PURBANCHAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING
KHWOPA ENGINEERING COLLEGE
LIBALI-08, BHAKTAPUR

A PROPOSAL REPORT ON

Virtual Assistance for Nepali Law

Project work submitted in partial fulfillment of the requirements for the degree of Bachelor of Engineering in Computer Engineering(Eight Semester)

Submitted by

Anit Kumar Shahi	(760303)
Ashal Upreti	(760307)
Jiban Rawal	(760318)
Madan Acharya	(760321)

Khwopa Engineering College
Libali-08, Bhaktapur
March 28, 2024

Abstract

The "Virtual Assistance for Nepali Law" project, detailed in this report, represents the culmination of our group's efforts for our final year project. Over the course of the next six months, we aim to develop a comprehensive tool to aid individuals, particularly students and researchers, in efficiently searching and generating results. Built upon the foundations of LLM and Langchain models, supported by Whisper and Transformer systems, the project seeks guidance from platforms such as Hugging Face and Generative AI tools. Initially focusing on Nepal law and the Constitution of Nepal, the project's scope will expand to encompass various fields including health, education, and natural geography. Our ultimate objective is to create robust learning, researching, and advising tools accessible across diverse knowledge domains, with a specific focus on tailoring educational assistance to the needs of students.

Keywords:*Generative AI,HuggingFace,LangChain, LLM,Transformer*

Contents

List of Figures

List of Abbreviation

Abbreviations	**Meaning**
GPU	Graphical Processing uni
GUI	Graphical User Interface
NMS	Non Maximum Suppression
RCNN	Region Based Convolutional Neural Network
RTX	Ray Tracing Texel eXtreme
Rcll	Recall
SORT	Simple Online Real time Tracking
SSD	Single shot Detection
TTS	Text To Speech
UML	Unified Modelling Language

Chapter 1

Introduction

1.1 Background Introduction

Constitutions and laws are essential for governing the systems of any democratic nation. Each country establishes its own set of regulations and legal frameworks, whether written or unwritten. Adherence to the rules and obligations outlined in the constitution is crucial for maintaining societal order. However, many citizens often lack awareness of even the fundamental rules, obligations, responsibilities, and rights conferred by federal and service regulations. As technologically adept students, we recognize the need for digital transformation and efficient communication systems that facilitate seamless interaction between people and government authorities. Through user-friendly tools, we aim to ensure that individuals remain well-informed about legal provisions, rights, and constitutional mandates. Our primary focus is to leverage Generative AI technology to enable easy access to information and facilitate natural communication responses. This tool will adeptly address inquiries related to Nepali policies and the constitution in a detailed and creative manner.

Initially, our model will function as a content generator and knowledge repository system pertaining to law and constitution. However, we envision expanding its scope to encompass fields such as education and healthcare. This project represents our modest attempt to harness the innovative capabilities of Large Language Models and Generative AI for intelligent, natural communication.

Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. Language models analyze bodies of text data to provide a basis for their word predictions. They are used in natural language processing (NLP) applications, particularly ones that generate text as an output. Some of these applications include , machine translation and question answering. Some common statistical language model are:

1.1.1 N-gram:

N-grams are a relatively simple approach to language models. They create a probability dis- tribution for a sequence of n The n can be any number, and defines the size of the "gram", or sequence of words being assigned a probability. For example, if n = 5, a gram might look like this: "can you please call me." The model then assigns probabilities using sequences of n size. Basically, n can be thought of as the amount of context the model is told to consider. Some types of n-grams are unigrams, bigrams, trigrams and so on.

1.1.2 Continuous Space:

This type of model represents words as a non-linear combination of weights in a neural network. The process of assigning a weight to a word is also known as word embedding. This type becomes especially useful as data sets get increasingly large, because larger datasets often include more unique words. The presence of a lot of unique or rarely used words can cause problems for linear model like an n-gram. This is because the amount of possible word sequences increases, and the patterns that inform results become weaker. By weighting words in a non-linear, distributed way, this model can "learn" to approximate words and therefore not be misled by any unknown values. Its "understanding" of a given word is not as tightly tethered to the immediate surrounding words as it is in n-gram models.

1.2 Motivation

Many societal issues stem from a lack of knowledge about basic rights, responsibilities, policies, and provisions. Citizens find it cumbersome to sift through lengthy textbooks or PDFs to understand these concepts. To address this, our project aims to create a creative and generative chatbot that promptly responds to queries regarding policies and provisions. This solution saves time and helps prevent issues arising from simple negligence, providing users with quick and accurate answers to their policy-related inquiries.

1.3 Problem Definition

Each citizen of the country is required to fulfill certain obligations outlined by national policies and provisions. Likewise, the government endeavors to provide services and enable the exercise of citizens' rights. At both national and local levels, conflicts and disputes often arise due to misunderstandings and disregard for policies. Therefore, to foster a system of good governance, it is imperative that everyone understands and adheres to the established rules and regulations. This objective can be efficiently achieved through the utilization of digital assistance tools accessible via mobile devices and various other digital platforms. From the general public to legal practitioners, individuals can utilize these generative assistance tools to seek guidance and support.

1.4 Objectives

To develop a Nepali response generator to help the citizen to be aware and update about the government policy, law and provisions.

1.5 Scope and Applications

The major scope of the project is to:

- be used by citizen to get information about policy and provision.

Chapter 2

Literature Review

2.1 A Neural Probabilistic Language Model

2.1.1 Introduction

Goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the curse of dimensionality. NPLM proposes to fight the curse of dimensionality by learning a distributed representation of words which allows each training sentence to inform the model about an exponential number of seman- tically neighboring sentences. The model learns simultaneously a distributed representation for each word along with the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar to the words forming an already seen sentence. [1] The idea of the proposed model can be summarized as follows

- associate with each word in the vocabulary a distributed word feature vector (a real-valued vector in IR^m, where m is the size of embedding vector.)
- express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence
- learn simultaneously the word feature vectors and the parameters of that probability function

2.1.2 NLPM Architecture

The basic form of the model is shown in Figure 2.1. The objective is to learn the function $f(w, w_{t-1}, w_{t-n}, t) = P(w|w_{t-1})$, in the sense that it gives high out-of-sample likelihood.

A mapping C from any element of V to a real vector $C(i) \in \mathbb{R}^m$. It represents the distributed feature vector associated with each word in the vocabulary. In practice, C is represented by a $|V| \times m$ matrix (of free parameters).

From the direct architecture Figure 2.1,
$f(i, w_{t-1}, w_{t-2}, ..., w_{t-n}) = g(i, C(w_{t-1}), C(w_{t-2}), ..., C(w_{t-n}))$. Softmax function is used in the output layer of the neural network to get the probability of the target word.

The function f is the composition of two mappings (C and g), with C being shared across all words in the context. The function g may be implemented by a feed-forward or recurrent neural network or another parameterized function, with parameters θ.

