational grammatical rules that assist in tasks such as language modeling and machine translation. [13]

3 REQUIREMENT ANALYSIS

3.1 Project Requirements

3.1.1 Hardware Requirements

- **Computer**

  A computer was required to train the model. Furthermore, computers are also required to keep track of codes, reports, results,etc.

- **GPU and TPU**

  Description: Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) are specialized hardware accelerators used for training and running ML models efficiently.

  Usage in Project: These will be used for fine-tuning the spell checker model to handle large datasets and complex computations.

- **Development Machines**

  Description: Standard personal computers or laptops used by developers.

  Specifications:

  - Processor: Multi-core processor (Intel i5/i7 or equivalent).
  - RAM: Minimum 8GB, recommended 16GB.
  - Storage: SSD for faster read/write operations, at least 256GB.

- **GPU Requirement**

  Description: A GPU is crucial for tasks that involve intensive computations, such as training and running ML models.

  Specifications: NVIDIA GeForce RTX 3050 or higher: A modern GPU with sufficient VRAM (at least 4GB) and CUDA cores to handle ML workloads.

  Its Advantages include:

  - Significant speed-up in training ML models compared to CPU-only systems.
  - Enhanced performance for real-time inference and batch processing.

  Usage in Project: For local development and testing of ML models, allowing

developers to iterate quickly and efficiently without relying solely on cloud resources.

3.1.2 Platform Requirements

- **HuggingFace**

  HuggingFace will be required to upload datasets and the trained models. The pre-trained transformer models are available in HuggingFace from where they can be downloaded. It is also utilized for knowing the details about the models and trainers that will be used during the project.

- **Google Colab**

  Description: Google Colab is a cloud-based service that provides free access to GPUs and TPUs, integrated with Jupyter notebooks for an interactive development experience. Its Advantages include:

  - Free tier with access to powerful hardware accelerators.
  - No setup required, accessible via web browser.
  - Integration with Google Drive for easy data management.

  Usage in Project: To provide the necessary computational power for model training and evaluation in a collaborative environment.

- **Kaggle**

  Description: Kaggle is a platform for data science competitions and learning, offering free access to GPUs and TPUs, along with a vast repository of datasets.
  Advantages:

  - Free access to powerful computing resources.
  - Extensive collection of datasets for training and testing models.
  - Community forums and notebooks for collaboration and learning.

  Usage in Project: For experimenting with models, accessing datasets, and leveraging community knowledge.

3.1.3 Software Requirements

- **Python**

  Description: Python is a versatile and widely-used programming language, particularly popular for machine learning (ML) tasks due to its simplicity and extensive libraries. Its advantages include:

  - Free and open-source.
  - Extensive support community.
  - Rich ecosystem of libraries and frameworks.

  Usage in Project: Python will be used for developing the core ML algorithms for the spell checker.

- **HTML, CSS, JS**

  Description: HTML, CSS, and JavaScript are the fundamental technologies for building web applications.

  - HTML (HyperText Markup Language): The standard language for creating web pages and web applications.
  - CSS (Cascading Style Sheets): A style sheet language used for describing the presentation of a document written in HTML.
  - JavaScript (JS): A programming language that enables interactive web pages

  Its Advantages include:

  - All three are free and open-source.
  - Widely supported across all modern web browsers.
  - Essential for developing user interfaces and enhancing user experiences.

  Usage in Project: These technologies will be used to create the front-end of the spell checker application, allowing users to interact with the tool via a web browser.
  Additional Libraries/Frameworks: React.js will be used for building dynamic user interfaces.

- **Flask or Django**

  Description: Flask and Django are popular Python web frameworks.

  Its advantages include:

  - Facilitate the development of web applications.
  - Provide robust routing, templating, and authentication systems.

  Usage in Project: These frameworks will be used to build the backend of the web application, handling requests, serving the front-end, and interfacing with the ML models.

- **Pandas**

  Pandas is a Python library used for to manipulate and analyze data. It helped in providing structures for datasets at the time of preprocessing which were favourable for further processing such as tokenization, embedding and so on.

- **Transformers**

  The transformers library will used in the project to import models, tokenizer from HuggingFace. Also,it will be utilized for handling trainer, that is required to train the model.

- **Sklearn**

  Sklearn is the updated Python library of scikit-learn. This library will be used in the project to import evaluation matrices like accuracy, precision, recall and f1 score.

- **Dataset**

  The dataset library will be used for manipulating and preparing data for training, validating and evaluating. It will be used for splitting dataset, converting parallel datasets into dictionary, etc.

3.2 Functional Requirements and Non-Functional Requirements

3.2.1 Functional Requirements

Functional requirements define what a system should do. It outlines specific functionalities and features that the system must exhibit to meet the user needs. Some

of the functional requirements are stated as:

- The user shall be able to give input as a sentence.
- The system shall suggest a correct words for the given input sentence.
- The user shall be able to know where the correction has been made.

3.2.2 Non-Functional Requirements

The requirements which should be there in a system in order to enhance the working of the system can be referred as non-functional requirements. Some of the non-functional requirements of the system can be stated as:

- The user interface shall be simple and responsive.
- The response time of the system shall be optimal.
- The system should be able to run on local machine.

4 METHODOLOGY

4.1 Process Model

For the development of this project, we will be using an agile development model. Agile development model is a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

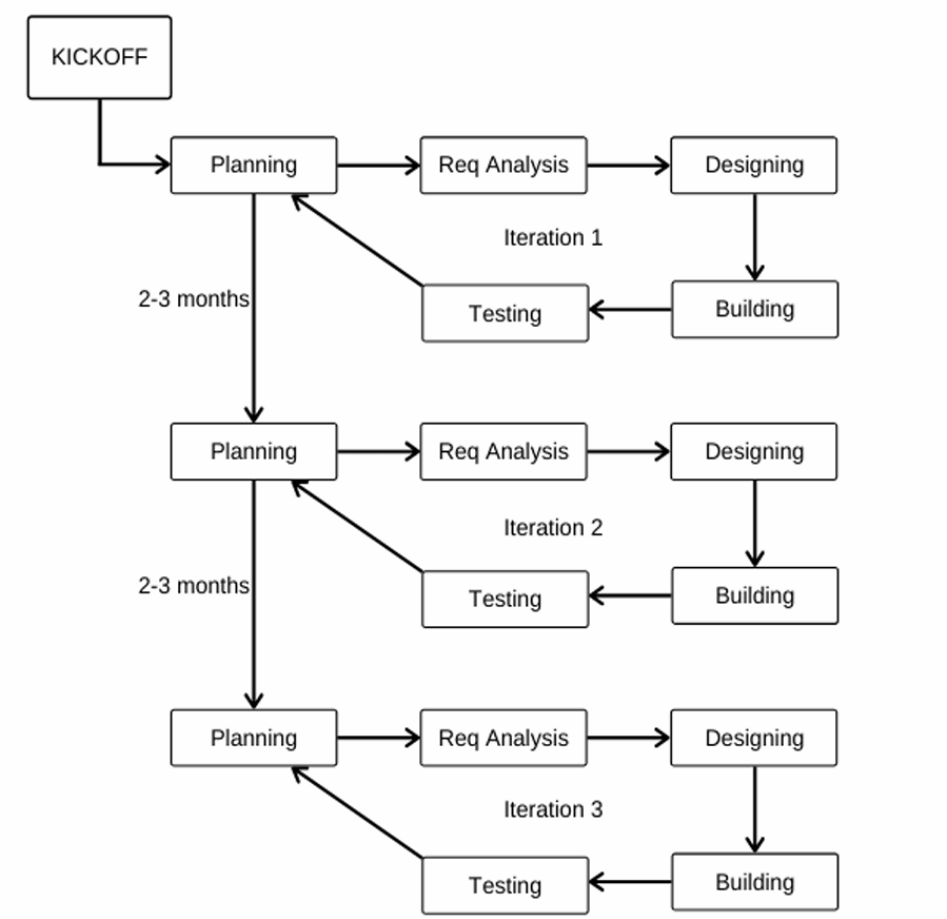


Figure 4.1: Agile Process Model

In Agile development, the software is built incrementally using short iterations of one to four weeks to align with changing business needs. Unlike other development methods in which a single pass development of six to eighteen months is undergone by predicting all the 1 requirements and risks upfront, Agile adopts frequent feedback by delivering workable products after every iteration.

In our project, we will be implementing the Scrum framework, which is currently one of the most well-known agile development life cycle models. One of the main reasons we are choosing to use the Scrum framework is its adaptability to changing requirements of the project.

All the three developers in the team will be in constant communication with each other to clarify what each developer will be doing. Every morning, a daily scrum meeting will be held where the task for the day to be done will be discussed. The tasks will be divided into time boxes (small time frames) to deliver specific features for a release. Each build will be incremental in terms of features, and the final build will hold all the required features.

4.2 System Block Diagram

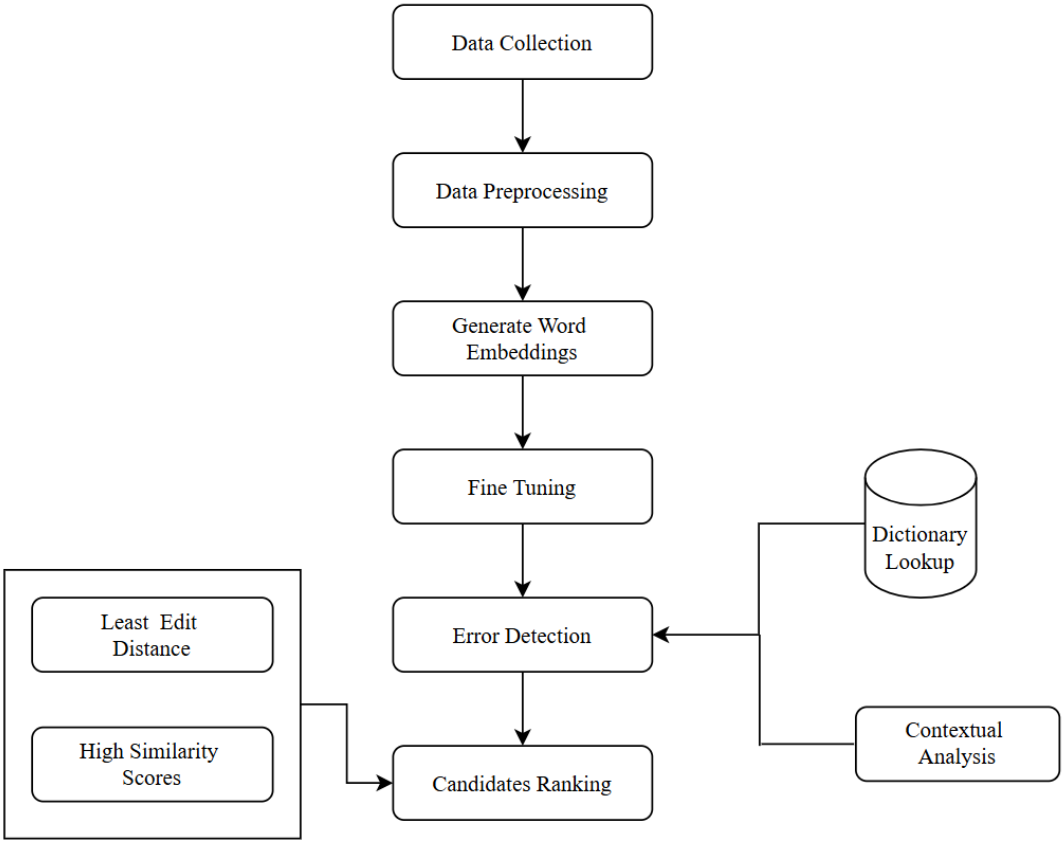


Figure 4.2: System Block Diagram

The diagram provided illustrates a system architecture for a context-based spelling checker. Let's break down each component in detail:

4.2.1 Data Collection

Data collection involves gathering a large corpus of Nepali text from various sources. This step is crucial because it ensures that the model is trained on diverse examples, capturing the full range of language use. Sources can include books, newspapers, websites, social media, and forums. For example, collecting articles from online Nepali news portals can provide formal language, while social media posts can offer more colloquial and conversational language. The goal is to build a dataset that represents different contexts and usage patterns in the Nepali language, which helps the model understand the nuances and variations.

4.2.2 Data Preprocessing

Data preprocessing is essential for preparing the raw text for model training. It involves several sub-steps:

- **Text Normalization**: This ensures consistency by converting different forms of characters into a standard form. For instance, Nepali characters can have variations due to different encoding standards, so normalization would convert all forms of ”आ” to a single standard form.
- **Tokenization**: This step splits the text into meaningful units like words, subwords, or characters. For example, the sentence "नेपाल एक सुन्दर देश हो।" is tokenized into [”नेपाल", "एक", "सुन्दर", "देश", "हो", "।"].
- **Noise Removal**: This involves removing unwanted characters, such as punctuation and special symbols, that are not useful for analysis. For example, social media texts might include ”@” and hashtags, which need to be removed.

In the paper [10], it has been tried to propose a standard of the preprocessing of Nepali corpus. So, we might be following these preprocessing pipline for cleaning the Nepali corpus we acquire.

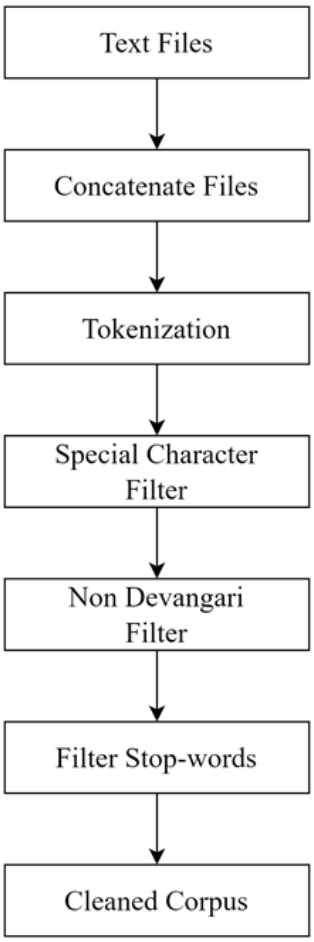


Figure 4.3: Preprocessing pipeline for Nepali Language

4.2.3 Model Training

Model training involves creating and refining models that can understand and correct spelling errors in Nepali.

Word2Vec: Word2Vec is a popular technique used for natural language processing (NLP) tasks that transforms words into high-dimensional vector representations. These vectors capture semantic relationships between words, such that words with similar meanings are located close to each other in the vector space. This model creates word embeddings, which are vector representations of words capturing their meanings based on context. By training on the collected corpus, words with similar meanings (e.g., "सुन्दर" and "राम्री" for ”beautiful”) will have similar vector representations, helping the model suggest corrections based on semantic similarity.

There are two main approaches to training Word2Vec models.They are:

Continuous Bag of Words (CBOW): In the CBOW model, the objective is to predict the target word from the surrounding context words. Given the context of words wt-2, wt-1, wt+1, wt+2 the goal is to predict wt.

The probability of the target word given the context is modeled as:

$$P(w_t|w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$$

Skip-gram: In the Skip-gram model, the objective is to predict the context words from a given target word. Given a word wt the goal is to predict the context words $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$. The probability of the context words given the target word is modeled as:

$$P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}|w_t)$$

Transformers: These are advanced models that leverage self-attention mechanisms to understand context deeply. Fine-tuning a pre-trained transformer model (such as BERT) on your Nepali corpus enables it to understand and correct spelling errors in a contextually aware manner, making it highly effective for nuanced language corrections.

4.2.4 Error Detection

Error detection involves identifying words in the text that are incorrect. This can be done using two main methods:

Dictionary Lookup: Checking if words exist in a predefined lexicon or dictionary of valid Nepali words. Words not found in the dictionary are flagged as potential errors.

Contextual Analysis: Using models to determine if a word fits within the given context. For example, the word "सुनदेर"would be flagged as incorrect in "नेपाल एक सुनदेर देश हो।" because it does not fit the context and is not a valid word.

4.2.5 Generate Candidate Corrections

Generating candidate corrections involves suggesting possible correct words for the detected errors. Two common methods are:

Embeddings: Using cosine similarity to find words with similar vector representations in the word embedding space. This suggests corrections that are semantically similar to the misspelled word.

Levenshtein Distance: Using the minimum edit distance to suggest corrections. This method finds words that are closest in spelling to the misspelled word by calculating the number of edits (insertions, deletions, substitutions) needed to transform one word into another.

4.2.6 Contextual Ranking of Candidates

Contextual ranking of candidates is crucial for selecting the most appropriate correction based on the surrounding text. This can be achieved by:

Cosine Similarity: Comparing the context vector (an average of vectors for surrounding words) with candidate vectors. The candidates with the highest similarity scores are ranked higher.

Neural Networks: Using Transformer models to rank candidates based on how well they fit into the context. For example, in the sentence "नेपाल एक सुनदेर देश हो।", "सुन्दर" would be ranked higher than "सुन्द्र" because it fits the context better.

4.2.7 Model Evaluation and Tuning

Model evaluation and tuning ensures the spell checker performs well. This involves:

Evaluation Metrics: Using precision, recall, F1-Score, and accuracy to measure performance.

Cross-Validation: Testing the model on different subsets of data to ensure robustness.

Hyperparameter Tuning: Adjusting model parameters to optimize performance. For example, fine-tuning the learning rate or the number of layers in a neural network. Example: User inputs "नेपाल एक सुनदेर देश हो।" and gets "नेपाल एक सुन्दर देश हो।".

4.3 Nepali language spelling checking

4.3.1 Nepali Language Inflection

In Nepali, part-of-speech categories include **Noun**, **Verb**, **Adjective**, **Preposition**, and **Adverb**. Nepali words within these categories are also inflected for various grammatical features, with unique rules due to Nepali's Indo-Aryan linguistic roots.

- **Nouns and Adjectives**: These are inflected based on *number* ("किताब" –> "किताबहरू") (singular or plural), *gender* (masculine or feminine), and *case* (घर"–>"घरको") . Nepali nouns and adjectives also mark *definiteness* ("यो"–>"त्यो"), using suffixes or postpositions to convey specificity.
- **Verbs**: Nepali verbs inflect primarily for *tense* (खानु"–>"खाँदैछ) (present, past, and future), *person* "खान्छु"–>"खान्छ" (first, second, third), *number* (singular or plural), and *gender*. Verbs also reflect the subject's *honorific level* ("खान्छौ"–> "खानुहुन्छ"), with distinct forms for respect and familiarity. Verb endings change according to these factors, following specific morphological rules.
- **Pronouns**: Pronouns are marked similarly, showing distinctions in *person*, *number*, *gender*, and *honorific levels* ("तिमी"–>"तपाईं").

