TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Project Report

On

‘Samadhan’ – A Local Government Service Query Chatbot

Submitted By:

Aarogya Bhandari (THA077BCT001)

Aayush Pokharel (THA077BCT002)

Piyush Luitel (THA077BCT031)

Prashant Bhusal (THA077BCT034)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

March, 2024

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Project Report

On

‘Samadhan’ – A Local Government Service Query Chatbot

Submitted By:

Aarogya Bhandari (THA077BCT001)

Aayush Pokharel (THA077BCT002)

Piyush Luitel (THA077BCT031)

Prashant Bhusal (THA077BCT034)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor’s Degree in Computer Engineering

Under the supervision of

Er. Bisikha Subedi

March, 2024

DECLARATION

We hereby declare that the report of the project entitled "**Samadhan – A Local Government Service Query Chatbot"** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Computer Engineering**, is a bonafide report of the work carried out by us. The materials contained in this report have not been submitted to any University or Institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

Aarogya Bhandari (Class Roll No.: 077/BCT/001) _____

Aayush Pokharel (Class Roll No.: 077/BCT/002) _____

Piyush Luitel (Class Roll No.: 077/BCT/031) _____

Prashant Bhusal (Class Roll No.: 077/BCT/034) _____

Date: March, 2024

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, a minor project work entitled "**Samadhan – A Local Government Service Query Chatbot"** submitted by **Aarogya Bhandari, Aayush Pokharel, Piyush Luitel** and **Prashant Bhusal** in partial fulfillment for the award of Bachelor's Degree in Computer Engineering. The Project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled and ready to undertake any related work to their field of study and hence we recommend the award of partial fulfillment of Bachelor's degree of Computer Engineering.

_____

Project Supervisor
Ms. Bisikha Subedi
Department of Electronics and Computer Engineering, Thapathali Campus

_____

External Examiner

_____

Project Co-ordinator
Mr. Umesh Kanta Ghimire
Department of Electronics and Computer Engineering, Thapathali Campus

_____

Mr. Kiran Chandra Dahal
Head of the Department,
Department of Electronics and Computer Engineering, Thapathali Campus

March, 2024

COPYRIGHT

ACKNOWLEDGEMENT

We would like to express our sincere appreciation to the Department of Electronics and Computer Engineering at the Institute of Engineering, Thapathali Campus, for the invaluable opportunity to enhance our programming skills and to engage with the fields of Natural Language Processing (NLP) and Artificial Intelligence (AI) during this project. We are particularly grateful to our supervisor, **Er. Bisikha Subedi**, for her expert guidance, which not only deepened our understanding of NLP and AI but also provided direction on proper project documentation.

Our heartfelt thanks also go to **Er. Dinesh Baniya Kshatri** and **Er. Umesh Kanta Ghimire**, whose insightful guidance and creative ideas were instrumental in shaping both the project and this report.

Aarogya Bhandari (THA077BCT001)

Aayush Pokharel (THA077BCT002)

Piyush Luitel (THA077BCT031)

Prashant Bhusal (THA077BCT034)

ABSTRACT

This project addresses the challenge of improving the efficiency of government services for Nepali citizens by developing a chatbot system named ‘Samadhan’. The chatbot can assist users with various services, such as citizenship and different certificates (birth, death, marriage, migration, divorce). The project uses the Rasa framework, a conversational AI platform with natural language processing features, to build the chatbot. The project follows a systematic method that includes data collection, analysis, chatbot design, implementation, and evaluation. The project aims to simplify and speed up the service process, improve the user experience, and ensure the chatbot’s effectiveness and quality.

Keywords: Chatbot, Gemini, Natural Language Generation, Natural Language Processing, Natural Language Understanding, Rasa.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
BERT	Bidirectional Encoder Representations from Transformers
ConveRT	Conversational Representations from Transformers
CRF	Conditional Random Field
CSS	Cascading Style Sheets
DAO	District Administration Office
DIET	Dual Intent and Entity Transformer
FAQ	Frequently Asked Questions
FFN	Feed Forward Network
GloVe	Global Vectors for Word Representation
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
NER	Named Entity Recognition
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
REST	Representational State Transfer
SDK	Software Development Kit
TED	Transformer Embedding Dialog
UI	User Interface

1 INTRODUCTION

1.1 Background

Accessing government services and information is a common need for citizens in the modern digital era. However, the conventional methods of seeking assistance, such as going to government offices physically, are often time-consuming and annoying. Since every citizen may require government services at some stage, the demand for efficient access is very high. In this situation, chatbots can be a beneficial tool. Many citizens have to get information or apply for various government services, such as citizenship, birth, death, marriage, migration, and divorce certificates. But these processes can be hard to follow, wasting time and causing confusion for some users.

Therefore, creating a chatbot that can respond to government service queries and offer user guidance can be a crucial factor in enhancing the overall user experience and satisfaction. The chatbot would be able to understand the natural language of the users, provide relevant and accurate information, and guide the users through the steps and procedures of accessing government services. The chatbot would also be able to handle exceptions, errors, and feedback from the users, and improve its performance over time. By doing so, the chatbot can help citizens access government services and information more easily and conveniently, and improve their overall satisfaction and trust.

1.2 Motivation

‘Samadhan’ focuses on positive change, driven by an unwavering commitment to improve the challenges faced by Nepali citizens in their interactions with government services. The current landscape, characterized by a lack of effective communication channels within government offices, leaves individuals seeking essential services in a state of perplexity. The lack of clear guidance on required steps, necessary documents, and expected timelines compounds the difficulties citizens encounter.

The driving force behind ‘Samadhan’ lies in the profound desire to reshape this narrative. Our project aspires to dismantle the barriers that hinder citizens from seamlessly accessing government services. At its core, ‘Samadhan’ aims to introduce an intelligent chatbot system that not only responds to government service queries but

also serves as a friendly guide, simplifying the entire application process. Motivated by the vision of empowering individuals, ‘Samadhan’ seeks to revolutionize the way citizens navigate the intricacies of government processes. By providing a user-friendly interface, we envision transforming frustration and disconnection into a seamless, efficient, and empowering experience. ‘Samadhan’ is not just a technological solution; it’s a commitment to building a centralized hub that acts as a trustworthy companion for citizens, ensuring they confidently access the services they need.

1.3 Problem Definition

In Nepal, people often find themselves facing a considerable challenge when dealing with government services, mainly because effective communication channels within government offices are notably lacking. Those trying to secure essential services are frequently left in the dark about the required steps, necessary documents, and the expected duration of the process. The absence of a central hub or clear guidance system means there’s no go-to resource for navigating the complexities of various government processes. Currently, information sharing is textual and tedious to read, making it difficult for individuals to receive the right guidance. This unreliable communication setup creates a noticeable divide between government offices and citizens, making it increasingly challenging for people to access the services they need and leaving them feeling frustrated and disconnected.