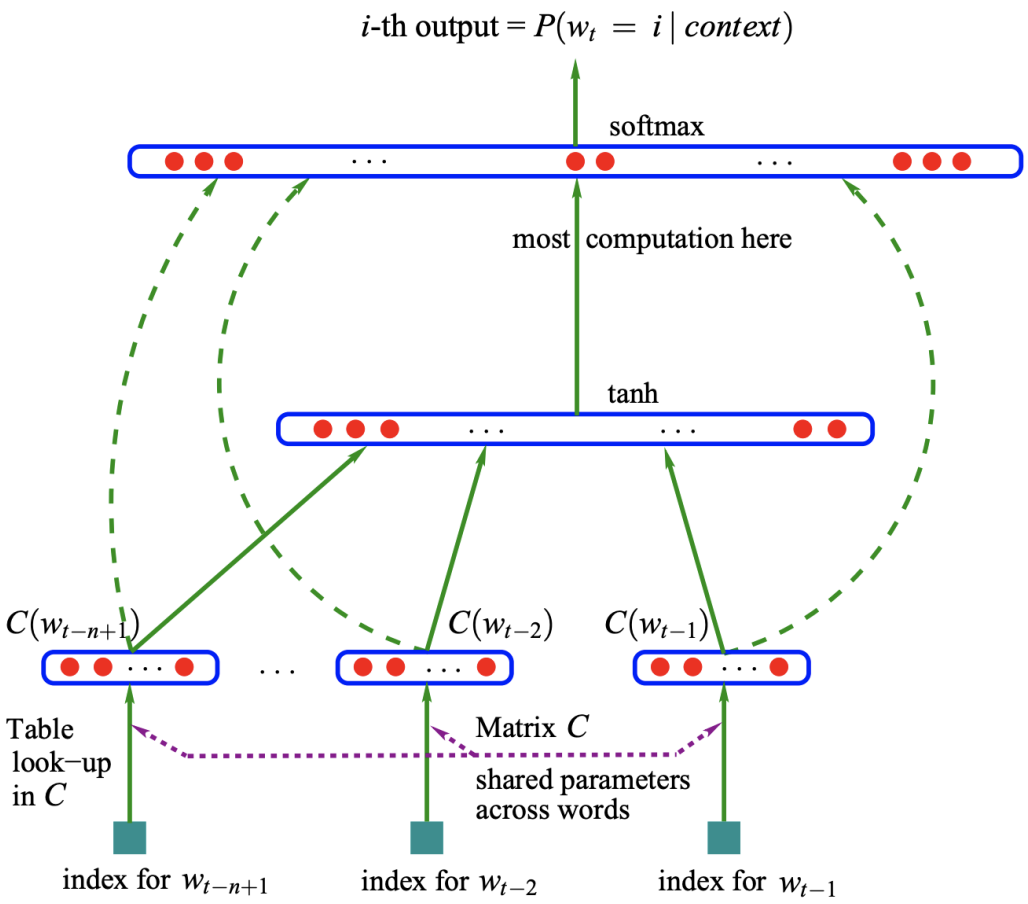

Figure 2.1: Direct Architecture of Probabilistic Language Model

2.1.3 Conclusion and Result

The main result of this experiment is that the neural network performs much better than the smoothed trigram. Greater the length of the context words and higher the number of hidden units, the model becomes more efficient. Moreover, direct architecture was found to be better by about 2It can be deduced that the neural probabilistic model performs better due to the advantage of the learned distributed representation to fight the curse of dimensionality.

2.2 Attention is all you need

2.2.1 Introduction

Before the introduction of the transformer, the dominant sequence transduction models were based on the complex recurrent or convolutional neural networks that include an encoder and a decoder. But this paper introduced a new network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Attention mechanisms have become an integral part of compelling sequence modeling and trans- duction models in various

tasks, allowing the modeling of dependencies without regard to their distance in the input or output sequences. In all but a few cases, however, such attention mech- anisms are used in conjunction with a recurrent network. [2]

2.2.2 Transformer model architecture

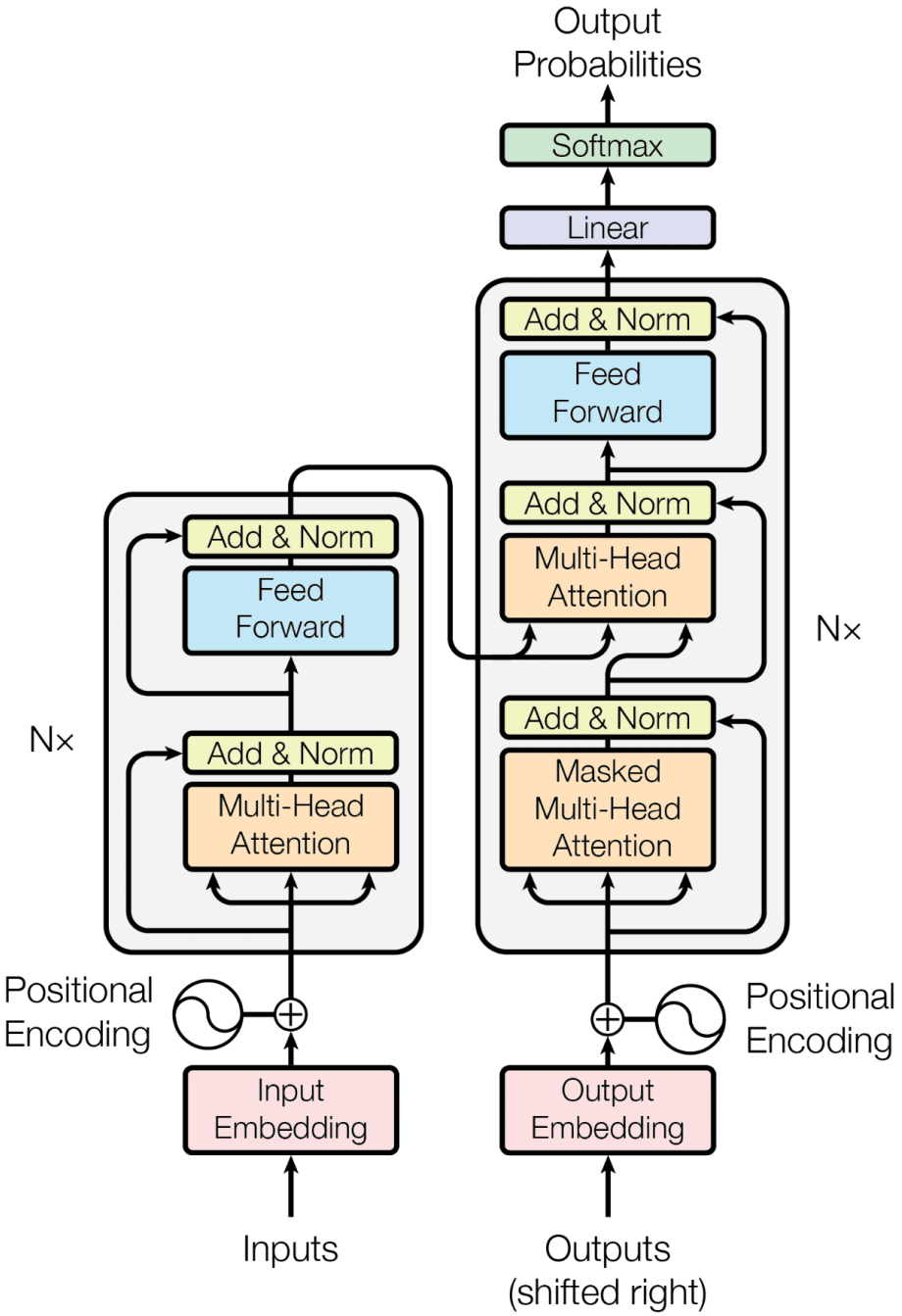

Figure 2.2: The Transformer - model architecture

Encoder and Decoder
The encoder is composed of a stack of N = 6 identical layers. Each layer has

two sublayers: multi-head self-attention mechanism, and the simple position wise fully connected feed-forward network. Residual connection around each of two sub layers, followed by a layer normalization is employed in the encoder. The decoder is also composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections around each of the sub layers, followed by layer normalization is employed. Self attention sub-layer in the decoder stack is modified to prevent the positions from attending to subsequent positions.

Attention
An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

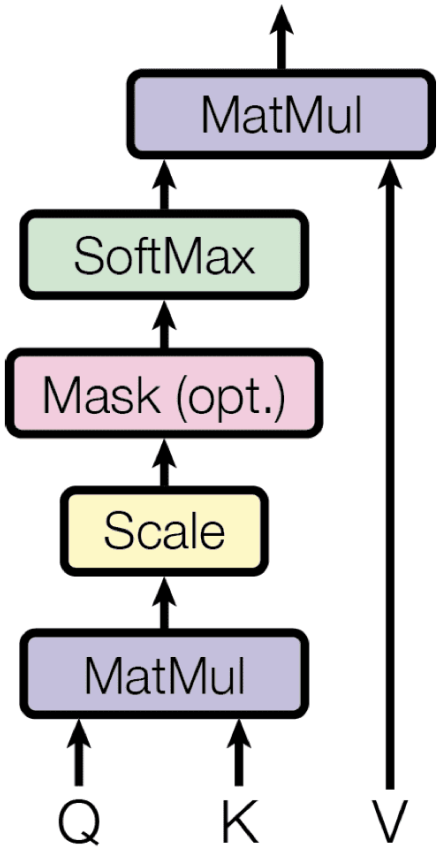

Figure 2.3: Scaled Dot-Product Attention

In scaled dot-product attention, the input consists of queries and keys of dimension dk and values of dimension dv. We compute the dot products of the query with all keys, divide each by dk and apply a softmax function to obtain the

weights on the values.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

In multi-head attention, instead of performing a single attention function with d_{model}-dimensional keys, values, and queries, we found it beneficial to linearly project the queries, keys, and values h times with different, learned linear projections to d_k, d_k, and d_v dimensions, respectively. On each of these projected versions of queries, keys, and values, we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in figure 2.4.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W_o \tag{2.2}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