4.3.2 Nepali Spelling Error Patterns

Nepali spelling errors arise from a range of linguistic and contextual factors. The following categories summarize key patterns that are common sources of spelling errors in the Nepali language.

Compound Words

Nepali uses compound words in1 various contexts without a standard for writing them as single units or separate terms. This lack of standardization often leads to variations in

compound word formation. For example, terms like "गृहकार्य" (grihakarya) may appear as both "गृहकार्य" and "गृह–कार्य", depending on context or individual preference.

Abbreviations

Nepali often shortens longer terms or phrases into abbreviations, leading to inconsistencies in spelling. Words such as "प्रा." for "प्राध्यापक (professor) and "डा." for "डाक्टर" (doctor) can appear differently across texts. Spelling checkers must account for such variations in order to recognize different forms of abbreviations.

Grapheme Confusion

Certain Nepali characters are visually similar, causing frequent errors in spelling due to misidentification. Examples include confusion between "स" (sa) and "ष" (ṣa), or "ब" (ba) and "व" (va). These similarities lead to accidental or habitual substitutions, especially in handwritten and digital texts.

Redundancy in Graphemes

Nepali has phonetic overlap in some characters, leading to redundancy where the same sounds can be represented by different graphemes. Examples include the characters "श" (sha), "ष" (ṣa), and "स" (sa), which can all represent similar sounds in certain dialects or regions. Spelling errors often arise from this redundancy, as writers may interchange these letters.

Loanword Adaptation

Nepali incorporates many terms from other languages, especially English and Sanskrit. Transcription methods for these loanwords are not standardized, resulting in multiple spellings. For example, "कम्प्युटर" (computer) and "कमप्युटर" represent the same word but are spelled differently. Such discrepancies often lead to inconsistent spelling.

Dialectal Variation

Nepali has significant dialectal differences that influence spelling. Words may be spelled as they are pronounced in various regions, leading to variations. For instance, "छैन" (chhaina) for may appear as "छैना" in some dialects. These regional differences are a source of spelling variation.

Mistyping

Typographical errors are common due to the use of various input methods on keyboards. Keyboards with fewer keys than required symbols or varied layouts in different typing software often result in mistyping. For instance, "स" (sa) might be typed as "ष" (ṣa) if the typist uses an unfamiliar keyboard layout or experiences a “shift-slip.”

By examining these error types, this study aims to create a comprehensive model for recognizing and correcting such errors in Nepali text processing systems.

4.3.3 Confusion sets

Confusion sets, made up of frequently mistaken characters or words, are essential for identifying spelling mistakes in text. In our system, we construct confusion sets by focusing on characters or words with similar pronunciations, enhancing error detection accuracy by flagging potentially incorrect or easily confused terms. We also use statistical analysis to consider characters with high occurrence rates or a tendency to be mistaken for others. Characters commonly misspelled or frequently confused in writing are prioritized within these sets.

For the Nepali language, confusion sets may include characters with similar pronunciations, such as homophones, nasal vs. non-nasal sounds, and retroflex vs. non-retroflex sounds. For instance, characters like "ष", "स", "श" can be grouped based on similar pronunciations. Characters prone to confusion due to visual similarity or frequency—like "ि" and "ी", or dirgha ukar and hraswa ukar—are also included in these sets.

Confusion sets for words of the same or differing lengths include frequently confused words. Same-length sets consist of words with identical lengths but distinct spellings or meanings, such as "फुल" and "फूल". These sets help in identifying mistakes when similar-length words are incorrectly swapped.

4.4 Datasets Error Issues

There are various types of errors found in this dataset. Here we have categorized the errors into following types:

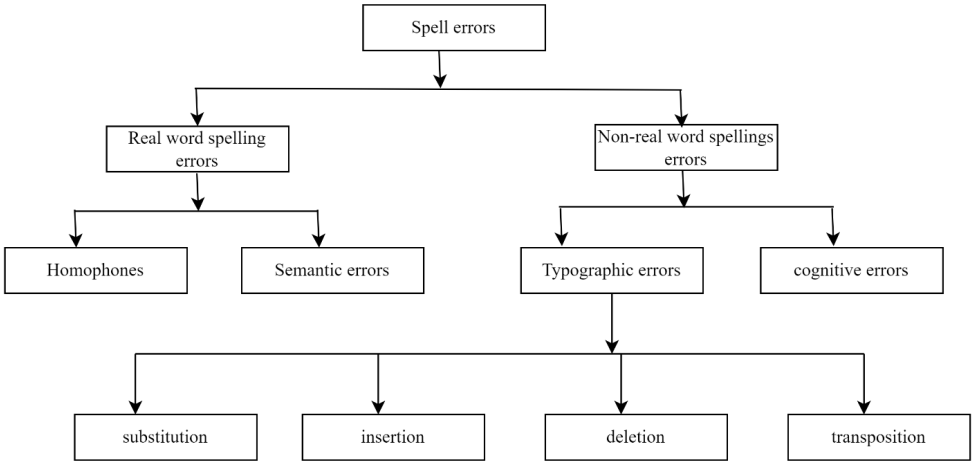


Figure 4.4: Error category

4.4.1 Real word spelling errors

Real-word spelling errors occur when a correctly spelled word is used incorrectly in a sentence, leading to contextually incorrect usage. These errors are challenging because the misspelled word is still a valid word in the language but does not fit the context.

Homophones

Homophones are words that sound the same when pronounced but have different meanings and spellings. Homophones can often lead to spelling errors, especially in languages like Nepali where phonetic similarities are common. These errors can be particularly challenging because even though the words are phonetically identical, they have distinct meanings and are used in different contexts. For example: फूल (phool) vs. फुल (phul): Semantic Errors Semantic errors occur when a word is used in the correct spelling but in an incorrect context, resulting in a sentence that does not make sense or conveys a wrong meaning. These errors are related to the meaning of the words and their appropriate use in context. For example: Incorrect: उ मिठो घोडा खान्छ। (He eats a sweet horse.) Correct: उ मिठो खाना खान्छ। (He eats delicious food.) Explanation: The word "घोडा" (horse) is used instead of "खाना" (food).

4.4.2 Non-Real word spelling errors

Non-word spelling errors occur when the misspelled word is not a valid word in the language. These errors are easier to detect because the resulting string is not in the dictionary of valid words.

Typographic Errors

Typographic errors, also known as typos, are mistakes made during the typing process. These errors can occur in various forms, such as substitution, insertion, deletion, and transposition.

Substitution Errors

A substitution error occurs when one character is replaced by another. Example: Incorrect: "केहि" (instead of "केही") Correct: "केही"

Insertion Errors

An insertion error occurs when an extra character is added.

Deletion Errors

A deletion error occurs when a character is omitted.

Transposition Errors

A transposition error occurs when two adjacent characters are swapped.

4.5 Encoder-decoder model

The Transformer Model is perhaps the most popular ML model of Encoder-Decoder models. As the name implies, these models can be divided into two parts - an encoder and a decoder.

While the Transformer Model is the popular one, it was not the first ML model to incorporate an Encoder - Decoder architecture. For example, the paper “Sequence to Sequence learning with Neural Networks” [14] in 2014 introduced an Encoder-Decoder architecture for translation tasks based on the RNN model. Translation is an ideal task

for studying seq2seq architecture

Looking at the Encoder-Decoder model as a blackbox system, it can be defined as a system that takes a sequence (say $\vec{F} = (f_0, f_1, f_2, ..., f_{n-1})$) as input and a sequence (say $\vec{E} = (e_0, e_1, e_2, ..., e_{n-1})$) as output .

Dividing a high level view into encoder and decoder. The encoder process converts (or encodes) the natural language sequence of input data into a vector. This encoded vector data is then the input provided to the decoder process. The decoder processes (or decodes) the encoded vector representation into a sequence of natural language data. This is the output sequence.

In a RNN model, each layer of the encoder layer ,say $F(\vec{k}, \vec{h})$, takes input vector $\vec{x}$ and a hidden state $\vec{h}$. The input vector is used for encoding and understanding the input text sequence word by word. The hidden state $\vec{h}$ is necessarily to understand the dependency and relationship of the given input word to other input words present in the sequence. This is often called Context.

Let $F(f_i, h_i)$ be the i^{th} layer of encoder RNN. The $F(f_0, h_0)$ takes the text input and generates an output vector and hidden state vector h1. The $F(f_1, h_1)$ takes h_1 as the input and calculates the hidden state vector h_2 and so on repeatedly for the entirety of F.

The decoder takes the output of the latest encoder layer as its input and has to output a sequence of words such as possess a categorical distribution over the target vocabulary.

Vocabulary is the set of words that the mode understands. If the encoder-decoder model is trained for NLP tasks over a single language, the input and output vocabulary may be the same. If, however, it is trained for multiple languages, like translation tasks, the source vocabulary associated with the encoder and the target vocabulary associated with the decoder are different.

The decoder process is fed not only with the output of the encoder process but also with

the output label sequence during the training phase. But this output label sequence will not be available for the inference. So, instead all the words after the specific words are masked and forced to guess thereby simulating the action during the inference action. This is known as Masked Language Modelling.

The output expected of the encoder-decoder model is a sequence of words and not (generally) a single word. So, while selecting the most probable words at each position, greedy decoding is used to reduce complexity. But the limitation of such technique is that though the words in the word sequence $W = w_0, w_1, w_2, , w_{n-1}$ may be the most probable in their respective positions, some other sequence say $T = t_0, t_1, t_2, , t_{n-1}$ may be the most probable when taken all together.

To solve this problem, the concept of beam search came to be implemented during the interface. Here, several different sequences are simultaneously kept track of and the sequence with the highest cumulative probability is provided . A hyper-parameter (often called beam width or number of beams) is specified for how many sequences are tracked.

Limitations of RNN based Encoder-Decoder Model

In RNN -based encoder-decoder models, there is only one hidden state for each layer of encoder.So, there naturally is going to be a limitation as to how much context can be understood. In addition to this, the input sequence is given sequentially from the first to the last. Due to this, for any given word f_i in the sequence only the context from words that came before the word f_i can be associated.

To solve this Bidirectional Encoders were introduced by the paper "Neural Machine Translation by Jointly Learning to Align & Translate" in 2016 [15]. Here, two hidden states are associated with each word of the input sequence : one going from the beginning to the end and another from the end to the beginning. Once all the words have been transversed, every word possesses two hidden state vectors associated with it. So, for any input sequence $\vec{F} = (f_0, f_1, f_2, ..., f_{n-1})$, the first RNN would go from $f_0 -> f_1 -> f_2 -> ... -> f_{n-1}$ generating a hidden state vector set $\vec{H} = (h_0, h_1, h_2, ..., h_{n-1})$ and the second RNN would go from $f_{n-1} -> f_{n-2} -> f_{n-3} -> ... f_2 -> f_1 -> f_0$ generating

a hidden state vector $\vec{J} = (j_0, j_1, j_2, ..., j_{n-1})$ Once both of the hidden vectors are ready they are concatenated into one matrix.

4.5.1 Attention

While the input encoding from the bidirectional encoders represent an advancement, they also introduce a problem. As described above, the number of hidden states associated with any input sequence will vary depending upon the number of words in the sequence. Naturally, then the variance in the hidden state matrix introduces complexity when provided to the decoder. Thus, the attention mechanism was developed to overcome the problem of variable hidden state matrix.

Attention mechanism mainly overcomes the problem associated with the variable hidden state by downsizing the set of hidden states into a weighted sum of all the constituent vectors. The algorithms used for the weighted sum differ but one of the more popular ones is the dot product attention mechanism where a vector dot product is taken between the constituent vectors.

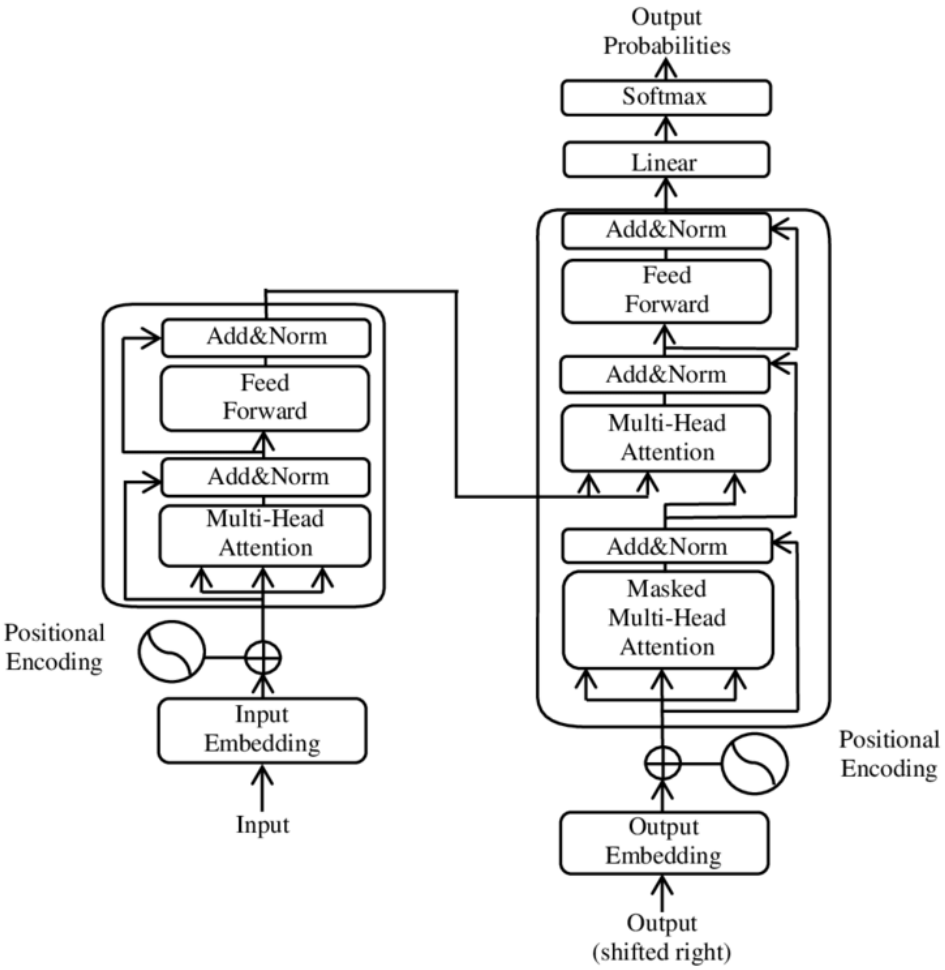


Figure 4.5: Encoders and Decoders in Transformer

Transformer, which was introduced by Vaswani et al in 2017 [16] , is an Encoder-Decoder model with multi-headed self attention mechanism and parallel input processing.

Even after the introduction of Bidirectional Encoder, one of the problems faced by Encoder-Decoder models is the difficulty in parallelizing input processing. Due to the lack of parallelization, higher amounts of resources are required for both pre-training and fine tuning and also more time is required to do so.

The introduction of attention based self attention makes the reliance on RNN entirely redundant as well as to parallelize the input processing. This removes the bottleneck in training and makes the transformer model more efficient.

As with previous encoder-decoder models, the ultimate goal of the transformer model is to take any input sequence $\vec{F} = (f_0, f_1, f_2, ..., f_{n-1})$ and to convert it into the required output sequence $\vec{E} = (e_0, e_1, e_2, ..., e_{n-1})$ according to the type of the task for which the model is trained and fine tuned.

All encoders and all decoders are respectively identical to each other but have their own respective weights and parameters that may or may not be the same.

Encoder

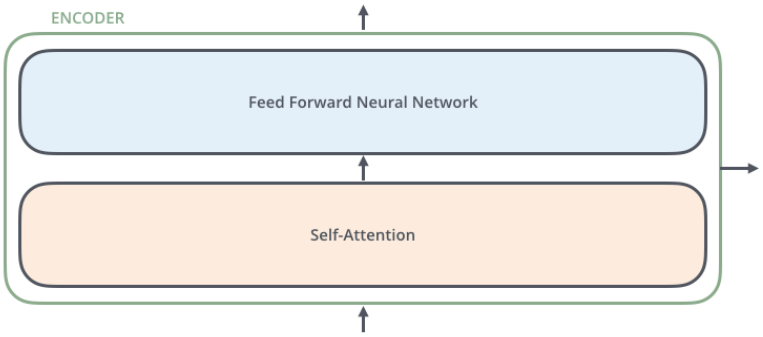


Figure 4.6: Encoder layer

Each encoder is composed of two parts- a self attention layer and a feed forward network layer. Each of these sub-layers can be further divided. 4.7

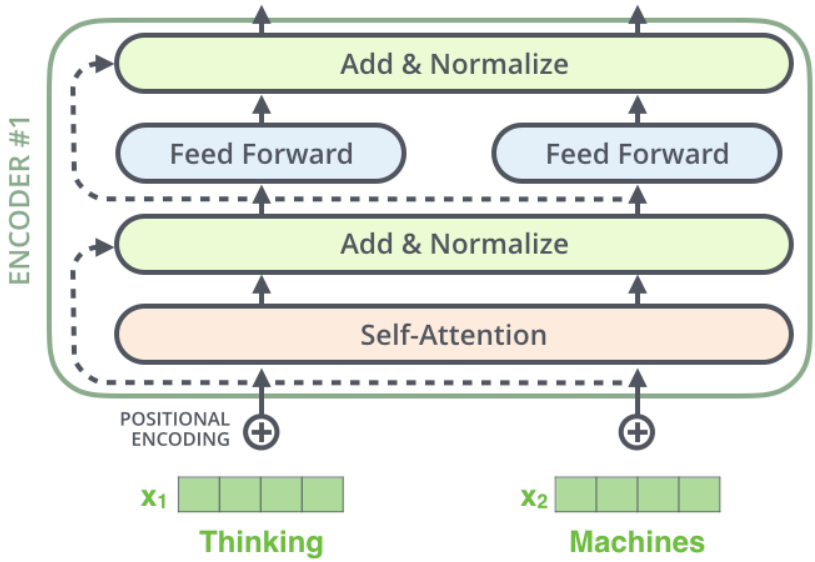


Figure 4.7: Encoder Sublayers

During the training process, the input sequence is provided to the first encoder. The input first passes through the attention layer. The output of the attention layer is passed to the add and normalization. Here the input to the attention and its output are added and layer normalization is performed.

The output thus obtained is sent to the feedforward layer. Here too, similar steps are taken for add and normalization.