1.4 Objectives

The objectives of this project are:

- To gather and categorize the inquiries related to the government services that include citizenship, birth, death, marriage, divorce and migration certificate.
- To develop a chatbot that can address those inquiries and offer detailed instructions to the users.

1.5 Application

The applications of ‘Samadhan’ are as follows:

- Citizen Queries Handling:

  Citizens can use Samadhan, a one-stop platform, to ask questions about government services like certificates for citizenship, birth, death, marriage, divorce, and migration. Samadhan provides clear information and advice through easy interactions, helping citizens avoid using old methods and save time.

- Romanized Nepali Support:

  With Samadhan, users can choose to communicate in either English or Romanized Nepali. This feature makes it easier for users who like to use English or Nepali. Samadhan improves user experience and inclusion by respecting different language choices, making sure that all citizens can use government services well. For Romanized Nepali input, the chatbot generates the responses in Nepali.

- Procedure Simplification:

  Samadhan makes it easier to use government services by giving step-by-step help for complicated official processes. It explains what is needed, and supports users at every step of the application process, making sure the experience is smooth and fast.

- User Empowerment:

  Samadhan teaches citizens about the government services through engaging chats. It helps citizens learn and access what they need, encouraging them to get involved and use government systems well, leading to a more aware and active society.

1.6 Scope of project:

The scope of the Samadhan project is vast, encompassing the efficient handling of inquiries related to key government services in Nepal, including citizenship, birth, death, marriage, divorce, and migration certificates. With a user-friendly interface, it not only provides accurate information but also facilitates the application process by offering links to application forms and token numbers, thereby enhancing accessibility and convenience for citizens seeking government assistance.

However, the project has some drawbacks. Firstly, it can answer questions about citizenship and vital certificates well, but it may not include all the government services available in Nepal, which may limit its usefulness for users with other queries. Secondly, the government service procedures may differ across different regions of the country, which may cause some inconsistencies in the help provided. Thirdly, the chatbot depends on a fixed knowledge base, which means it does not give real-time data, which may lead to outdated information for users, showing the need for regular updates and maintenance to keep it accurate and relevant.

1.7 Report Organization

The report is organized into ten chapters, each crucial to the project's full explanation. Chapter 1 introduces the project, outlining its background, purpose, challenges, goals, practical use, and extent, highlighting its importance. Chapter 2 gives a brief review of related works, such as 'Muna' by Diyo.ai and 'AskGov' by the Singapore government, recognizing past contributions in this area.

Chapter 3 details the software requirements and assesses the project's technical, operational, and financial viability, setting the stage for development. Chapter 4 focuses on the careful gathering of training data, essential for the project's natural language understanding, actions, responses, and dialogue management. Chapter 5 explores the project's design, explaining the system structure and how it operates, illustrated with sequence diagrams, flowcharts, and query processing techniques. Chapter 6 describes the transition from design to implementation, detailing the natural language understanding pipeline, strategies, and the complexities of model training.

Chapter 7 displays the project's results, including the user interface, testing outcomes, and performance metrics, providing both qualitative and quantitative analysis. Chapter 8 suggests future enhancements and improvements to the project. Chapter 9 reviews the project's progress, summarizing the main discoveries and successes. Chapter 10 credits the academic and technical references that supported the project, ensuring scholarly honesty and openness. This clear and systematic structure aids in understanding the project from start to completion.

2 LITERATURE REVIEW

The increasing integration of Artificial Intelligence (AI) into daily life is evident through the production of intelligent agents, such as chatbots, which perform diverse tasks from labor to complex operations [1]. Chatbots, an elementary yet widespread form of intelligent Human-Computer Interaction (HCI) [2], simulate human-like conversation through text or voice, employing Natural Language Processing (NLP) to understand multiple human languages.

Their popularity stems from several advantages, including platform-independence, immediate user accessibility, and re-engagement of inactive users through notifications. The widespread utility and advantages of chatbots underscore their significance in modern AI-driven interactions. In the private sector, chatbots, also referred to as conversational agents, have seen extensive utilization in various domains like banking, media, tourism, and retail. These AI-driven solutions, exemplified by virtual assistants like Siri, Alexa, and Google Now, have transformed customer service interactions and transactional processes [3].

Rasa NLU (Natural Language Understanding) is an open-source natural language processing library primarily used for understanding and extracting intent and entities from user messages. Operating as part of the Rasa framework, it facilitates the creation of conversational AI applications. Rasa NLU employs machine learning-based models to decipher the intentions behind user messages (intents) and extract relevant information (entities) crucial for responding effectively in chatbot interactions. Its pipeline-based architecture allows for the configuration of various components, including tokenizers, featurizers, intent classifiers, and entity extractors, enabling developers to tailor the NLU pipeline to suit specific use cases and data. By training on annotated datasets, Rasa NLU learns to recognize patterns in user input, enhancing its ability to accurately comprehend natural language queries and generate appropriate responses in conversational applications [4].

Rasa Core, an integral component of the Rasa framework, is an open-source dialogue management library used for building conversational AI applications. It orchestrates the flow of conversations in chatbots by employing machine learning-based models for

managing dialogue policies. Operating on a probabilistic framework, Rasa Core utilizes Conditional Random Field (CRF) tagging layer on top of the user sequence transformer encoder output corresponding to the input sequence of tokens, to predict and determine the next best action or response based on the conversational context and history of interactions. Its dialogue management system allows for handling multi-turn conversations by learning from user inputs and feedback, enabling the chatbot to make contextually appropriate decisions and responses. Rasa Core's flexibility lies in its capability to create custom dialogue policies, such as rule-based policies or machine learning-based policies, empowering developers to design sophisticated conversational flows and optimize chatbot interactions for diverse use cases and domains.

The Dual Intent and Entity Transformer (DIET) is a lightweight transformer architecture designed for Natural Language Understanding (NLU) in conversational AI, introduced as a state-of-the-art solution. Its key features include enhanced accuracy and faster training compared to fine-tuning BERT(nearly 6 times faster). Operating as a multitask model, DIET handles both intent classification and entity recognition, providing the flexibility to integrate pre-trained embeddings such as BERT, GloVe, and ConveRT. In contrast to large-scale pre-trained language models, which pose challenges in terms of computational intensity and training time, DIET presents a modular architecture that seamlessly fits into standard software development workflows. The model's versatility allows developers to achieve high-level performance without relying on extensive pre-training, addressing the need for efficient and rapidly trainable AI assistants. Before DIET, Rasa's NLU pipeline utilized a bag-of-words model, while DIET leverages a sequence model considering word order, resulting in superior performance. It is a compact model with a plug-and-play structure, suitable for both intent classification and entity extraction. The DIETClassifier incorporates dense and sparse featurizers, offering flexibility in converting user messages into numerical vectors. Examples include the ConveRTFeaturizer and CountVectorsFeaturizer. Moreover, DIETClassifier is fine-tunable, allowing developers to configure neural network architecture using various hyperparameters for optimized performance in NLU tasks, making it a powerful and customizable tool for conversational AI development [5].