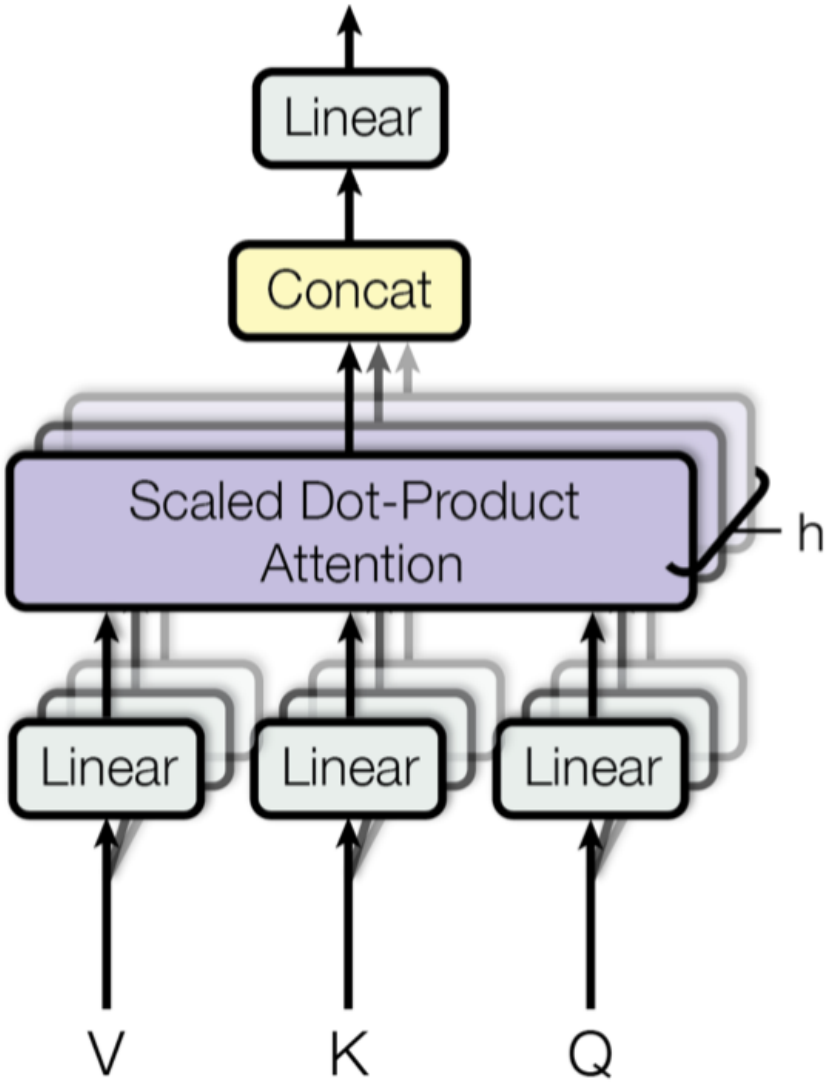

Figure 2.4: Multi-Head Attention consists of attention layer running in parallel

Position wise Feed Forward Neural Network
In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically.
This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.3}$$

Embedding and Softmax
Similarly to other sequence transduction models, Learned embeddings are used to convert the input tokens and output tokens of dimension dmodel. Usual learned linear transformation and softmax function are used to convert the decoder output to predict next-token probabilities.

Positional Encoding
Since the model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, some information about the relative or absolute position of the tokens in the sequence must be injected. To this end, "positional encodings" are added to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension model as the embeddings, so that the two can be summed. Sine and cosine functions of different frequencies are used for positional encoding as follows:

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{(2i/d_{\mathrm{model}})}}\right) \tag{2.4}$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{(2i/d_{\mathrm{model}})}}\right) \tag{2.5}$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid.

2.2.3 Conclusion and Result

In the machine translation task, the transformer model outperformed all previous state of the art models which have used the RNNs, GRUs and LSTMs. To evaluate if the Transformer can generalize to other tasks, experiments on English constituency parsing were performed and de- spite the lack of task-specific tuning, the transformer model performed surprisingly well, yielding better results than all previously reported models with the exception of the Recurrent Neural Network Grammar.

In this work, the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi- headed self-attention is presented.

Till today, various transformer models have been developed which have performed better on all kinds of natural language processing tasks like language modeling, text classifications, questions answering, machine translation, sentence similarity, summarization, etc.

Chapter 3

Requirement Analysis

3.1 Software Requirement

Software requirement for our prepared system includes:

1. Programming Language : Python, Javascript
2. Libraries : Tensorflow, keras, scikit-learn, spacy, nltk, and other python libraries
3. Framewoks : Django, Django Rest Framework, React
4. Application software : Postman
5. Target Platform : Web

3.2 Hardware Requirement

Our prepared system of virtual assistant requires following hardware requirements:

1. Android Mobile for deployment.
2. For development, laptop or desktop of at least RAM 16GB and storage of 250GB is required.

3.3 Functional Requirement

1. **R1: Word Embedding Visualization**

   Description

   Word embeddings of nepali words created using various methods like word2vec, embedding layers should be visualized in 2d and 3d graphs and show the required relationship between the words.

   (a) **Visualize word embedding**

   Input : Select word embeddings visualization tab

   Output : 2d and 3d graphs of the word embeddings vector of few selected words or default 100 or 200 words(Number of words may vary as per requirements).

   Processing : All the default word embeddings vectors will be returned and plotted in 2d and 3d graphs.

(b) **Search Words**

Input : Keyword to be searched in the graph

Output: Vector of keyword and position in the graphs highlighted

Processing: Keyword searched in the vocabulary and if found its vector is returned and plotted with highlight color in the graph. If not found, display NOT FOUND message. If the keyword is not found in the vocabulary, archive it to the vocabulary database for future references.

2. **R2: Sentiment Classification**

Description

When the user enters a sentence and performs sentimental classification, the sentimental classification should be able to distinguish the sentence into positive, negative and neutral with some cutoff probability (about 70%).

(a) **Classify the result**

Input : A sentence

Output : Probabilities of the sentence being positive, negative and neutral.

Processing :

(b) **Store the result**

Input : A sentence to be classified

Ouput : Response message for successful storage

Processing : Store the sentence and its predicted label in the sentimental classification table. If the predicted label is incorrect , get the correct label from the user and store it in the database.

3. **R3: Language Modelling**

Description

Nepali Language modeling based on both probabilistic and neural approach. Based on the trained language model, it should be able to auto generate the next word given context words.

(a) **Predict next word**

Input : An incomplete sentence

Output : List of words along the probabilities of being the next word. Processing : The language model should be able to predict the next word from vo- cabulary based on the previous words provided by the user.

4. **R4: Spelling Correction**

   Description

   Based on the trained language model, the system should suggest correct spelling for a given sentence.

   Input : A sentence

   Output : Provide suggestions to correct the spelling of the words in the input sentence.

   Processing : Firstly, candidate sentences are found using 1 or 2 minimum edit distance. Probability of each candidate sentence and likelihood of error of given sentence is calculated. And suggestions should be made based on maximum posterior.

5. **Interactive Web Application**

   Description

   All the requirements R1, R2, R3 and R4 should be easily accessible through a website and APIs.

3.4 Non-Functional Requirement

These are essential for the better performance of the system. The points below focus on the non-functional requirement of the system prepared.

1. **R1: Performance and Scalability**

   (a) The load time for the user interface screens shall take no longer than 3 seconds.

   (b) Queries shall return results within 3 seconds.

2. **R2: Design Constraints**

   (a) The system shall be developed using python and postgresql databases.

3. **R3: Standard Compliance**

(a) The graphical user interface shall have a consistent look and feel

4. **R4: Availability**

   (a) The system shall be available all time.

5. **R5: Portability and Compatibility**

   (a) The system shall be able to run in all browsers.

6. **R6: Reliability and Maintainability**

   (a) The mean time to recover from a system failure must not be greater than 2 hours.

   (b) Rate of system failure should not be greater than twice a month.

Chapter 4

Feasibility Study

Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software.

4.1 Technical Feasibility

Technical Feasibility is the formal process of assessing whether it is technically possible to man- ufacture a product or service. For our system, the required software and hardware were readily available to us. So we can state with confidence that the project is technically feasible.

4.2 Operational Feasibility

Once the full-fledged chatbot is developed, it will require proper management and maintenance to ensure consistent operation and servicing. Despite being a small-scale chatbot, it does not demand extensive resources for service operation. However, stable internet connectivity and server performance are imperative for enhancing user experience and productivity. Additionally, operational overhead is a crucial aspect to consider, particularly its impact on devices like mobile phones or other digital systems. At times, it inadvertently elevates network traffic, leading to adverse consequences.

4.3 Economic Feasibility

The purpose of an Economic Feasibility Study (EFS) is to demonstrate the net benefit of a proposed project, taking into consideration the benefits and costs to the agency, other state agencies, and the general public as a whole. It is the process of determining whether the project is worth the cost and time investment. Since most of the hardware was already in our possession and the softwares used was mostly free, we can declare that the system is economically feasible.

4.4 Legal Feasibility

Since we are using open source data for building language model, we can state that the project is legally feasible.