Decoder

Each decoder in the decoder process can be divided into two sub layers: Attention Mechanism and Feed Forward.

A difference from the encoder is that the decoder layer contains an additional masked multi headed attention sub layer as well as containing the usual attention sub layer. The purpose of the masked multi-headed sublayer has been explained above.

While the first (i.e. masked) attention sublayer is used for the processing of encoded output sequence during training, the second (i.e usual) attention sublayer is used for encoder-decoder attention. This is used to communicate effectively the attention

understood by the encoder from the given input sequence.The other processing is the same as that of the encoder process.

Positional Encoding

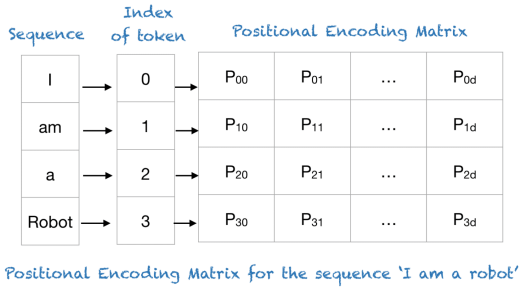


Figure 4.8: Positional Encoding Example

In Nepali, as in many other languages, the order of words in a sentence matters and the order of words can sometimes change the meaning of the sentence drastically. So, it is crucial that the position of the words be taken into consideration when they are encoded. In earlier vector representations, words had the same vector representation no matter the position.

Positional Encoding encodes the text sequence in a vector such that each position in the sequence is assigned a unique representation. Such representations are used instead of other, more simpler, parameters like index number. Index number can increase with the increase in sequence length.

In Transformer based models too, simple vector representations of the input sequence is positionally encoded. This positionally encoded vector representation is the input provided to the first encoder layer.For any input sequence of length L, the following formula is used to calculate the positional encoding of any k^{th} object:

$$P(k, 2i) = \sin(\frac{k}{n^{\frac{2i}{d}}}) \tag{1}$$

$$P(k, 2i+1) = \cos(\frac{k}{n^{\frac{2i}{d}}}) \tag{2}$$

where:

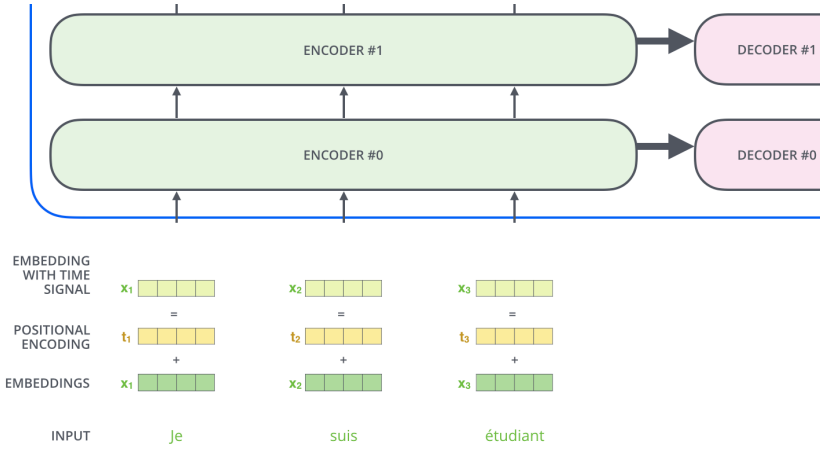


Figure 4.9: Input to Encoder from Positional Encoding

k= position of the object within the input sequence. $0 \leq k \leq \frac{L}{2}$

d= dimension of output embedding space

P(k, j)= Position function for mapping a position k in the input sequence to index (k,j) of the positional matrix

n= user-defined scalar

j= used for mapping to column indices $0 \leq i \leq \frac{d}{2}$ where a single value of i maps to both sine and cosine functions.

4.5.2 Mathematics of Attention Mechanism

As ML models cannot understand natural language texts by themselves, the input text is first encoded into suitable vector representation. Suitable techniques are used for this representation. Each of the words are encoded by a vector of the same size.

नेपाल as x_1

देस as x_2

हो as x_3

Here,x_1, x_2, x_3 represent the vector embedding of the corresponding words.How long the list $X = \{x_1, x_2, x_3, .., x_{n-1}\}$ can be depends upon the value of n.'n' is a hyperparameter to be defined that should be greater than the number of words in the longest sentence present in the dataset.

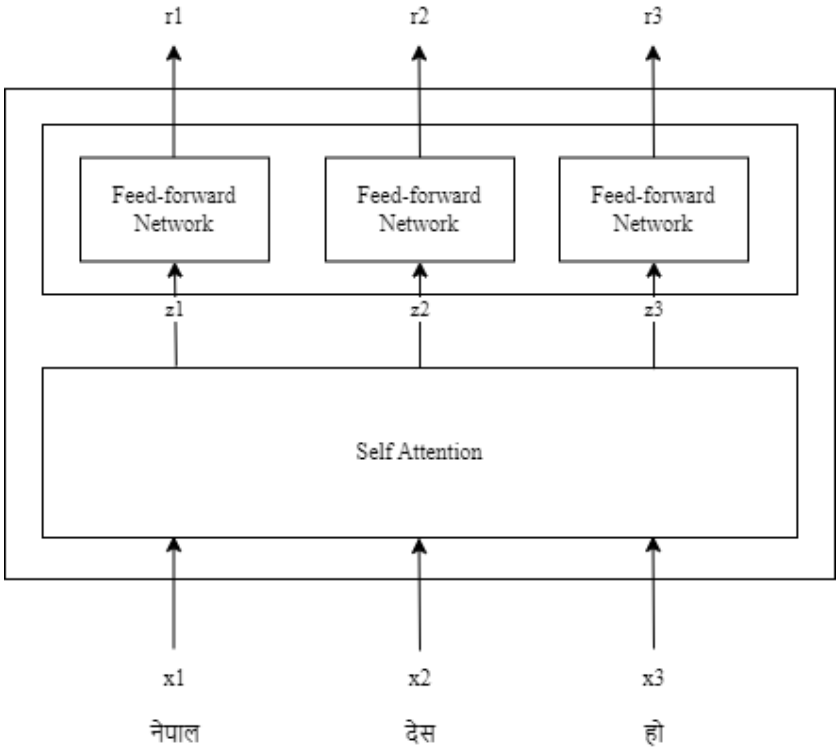


Figure 4.10: View of Encoder Layer

Here, a representation of the first encoder layer is shown. Here, x_i represents the i^{th} embedding of the word. z_i represents the i^{th} output of the self attention sub layer, which is also the input of the feed forward neural network. r_i represents the i^{th} output vector of the feed forward network and thus also the overall output of the first encoder layer.

Here, the x_i is the embedding of the input sequence and is the input to the self attention layer because it is the first encoder layer. For every other encoder layer, the input is the output of the previous encoder layer and so on until the input passes to all the encoder layers present.

For every input embedding x_i, there are 3 vectors associated with it. These are called query vector, key vector and value vector. These are represented by q_i, k_i and v_i respectively.These vectors are created by the multiplication of weights at each layer to the input embedding (i.e query weight, key weight and value weight). Let these weights be represented by w^Q, w^K and w^V respectively. These weights are initialized randomly at the beginning of the training process. Finding the suitable values of these weights is

the purpose of the training process. If x_i is any input vector embedding:

$q_i = w^Q x_i$

$k_i = w^K x_i$

$vi = w^V x_i$

The transformer model uses a dot product for calculating self-attention. For each word a score is calculated by the dot product between the query vector of that specific word and the key vector of every other word in the sequence. Using the sentence "नेपाल देस हो |" as an example with w^Q, w^K, w^V as scalar weights, the self attention of word 1 (नेपाल) can be calculated as :

Table 4.1: Calculation of Scores

Input	नेपाल	देस	हो
Embedding	x_1	x_2	x_3
Query	q_1	q_2	q_3
Key	k_1	k_2	k_3
Value	v_1	v_2	v_3
Scores	q_1k_1	q_2k_2	q_3k_3

The values of score can be very high due to it being a scalar product of two vectors whose value may fluctuate highly. High values of score may also lead to processing overheads. To stabilize these gradients, the value of score is divided by the square root of the dimension of the key vector. Let this be represented as $\sqrt{d_k}$.

To calculate the probability associated with each word from the divided score value, softmax function is used. Softmax function , which converts a vector of size N to probability distribution of N possible outcomes, is defined as:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{3}$$

where,

σ = softmax

$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

k = number of classes in multiclass classifier

e^{z_i} = standard exponential function of output vector

The sum of all the softmax values, as probability , is always 1. The outcome of the attention is the sum of the multiplication of each corresponding value matrix and softmax value.

Table 4.2: Calculation of Softmax value

Input	नेपाल	देस	हो
Embedding	x_1	x_2	x_3
Query	q_1	q_2	q_3
Key	k_1	k_2	k_3
Value	v_1	v_2	v_3
Scores	s_1	s_2	s_3
Divide by $\sqrt{d_k}$	$\frac{s_1}{\sqrt{d_k}}$	$\frac{s_2}{\sqrt{d_k}}$	$\frac{s_3}{\sqrt{d_k}}$
Softmax	$\sigma(\frac{s_1}{\sqrt{d_k}})$	$\sigma(\frac{s_2}{\sqrt{d_k}})$	$\sigma(\frac{s_3}{\sqrt{d_k}})$
Softmax $*$ value	$\sigma(\frac{s_1}{\sqrt{d_k}}) * v_1$	$\sigma(\frac{s_2}{\sqrt{d_k}}) * v_2$	$\sigma(\frac{s_3}{\sqrt{d_k}}) * v_3$
Sum	sum_1		

In terms of matrix calculation, attention can be calculated directly as:

$$\text{Self} - \text{attention} = \sigma(\frac{QK^T}{\sqrt{d_k}})V = Z(\text{say}) \tag{4}$$

This calculation is just for a single word in a single layer. In any transformer model, there are multiple attention heads for any single word. Thus, for any single word there will be 'n' Z sum values if there are 'n' attention heads as $Z_0, Z_1, Z_2, ..., Z_{n-1}$

The vectors thus found are concatenated with each other as:

$$Z_{tot} = \text{concat}(Z_0, Z_1, Z_2, , Z_{n-1}) \tag{5}$$

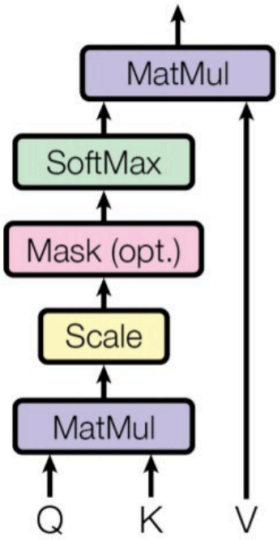


Figure 4.11: Attention Calcualtion Steps

s And multiplied with the weight matrix W^O that was trained with the model. So,

$$Z = Z_{tot} * W^O \tag{6}$$

Thus, attention heads from 0 to n-1 are calculated for each n inputs. Before sending the output to feed forward network, layer normalization is performed as:

$$Z_{normalized} = \text{layerNorm}(X + Z_{concat}) \tag{7}$$

4.5.3 Sample Calculation

The relative importance of different words in a string and their context can be weighed efficiently using the self-attention mechanism. In the self-attention mechanism used in Transformer models, three vectors: Query(Q), Key(K) and Value(V) are important. For more efficient calculation, the operations between various vectors are performed using metrics.

To understand the calculation better, let us consider the values of the three vectors as:
$Q = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$

$$K = \begin{bmatrix} 4 & 4 & 6 \\ 5 & 5 & 4 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.11 & 0.22 \\ 0.44 & 0.55 \end{bmatrix}$$

In Self-Attention mechanism, the dimensionality of the Query and Key metrics should always match, but the dimensionality of the Value matrix may be different.In the above example, both the Query and Key metrics are $2 * 3$ metrics but the Value matrix is $3 * 2$ matrix.

In the figure 4.11, the first step in attention calculation is matrix multiplication. To achieve a dot product result with matrix multiplication, we take the transpose of the second matrix. The overall formula for self-attention in using matrix calculation is :

$$\text{Self} - \text{attention} = \sigma(\frac{QK^T}{\sqrt{d_k}})V = Z(\text{say}) \tag{8}$$

Calculating QK^T

$$QK^T = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 & 4 & 6 \\ 5 & 5 & 4 \end{bmatrix}^T$$

$$QK^T = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 & 5 \\ 4 & 5 \\ 6 & 4 \end{bmatrix}$$

$$QK^T = \begin{bmatrix} 30 & 27 \\ 30 & 27 \end{bmatrix}$$

d_k is the dimension of the key matrix. Here, the dimension of the key is 2. So, $d_k = 2$

So, scaling down the multiplied matrix by dividing it with $\sqrt{d_k}$ $\frac{QK_T}{\sqrt{d_k}} = \frac{\begin{bmatrix} 30 & 27 \\ 30 & 27 \end{bmatrix}}{\sqrt{2}}$

Applying softmax function: $\sigma(\frac{QK^T}{\sqrt{d_k}}) = \begin{bmatrix} 0.476287 & 0.023713 \\ 0.476287 & 0.023713 \end{bmatrix}$

Multiplying by Value (V): $A = \sigma(\frac{QK^T}{\sqrt{d_k}}) * V$

$$A = \begin{bmatrix} 0.476287 & 0.023713 \\ 0.476287 & 0.023713 \end{bmatrix} * \begin{bmatrix} 0.11 & 0.22 \\ 0.44 & 0.55 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.06282 & 0.11782 \\ 0.06282 & 0.11782 \end{bmatrix}$$

Thus the self-attention is calculated for any one attention head at any one layer. There are multiple layers of encoder and decoders in any Transformer based system and correspondingly the number of attention heads also is large.

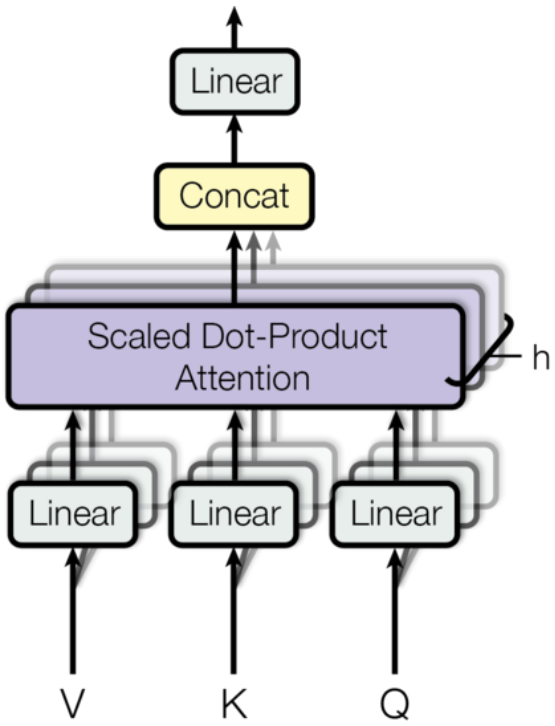


Figure 4.12: Multi Headed Attention

The attentions of different attention heads are concatenated before providing to the Linear Optimization layer as described above.

4.5.4 Feed-forward Neural Network

In order to introduce non-linearity into the model and to more accurately mimic the relationship between words in the sentence, a feed-forward Neural Network is used. In the feed-forward Neural Network.In each feed-forward neural network, there are two linear transformations with an activation function between them. This activation function helps to introduce non-linearity in the model. Rectified Linear Unit (ReLU) is one of the most common activation functions and is generally used in Transformer based models.

The ReLU activation function can be represented mathematically as :

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{9}$$

4.5.5 Optimizer

The main task of the any ML model in the training (or fine tuning) process is to minimize the losses. In this minimizing process, weights of the model have to be changed so that there are lower losses. During this, optimizers play an important part. Optimizers are algorithms that update the weights of the model iteratively to reduce the loss function. The loss functions used in the Transformer based models have been described above. In simplified terms, the tasks performed by optimizers are:

- Forward Pass
- Loss Calculation
- Backward Pass
- Parameter Update
- Repeat Iteratively

Adam Optimizer

Adam stands for Adaptive Moment Estimation with Weight Decay Optimizer. It is a popular optimizer for models using transformers architecture and represent development over previous optimizing algorithms. AdamW optimizer adapts the learning rate for each parameter individually. If the parameters have larger gradient in the past, they receive lower learning rate and vice versa.

For any parameter w_i at time t, the update rule is specified in AdamW as:

$$(w_i)_t = (w_{i-1})_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{(\hat{v_i})_t} + \varepsilon} \tag{10}$$

4.6 Models

4.6.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google. BERT was created and published in 2018 by Jacob Devlin and his colleagues from Google. A 2020 literature survey concluded that ”in a little over a year, BERT has become a ubiquitous baseline in NLP experiments”, counting over 150 research publications analyzing and improving the model.

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. BERT’s goal is to generate a language model, only the encoder mechanism is necessary. BERT uses two training strategies.

Masked Language Modeling (MLM)
Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

- Adding a classification layer on top of the encoder output.
- Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
- Calculating the probability of each word in the vocabulary with softmax.

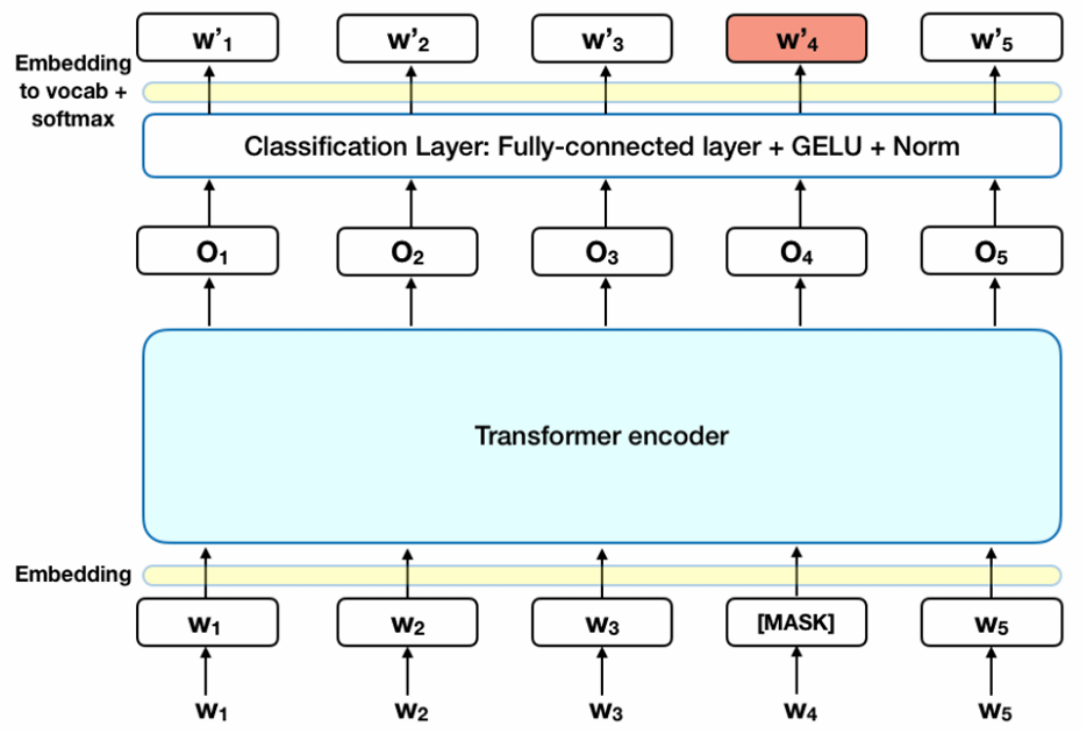


Figure 4.13: Masked Language Modeling

The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness.