Anran Jiao, a researcher from Nankai University, China, proposed an intelligent chatbot system that extracts entities using Rasa NLU and Neural Network methods in his 2020 paper. The Rasa NLU method employs spaCy, scikit-learn, and sklearn-crfsuite libraries for intent classification and entity extraction, while the Neural Network method uses TensorFlow, word embeddings, tanh activation function, softmax layer and cross-entropy loss function to train and predict entities. The paper shows that the Rasa NLU method outperforms the Neural Network method in terms of accuracy (1.0) and integrity of sentence (0.92). The Neural Network method has an accuracy (0.95) and an integrity of sentence (0.70). This means that the Neural Network method is less accurate and less complete in identifying the sentences that contain the user's query than the Rasa NLU method [6].

2.1 Related Works

2.1.1 Muna by Diyo.ai

Diyo.ai creation Muna is conceived as a user-friendly guide to facilitate hassle-free communication between citizens and their local government whether to handle application processes properly, document a clarifying analysis, or provide guidance on workshops, Muna emerges as a versatile partner. Acting as a reliable source of general information on government services, Muna addresses public inquiries on aspects of local governance.

Furthermore, Muna is now incorporated into Lalitpur Metropolitan Municipality and Butwal Sub-Metropolitan Municipality. In addition to its role in information dissemination, Muna not only facilitates public participation but also contributes to the development of responsible and accountable local government in areas of meaningful implementation difficult to address public complaints, empowering individuals to express concerns about issues such as traffic, public waste disposal, or road conditions [7].

2.1.2 AskGov by Singapore government

AskGov.sg, a Singapore government-led initiative acts as a comprehensive website and interactive chatbot, providing citizens with a versatile platform to inquire about government services In this digital space, users the user can refer their queries to various

government agencies, including the finance ministry, health including ministry, public -Transport council, immigration, checkpoint authority, etc. The integrated chatbot feature ensures that citizens can easily interact with the platform and get fast and accurate answers to their queries.

The user-friendly AskGov.sg homepage not only displays a careful selection of frequently asked questions but also uses chatbot functionality to provide concise and informative answers This unique integration provides accessibility and visibility transparently, enabling citizens to access government services and speak up in real-time effortlessly to clarify doubts or gather information AskGov.sg stands as a prime example of the usefulness of technology to engage citizens encourage government communications and facilitate access to critical information [8].

3 REQUIREMENT ANALYSIS AND FEASIBILITY STUDY

3.1 Software Requirements

The software required for this project is listed below

Table 3-1 Software Requirements

Software	**Reason**
Python	Python for coding and utilizing various modules and libraries.
HTML, CSS and Javascript	For front-end development and backend integration
Rasa framework	Rasa for natural language understanding and natural language generation.
VS code IDE	Visual Studio Code (VS Code) for development and project management.

3.2 Technical Feasibility

The technical feasibility assessment for our government services chatbot project revolves around evaluating the practicability of its implementation. Leveraging open-source tools such as Rasa framework, we aim to enhance the chatbot's ability to comprehend user queries by intent and entity extraction. Our development strategy prioritizes open-source libraries and frameworks, particularly relying on Python and its associated libraries. An integral aspect of our approach involves establishing a comprehensive knowledge base ensuring the chatbot's access to essential information. This application will be hosted on a website, providing users with an efficient and accessible solution.

3.3 Operational feasibility:

The operational feasibility of our government services chatbot project is rooted in its practicality and effectiveness for day-to-day use. The availability of data from the Kathmandu Metropolitan City website and the DAO Kathmandu supports the development of the knowledge base of the chatbot. Considering the utilization of free

software framework like Rasa, the project stands as an operationally sustainable solution. The project's success depend upon the collaboration with relevant government offices and the responsiveness of the chatbot to user needs, making it a operationally viable tool for improved citizen engagement with government services.

3.4 Economic feasibility:

The project is economically feasible because we use open-source tools that do not require any development costs unlike proprietary software. We chose Rasa, an open source framework that is free of charge, to match our project's goal of being cost-effective. Moreover, the chatbot hosting platforms that we selected have some extra costs but they are within our team's budget. The project is economically viable and sustainable due to use of open-source resources.

4 DATASET PREPARATION

4.1 Training Data Collection

We collected the training data for our chatbot model from two sources: online and offline. First, we collected the documents required from Nagarik Woda Patra, which is a citizen charter that provides information about the services and procedures of the Kathmandu Metropolitan city. We downloaded the documents from the official website of the city and extracted the relevant information for our chatbot domain (‘domain.yml’ file). Second, we visited Lalitpur Metropolitan City Ward no. 1 and inquired with the ward secretary to collect the steps of the procedure for obtaining a birth, death, marriage, migration and divorce certificate. We recorded the conversation and transcribed it into text. We then annotated the text with the intent and entities of the chatbot. We combined the data from both sources and preprocessed it for our chatbot model.

4.1.1 Data for NLU

```
- intent: birth_certificate_query
  examples: |
    - how to make [birth](type) certificate?
    - [birth](type) certificate
    - how to obtain [birth](type) certificate?
    - [birth](type) certificate process
    - process to make [birth](type) certificate.
    - how to get a birth certificate in nepal?
    - what is the procedure to make [birth](type) certificate.
    - how can I process for [birth](type) certificate.
    - complete process to obtain [birth](type) certificate.
```

```
- intent: migration_documents_certificate_query_romanized
  examples: |
    - [basai sarai darta](type) banauna kun kun [kagajaat](specification) chainxa?
    - [basai sarai](type) darta ko lagi chaine [kagajaat](specification) k k hun?
    - k k [kagaj](specification) liyera [basai sarai darta](type) banauna jane?
    - [basai sarai](type) banaune [kagajaat](specification) k ho?
    - [basai sarai](type) darta banauna kun kun [kagajaat](specification) chainxa.
    - [basai sarai darta](type) banaune [kagajaat](specification) k ho?
```

User inquiries suitable for input into the chatbot were gathered and compiled into the 'nlu.yml' file as examples of intents. Our dataset encompasses a total of 94 intents, reflecting the diverse intentions behind user queries that the chatbot can comprehend.