4.5 Scheduling Feasibility

This assessment is the most important for project success; after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete. Since we have properly planned our approach to completing the project in components using agile model, we can say that it is feasible time-wise.

Chapter 5

Methodology

5.1 Agile Method as Software Development Model

Teams use the agile development methodology to minimize risk (such as bugs, cost overruns, and changing requirements) when adding new functionality. In all agile methods, teams develop the software in iterations that contain mini-increments of the new functionality. There are many different forms of the agile development method, including scrum, crystal, extreme programming (XP), and feature-driven development (FDD).

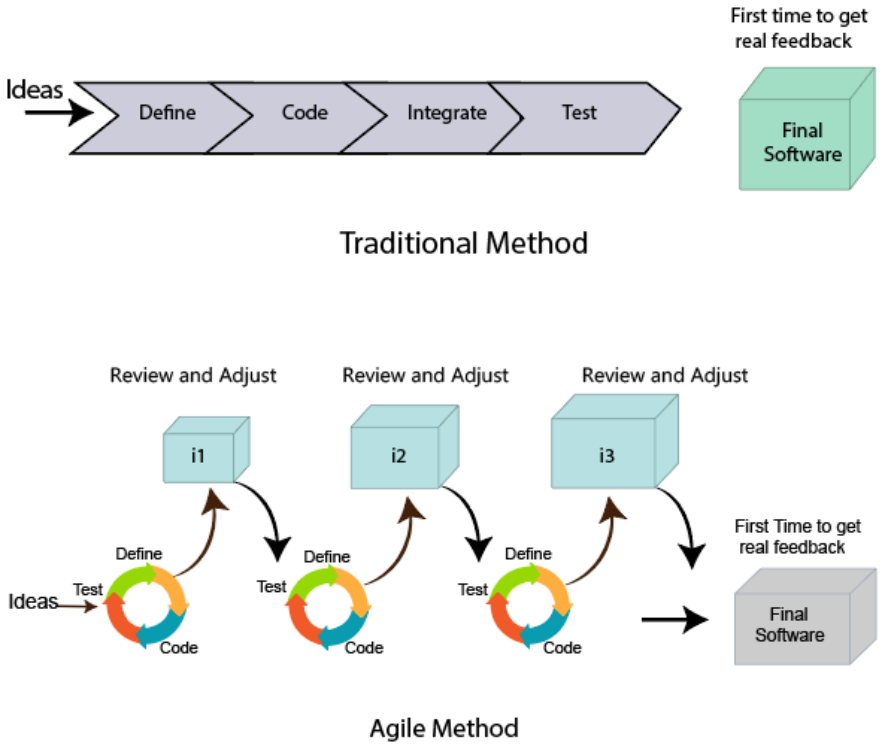

Figure 5.1: Agile Model as Software Development Model

5.2 Overall Phase Followed

The overall project has been completed in three main phase which are:

a. Planning Phase

b. Development Phase

c. Integration

5.2.1 Planning Phase

In planning phase, necessary works to be done and planning of overall project were discussed and the decision were documented for further procedures.
First the project was divided into four parts:

a. Input processing

b. Model analysis

c. Model development

d. Integration and UI/UX development

These parts were then assigned to each project members who then studied about the respective parts in detail.

5.2.2 Development Phase

In this phase, the divided parts were well studied and developed. Then each of those developed parts were tested separately.

5.2.3 Integration

In this phase, the separately developed parts were integrated to form a system and integration testing was done.

5.3 Gantt Chart

A gantt chart structure is given below:

S.N	Weeks / Job Description	2nd Week	4th Week	6th Week	8th Week	10th Week	12th Week	14th Week	16th Week
1	Problem Identification								
2	Problem Analysis								
3	Design								
4	Coding or Developing								
5	Testing and Debuging								
6	Documentation								

Figure 5.2: Gantt Chart

5.4 Neural Probabilistic Language Model

5.4.1 Introduction

Language model is used to estimate the probability of the word sequences and to estimate the probability of a word following the sequence of words. The concept can be used to auto-complete a sentence with most likely suggestions as shown below:

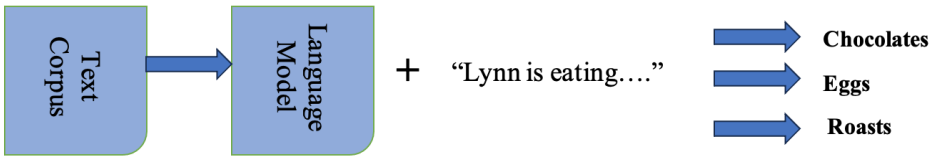

Figure 5.3: Language model for autocomplete

Some other applications of the language model are:

1. Speech Recognition
2. Spelling Correction
3. Augmentative Communication

5.4.2 N-gram

A N-gram is a sequence of N-words. For example:

Corpus : I am happy because I am learning.

Uni-grams : {I, am, happy, because, I, am, learning}

Bi-grams : {I am, am happy, happy because, . . .}

Tri-grams : {I am happy, am happy because, . . . }

5.4.3 Sequence Notation

Let's define a standard sequence notation to represent the sequence of words in our corpus. Let our corpus be as shown below where n^{th} is represented as W_n. Let the number of words in the corpus be m = 500

Corpus: This is great. ... teacher drinks tea.

Hence, W_1 = This, W_2 = is, W_3 = great and so on...

Let us represent the sequence of words as W_s^e where s represent the starting index and e represent the ending index of the words in the corpus. So,

$W_1^m = W_1 W_2 W_3 \dots W_{m-1} W_m$

Similarly, $W_1^3 = W_1 W_2 W_3$ = "This is great".

5.4.4 N-gram probability

N-gram probability is the probability of the occurrence of the sequence of N-words in a corpus. From uni-gram, bi-gram and tri-gram probability, we can deduce for N-gram probability denoted $P(W|W_1^{N-1})$ by

$$P(W|W_1^{N-1}) = \frac{C(W_1^{N-1} W_N)}{C(W_1^{N-1})} \tag{5.1}$$

5.4.4.1 Uni-gram probability

Uni-gram Probability is the probability of occurrence of a word in a given corpus.
Corpus : I am happy because I am learning.

Size of corpus: m=7

$$P(I) = \frac{2}{7} \tag{5.2}$$

$$P(happy) = \frac{1}{7} \tag{5.3}$$

Hence, we can see that Probability of uni-gram is given by:

$$P(w) = \frac{C(w)}{m} \tag{5.4}$$

where, C(w)= frequencey of the word in the corpus.

5.4.4.2 Bi-gram Probability

Bi-gram Probabiity is the probability of occurrence of sequence of two words in a corpus. **Corpus:** I am happy because I am learning. **Size of corpus:** m=7

$$P(am|I) = \frac{C(Iam)}{C(I)} = \frac{2}{2} = 1 \tag{5.5}$$

$$P(happy|I) = \frac{C(Ihappy)}{C(I)} = \frac{0}{7} = 0 \tag{5.6}$$

$$P(learning|am) = \frac{C(amlearning)}{C(am)} = \frac{1}{2} \tag{5.7}$$

Here, we can see that Probability of bi-gram is given by:

$$P(y|x) = \frac{C(xy)}{x} \tag{5.8}$$

Where C(w)= frequency of the word in the corpus

5.4.4.3 Tri-gram Probability

Tri-gram Probability is the probability of the occurrence of the sequence of three words in a corpus. **Corpus** : I am happy because I am learning. **Size of corpus**: m=7

$$P(happy|Iam) = \frac{C(Iamhappy)}{C(Iam)} = \frac{1}{2} \tag{5.9}$$

Hence, we can see the Probability of bi-gram is given by:

$$P(W_3|W_1^{N2}) = \frac{C(W_1^2 W_3)}{C(W_1^2)} \tag{5.10}$$

Where, C(w)= frequency of the word in the corpus

5.4.5 Probability of a Sequence

From the conditional probability and chian rule, we have

$$P(B/A) = \frac{P(A,B)}{P(A)} \tag{5.11}$$

from 5.11, we get

$$P(A,B) = P(A).P(B/A) \tag{5.12}$$

Using 5.12, we can deduce

$$P(A,B,C,D) = P(A).P(B/A).P(C/A,B).P(D/A,B,C) \tag{5.13}$$

Now, the probability of the sequence of words: "the teacher drinks tea" is given by:

$$P(theteacherdrinkstea) = P(the).P(teacher/the).P(drinks/theteacher).P(tea/theteacherdrin \tag{5.14}$$

5.4.6 Approximation of Sequence Probability

When using n-gram language model, the exact sentence for the required n-gram might not be present exactly in corpus due to which the frequency of the words sequence becomes zero and hence the probability. For example:

Input: the teacher drinks tea

P(the teacher drinks tea) = P(the) ·P(teacher|the)·P(drinks|the teacher)·P(tea|the teacher drin

Here,

$$P(\text{tea}|\text{the teacher drinks}) = \frac{C(\text{the teacher drinks tea})}{C(\text{the teacher drinks})} \tag{5.15}$$

In the above equation 5.15, the frequency of "the teacher drinks tea" and "the teacher drinks" are both likely 0.