Next Sentence Prediction (NSP)

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence. To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model.

- A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token

is inserted at the end of each sentence.

- A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
- A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

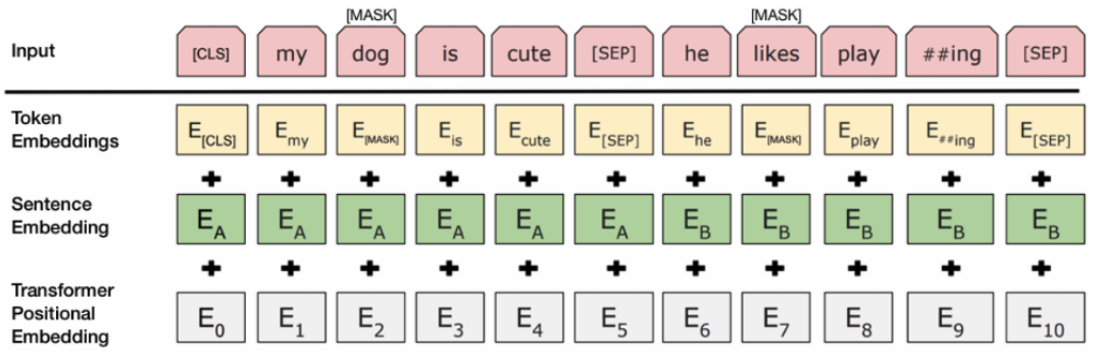


Figure 4.14: Next Sentence Prediction

To predict if the second sentence is indeed connected to the first, the following steps are performed:

- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
- Calculating the probability of IsNextSequence with softmax.

4.6.2 RoBERTa

RoBERTa (Robustly Optimized BERT Approach), an optimized version of BERT, builds on its architecture while addressing some of its limitations to enhance performance. It removes the Next Sentence Prediction (NSP) objective, as studies found it did not significantly benefit the model. Instead, RoBERTa focuses solely

on the Masked Language Modeling (MLM) objective with dynamic masking, where different tokens are masked during each epoch, exposing the model to more diverse training examples. Additionally, RoBERTa is trained on significantly larger datasets with longer sequences and larger batch sizes. These optimizations allow it to outperform BERT on various NLP benchmarks. While the architecture remains the same as BERT, RoBERTa's robust training strategy makes it more effective in capturing nuanced language representations.

These differences in training objectives and optimizations make BERT and RoBERTa complementary for exploring tasks like context-based spelling correction, where understanding sentence-level context is crucial.

4.7 UML Diagrams

4.7.1 Use Case Diagram of the Overall System

A use case diagram provides a visual representation of the interaction of the actors with the system to accomplish a particular goal. It is a high level overview of the system functionality and the ways in which a user can interact with the system. It shows what a user/system can do without including the implementation details.

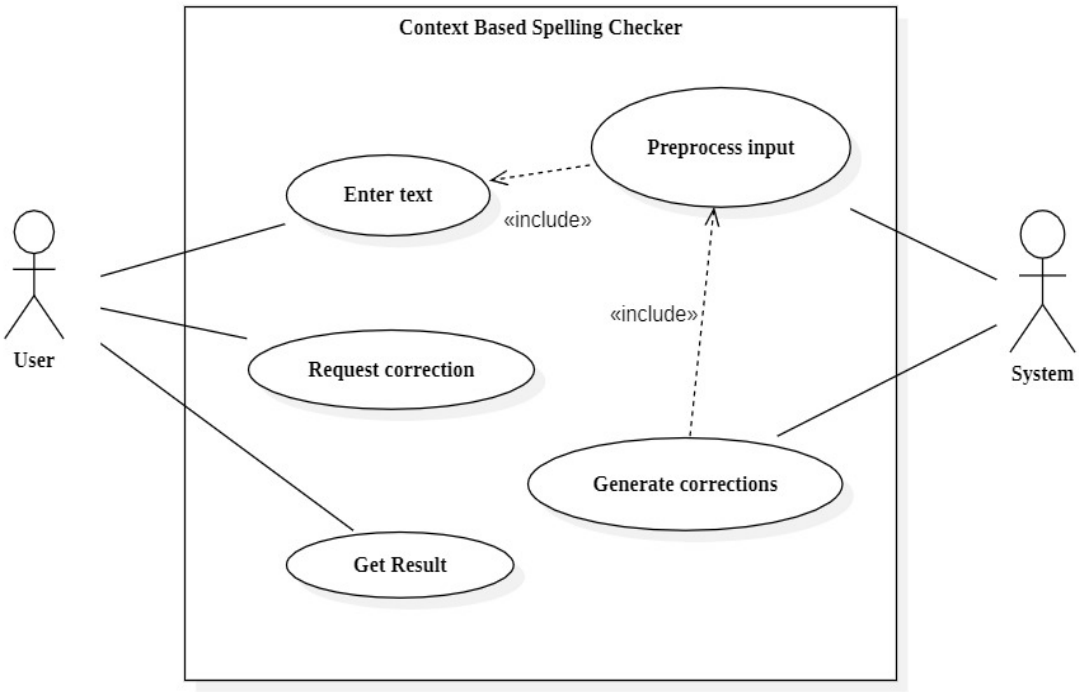


Figure 4.15: Use Case Diagram of the Overall System

User entity

A user entity used the correction system. The user is able to select a model from the available list. The user can enter the Nepali text which has to be checked by the system. The user need to trigger the system to check the spelling on the input text and the user gets the result from the system, which is the spelling corrected text.

System entity

A system entity is the vital entity Nepali spelling system who is actually responsible for correcting the spelling on the input text. It is triggered by the user, the content in the text field is preprocessed and result is generated by the system. The output is converted to appropriate format before giving it to the user as result.

4.8 Metrics

4.8.1 Accuracy

Accuracy is a widely used metric to calculate how often the ML model gives correct outcomes. It is the ratio of correctly identified outcomes (or correct predictions) to the total number of outcomes (or total predictions). So,

$$\text{Accuracy} = \frac{\text{No. of correct predictions}}{\text{Total no. of predictions}} \tag{11}$$

If TP, TN, FP, FN represent the True Positive, True Negative, False Positive and False Negative outcomes respectively , then accuracy can be defined as :

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{12}$$

4.8.2 Loss

The **loss** metric is a measure of how well the model's predictions match the expected outputs during training. In this project, the loss is computed using the Cross-Entropy Loss, which is widely used for tasks involving classification or token prediction. It is given by the formula:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}),$$

where N is the number of samples, C is the number of classes, y_{ij} represents the true label (one-hot encoded), and $\hat{y}_{ij}$ is the predicted probability for the j-th class. Lower loss values indicate better model performance on the given task.

4.8.3 Perplexity

The **perplexity** metric is a widely used measure for evaluating language models. It quantifies how well the model predicts a sequence of words and is defined as the exponential of the average negative log-likelihood of the predicted tokens:

$$\text{Perplexity} = \exp\left(-\frac{1}{T} \sum_{t=1}^{T} \log P(w_t | w_1, w_2, \ldots, w_{t-1})\right),$$

where T is the length of the sequence, and $P(w_t|w_1, \ldots, w_{t-1})$ represents the probability assigned by the model to the t-th token given the preceding tokens. A lower perplexity indicates that the model is better at predicting the next word in a sequence, thus demonstrating higher language modeling quality.

5 IMPLEMENTATION DETAILS

5.1 Word2Vec

5.1.1 Dataset Preparation

This dataset consists of a collection of Nepali news articles categorized into 20 distinct categories. The data has been extracted from the most trusted Nepali newspapers, such as Kantipur and Gorkha Patra, with a total of over 73,000 newspaper articles.

Table 5.1: Data Statistics

Category	Number of docs
Agriculture	200
Automobiles	246
Bank	617
Blog	259
Business	307
Economy	600
Education	185
Employment	304
Entertainment	634
Health	180
Interview	330
Literature	251
Migration	111
Opinion	500
Politics	550
Society	353
Sports	700
Technology	118
Tourism	265
World	313

The dataset also includes texts extracted from various Nepali novels and literature, adding depth and variety to the linguistic resources available for analysis and model training.

We performed several key operations to clean and prepare the dataset. These included tokenization, normalization, and filtering out non-Devanagari characters and special symbols. A sample of the dataset is given below:

S.N.	Content

1	{
	”newsId”: ”N1381600368”,
	”newsSource”: ” ०७४/७५ को नीति तथा कार्यक्रम प्राथमिकतामा उच्च आर्थिक वृद्धि कैलाली काँग्रेस – उम्मेदवार छनोटमा विवाद धनगढी, जेठ १२ गते । धनगढी उपमहानगरपालिकाका विभिन्न वडाध्यक्षका उम्मेदवार छनोटको विषयलाई लिएर नेपाली काँग्रेस कैलालीमा विवाद चुलिएको छ । ”,
	}

2	{
	”newsId”: ”P656291723”,
	”newsSource”: ”https://ekantipur.com/”,
	”newsText”: ”करिब ४ सय वर्षपछि बृहस्पति र शनि ग्रह एकआपासमा धेरै नजिक देखिए गोविन्द पोखरेल काठमाडौं — सन् १६१० मा खगोलशास्त्री ग्यालिलियो गेलिलीले बृहस्पति गृहका चार वटा चन्द्रमा पत्ता लगाए ।”,
	}

Table 5.2: Sample dataset of un-preprocessed Nepali News Corpus

Preprocessing Results

- Before Preprocessing: The corpus contained 6,961,006 words.
- After Preprocessing: The number of words was reduced to 4,470,401.

Vocabulary

- Before Preprocessing: The vocabulary size was 207,458 unique words.
- After Preprocessing: This number remained the same, indicating that the preprocessing focused more on cleaning and reducing the total word count rather than altering the vocabulary.

Number of Sentences

- The corpus was segmented into 492,872 sentences.
- The preprocessing steps significantly reduced the size of the dataset, focusing on retaining only the most relevant and clean text data. This reduction ensured that the dataset was well-prepared for training the context-aware spelling correction model.

Word Cloud

A word cloud was generated from the dataset to visualize the most frequent words, providing an overview of the dataset's content. This visualization helped in understanding the distribution and frequency of words within the dataset. The preprocessing of the dataset involved several steps to ensure the data was clean and suitable for use in the spelling correction tool. This included tokenizing the text into sentences and words, filtering out special characters not used in Nepali, and removing non-Devanagari characters using regular expressions. Additionally, stop words were filtered out, resulting in the removal of 255 common stop words. The preprocessing also included creating word embeddings using Word2Vec, which transformed words into high-dimensional vector representations, capturing semantic relationships between words.

5.1.2 Word2Vec Parameters

Table 5.3: CBOW Parameter Specification

Dimension	300
Architecture	CBOW
Epochs	15
Window	10
Minimum count	2
Negative Sampling	15

5.1.3 Sentence Correction

We used the Word2Vec model to convert words into high-dimensional vector representations. We then trained the Word2Vec model on a large corpus of Nepali text to capture the semantic relationships between words. The model placed similar meanings close to each other in vector space.

Candidate Generation

We began by identifying the erroneous word in the text. This word, along with a comprehensive vocabulary of correct words sourced from a dictionary or a pre-built corpus, formed the input for our candidate generation process.

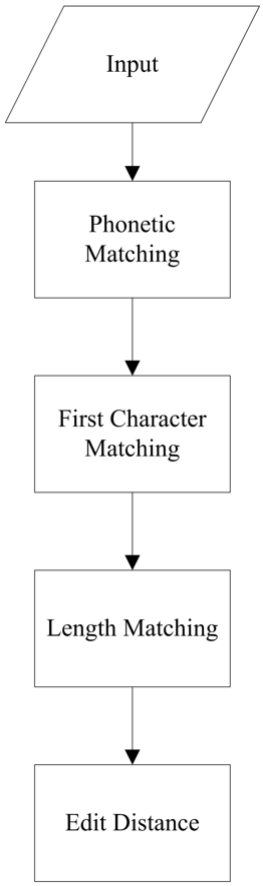


Figure 5.1: Candidate Generation

Phonetic Matching

To find words that sound similar to the erroneous word, considering phonetic similarities often result in similar spelling errors, we converted both the erroneous word and all vocabulary words into their phonetic codes using a phonetic algorithm specifically adapted for the Nepali language. This algorithm mapped characters to their phonetic equivalents, creating a phonetic representation of each word. By comparing these phonetic codes, we could identify words that sounded similar to the erroneous word, narrowing down our list of potential corrections.

First Character Matching

To quickly filter out unlikely candidates by ensuring the initial letters match. We checked if the first character of the erroneous word and matched the first character of each candidate word from the vocabulary. This step reduced the pool of potential corrections, focusing only on those words that shared the same initial letter as the erroneous word.

Length Matching

To further refine the list of candidate corrections by considering word length. We compared the length of the erroneous word to the lengths of the candidate words. Words of the same or similar length (within a reasonable range) were considered more likely candidates. This helped to exclude words that were too short or too long to be plausible corrections.

Edit Distance Calculation

To measure how similar the erroneous word is to each candidate word based on the number of changes required to transform one into the other. We calculated the edit distance (Levenshtein distance) between the erroneous word and each candidate word. The edit distance quantifies the number of single-character edits (insertions, deletions, or substitutions) needed to change one word into the other. Candidate words with an edit distance within an acceptable range (typically 1 or 2) were added to the list of possible corrections. This metric helped prioritize words that were more likely to be the intended

correct word, given the minimal edits required.

Contextual Filtering

For each identified erroneous word and its list of candidate corrections, we considered the surrounding context words within the sentence.

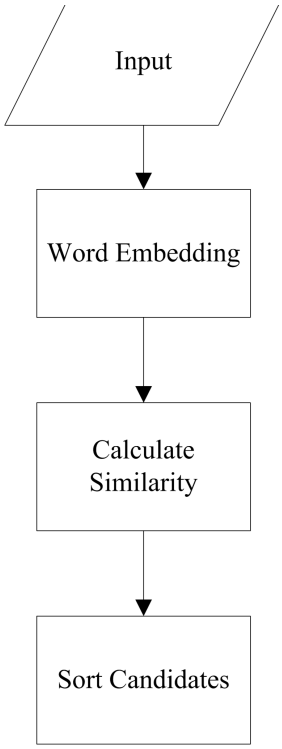


Figure 5.2: Candidate Generation

- **Word Embeddings:** To represent words in a high-dimensional vector space, capturing semantic similarities based on context. We used pre-trained Word2Vec embeddings to obtain vector representations foeach candidate word and each context word. Word2Vec models, trained on a large Nepali text corpus, mapped words to vectors such that words with similar meanings or usage contexts were positioned closely together in the vector space.
- **Calculate Similarity:** To evaluate how well each candidate word fits within the context of the sentence. For each candidate word, we calculated the average

cosine similarity between its embedding and the embeddings of the context words. Cosine similarity measures the cosine of the angle between two vectors, providing a metric for how similar the two vectors are. A higher cosine similarity indicated a better contextual fit for the candidate word within the sentence. The mathematical formula corresponding to the cosine similarity is given as:

$$\text{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

where:

- $\mathbf{A} \cdot \mathbf{B}$ is the dot product of the vectors **A** and **B**.
- $\|\mathbf{A}\|$ is the magnitude (or norm) of the vector **A**.
- $\|\mathbf{B}\|$ is the magnitude (or norm) of the vector **B**.

- **Sort Candidates:** To identify the most contextually appropriate correction for the erroneous word. We sorted the candidate words based on their similarity scores and edit distances. Candidates with higher similarity scores (indicating better contextual fit) and lower edit distances (indicating fewer changes needed) were prioritized. The candidate word with the highest similarity score and the lowest edit distance was selected as the best correction, ensuring both contextual relevance and minimal deviation from the original word.

Correct Sentence Generation

We took the erroneous sentence as input and for each word in the sentence, If the word had a low context similarity score, we generated candidate corrections. We filtered the candidates based on their context similarity scores. We chose the best candidate as the corrected word. If the word had a high context similarity score, we retained it as is. We returned the corrected sentence along with the context similarity scores for each word. Additionally, we provided a list of similar words for each corrected word.

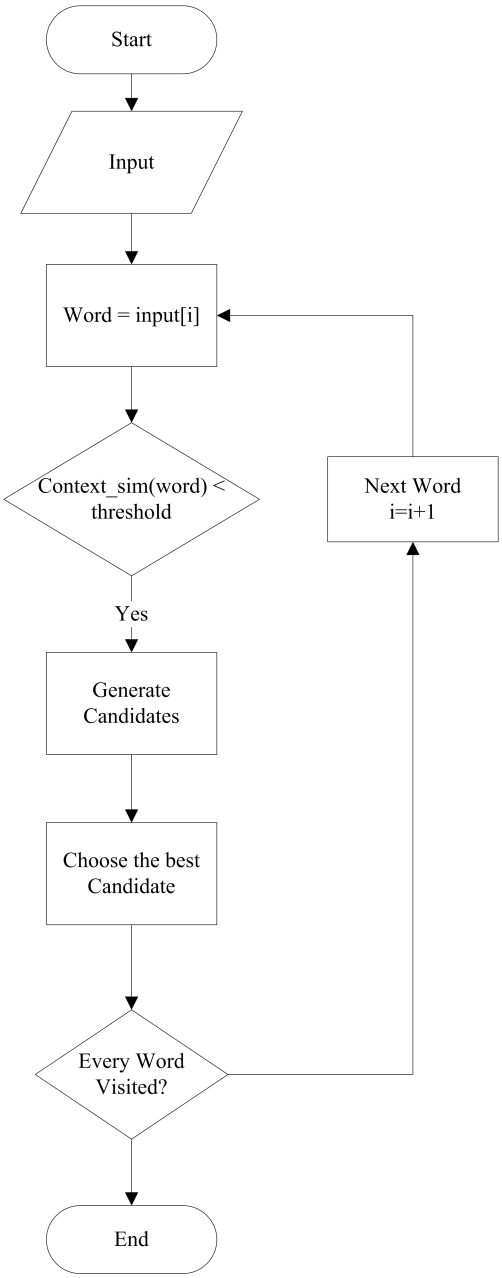


Figure 5.3: Correct Sentence Generation

5.1.4 Test Set Generation

Confusion Set Generation

We generated sets of words that are often confused with each other based on phonetic and contextual similarities. This involved creating a list of common errors and their potential corrections.