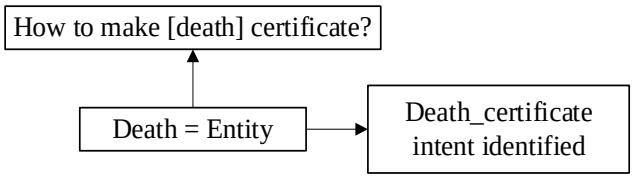


Figure 4-1 Examples of Intent and Entities Identification in English Query

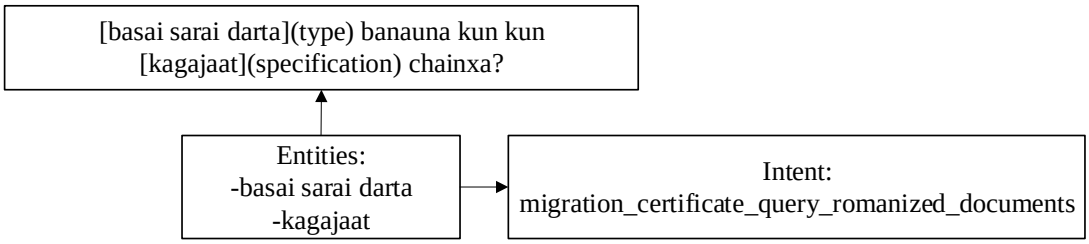


Figure 4-2 Examples of Intent and Entities Identification in Roman Nepali Query

4.1.2 Data for actions and responses

The domain.yml file consists of the data for actions and responses, organizing them in a structured manner to define the behavior and functionality of the chatbot within the conversational AI system. Actions, which encompass the range of responses or behaviors that the chatbot can execute, are listed under the 'actions' section of the file. Each action is associated with one or more potential responses, typically organized under the 'responses' section. These responses include various messages or utterances that the chatbot can deliver to users. There are total of 52 actions and 49 responses listed in our domain.yml file.

The responses in domain.yml file are defined as:

```
utter_citizenship_by_descent_eligibility:
- text: |-
    Eligibility criteria:
    1. At least one parent must be Nepali.
    2. Age must be 16 or above.
utter_citizenship_by_descent_eligibility_romanized:
- text: |-
    Yogyata Maapdanda:
    1. Kamti ma ek jana abhibhavak Nepali nagarik hunu parchha.
    2. Tapai ko umer 16 barsa mathi hunu parchha.
utter_citizenship_by_descent_documents:
- text: |-
    Required documents:
    1. Recommendation form from District Administration Office which costs Rs. 20
    2. Birth certificate original and its 2 photocopies
    3. Parents citizenship original and 2 photocopies
    4. Migration certificate original and 2 photocopies (if migrated from any place).
    5. School/College character certificate original and its 2 photocopies
    6. 7 Photos (passport or auto-sized)
```

This is how the actions are defined in the domain.yml file:

```
actions:
- action_default_fallback
- utter_certificate_query_romanized
- utter_divorce_certificate_documents
- utter_migration_certificate_documents_romanized
- action_handle_citizenship_query_romanized
- utter_greet3
- utter_halkhabar
- utter_greet1
- action_handle_citizenship_query
- utter_greet2
- utter_citizenship
- utter_citizenship_romanized
- utter_citizenship_by_birth_eligibility
- utter_citizenship_by_birth_eligibility_romanized
- utter_citizenship_by_birth_steps
- utter_citizenship_by_birth_steps_romanized
- utter_citizenship_by_birth_documents
- utter_citizenship_by_birth_documents_romanized
```

4.1.3 Data for Dialog Management

The 'stories.yml' file in Rasa plays a crucial role in orchestrating the flow of conversation between the chatbot and the user by defining sequences of interactions based on specific user intents. Each story represents a potential path that the conversation can take, guiding the chatbot's responses to effectively address the user's needs or inquiries.

For instance, let's examine the story titled "User interaction for divorce certificate":

```
- story: User interaction for divorce certificate
  steps:
  - intent: divorce_certificate_query
  - action: utter_divorce_certificate_documents
  - action: utter_divorce_certificate_steps
```

In this example, the story outlines the following sequence of steps:

1. The user expresses the intent to inquire about a divorce certificate (‘divorce_certificate_query’).

2. Upon recognizing this intent, the chatbot responds with information about the required documents for obtaining a divorce certificate (‘utter_divorce_certificate_documents’).
3. Subsequently, the chatbot provides guidance on the steps involved in obtaining the divorce certificate (‘utter_divorce_certificate_steps’).

By structuring the conversation flow in this manner within the 'stories.yml' file, the chatbot can effectively engage with users, anticipate their needs, and provide relevant information or assistance in a coherent and intuitive manner.

```
- story: User interaction for divorce certificate romanized
  steps:
  - intent: divorce_certificate_query_romanized
  - action: utter_divorce_certificate_documents_romanized
  - action: utter_divorce_certificate_steps_romanized



story: User asks for migration certificate and death certificate romanized
steps:
- intent: migrationcertificate_and_deathcertificate_query_romanized
- action: utter_for_migrationcertificate_romanized
- action: utter_migration_certificate_documents_romanized
- action: utter_migration_certificate_steps_romanized
- action: utter_for_deathcertificate_romanized
- action: utter_death_certificate_documents_romanized
- action: utter_death_certificate_steps_romanized
```

```
- story: User interaction for marriage certificate documents
  steps:
  - intent: marriage_documents_certificate_query
  - action: utter_marriage_certificate_documents



- story: User asks for birth certificate and marriage certificate romanized
  steps:
  - intent: birthcertificate_and_marriagecertificate_query_romanized
  - action: utter_for_birthcertificate_romanized
  - action: utter_birth_certificate_documents_romanized
  - action: utter_birth_certificate_steps_romanized
  - action: utter_for_marriagecertificate_romanized
  - action: utter_marriage_certificate_documents_romanized
  - action: utter_marriage_certificate_steps_romanized
```

There are total of 105 stories in our stories.yml file.These are few examples of the stories created in our ‘stories.yml’ file.