Hence, in order to calculate the required probability of the sequence of the words, as per Markov assumption, only last N words matter. Suppose, only last one words matter then, we have

P(the teacher drinks tea) ≈ P(the)·P(teacher|the)·P(drinks|teacher)·P(tea|drinks)

Hence, we can model the entire sentence with n-gram following Markov Assumption as follows:

$$P(W_n|W_1^{n-1}) \approx P(W_n|W_{n-N+1}^{n-1}) \tag{5.16}$$

5.4.7 Starting and Ending Sentence

5.4.7.1 Start of sentence symbol $\langle s \rangle$

For an input sentence, "the teacher drinks tea", we have

P(the teacher drinks tea) ≈ P(the)·P(teacher|the)·P(drinks|teacher)·P(tea|drinks)

After, we add a start token $\langle s \rangle$, we have

Input: $\langle s \rangle$ the teacher drinks tea

P(the teacher drinks tea) ≈ P(the|$\langle s \rangle$)·P(teacher|the)·P(drinks|teacher)·P(tea|drinks)

In this way, we can add start token to determine the possible starting words and calculate the probability of sequence considering the first word in an N-gram language model. For Tri-gram and N-gram, we add $(N-1)$ start tokens $\langle s \rangle$ in the beginning of the sequence.

5.4.7.2 End of sentence symbol $\langle /s \rangle$

For an input sentence, "the teacher drinks tea", we have

P(the teacher drinks tea) ≈ P(the)·P(teacher|the)·P(drinks|teacher)·P(tea|drinks)

After, we add a start token $\langle s \rangle$ and an end token $\langle /s \rangle$, we have Input: $\langle s \rangle$ the teacher drinks tea $\langle /s \rangle$

P(the teacher drinks tea) ≈ P(the|$\langle s \rangle$)·P(teacher|the)·P(drinks|teacher)·P(tea|drinks)·P($\langle /s \rangle$|t

For bi-gram, tri-gram, or any N-gram model, we add the end token just once at the end of the sentence, unlike the start token.

5.4.8 Count Matrix

Count matrix is the matrix containing the frequency of occurrence of N-gram. Rows in the count matrix represent the unique corpus of $(N-1)$ gram and columns represent the unique corpus word.

Corpus: $\langle s \rangle$ I study I learn $\langle /s \rangle$

5.4.9 Probability Matrix

Probability matrix is the matrix containing the probability of the occurrence of N-gram words sequence. It can be created by dividing each element of the count matrix by the total sum of all row elements. For example:

Corpus: $\langle s \rangle$ I study I learn $\langle /s \rangle$

5.5 Sequence Modelling

5.5.1 Introduction

Sequence Modelling is the ability of a computer program to model, interpret, make predictions about or generate any type of sequential data, such as audio, text etc. For example, a computer program that can take a piece of text in English and translate it to French is an example of a Sequence Modelling program (because the type of data being dealt with is text, which is sequential in nature) [3]. An AI algorithm called the Recurrent Neural Network, is a specialised form of the classic Artificial Neural Network (Multi-Layer Perceptron) that is used to solve Sequence Modelling problems. Recurrent Neural Networks are like Artificial Neural Networks which has loops in them. This means that the activation of each neuron or cell depends not only on the current input to it but also its previous activation values

5.5.2 Recurrent Neural Networks

Humans don't start their thinking from scratch every second. As we read a text, we understand each word based on your understanding of previous words. We don't throw everything away and start thinking from scratch again. Our thoughts have persistence.

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

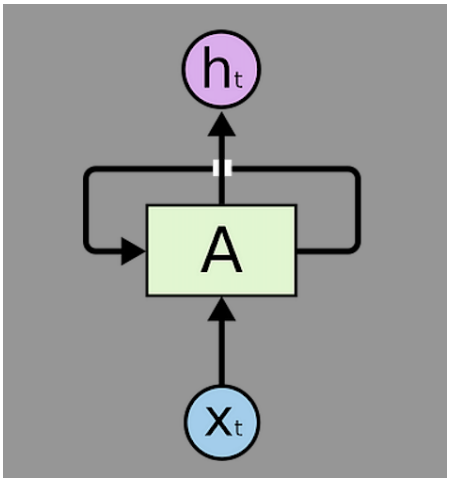

Figure 5.4: Recurrent Neural Networks

In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

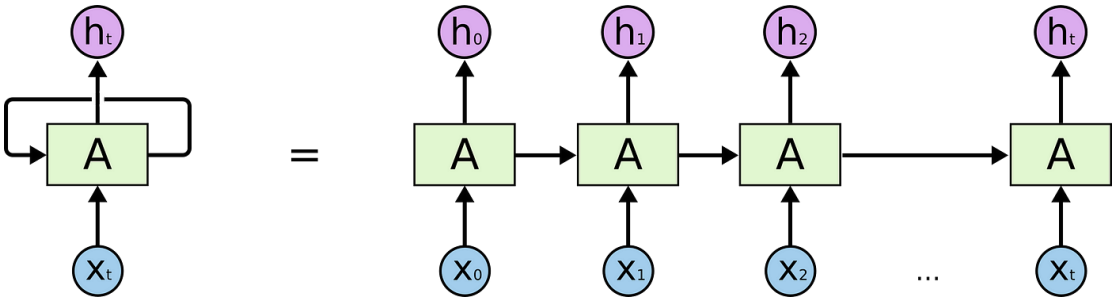

Figure 5.5: An unrolled recurrent neural networks

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data. And they certainly are used! In the last few years, there have been incredible success apply- ing RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning, etc.

5.5.3 Vanishing Gradients with RNNs

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they'd be extremely

useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context - it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.

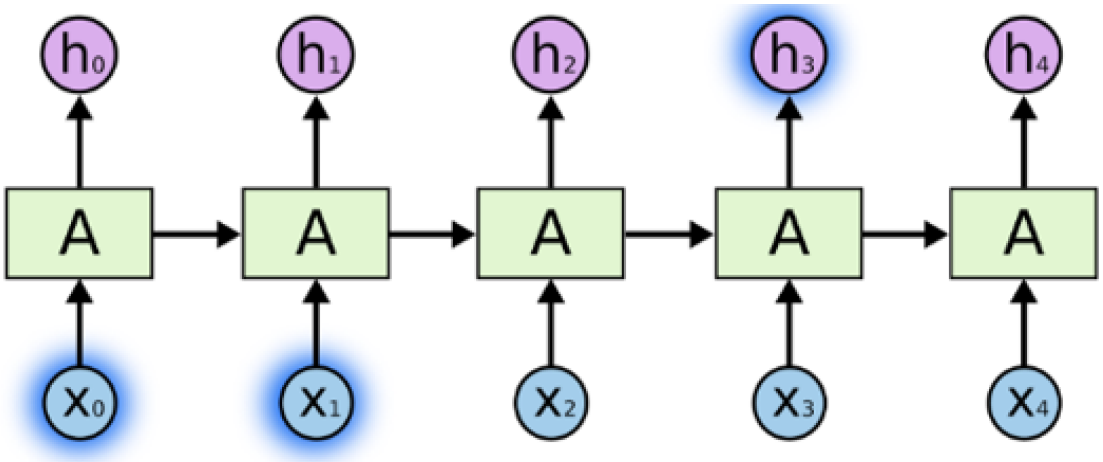

Figure 5.6: Short Sequence in RNNs

But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France. . . . I speak fluent French." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

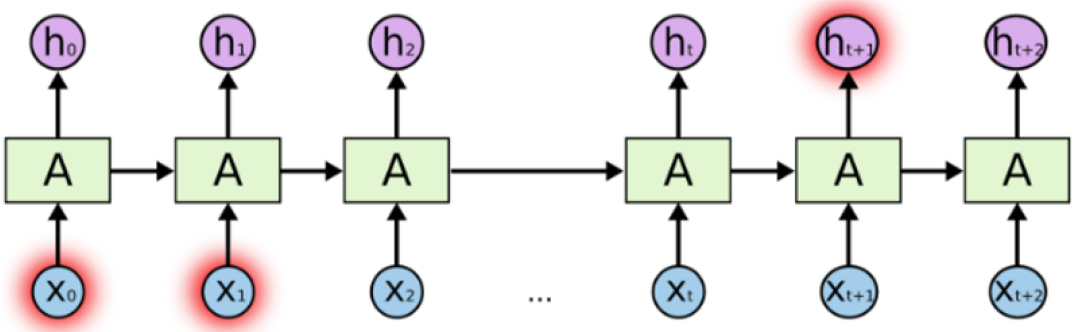

Figure 5.7: Caption

In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of

this form. Sadly, in practice,

RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.