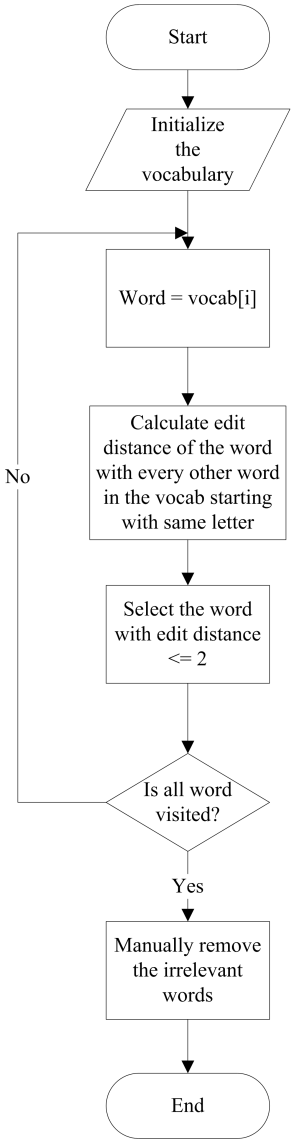


Figure 5.4: Confusion Set Generation

Test Set Generation

We created a test set to evaluate the performance of the spelling correction tool. This test set included sentences with known errors and their correct versions for benchmarking and validation purposes.

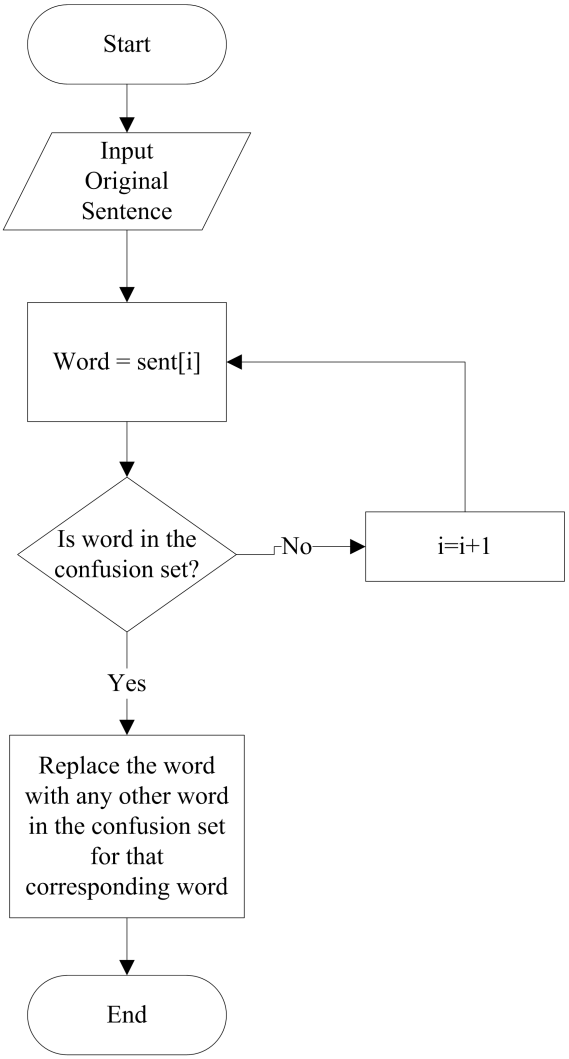


Figure 5.5: Test Set Generation

5.2 Transformer

As the result obtained from Word2Vec was not good. So, we tried to solve the problem of checking and correcting the contextual errors in the sentence by using Transformer model.

5.2.1 Dataset Description

This dataset is a comprehensive collection of Nepali-language articles, including details like categories, headlines, full text, publication dates, and links to the original articles. It's designed to make it easy to analyze trends, categories, and content over time.

- **Size:** The dataset contains over 2.5 million articles across 34 unique categories. ”Politics” is the most common category in the sample.
- **Total Word Count:** The articles have a combined total of 765,220,852 words.

Most of the data is in Nepali, while the categories are also available in English for ease of analysis and translation. This dataset is well-suited for exploring trends, examining the prevalence of topics like politics, and studying patterns in Nepali-language media over time.

The distribution of the words per sentence in raw dataset are:

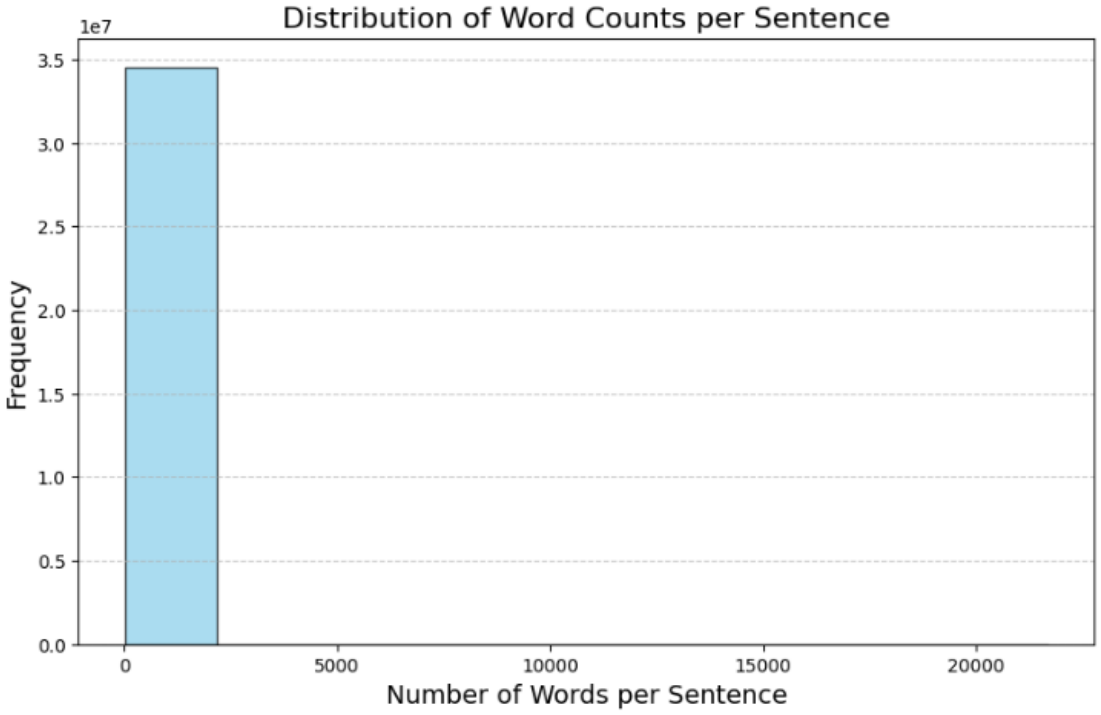


Figure 5.6: Distribution of words per sentence with outliers

The distribution is dominated by an extreme outlier or outliers with sentence lengths much greater than the majority of the data. The bulk of the data appears clustered near the lower end of the x-axis, likely around small sentence lengths, while the long tail reaches up to 20000 words. This suggests that the dataset contains sentences with unusually long lengths, which skew the overall visualization and make it hard to interpret the bulk of the data.

The distribution of the words per sentences after removal of the sentence with word length that is outlier:

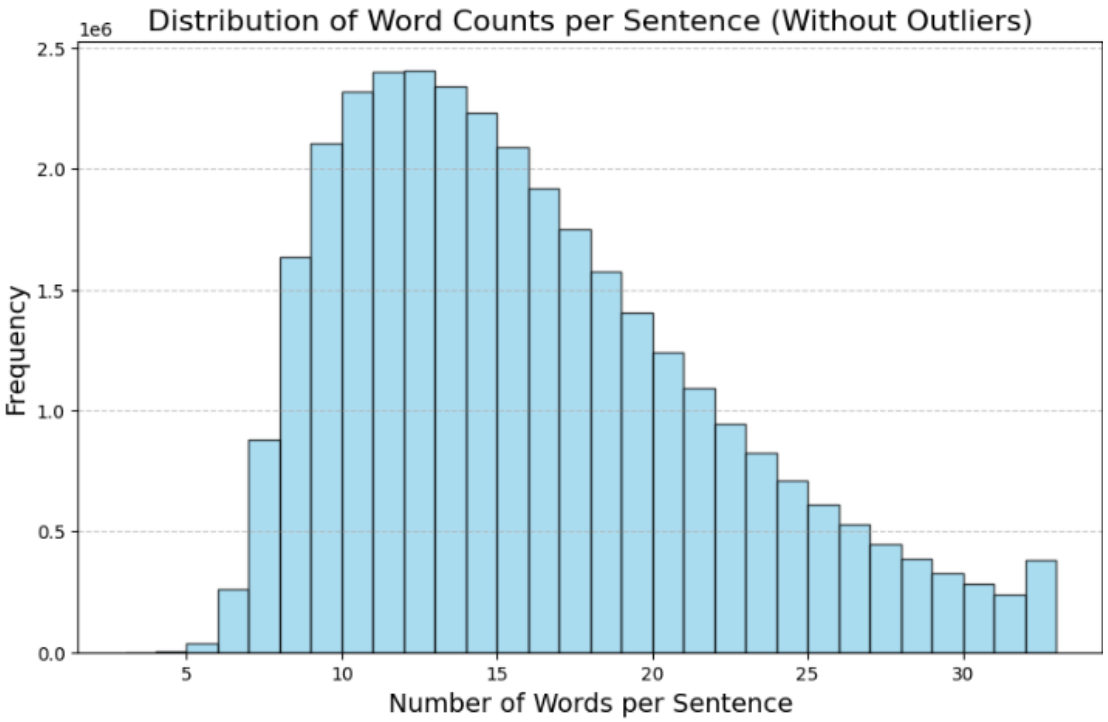


Figure 5.7: Distribution number of words per sentence without outliers

From this plot we can see the dataset has almost normal distribution of the data. But it is slightly right skewed.

The above plot shows the occurrence of commonly confusion words in the training datasets. Most of the data resides at the left side of the distribution shows less occurrence in the dataset. Blue line smoothly approximates the probability density function, showing the general trend of word frequencies. The data confirms a typical natural language distribution where a few words occur very frequently, and the vast majority occur rarely. The dataset is heavily imbalanced, with most confusion words being either very rare or very frequent. This imbalance may affect how the model learns to resolve these words. Low frequency confusion words contribute little to the overall dataset, which might lead the model to under-prioritize them during training.

We had not trained entire datasets instead done on 4,79,232 sentences. The commonly used confusion words were collected and their occurrences in the training datasets were

studided. Some frequent and infrequent along with their occurences count was shown below as a sample.

Table 5.4: Frequent Words

Word	Count
आँसु	3260
खाली	3260
देखि	3260
देखी	3260
नाउँ	3260
चीन	3260
जाति	3260
सीता	3260
बिना	3260
जेठा	3260

Table 5.5: Infrequent Words

Word	Count
होडा	0
दुबाउनु	0
ईष	0
धुबाउनु	1
खोप	1
अरणी	2
जुवारी	2
लथ	3
आंत	4
हत	4

In order to balance the occurrence of confusion words in the datasets, we clipped the occurrence of the words of upper limit to 3,260.

5.2.2 Confusion Set Creation

Directly iterating through the vocabulary and would be computationally complex. So, to reduce computational complexity we first grouped the words in the Nepali Brihat Sabdakosh [17] by transliterating the first letter in the word. The words which have same transliteration in the first letter are placed in same group.

Table 5.6: Transliteration map

Transliteration	**Letters**
sa	स, श, ष
ka	क, ख, ऋ
jha	झ, ज
ba	ब, भ, व
dha	ध, द, ढ, ड
na	ण, न, ङ, ञ
ta	त, ट, ठ, थ
cha	च, छ
la	ल
ra	र
aa	आ, अ, ए, ह, ओ
ii	ई, इ
uu	ऊ, उ, औ, ौ
pa	प, फ
o	ो
aai	ऐ, ै
a	ा, े
i	ी, ि
u	ु, ू

Algorithm: Grouping Words Based on Transliteration of the First Letter

1. **Input:**

- **Transliteration Map:** A dictionary mapping Nepali letters to their respective transliterations.
  Example: {'स': 'sa', 'श': 'sa', 'ष': 'sa', 'क': 'ka', 'ख': 'ka', 'ग': 'ga', ...}
- **Word List:** A list of Nepali words to be grouped.
  Example: {"संगीत", "शिक्षा", "षड्दर्शन", "शंख", "कथा", "खेल"}.

2. **Output:** A dictionary grouping words by the transliteration of their first letter. Example:

Table 5.7: Example output after grouping by first letter

sa	संगीत, शिक्षा
ka	कथा, खेल

3. **Steps:**

a) **Initialization:** Create an empty dictionary `grouped_words` to store lists of words grouped by their transliterated first letter.

b) **Define Transliteration Function:** Implement a function `transliterate_first_letter(word)`:
   - Extract the first letter of the word.
   - Use the transliteration map to convert the first letter into its transliterated form.
   - If the first letter does not exist in the map, use the letter itself.

c) **Group Words:** For each word in the word list:
   - Apply the transliteration function to the first letter of the word.
   - Append the word to the corresponding group in `grouped_words`.

d) **Display Results:** Print the grouped words for verification.

After grouping the words by using the algorithm specified above then the confusion set was created by transliterating the entire word. The algorithm for this is given below:

Algorithm: Generating Confusion Sets Based on Whole Word Transliteration

Step 1: Input Preparation:

Take as input:

- A dictionary of grouped words where each group contains words starting with the same transliterated first letter.
- A transliteration mapping that provides the transliteration for every character in the word.

Step 2: Transliteration Function:

Define a function `transliterate(word)` that takes a word as input and replaces each character with its corresponding transliteration from the transliteration map. If no transliteration exists for a character, the original character is retained.

Step 3: Initialize Data Structures:

Create an empty dictionary, `confusion_sets`, to store confusion sets. Each word will map to a list of similar words based on transliteration.

Step 4: Process Each Group:

For each group of words in the dictionary:

(a) For each word in the group, compute its full transliteration using the `transliterate` function.

(b) Create a temporary dictionary `transliteration_map` to store words by their full transliteration.

(c) Append each word to the corresponding transliteration key.

Step 5: Identify Confusion Sets:

For each transliteration key in the `transliteration_map`, if there is more than one word associated with the transliteration, those words are considered to be in the same confusion set and should be added to each other's confusion sets.

Step 6: Output Results:

Save or return the confusion sets. The result will be a dictionary where each word maps to a list of words that are similar to it based on full word transliteration.

Example:

Consider the following input:

`grouped_words`:

ka: कथा, खेल, कता

ma: मैला, महिला, माला

`Transliteration Map`: {'स': 'sa', 'श': 'sa', 'ष': 'sa', 'क': 'ka', 'ख': 'ka', 'ग': 'ga', ...}

- **Transliteration of Words:**
  - Group `'ka'`:
    - 'कथा' → 'kata'
    - 'खेल' → 'khela'
    - 'कता' → 'kata'
  - Group `'ma'`:
    - 'मैला' → 'maaila'
    - 'महिला' → 'maaila'
    - 'माला' → 'maala'
- **Output:**

  Confusion sets:

कथा	कता
कता	कथा
महिला	मैला
मैला	महिला

Pseudocode for this algorithm is given in the appendix at page number 91.

After generating the confusion set using these algorithm most of the words in the confusion set was relevant and were sounding similar. But there were cases where we had to manually remove some words which were not sounding similar. By using this technique we were able to collect around 18000 similar sounding words in Nepali.

5.2.3 Fine Tuning: Applying Mask for Masked Language Modeling

Input:

- sentence: The input sentence to be masked.
  For example: "यो बगैचामा, धेरै फुलहरु छन ।"
- confusion_set: A predefined set of confusion words.
  For example: ['फुल', 'फूल']
- mask_prob: The probability of masking tokens.

Output:

- masked_sentence: The sentence with selected tokens replaced by [MASK].
  For example: यो बगैचामा धरै [MASK] छन

Steps:

1. **Preprocess the Sentence:** Clean the sentence by removing unwanted characters. Split the cleaned sentence into tokens. Exclude the final punctuation token if present.

   In the above example, comma and quotation are removed to get:
   [यो, बगैचामा, धेरै, फुलहरु, छन]

2. **Identify Confusion Words:** Extract tokens from the sentence that belong to the confusion_set. And remove the post-position of the confusion-word.

   In the above example फुल is the confusion word. So, remove हरु.
   [यो, बगैचामा, धेरै, फुल, छन]

3. **Calculate Number of Tokens to Mask:** Determine the total number of tokens to mask based on the mask_prob. Ensure that at least one token is always masked if the confusion word is present.

   In the above example, the number of token is 5 and masking probability is 15%. Since, a confusion word is present the number of token to be masked is 1.

4. **Select Tokens to Mask:**

   - If confusion words are present, randomly select one confusion word to mask.
   - Reduce the remaining token mask count accordingly.
   - If additional tokens need to be masked, randomly select non-confusion words for masking.

   In the above example, फुल is selected.

5. **Mask the Tokens:** Replace the selected tokens in the sentence with `[MASK]`.

   [यो, बगैचामा, धेरै, [MASK], छन]

6. **Return Masked Sentence:** Join the modified tokens to create the masked_sentence.

   In the above example:
   यो बगैचामा धरै [MASK] छन

5.2.4 Integrating with Transformer

The core of our spelling correction system for Nepali is based on leveraging a confusion set and a pre-trained language model. The confusion set is a dictionary that maps each potentially similar word to its corresponding set of alternative words (typically homophones or similar words that are contextually interchangeable). This approach aims to identify and replace misspelled words with the most probable correct word based on the context of the sentence.