5 METHODOLOGY AND SYSTEM ARCHITECTURE

5.1 System Architecture

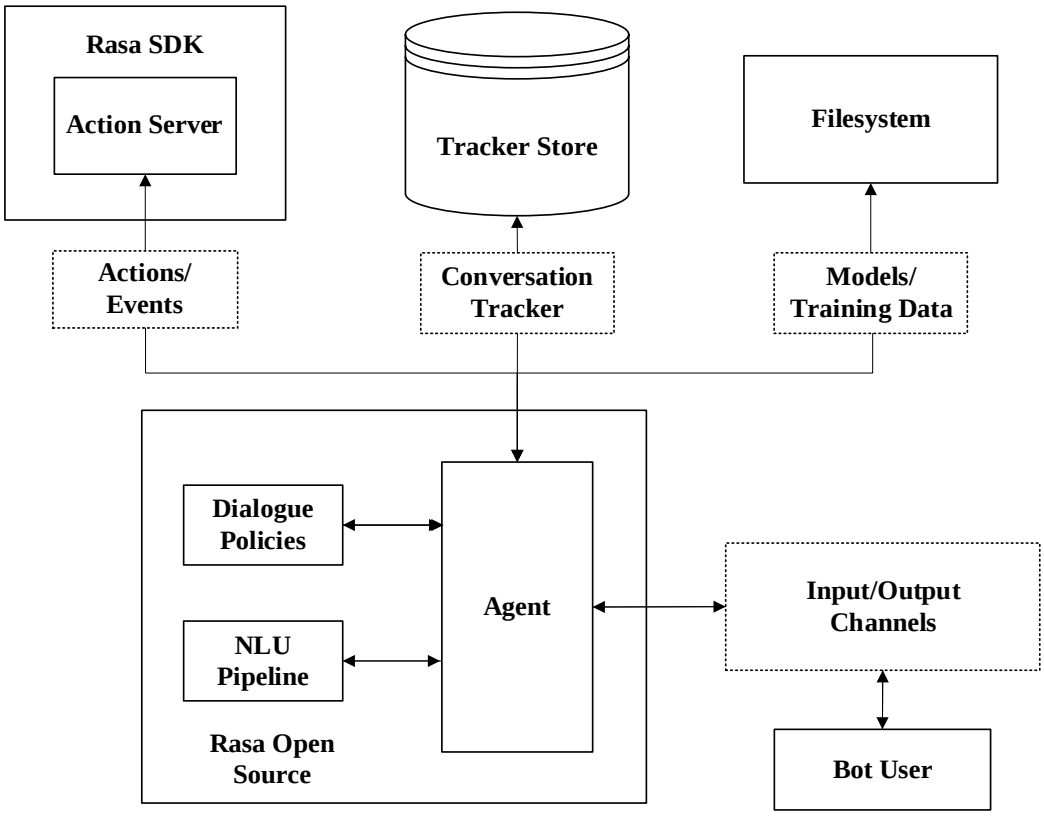


Figure 5-1 System architecture

The diagram presents the chatbot system's structure, emphasizing the main elements and how they interact. The chatbot's success relies on the integrated operation of these elements, which guarantees that users receive information both correctly and promptly. The Input/Output Channels are the interfaces for communication between the Bot User and the Agent, facilitating interactions with the system. The Agent is the core element, responsible for interpreting user inputs, consulting the conversation log in the Tracker Store, carrying out actions through the Rasa SDK Action Server, and applying models from the Filesystem.

The NLU Pipeline is essential, comprising modules for recognizing intents, filling slots, and performing Named Entity Recognition (NER). This pipeline is key to grasping what users want and prefer. Dialogue Policies are the guidelines or tactics that determine the chatbot's reactions to user inputs. These policies may be manually crafted or derived

from data, taking into account factors like user objectives, dialogue conditions, and system operations. The Tracker Store archives the dialogue history, including user queries, bot replies, and executed actions, which helps the Agent refer to previous exchanges. Finally, the Models/Training Data repository is where the chatbot retrieves stored models and training data, enabling it to respond intelligently based on past learning, thus enhancing the system's flexibility and performance.

5.2 Interaction flow

5.2.1 Sequence Diagram

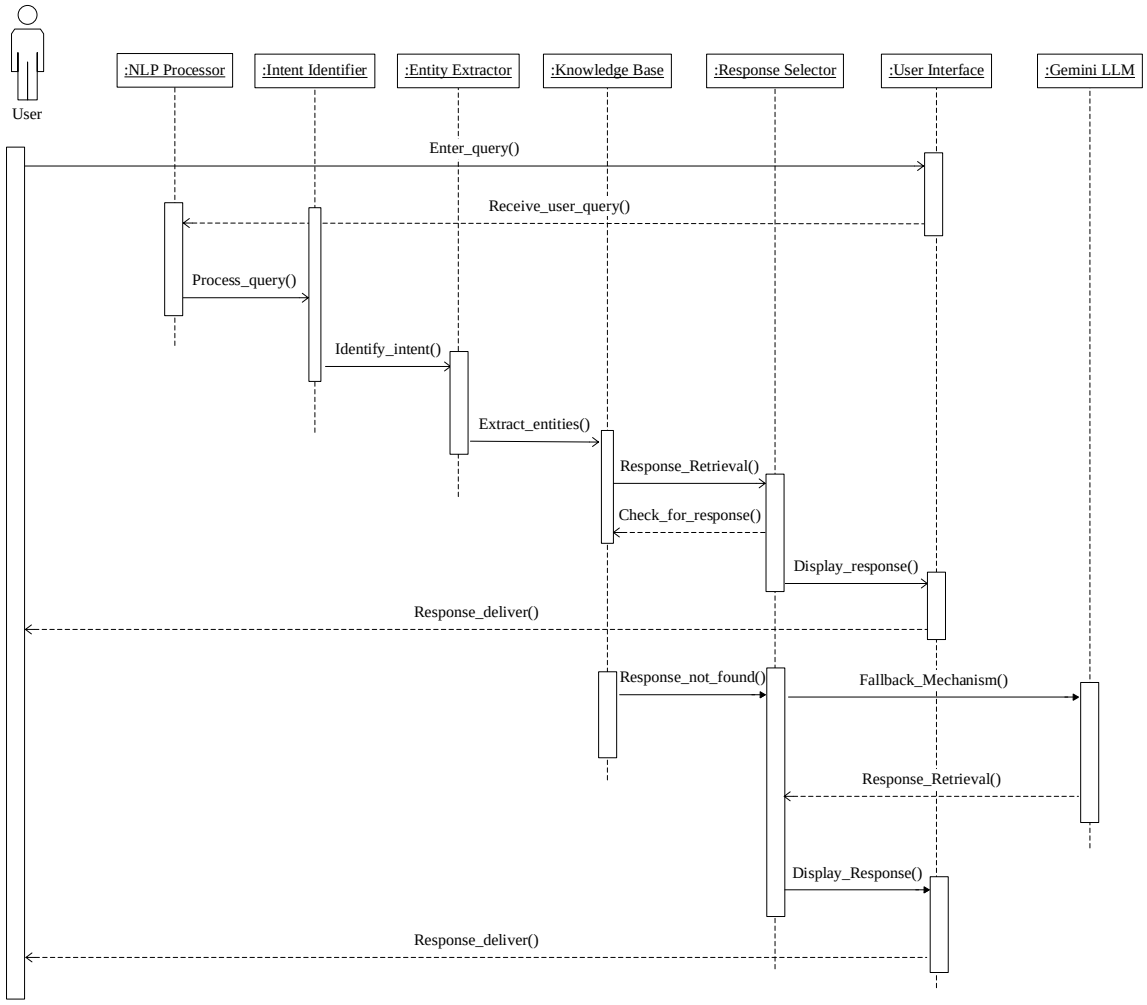


Figure 5-2 Sequence Diagram

User query reception initiates the process as the user inputs a query into the interface. The user query undergoes processing by the NLP processor, breaking it down into its

elements, such as words and phrases. Intent identification follows, wherein the system endeavors to understand the user's goal with the query, whether it's a question, information request, or instruction. The entity extractor then pinpoints any named entities like people or places in the query. Subsequently, the system checks the knowledge base for a pre-prepared response. If none is found, the response selector retrieves potential responses. The selected response is delivered to the user interface, where it is displayed. In cases where no suitable response is available, a fallback mechanism generates a more general response.