5.5.4 LSTMs

Long Short Term Memory networks - usually just called "LSTMs" - are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmid- huber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used [3].

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering infor- mation for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer as shown below.

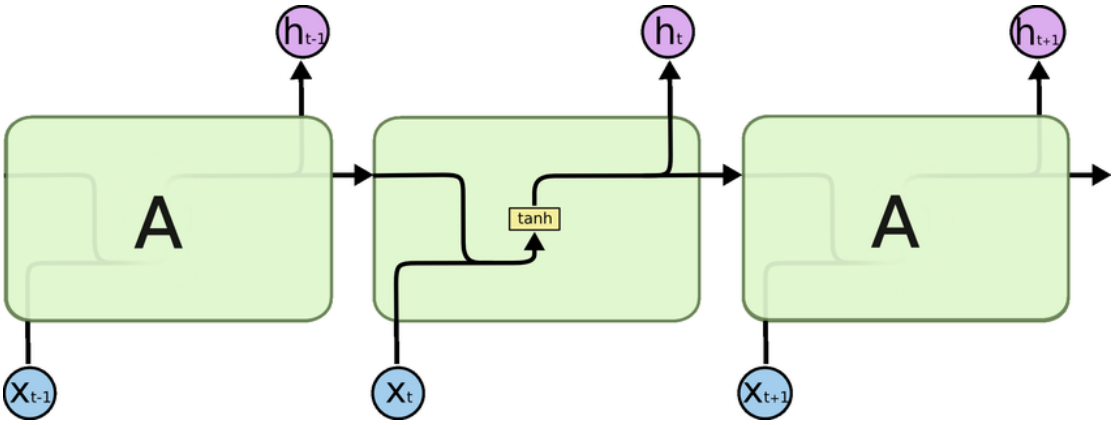

Figure 5.8: The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

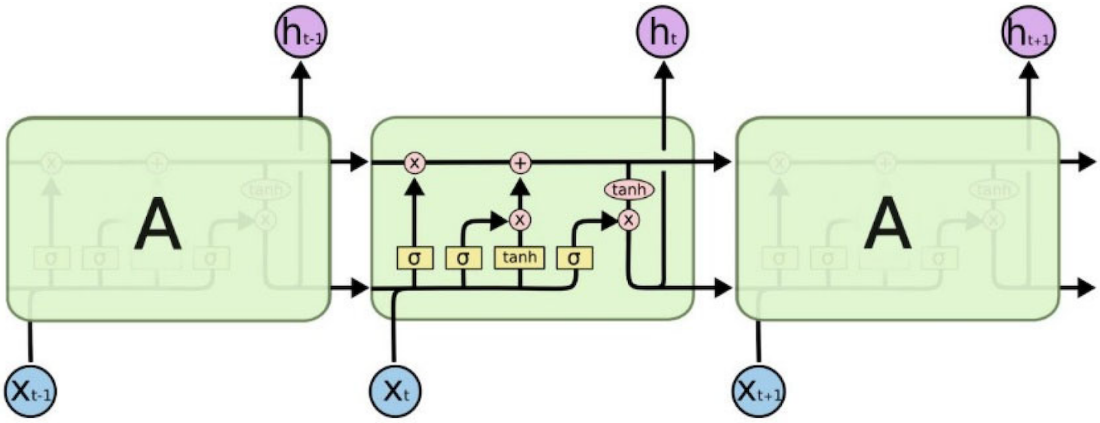

Figure 5.9: The repeating module in an LSTM contains four interacting layers.

5.5.5 LSTM Cell

The following figure shows the operations of an LSTM cell:

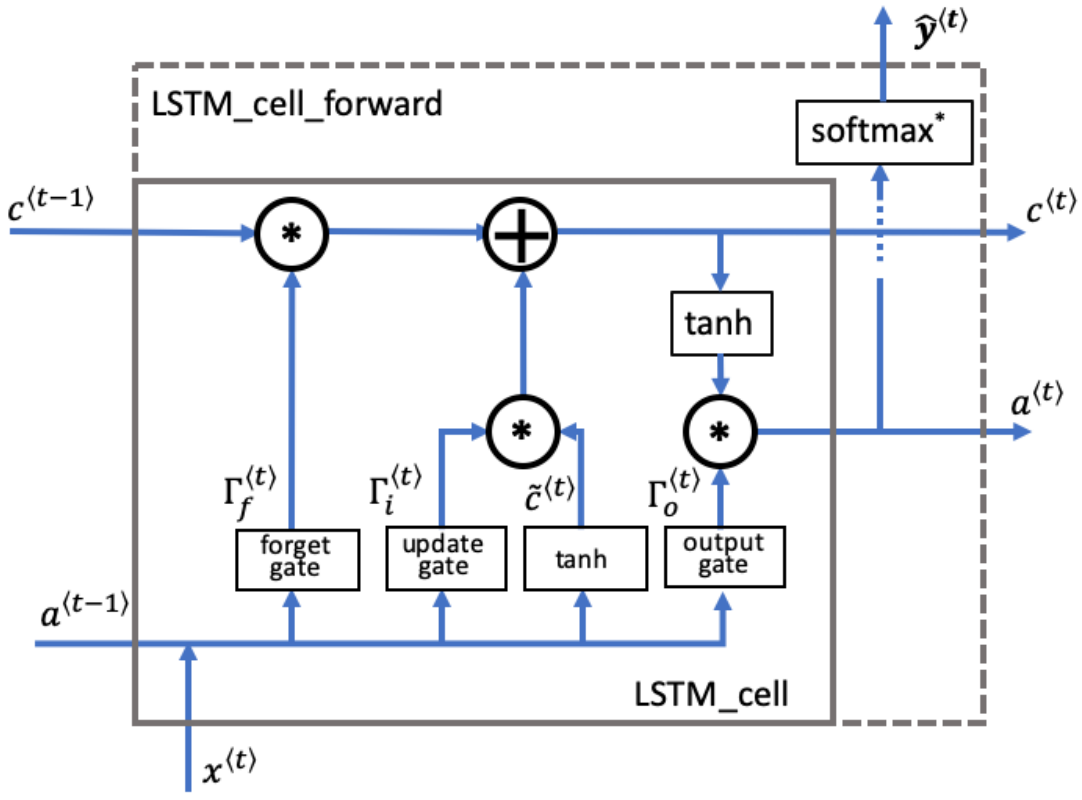

Figure 5.10: LSTM Cell

Forget Gate Γf

Assume, we are reading words in a piece of text and plan to use an LSTM to keep track of grammatical structures, such as whether the subject is singular("dog") or

plural("dogs). If the subject changes its state (from a singular word to a plural word), the memory of the previous state becomes outdated, so it is better to forget that outdated state.

The forget gate is a tensor containing values between 0 and 1. If a unit in the forget gate has a value close to 0, the LSTM will forget the stored state in the corresponding unit of the previous cell state. If a unit in the forget gate has a value close to 1, the LSTM will mostly remember the corresponding value in the stored state.

$$\Gamma_f^{<t>} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{5.17}$$

where,

- W_f = forget gate weight W_f
- b_f = forget gate bias b_f
- $\Gamma_f^{<t>}$ = Forget Gate

Explanation of the equation:

- W_f contains the weights that govern the forget gate's behavior.
- The previous time step's hidden state $a^{<t-1>}$ and current time step's input $x^{<t>}$ are concatenated together and multiplied by W_f .
- A sigmoid function is used to make each of the gate tensor's values $\Gamma_f^{<t>}$ range from 0 to 1.
- The forget gate $\Gamma_f^{<t>}$ has the same dimensions as the previous cell state $c^{<t-1>}$.
- Multiplying the tensors $\Gamma_f^{<t>} * c^{<t-1>}$ is like applying a mask over the previous cell state.
- If a single value in $\Gamma_f^{<t>}$ is 0 or close to 0, then the product is close to 0. This keeps the f information stored in the corresponding unit in $c^{<t-1>}$ from being remembered for the next time step.
- Similarly, if one value is close to 1, the product is close to the original value in the previous cell state. The LSTM will keep the information from the corresponding unit of $c^{<t-1>}$, to be used in the next time step.