The NepaliBERT and NepBERTa model is a pre-trained language model fine-tuned for Nepali text. It helps to evaluate the most probable candidates for each misspelled word based on the context in which it appears. The approach works by:

Step 1: Tokenization

The sentence is tokenized using the NepaliBERT tokenizer. This prepares the sentence for model inference by converting it into token IDs.

Step 2: Confusion Set Lookup

For each word in the sentence, we check if it exists in the confusion set. If a word is found, its possible corrections (confusion candidates) are retrieved.

Step 3: Masked Word Prediction

The identified word is replaced by a mask token [MASK], and the model predicts the most probable replacement for the masked word based on the context in the sentence.

Step 4: Probability Calculation

Using softmax, the model outputs a probability distribution for each candidate word. The candidate with the highest probability is selected as the correction.

Step 5: Sentence Correction

After processing all words in the sentence, the corrected sentence is returned, where misspelled words are replaced with their most probable correction.

These steps can be summarized using this system flow given below:

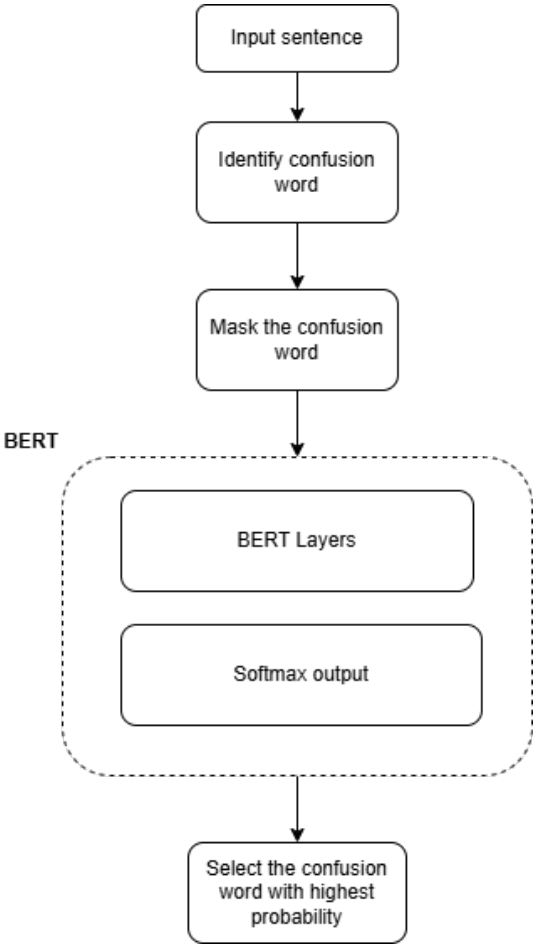


Figure 5.8: End to end pipeline

5.2.5 Hyperparameter Specification

The hyperparameter specification of both model i.e. NepaliBERT and NepBERTa used while fine tuning are same. These are both BERT based architecture.

Table 5.8: Hyperparameter Specification

Dimension	768
Layers	12
Epochs	2
Layer Unfreezed	2
Embedding	Trainable
Beta 1	0.9
Beta 2	0.999
Epsilon	10^{-8}

6 RESULT AND ANALYSIS

6.1 Word2Vec

We trained the Word2Vec model specifically continous bag of word till 15 epoch as specified in the table 5.3. When inputs such as आइतबार, जङ्गल & नुन were given as the input we can see that the model was able to give similar vectors to the similar words resulting in the high similarity scores between the similar words.

Table 6.1: Similarity Scores of the similar words

'आइतबार'	'जंगल'	'नुन'
बुधबार: 0.9195	जङ्गल: 0.8666	दाल: 0.9016
सोमबार: 0.9191	घना: 0.8650	नून: 0.8774
बिहीबार: 0.9125	आसपासमा: 0.8510	दाना: 0.8659
शुक्रबार: 0.9104	आसपासको: 0.8407	अन्न: 0.8657
मंगलबार: 0.9033	आसपास: 0.8312	बिस्कुट: 0.8591
शनिबार: 0.8834	वरिपरिको: 0.8257	तोरिको: 0.8590
मंगलबार: 0.8438	तालतलेया: 0.8241	रक्सी: 0.8577
सोमवार: 0.8023	जङ्गले: 0.8233	चाउचाउ: 0.8543
शुक्रवार: 0.7719	किनार: 0.8225	मिसाएर: 0.8532
शनिवार: 0.7700	जङ्गलको: 0.8217	चिउरा: 0.8524

By analyzing the table 6.1 given above we can say that the Word2Vec performs well on clustering the similar words.

The output of the candidate generation based only on the edit distance is shown below as discussed in the section 5.1.3. Here the candidate of only first word is shown.

Sentence	Candidates of first word (with edit distance)
”नीति तथा कार्यक्रम प्राथमिकतामा उच्छ आर्थिक वृद्धि ‘हिलारी स्टेप भत्किएको छैन”	[(नीति, 0), (नाति, 1), (नीजि, 1), (नीता, 1), (निति, 1), (नीतु, 1), (नीलि, 1), (नीती, 1), (नीधि, 1), (नीनि, 1), (नीतू, 1), (न्ति, 1), (नेति, 1)]

The contextual filtering of the generated candidate as discussed in the section 5.1.3 is done on the basis of the output is shown below as discussed in the section

Sentence	Contextual filtered candidates of the first word (with edit distance and similarity score)
”नीति तथा कार्यक्रम प्राथमिकतामा उच्छ आर्थिक वृद्धि ‘हिलारी स्टेप भत्किएको छैन”	[(निति, 1, 0.8043), (नीती, 1, 0.3043), (नीजि, 1, 0.2656), (नीधि, 1, 0.1938), (नीनि, 1, 0.1801), (न्ति, 1, 0.1181)]

As explained in the section 5.1.4 we followed the process as shown in the figure 5.4. We were able to generate collection of word that mostly sound similar but mean different. Here is a sample of the confusion set we have generated:

Table 6.2: Generated Sample Confusion Set

Sets
अशेष, अशिष्ट, अविशेष
खेल्छ, खेलपछि, खेल्ने, खान्छ
बढ्नु, बढनेछु, बढिन्, बाढ्नु
छेउमा, छेउँमै, छतमा
भेगमा, भलीमा, भेटेमा, भिरमा

As explained in the section 5.1.4 we followed the process as shown in the figure 5.5. We were able to generate test set of that consisted of correct sentences and the corresponding

incorrect sentences. Here is a sample of the test set we have generated:

Table 6.3: Generated Sample Test Set

Original Sentence
पानी कम हुने सिजनमा बाँध भत्किएको भन्दा पनि भत्काइएको हुन सक्ने उनी दाबी गर्छन्
भन्नुहुन्छ रुचि त छ तर राम्रो कथा र निर्माता भेटेमा
गाईवस्तु सकुशल भेटेमा र सञ्चो भएमा परेवाको बली दिइन्छ
सबै ठाउँमा हामीले आफूलाई लागेका कुराहरु बोल्यौं
गाउँले खोल्साखोल्सीमै सौच गर्थे

Incorrect Sentence
पानी कम हुने सिजनमा बाँध भत्किएको भन्दा पनि भत्काइएको हुन सक्ने उनी दाबी गर्जिन्
भन्नुहुन्छ रुचि त छ तर राम्रो कथा र निर्माता भेगमा
गाईवस्तु सकुशल भलीमा र सञ्चो भएमा परेवाको बली दिइन्छ
सबै ठाउँमा हामीले आफूलाई लागेका कुराहरु बोलेनौं
गाउँले खोल्साखोल्सीमै साचि गर्थे

After training the Word2Vec model we evaluated the model on the test set for the correction of the real-words error. The model was able to get the BLEU Score of 50.67 in this specific task. This score indicates that the Word2Vec performs not that good in the task of real-word error correction in a sentences. From the research and experimentation these results might be attributed to the following reasons:

- Word2Vec generates static embeddings, meaning a word has the same representation regardless of context. This limitation can be significant in tasks requiring nuanced context understanding.
- Word2Vec doesn't contain positional encoding so the place where the word appear doesn't matter. For this case we tried to make it aware about the position where the word appear by using the techniques implemented using cosine similarity as shown in the flowchart 5.3

In the Part B phase of this project we might be dealing with the task of making the model more context aware by using more complex and sophisticated model having the high capacity of learning contextual meaning of words. The solution to the above specified problem will be handled using the following techniques:

- Consider using context-aware models like BERT, which capture more nuanced relationships between words
- Fine-tune pre-trained models on domain-specific data to better capture relevant word usages and corrections.

So, through the experimentation we found that the Word2Vec is not suitable for the task of the context aware spelling correction on the Nepali language but performs well on the task of similar word clustering.

6.2 Transformer

In the Part A of this project we tried to implement the system using Word2Vec. But due to the dynamic limitations of the Word2Vec model, the model was unable to perform well on this task. We also noticed that the approach we used for solving this problem was also not very promising. So, we tried to implement the whole system using the transformer based model.

To create the confusion set words from Nepali Brihat Sabdakosh [17] was used. The words are grouped based on transliteration of the first letter using the algorithm at 5.2.2.

Table 6.4: Count of words after grouping the words by transliterating first letter

Transliteration of first letter	Count
aa	11702
aaa	2416
ii	591
uu	2700
ka	9847
aai	145
aao	496
aau	179
gha	5591
na	4450
cha	5967
jha	5078
ta	6598
dha	7744
pa	12758
ba	13449
ma	6004
ya	1005
ra	3377
la	3474
sa	11666

As the words are grouped the number of comparison required to create a confusion set is decreased. After this step the confusion set was created by using the algorithm specified in 5.2.2. The confusion set of around 18000 words were gathered from these algorithm. These words were checked manually and irrelevant words were removed.

6.2.1 NepBERTa

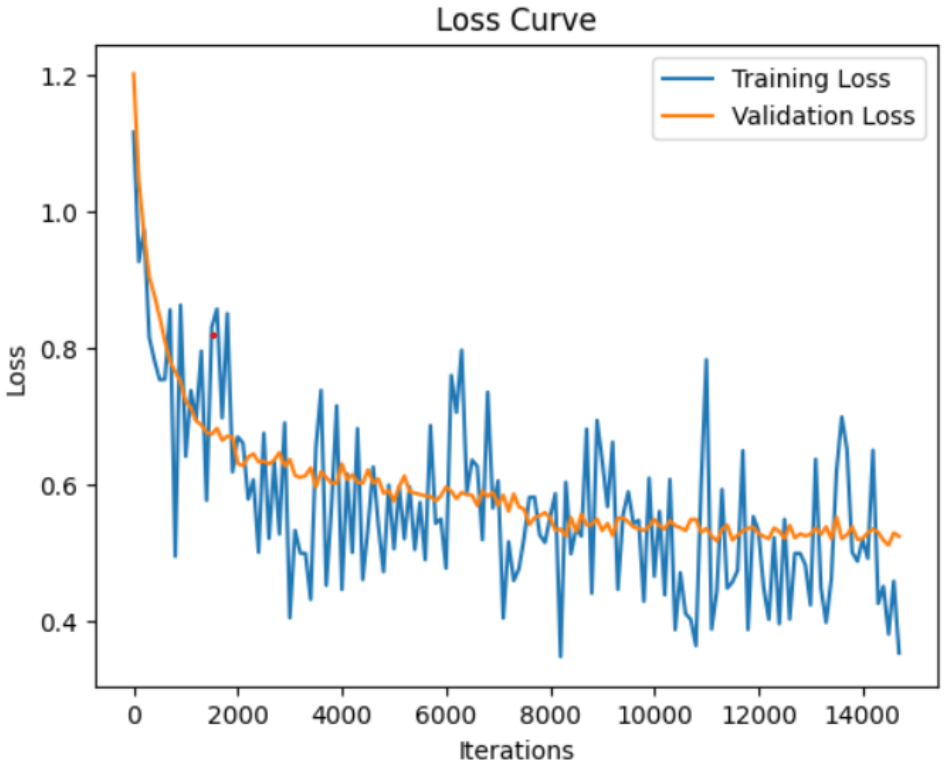


Figure 6.1: Loss curve for NepBERTa

The loss curve demonstrates the progression of training and validation loss over iterations. Initially, both losses start high but decrease steadily as training progresses. Training Loss Started at 0.96 and reduced to as low as 0.35, with an average of 0.52, showing significant improvement during the training process. Validation Loss: Started at 0.57, fluctuated slightly, and stabilized with a mean of 0.54, indicating consistent generalization on unseen data.

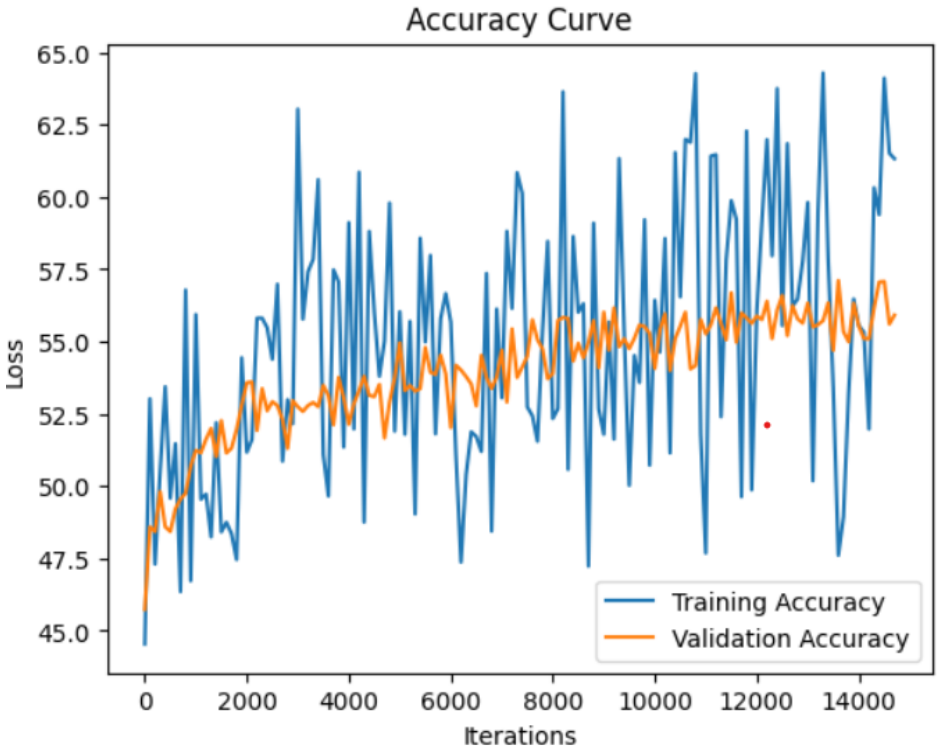


Figure 6.2: Accuracy curve for NepBERTa

- Training Accuracy : Improved from 44.93% to a maximum of 64.29%, averaging 56.22%, reflecting the model's learning capability.
- Validation Accuracy : Maintained between 53.71% and 57.11%, with an average of 55.41%, demonstrating stable validation performance without overfitting.

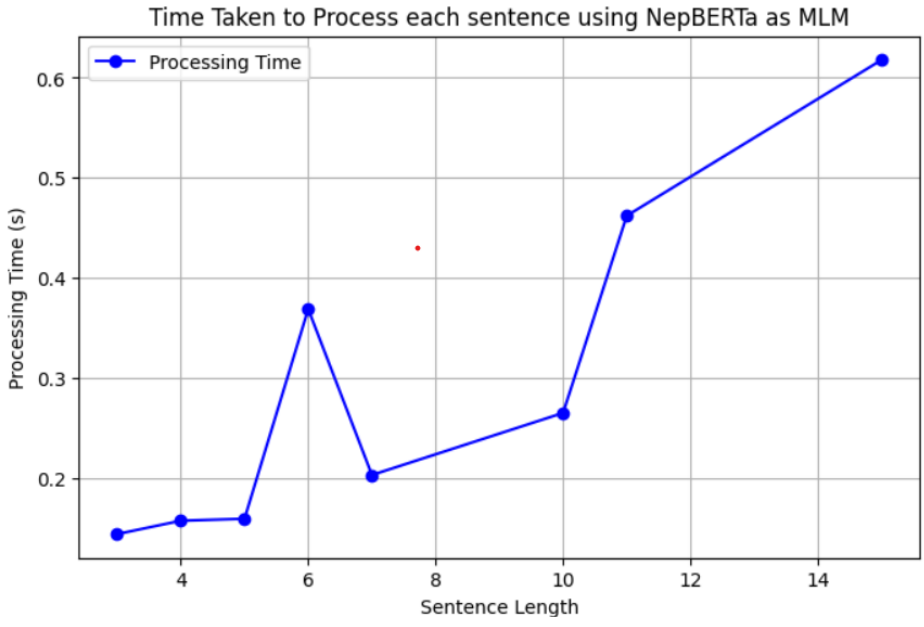


Figure 6.3: Processing Time for NepBERTa as MLM

The graph shows the relationship between sentence length and processing time using NepBERTa as a masked language model (MLM). Processing time increases with sentence length due to the model's computational complexity, as longer sentences require handling more tokens. A noticeable spike at sentence length 6 suggests additional overhead or threshold effects, possibly linked to the attention mechanism. Beyond this, processing time grows steadily, indicating predictable scalability for larger inputs. The model performs efficiently for shorter sentences, making it suitable for such use cases, but optimizations may be needed for handling longer sentences to improve performance.

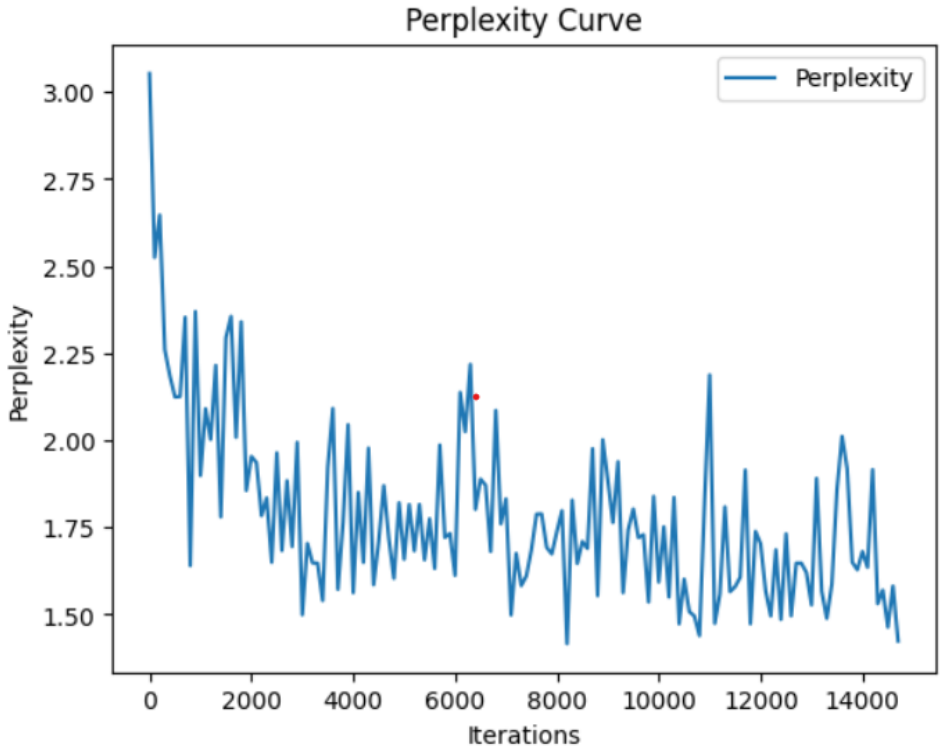


Figure 6.4: Training Perplexity for NepBERTa

- Training Perplexity: Decreased from 2.62 to 1.42, averaging 1.69, highlighting the model's increasing confidence.