5.2.2 System Flowchart

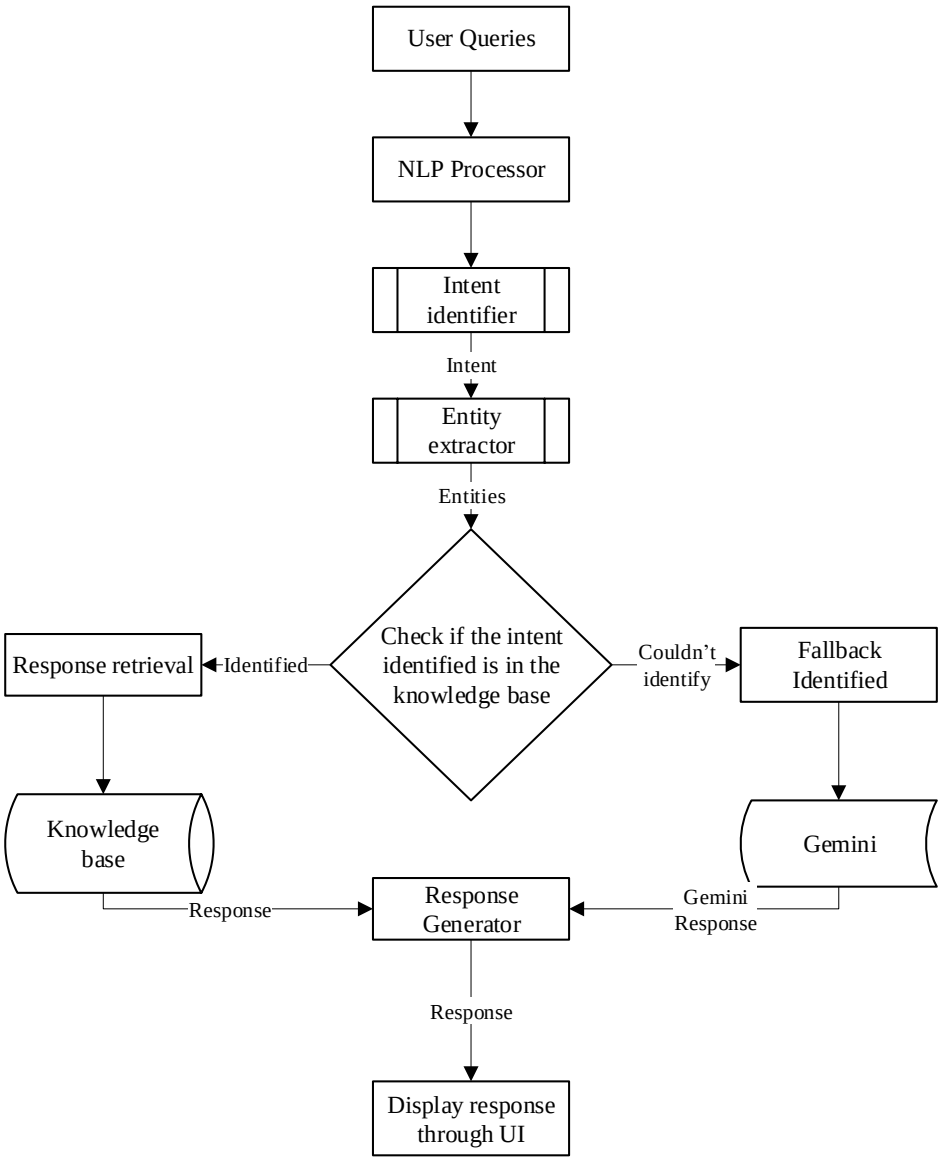


Figure 5-3 Flowchart of the system

5.2.3 Query Processing

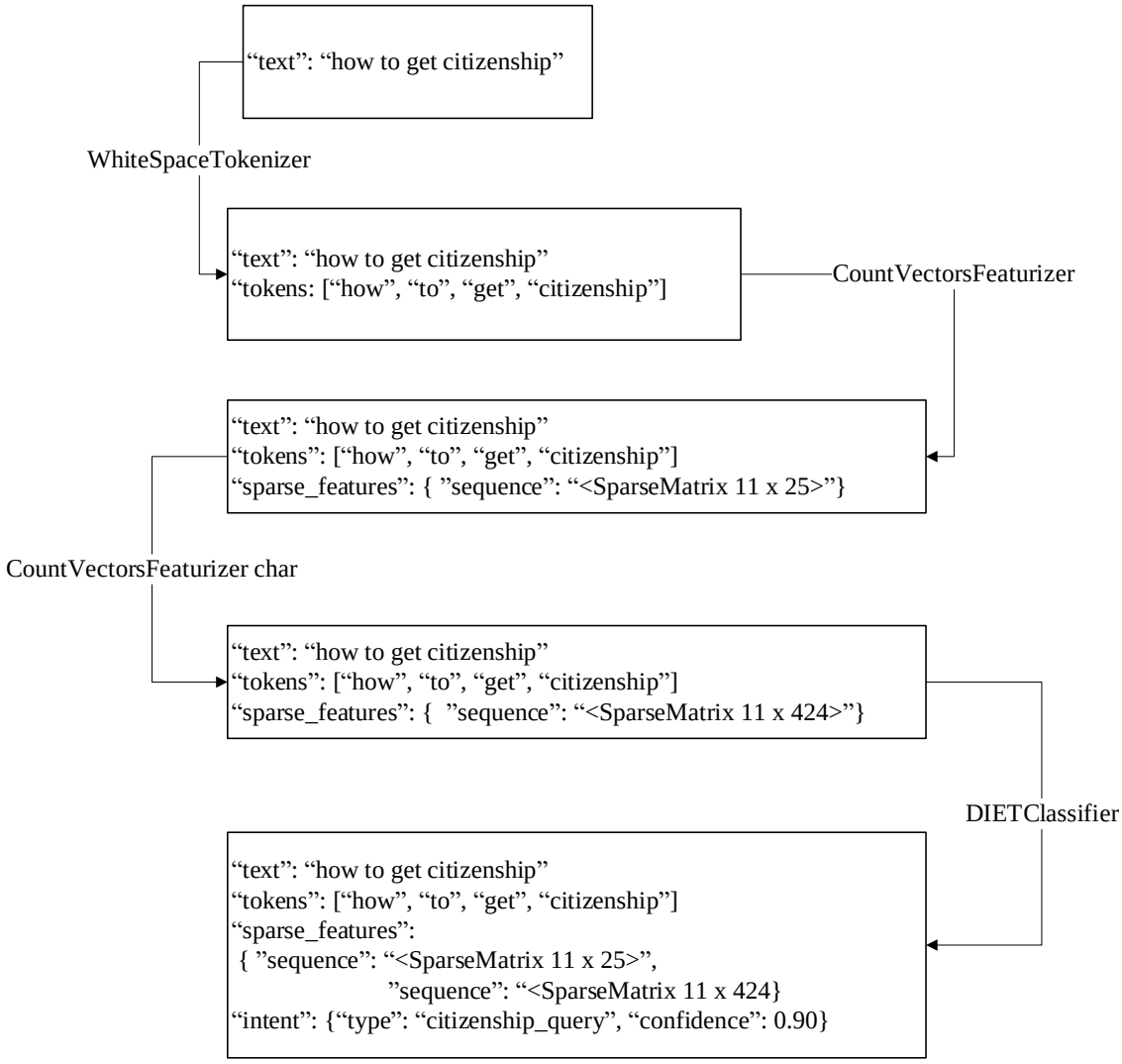


Figure 5-4 Query Processing

The Figure 5-4 shows a text processing pipeline, which is a series of steps used to break down and analyze text. In this case, the pipeline is being used to process the sentence "How to get citizenship?"