Candidate Value $c^{<t>}$

The candidate value is a tensor containing information from the current time step that may be stored in the current cell state $c^{<t>}$. The parts of the candidate value that get passed on depend on the update gate.The candidate value is a tensor containing values that range from -1 to 1. The tilde "" is used to differentiate the candidate $c^{<t>}$ from $c^{<t>}$

$$c^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{5.18}$$

Update Gate Γ_i

Update gate can be used to decide what aspects of the candidate $c^{<t>}$ to add to the cell state $c^{<t>}$. The update gate is a tensor containing values between 0 and 1. When a unit in the update gate is close to 1, it allows the value of the candidate $c^{<t>}$ to be passed onto the hidden state $c^{<t>}$ When a unit in the update gate is close to 0, it prevents the corresponding value in the candidate from being passed onto the hidden state.

$$\Gamma_i^{<t>} = \sigma(W_i[a^{<t-1>}, x^{<t>}] + b_i) \tag{5.19}$$

Explanation of the equation:

- Similar to the forget gate, here $\Gamma_i^{<t>}$, the sigmoid produces values between 0 and 1.
- The update gate is multiplied element-wise with the candidate, and this product $\Gamma_f^{<t>} * c^{<t>}$ is used in determining the cell state $c^{<t>}$

Cell State $c^{<t>}$

The cell state is the "memory" that gets passed onto future time steps. The new cell state $c^{<t>}$ is a combination of the previous cell state and the candidate value.

$$c^{<t>} = \Gamma_f^{<t>} * c^{<t-1>} + \Gamma_i^{<t>} * c^{<t>} \tag{5.20}$$

Explanation of the equation:

- The previous cell state $c^{<t-1>}$ is adjusted (weighted) by the forget gate $\Gamma_f^{<t>}$.
- the candidate value $c^{<t>}$ is adjusted (weighted) by the update gate $\Gamma_i^{<t>}$.

Output Gate Γ_o

The output gate decides what gets sent as the prediction (output) of the time step. The output gate is like the other gates, in that it contains values that range from 0 to 1.

$$\Gamma_o^{<t>} = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \tag{5.21}$$

Explanation of the equation:

- The output gate is determined by the previous hidden state $a^{<t1>}$ and the current input x^t
- The sigmoid makes the gate range from 0 to 1.

Hidden State $a^{<t>}$

The hidden state gets passed to the LSTM cell's next time step. It is used to determine the three gates $(\Gamma_f, \Gamma_u, \Gamma_o)$ of the next time step. The hidden state is also used for the prediction $y^{<}t>$.

$$a^{<t>} = \Gamma_o^{<t>} * tanh(c^{<t>}) \tag{5.22}$$

Explanation of the equation:

- The hidden state $a^{<t>}$ is determined by the cell state $c^{<t>}$ in combination with the output gate Γ_o.
- The cell state state is passed through the tanh function to rescale values between -1 and 1.
- The output gate acts like a "mask" that either preserves the values of $tanh(c^{<t>})$ or keep those values from being included in the hidden state $a^{<t>}$.

Prediction $y_{pred}^{<t>}$

The prediction in this use case is a classification, so you'll use a softmax.

$$y_{pred}^{<t>} = softmax(W_y a^{<t>} + b_y) \tag{5.23}$$

5.6 Bidirectional RNNs

A typical state in an RNN (simple RNN, GRU, or LSTM) relies on the past and the present events. A state at time t depends on the states x_1, x_2, ..., x_{t1} and x_t. However, there can be situations where a prediction depends on the past, present, and future events.

For example, predicting a word to be included in a sentence might require us to look into the future, i.e., a word in a sentence could depend on a future event. Such linguistic dependencies are customary in several text prediction tasks.

Thus, capturing and analyzing both past and future events is helpful in the above-mentioned scenarios.

To enable straight (past) and reverse traversal of input (future), Bidirectional RNNs, or BRNNs, are used. A BRNN is a combination of two RNNs - one RNN moves forward, beginning from the start of the data sequence, and the other, moves backward, beginning from the end of the data sequence. The network blocks in a BRNN can either be simple RNNs, GRUs, or LSTMs [4].

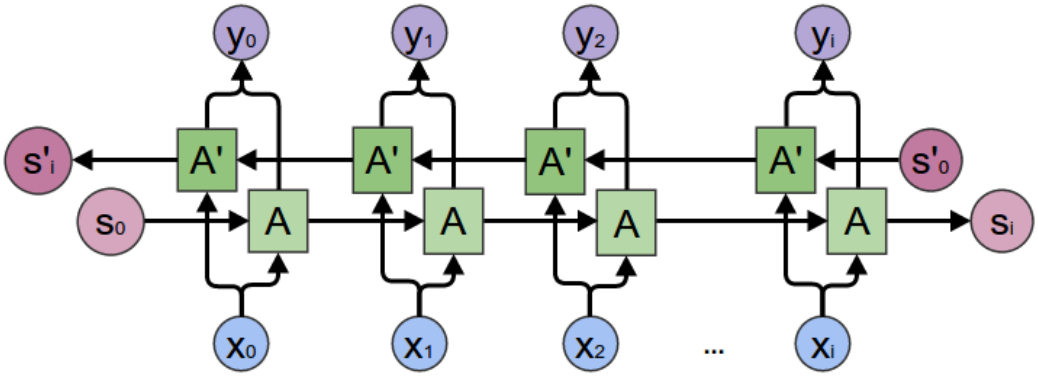

Figure 5.11: Bidirectional RNN

5.7 Sequence Modelling Application

Some aplications of sequence modelling are:

- **Language Modelling** : Auto-generation of next probable word.(Previous sequence of words are sequential data)
- **Image Captioning** (with the help of computer vision): Generating captions for images. (captions are sequential data)
- **Video Frame Prediction** (with the help of computer vision): Predict the subsequent frames of video given the previous ones. (the frames in a video are sequential in nature)
- **Classifying songs** (audio) as Jazz, Rock , Pop etc (genre). Here, audio is of sequential nature.
- **Composing Music** (music is sequential in nature)

Chapter 6

System Design and Architecture

6.1 Use Case Diagram

The Use Case Diagram of the prepared inference system.

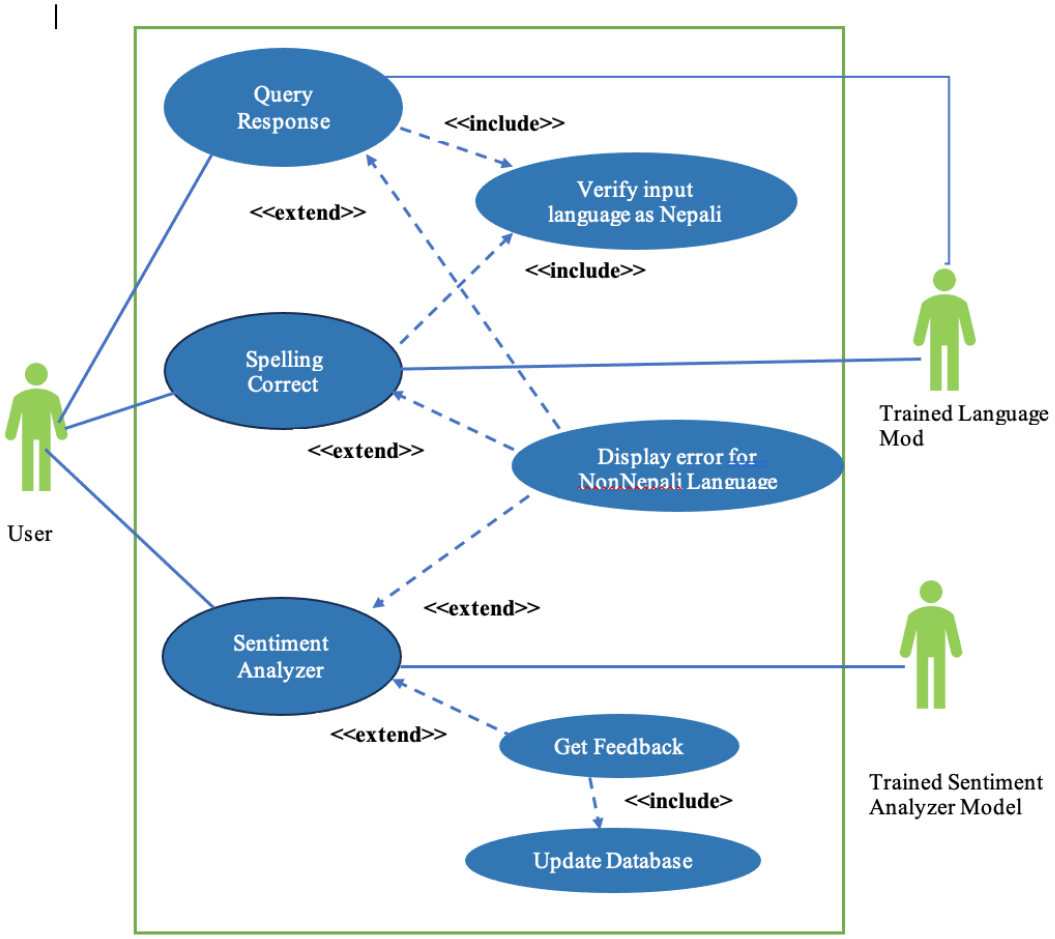

Figure 6.1: Use Case Diagram

6.2 Context Diagram

The Context Diagram shows the top level picture of the system.