NepBERTa is performing well after 2 epochs, with both losses decreasing and stabilizing. Further training or fine-tuning could enhance performance, but the current results indicate a solid foundation for contextual spell-checking in Nepali. These curve confirm that during the second epoch, the model continued to learn effectively while maintaining generalization across validation data.

6.2.2 NepaliBERT

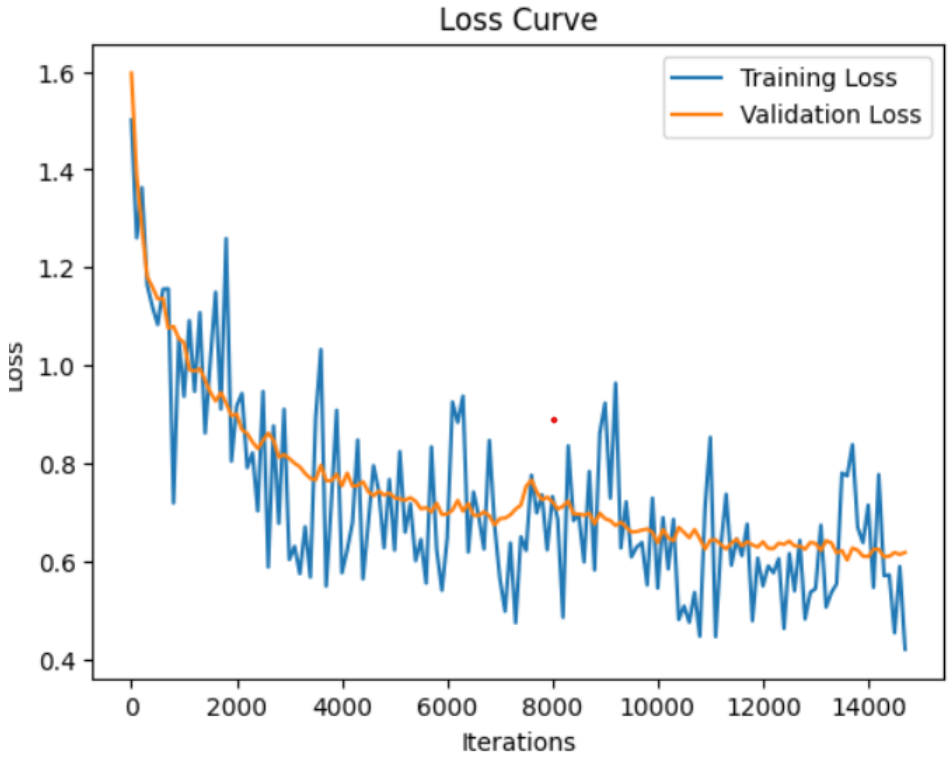


Figure 6.5: Loss curve for NepaliBERT

- Training Loss: Starts at 1.10, decreases to a minimum of 0.42, with an average of 0.64, indicating steady learning progress.
- Validation Loss: Starts at 1.65, stabilizes at a minimum of 0.60, and averages 0.67, showing consistent generalization, albeit slightly higher loss compared to training.

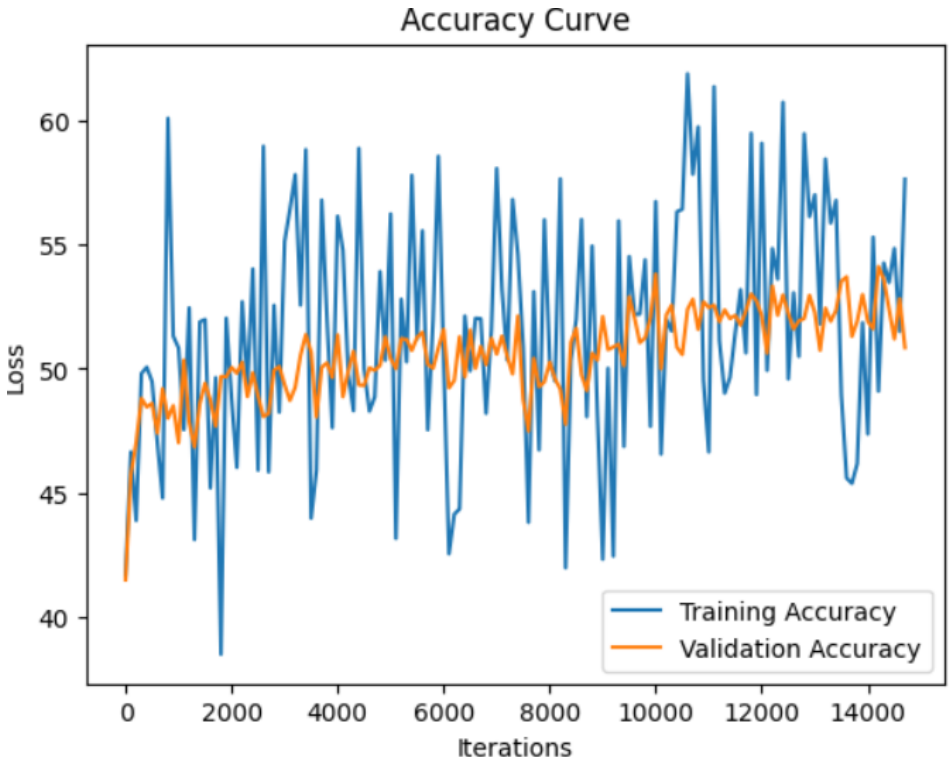


Figure 6.6: Accuracy curve for NepaliBERT

- Training Accuracy: Improves from 40.50% to a maximum of 61.89%, with an average of 52.16%, demonstrating noticeable learning throughout the epoch.
- Validation Accuracy: Ranges from 44.86% to 54.13%, averaging 51.50%, reflecting moderate generalization but slightly trailing training accuracy.

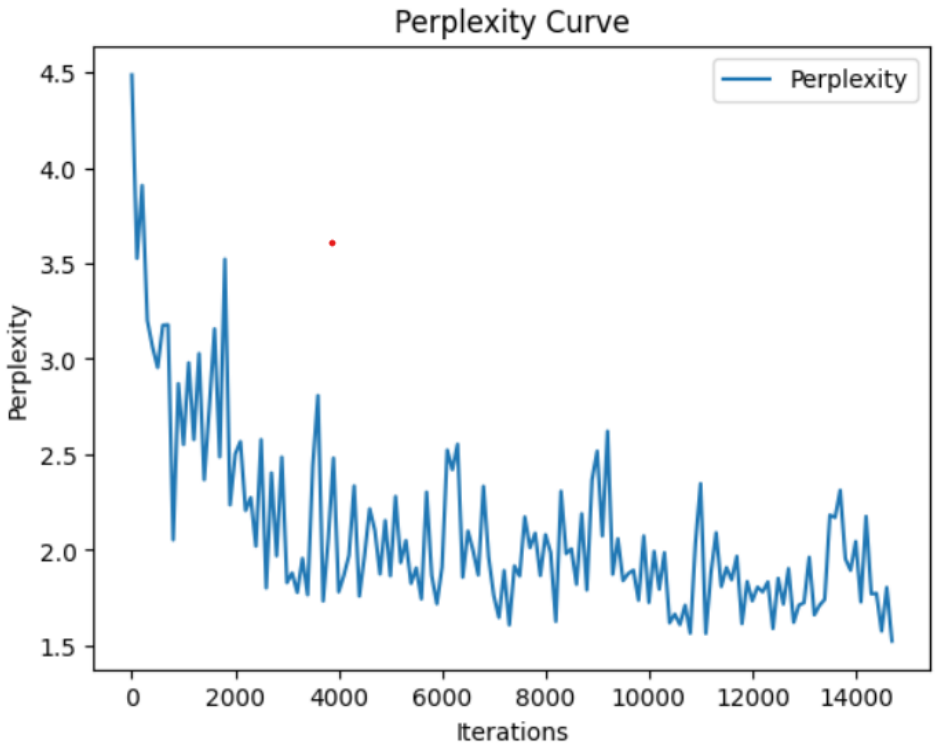


Figure 6.7: Perplexity Curve for NepaliBERT

- Perplexity: Drops from 2.99 to 1.52, with a mean of 1.91, showing growing model confidence.

Compared to the NepBERTa, NepaliBERT's loss values are higher, and its accuracy is slightly lower, suggesting NepBERTa might generalize better in this context

A lack of sufficient training epochs could limit the model's ability to fully learn the underlying structure of the data, resulting in suboptimal performance. Additionally, the batch size of 64 might introduce variability in gradient updates, leading to fluctuations in the loss curve during training. These fluctuations often indicate that the model has not fully converged, as the optimizer is still exploring the loss landscape in search of an optimal solution.

And not much fluctuation in the validation curve can be explained due to the lack of variation in the validation data set which consisted of only 1000 sentences. Further the accuracy calculated during the training is the accuracy of the Masked Language Modeling (MLM) task. Due to this reason the accuracy appears to be low as the accuracy

is calculated by considering whole corpus.

6.2.3 Evaluation

From the loss curve seen above, the NepBERTa's curves are quite impressive. So, we evaluated the performance of the model for the weights of NepBERTa only.

In the training dataset there were confusion words that were frequently repeated, while some other confusion words were not repeated much. So, to evaluated the performance of the data we created a validation set of 1000 sentences. From which the 909 sentences consisted of mostly repeated confusion words. And other consisted of not much occurring confusion words in the training set.

For the words that occured the most in the training set, the validation accuracy appeared to be 82.06%. And for the words that occured least in the training set, the validation accuracy appeared to be 63.63%.

6.2.4 Monitoring Output

In this section we take an example sentence and give the sentence to model before and after after training.

Sample 1:

Before Training:

```
{'यस': 0.00046574469888582826, 'यश': 4.60435840068385e-05}
Input Sentence: कठिन परिस्थितिमा पनि उसको यस घटेन।
Corrected Sentence: कठिन परिस्थितिमा पनि उसको यस घटेन।
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Figure 6.8: Output before training sample 1

Here, we can see that the probability of the word यस before training is high which is incorrect in that context.

After Training:

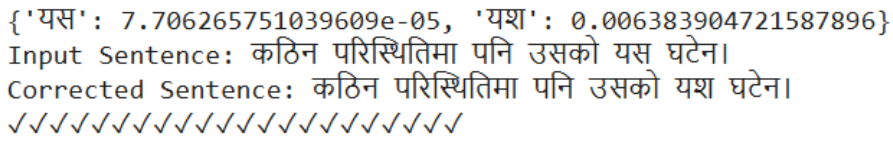


Figure 6.9: Output after training sample 1

Now, after training the model on the dataset for two epoch we can see the model is able to correctly gave high probability to the word यश. Similar result is also seen in the case of other words.

Sample 2:

Before Training:

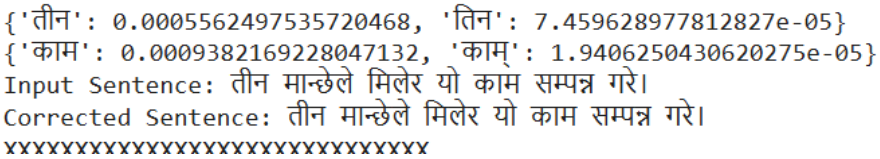


Figure 6.10: Output before training sample 2

Here, we can see that the probability of the word तीन before training is high which is incorrect in that context.

After Training:

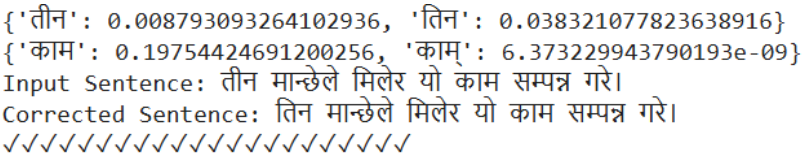


Figure 6.11: Output after training sample 2

Now, after training the model on the dataset for two epoch we can see the model is able to correctly gave high probability to the word तिन. Similar result is also seen in the case of other words.

Sample 3:

Before Training:

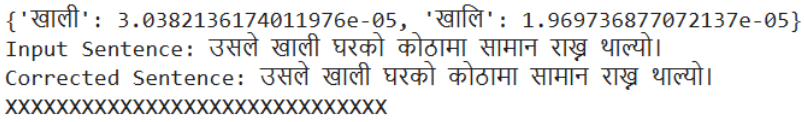


Figure 6.12: Output before training sample 3

Here, we can see that the probability of the word खालि before training is high which is incorrect in that context.

After Training:

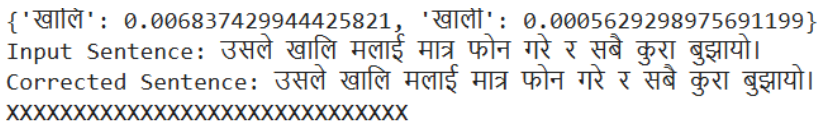


Figure 6.13: Output after training sample 3

Here, we can see that the model is unable to correct the confusion word. This may be attributed to the fact the word was presented incorrectly in the training dataset or not enough training. But we can see the probability difference between these two confusion words was decreased after training.

From this output, we can say that model is able to learn from the dataset and the task of Masked Language Modeling is fit for this task of context aware spelling correction. But still the model lags behind in some cases. This may be reasoned as follows:

- Not enough training.
- Less variation in the dataset.
- Less accurate dataset.

7 REMAINING TASKS

In the first part of the our Major Project on 'Nepali Context-Aware Spelling Tool' we developed the system based on the Word2Vec specifically continous bag of word. We understood how the Word2Vec model functioned to provide the word embedding that carries semantic meaning. The Word2Vec being a static model it lagged behind in some situation. Further, there are still errors present in the corpus which we will be focusing in the coming time.

In the part B, we used quite different approach to solve this existing real world problem using Transformer Based Model. As the BERT, is a transformer based that has self attention module which is capable of capturing dynamic relationship between the words in a sentence. We successfully built a end-to-end pipeline which is capable of correcting the words based on contextual relationship with other words. The results are pretty good, but could still be improved by cleaning the dataset more. Furthermore, the remaining task of our project are listed below:

- Fine tune the model with the data of around 40 million sentences.
- Improve the accuracy of the model.
  - Remove the stop words.
  - Remove the post-positions from the whole corpus, instead just from the confusion words.
  - Train for more epoch.

8 APPENDICES

Appendix A: Project Timeline (Gantt Chart)

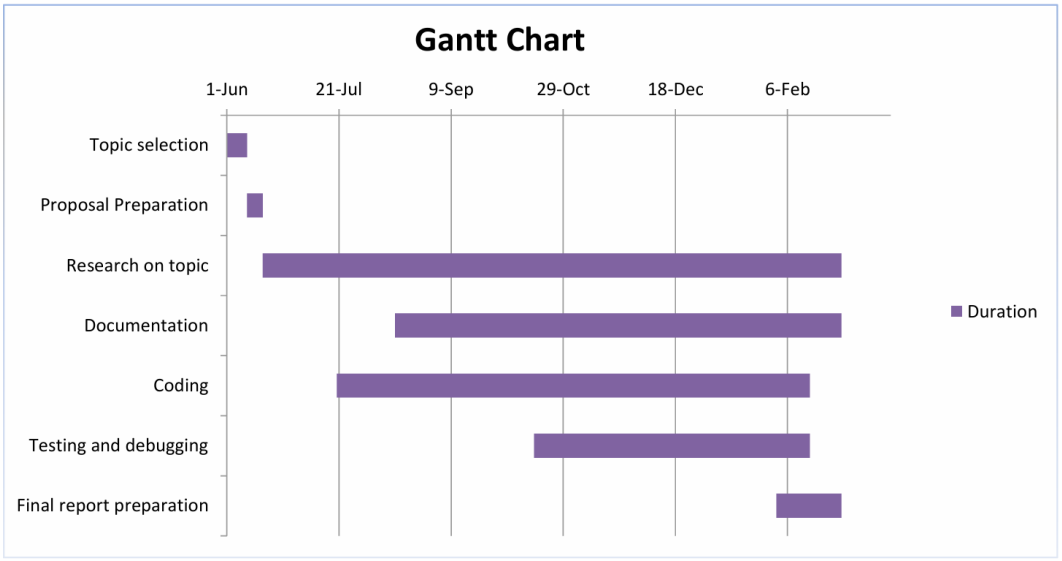


Figure 8.1: Gantt Chart

Appendix B: Project Budget

Table 8.1: Project Budget

Activity	**Amount (Rs)**
Printing	2000
Data Collection	1000
Google Colab Pro	5000
Miscellaneous	1000
Total=	9000

Appendix C: Dynamic Programming for Edit Distance

Input: Two strings, str_1 and str_2, where n and m denote their lengths respectively.

Output: Minimum edit distance D[n][m] between str_1 and str_2.

1. **Initialize Matrix**

   Define a 2D matrix D of size $(n + 1) \times (m + 1)$.

   Set $D[i][0] = i$ for all $i \in \{0, \ldots, n\}$.

   Set $D[0][j] = j$ for all $j \in \{0, \ldots, m\}$.

   (*This initializes base cases where one of the strings is empty.*)

2. **Fill Matrix Using Recurrence Relation**

   For each $i = 1$ to n:

   For each $j = 1$ to m:

   a) If $\text{str}_1[i - 1] = \text{str}_2[j - 1]$:

   $$D[i][j] = D[i - 1][j - 1]$$

   (*No edit is needed if characters match.*)

   b) Else, set

   $$D[i][j] = \min\left(D[i - 1][j] + 1, D[i][j - 1] + 1, D[i - 1][j - 1] + 1\right)$$

   where:

   - $D[i - 1][j] + 1$ represents a deletion,
   - $D[i][j - 1] + 1$ represents an insertion, and
   - $D[i - 1][j - 1] + 1$ represents a substitution.

   (*Choose the minimum cost among these operations.*)

3. **Return Final Result**

   The minimum edit distance is given by D[n][m], the value at the bottom-right cell of the matrix.

Complexity Analysis:

- *Time Complexity*: $O(n \times m)$ — each cell is computed once.
- *Space Complexity*: $O(n \times m)$ — matrix D requires $(n + 1) \times (m + 1)$ storage.

Appendix D: Pseudocode for confusion set creation based on transliteration

Algorithm 1 Confusion Set Generation Based on Transliteration

0

Input: Grouped word list *grouped_words*, transliteration map *transliteration_map*

Output: Confusion sets *confusion_sets*
Initialize an empty dictionary *confusion_sets*.
for each letter group (*letter*, *words*) in *grouped_words* **do**
    Initialize an empty dictionary *transliteration_to_words*.
    for each word *word* in *words* **do**
        Add *word* to *transliteration_to_words[transliterated]*.
    end for
    for each transliteration group (*transliterated*, *word_group*) in *transliteration_to_words* **do**
        if *len(word_group)* > 1 **then**
            for each word *word* in *word_group* **do**
                for each other word *w* in *word_group* such that *w* ≠ *word* **do**
                    Add *w* to *confusion_sets[word]*.
                    Add *word* to *confusion_sets[w]*.
                end for
            end for
        end if
    end for
end for=0

Appendix E: User Interface Snapshots For Word2Vec

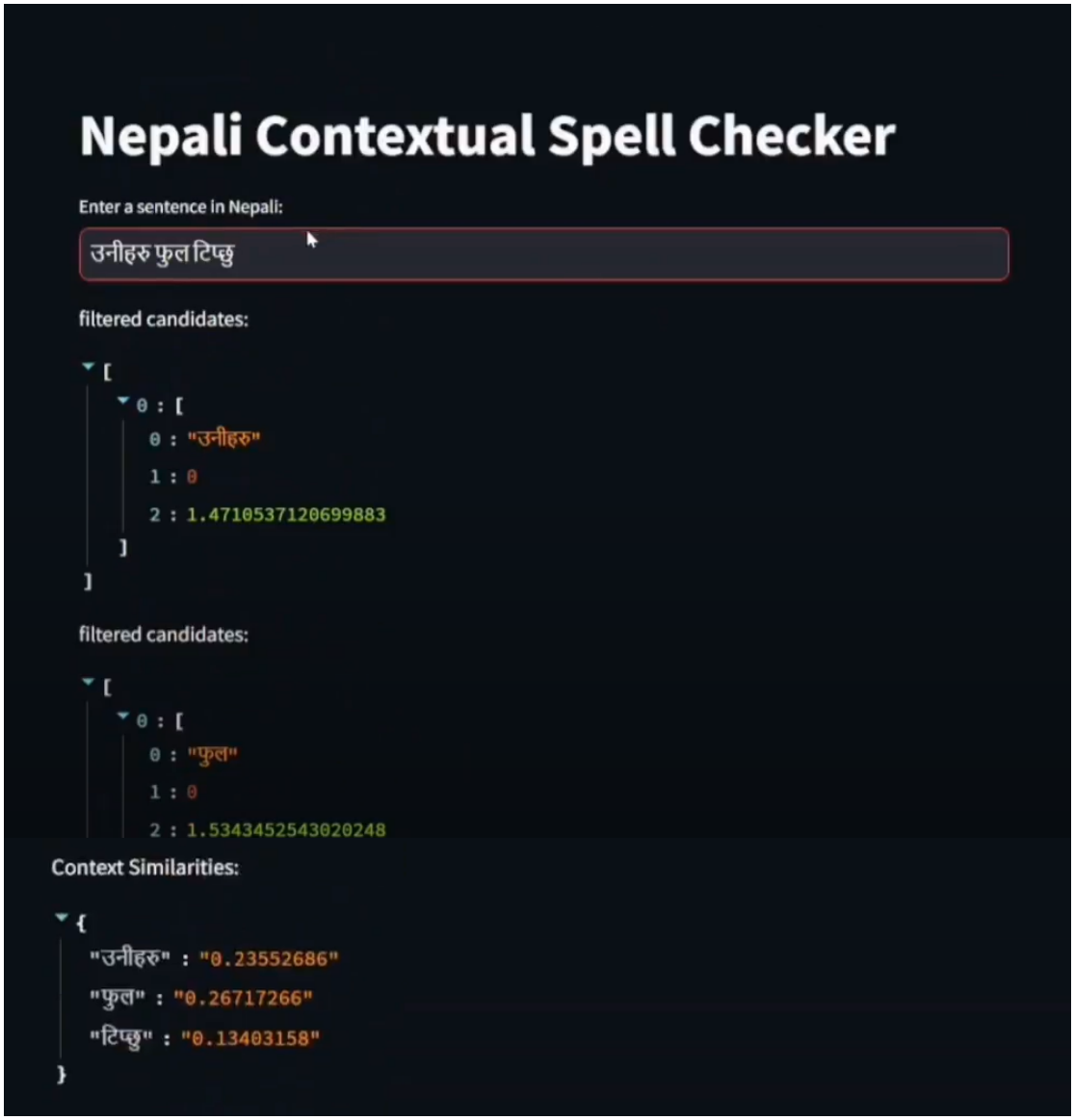


Figure 8.2: Spelling correction interface using Word2Vec

Appendix F: User Interface Snapshots For BERT

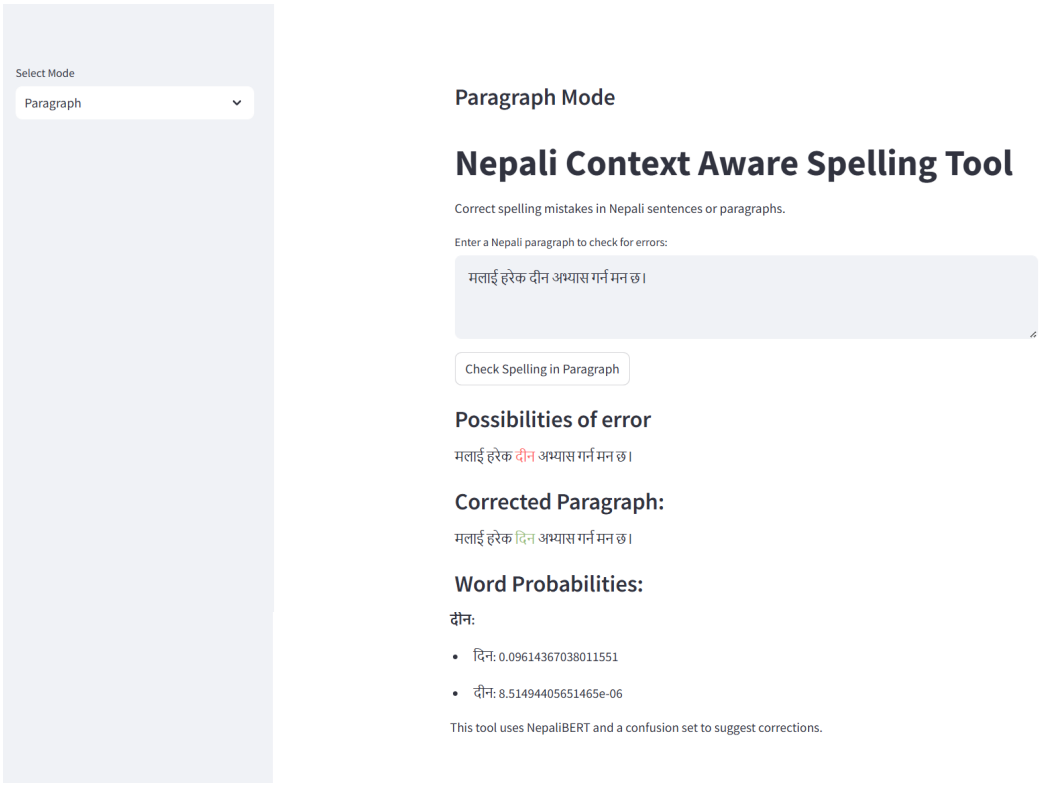
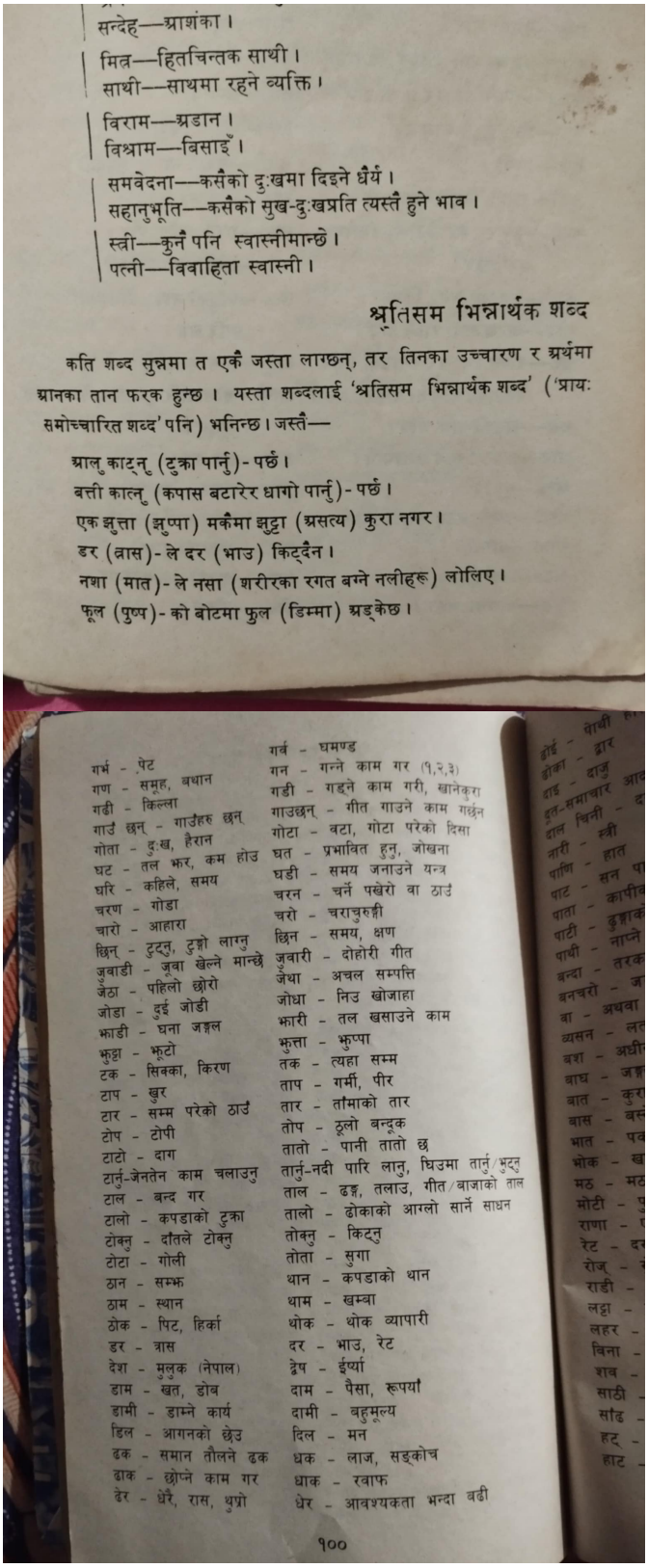


Figure 8.3: Spelling correction interface using BERT

Appendix G: श्रुतिसमभिन्नार्थक शब्द **collection**



सन्देह—आशंका।
मित्र—हितचिन्तक साथी।
साथी—साथमा रहने व्यक्ति।
विराम—अडान।
विश्राम—बिसाइँ।
समवेदना—कसैको दुःखमा दिइने धैर्य।
सहानुभूति—कसैको सुख-दुःखप्रति त्यस्तै हुने भाव।
स्त्री—कुनै पनि स्वास्नीमान्छे।
पत्नी—विवाहिता स्वास्नी।

श्रुतिसम भिन्नार्थक शब्द

कति शब्द सुन्नमा त एकै जस्ता लाग्छन्, तर तिनका उच्चारण र अर्थमा आनका तान फरक हुन्छ। यस्ता शब्दलाई 'श्रुतिसम भिन्नार्थक शब्द' ('प्रायः समोच्चारित शब्द' पनि) भनिन्छ। जस्तै—

आलु काट्नु (टुक्रा पार्नु)- पर्छ।
बत्ती काल्नु (कपास बटारेर धागो पार्नु)- पर्छ।
एक झुत्ता (झुप्पा) मकैमा झुट्टा (असत्य) कुरा नगर।
डर (त्रास)- ले दर (भाउ) किट्दैन।
नशा (मात)- ले नसा (शरीरका रगत बग्ने नलीहरू) लोलिए।
फूल (पुष्प)- को बोटमा फुल (डिम्मा) अड्केछ।

गर्भ - पेट
गण - समूह, बथान
गढी - किल्ला
गाउँ छन् - गाउँहरु छन्
गोता - दुःख, हैरान
घट - तल झर, कम होउ
घरि - कहिले, समय
चरण - गोडा
चारो - आहारा
छिन् - टुट्नु, टुक्रो लाग्नु
जुवाडी - जुवा खेल्ने मान्छे
जेठा - पहिलो छोरो
जोडा - दुई जोडी
झाडी - घना जङ्गल
झुट्टा - झूटो
टक - सिक्का, किरण
टाप - खुर
टार - सम्म परेको ठाउँ
टोप - टोपी
टाटो - दाग
टार्नु-जेनतेन काम चलाउनु
टाल - बन्द गर
टालो - कपडाको टुक्रा
टोक्नु - दाँतले टोक्नु
टोटा - गोली
ठान - सम्झ
ठाम - स्थान
ठोक - पिट, हिर्का
डर - त्रास
देश - मुलुक (नेपाल)
डाम - खत, डोब
डामी - डाम्ने कार्य
डिल - आगनको छेउ
ढक - समान तौलने ढक
ढाक - छोप्ने काम गर
ढेर - धेरै, रास, थुप्रो

गर्व - घमण्ड
गन - गन्ने काम गर (१,२,३)
गडी - गड्ने काम गरी, खानेकुरा
गाउछन् - गीत गाउने काम गर्छन
गोटा - बटा, गोटा परेको दिसा
घत - प्रभावित हुनु, जोखना
घडी - समय जनाउने यन्त्र
चरन - चर्ने पखेरो वा ठाउँ
चरो - चराचुरुङ्गी
छिन - समय, क्षण
जुवारी - दोहोरी गीत
जेथा - अचल सम्पत्ति
जोधा - निउ खोजाहा
झारी - तल खसाउने काम
झुत्ता - झुप्पा
तक - त्यहा सम्म
ताप - गर्मी, पीर
तार - तामाको तार
तोप - ठूलो बन्दूक
तातो - पानी तातो छ
तार्नु-नदी पारि लानु, घिउमा तार्नु/भुट्नु
ताल - ढङ्ग, तलाउ, गीत/बाजाको ताल
तालो - ढोकाको आग्लो सार्ने साधन
तोक्नु - किट्नु
तोता - सुगा
थान - कपडाको थान
थाम - खम्बा
थोक - थोक व्यापारी
दर - भाउ, रेट
द्वेष - ईर्ष्या
दाम - पैसा, रूपयाँ
दामी - बहुमूल्य
दिल - मन
धक - लाज, सङ्कोच
धाक - रवाफ
धेर - आवश्यकता भन्दा बढी

Figure 8.4: श्रुतिसमभिन्नार्थक शब्द collection

REFERENCES

[1] A. M. Turing, *Computing machinery and intelligence*. Springer, 2009.

[2] P. Gupta, "A context-sensitive real-time spell checker with language adaptability," 2020, *10.1109/ICSC.2020.00023*. [Online]. Available: 10.1109/ICSC.2020.00023

[3] B. Prasain, N. lamichhane, N. Pandey, P. Adhikari, and P. Mudbhari, "Nepali spell checker," 2023, *https://doi.org/10.3126/jes2.v1i1.58461*.

[4] S. Bista, Kumar, B. Keshari, L. Khatiwada, Prasad, P. Chitrakar, and S. Gurung, "Nepali lexicon development," 2004-2007, *https://www.yumpu.com/en/document/view/25135568/nepali-lexicon-development-pan-localization*.

[5] X. Ziang, A. Anand, A. Naveen, J. Dan, and A. Y. Ng, "Neural language correction with character-based attention," 2016, *https://doi.org/10.48550/arXiv.1603.09727*.

[6] N. Luitel, N. Bekoju, A. Kumar Sah, and S. Shakya, "Contextual spelling correction with language model for low-resource setting," 2024, *https://doi.org/10.48550/arXiv.1603.09727*.

[7] A. PAL1 and A. MUSTAFI2, "Automatic context-sensitive spelling correction of ocr-generated hindi text using bert and levenshtein distance," 2020, *https://doi.org/10.48550/arXiv.2012.076527*.

[8] Y. Bassil and M. Alwani, "A context-sensitive spelling correction using google web 1t 5-gram information," 2020, *https://doi.org/10.48550/arXiv.1204.5852*.

[9] B. Rijal and S. B. Basnet, "Vector distance based spelling checking systemin nepali with language-dependent," 2020.

[10] B. Rijal, S. Basnet, S. Awale, and S. Prasai, "Preprocessing of nepali news corpus for downstream tasks," 2022, *https://doi.org/10.3126/nl.v35i01.46553*.

[11] S. Pudasaini, S. Shakya, and A. Tamang, "Nepalibert: Pre-training of masked language model in nepali corpu," *arXiv preprint arXiv:1409.0473*, 2020, *https://www.researchgate.net/publication/375019515_NepaliBERT_Pre-training_of_Masked_Language_Model_in_Nepali_Corpus*.

[12] S. Timalsina, M. Gautam, and B. Bhattarai, “Nepberta: Nepali language model trained in a large corpus,” 2022, *https://aclanthology.org/2022.aacl-short.34.pdf*.

[13] B. K. Bal, “Structure of nepali grammar,” 2004, *https://www.researchgate.net/publication/237261579*.

[14] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *CoRR*, vol. abs/1409.3215, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215

[15] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[17] B. Pokharel, “Nepali brihat sabdakosh,” 2075, *ark:/13960/t2c91g80m*.