The first step in the pipeline is the WhitespaceTokenizer. This component simply splits the text into individual words. So, for the sentence "How to get citizenship?", the WhitespaceTokenizer would output the words "how", "to", "get", and "citizenship".

The next step in the pipeline is the CountVectorsFeaturizer. This component takes the words from the WhitespaceTokenizer and converts them into a format that can be

understood by a machine learning model. It does this by creating a sparse matrix, which is a data structure that efficiently stores data that is mostly empty.

The final step in the pipeline is the DIETClassifier. This component takes the sparse matrix from the CountVectorsFeaturizer and tries to identify the intent of the text. In this case, the DIETClassifier would identify the intent as "citizenship_query", which means that the user is asking a question about how to get citizenship. The DIETClassifier can also identify any named entities in the text, but there are no named entities in the sentence "How to get citizenship?"

6 IMPLEMENTATION DETAILS

6.1 NLU Pipeline

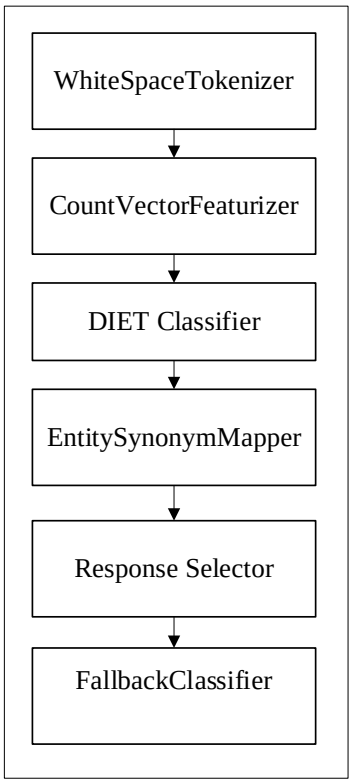


Figure 6-1 Pipeline

The Rasa pipeline refers to the sequence of processing steps that the framework uses to understand and respond to user inputs in a conversational context.

Let us take the intent citizenship_query which contains the following examples.

```
- intent: citizenship_query
  examples: |
    - I want information about citizenship.
    - how to apply for citizenship?
    - tell me about citizenship.
    - what are the requirements for citizenship?
    - help me with citizenship
    - can you help me with citizenship?
    - How to get citizenship?
    - complete process to obtain citizenship?
    - what is the process to make citizenship?
    - procedure to make citizenship?
    - i want to make citizenship
```

Now, for the queries “How to get a citiznship?” and “Jamma Darta banaune kasari” the way in which the pipelines work is described below. Some words in these queries are intentionally misspelled to show how the chatbot model reacts to them.

6.1.1 WhitespaceTokenizer

This is a tokenizer used in ‘Samadhan’ pipeline. This tokenizes the user input into individual words which are separated by spaces. Also, if there are any characters which are not considered as a letter, number, underscore, or a few special symbols (like @, #, &) then it is removed before splitting the word. This process is shown below for English as well as roman-Nepali query.

English:

User message: "How to get a citiznship?"

Tokenized: [“how”, “to”, “get”, “a”, “citiznship”]

Romanized Nepali:

User message: "Jamma Darta banaune kasari?"

Tokenized: ["Jamma", "Darta", "banaune", "kasari"]

6.1.2 CountVectorsFeaturizer

The tokens are passed through the CountVectorsFeaturizer, where sparse features are created. There are two layers of this vectorizer. They use different analyzer. One creates a sparse feature of the tokens or words which uses the analyzer as ‘word’ and the other creates the sparse feature of the characters within the word boundaries which uses the analyzer as ‘character.’ The two layers are:

6.1.2.1 ‘word’ as Analyzer

The parameters for the First layer of CountVectorsFeaturizer is: Analyzer = ‘word’, min_ngram = 1, max_ngram = 1.

For the given intent, 11x25 sparse matrix of type class ‘numpy.int64’ with 53 stored elements gets created in compressed sparse row format. And the feature names are:

['about', 'apply', 'are', 'can', 'citizenship', 'complete', 'for', 'get', 'help', 'how', 'information', 'is', 'make', 'me', 'obtain', 'procedure', 'process', 'requirements', 'tell', 'the', 'to', 'want', 'what', 'with', 'you'].

Table 6-1 Sparse Matrix of Individual Tokens

index	about	apply	are	can	citizenship	…	complete	is	tell	the
0	1	0	0	0	1	…	0	0	0	0
1	0	1	0	0	1	…	0	0	0	0
2	1	0	0	0	1	…	0	0	1	0
3	0	0	1	0	1	…	0	0	0	1
4	0	0	0	0	1	…	0	0	0	0
5	0	0	0	1	1	…	0	0	0	0
6	0	0	0	0	1	…	0	0	0	0
7	0	0	0	0	1	…	1	0	0	0
8	0	0	0	0	1	…	0	1	0	1
9	0	0	0	0	1	…	0	0	0	0
10	0	0	0	0	1	…	0	0	0	0

6.1.2.2 ‘char’ as Analyzer

This layer is used to correctly identify the intents and entities. The Featurizer and training data is adjusted to comply with the misspellings. The parameters are: Analyzer = ‘word’, min_ngram = 1, max_ngram = 4.

For the given intent, 11x424 sparse matrix of type class ‘numpy.int64’ with 1153 stored elements gets created in compressed sparse row format.

Table 6-2 Sparse Matrix of characters inside word boundaries

index	a	c	ci	cit	f	…	fo	for	g	ge	get
0	1	1	1	1	0	…	0	0	0	0	0
1	1	1	1	1	1	…	1	1	0	0	0
2	1	1	1	1	0	…	0	0	0	0	0
3	1	1	1	1	1	…	1	1	0	0	0
4	0	1	1	1	0	…	0	0	0	0	0
5	0	1	1	1	0	…	0	0	0	0	0
6	0	1	1	1	0	…	0	0	1	1	1
7	0	1	1	1	0	…	0	0	0	0	0
8	0	1	1	1	0	…	0	0	0	0	0
9	0	1	1	1	0	…	0	0	0	0	0
10	0	1	1	1	0	…	0	0	0	0	0

6.1.3 DIETClassifier

DIET is a joint transformer model that performs both intent detection and slot filling simultaneously.