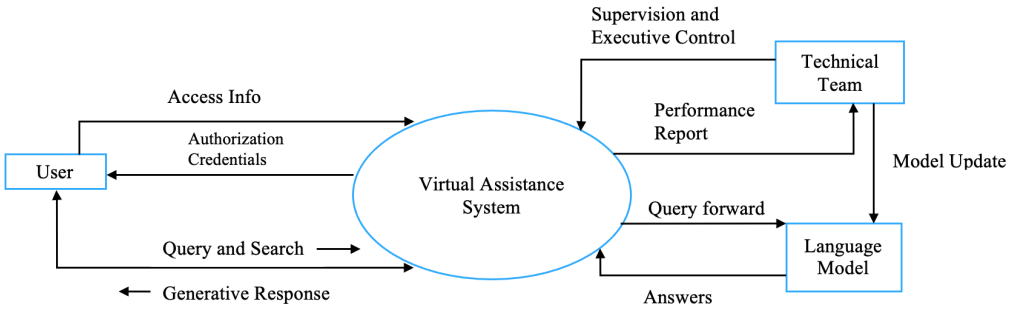

Figure 6.2: Context Diagram of Inference System

6.3 Data Flow Diagram

The Data Flow Diagram shows the flow of the data between the subsystem of the inference system. The level-1 data flow diagram of the inference system is shown below:

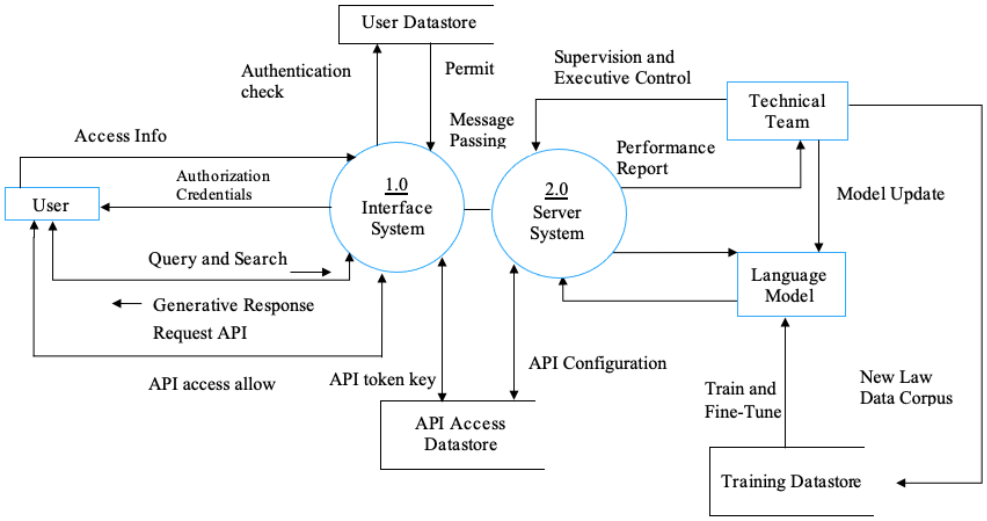

Figure 6.3: Level-1 Data Flow Diagram of Inference System

6.4 Sequence Diagram

The Sequence Diagram shows the flow of the process in between the subsystem. And the state when the subsystem are active and when the sub system are passive. The Sequence Diagram of the inference system is shown below:

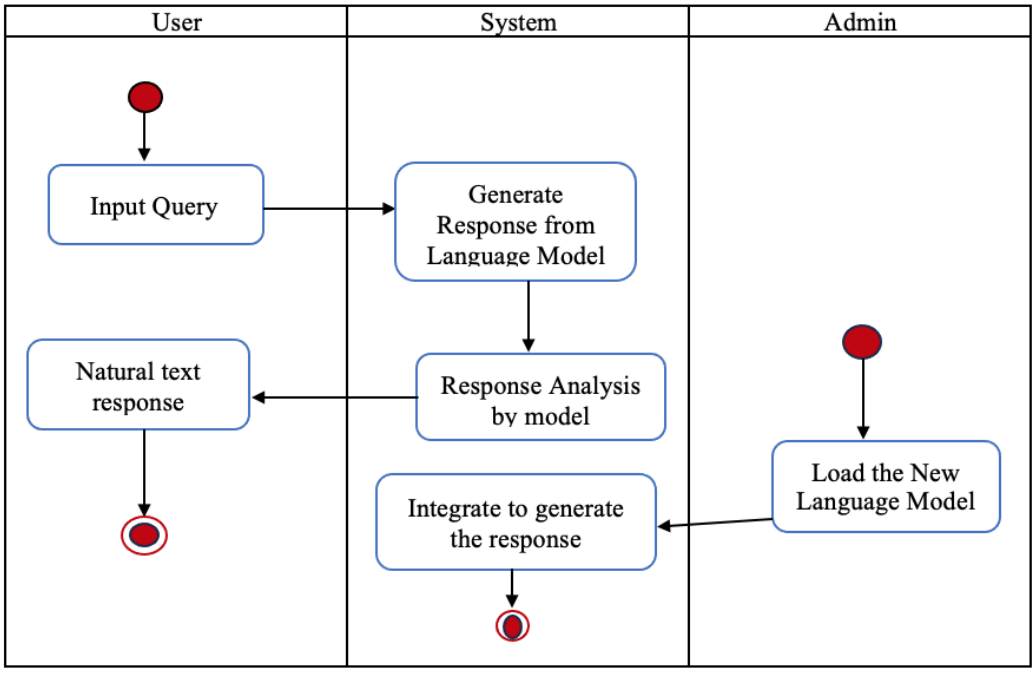

Figure 6.4: Sequence Diagram of Inference System

Chapter 7

Expected Outcome

7.1 Desktop Outcome Expected

We've expected outcome in following scenario:

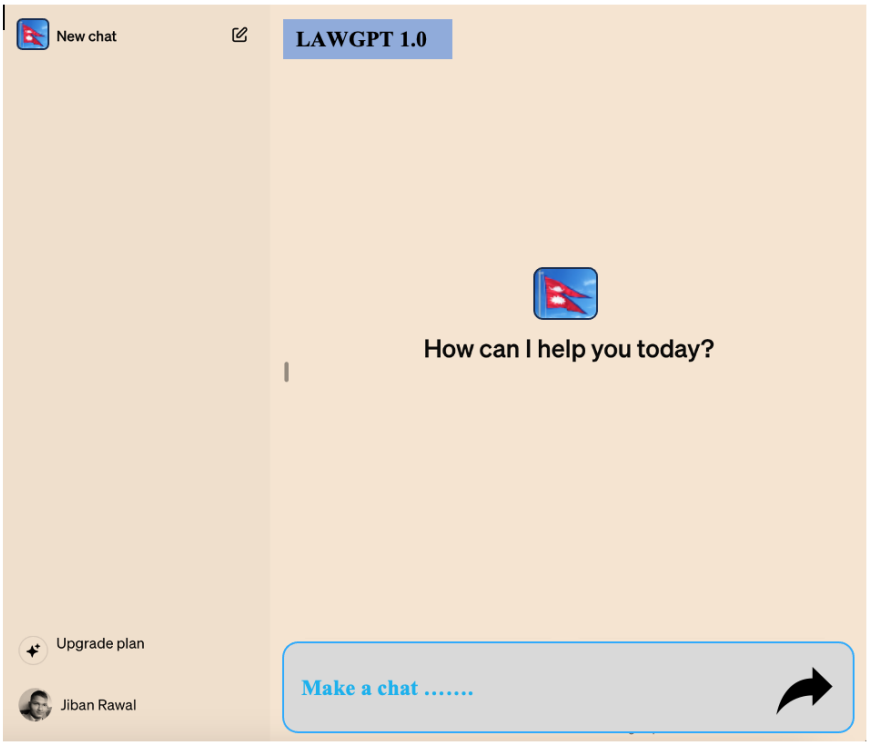

Figure 7.1: Desktop Interface

7.2 Mobile Outcome Expected

The expected mobile outcome was as in figure:

Bibliography

[1] N. L. Nabin Da Shrestha, Nirajan Bekoju, "Nepli language processing," *Pulchwok Engineering Conference*, 2023.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 6000–6010, 2017.

[3] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.