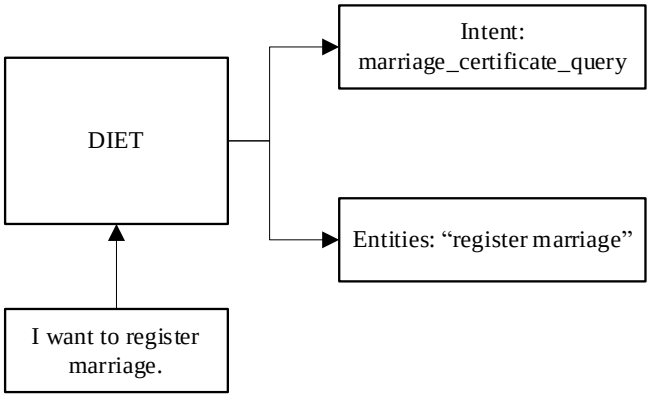


Figure 6-2 High Level Block Diagram of DIET Classifier

6.1.3.1 Architecture of DIET classifier

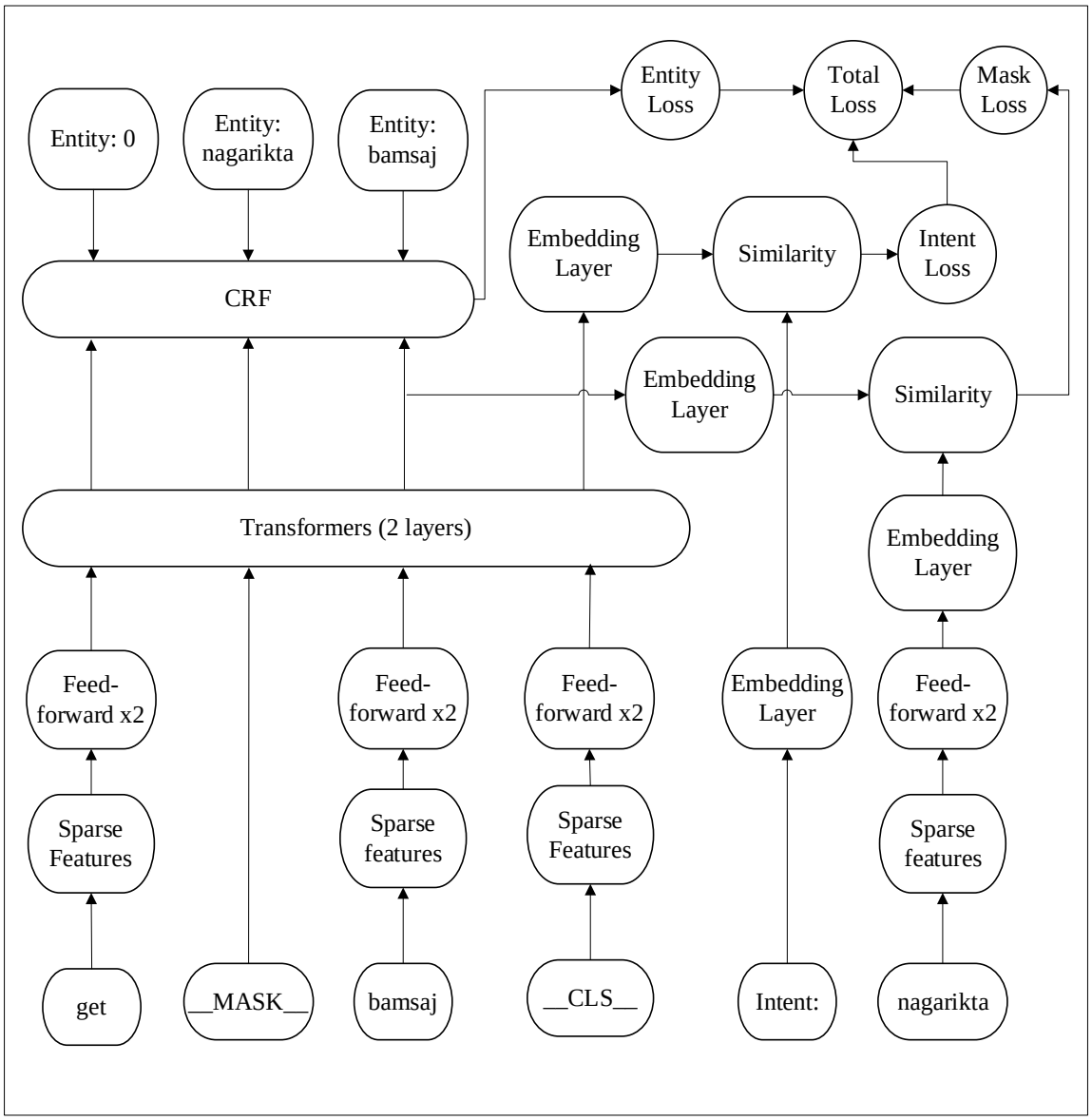


Figure 6-3 Architecture of DIET Classifier

DIET's architecture is based on modules, which have a multi-layer transformer as a key component of the architecture. The different modules are:

6.1.3.1.1 Input Preprocessing

The modules in the input preprocessing are responsible for preprocessing the input data so as to generate the features necessary for the transformer module:

Sparse Features: This module generates sparse features for the tokens in training; it's essentially a FFN, with sparse connections.

Feed forward: This module consists of a fully connected neural network (NN) which may have different inputs depending on which part of the architecture it's connected to, but its output is always a 256 size array. There are 2 different Feed Forward Networks in the architecture, the input of the first one is the sparse features of some token, and the input of the second one is the feed forward of the first, concatenated to the output of the first FFN. The use case is the same in both cases, to learn features about tokens which will be later used as inputs of the Transformer module.

Mask & CLS tokens: The mask token is applied randomly at a word of the sentence. This is then feeded to the transformer, which allows it to predict which token was masked, then allow learning through gradient descent.

$$L_M = -\langle S_M^+ - \log(e^{S_M^+} + \sum_{\Omega_M^-} e^{S_M^-}\rangle \quad (1)$$

Where $S_M^+ = h_{MASK}^T h_{token}^+$ is the similarity with the target label y_{token}^+ and $S_M^- = h_{MASK}^T h_{token}^-$ are the similarities with negative samples y_{token}^-, $h_{MASK} = E(a_{MASK})$ and $h_{token} = E(a_{token})$ are the corresponding embedding vectors $h \in IR^{20}$; the sum is taken over the set of negative samples Ω_M^- and the average $\langle . \rangle$ is taken over all examples.

The CLS token is a representation of the whole sentence, similar to an embedding of the whole sentence, and provides the ability to learn the intent of the sentence to the transformer.

6.1.3.1.2 Intent Recognition

Embedding Layer: The module is used in training, so as to generate the Mask loss, associated with the mask generated by the model. This basically generates a loss measurement between the token "predicted" or "filled in" by the transformers output, and the ground truth generated by the FFN which uses inputs from the pretrained embedding and the sparse feature modules from the token.

Intent Loss: This module calculates the loss of the output of the transformer and the intent classification of the sentence (given by the similarity module). Then goes as an input to the total loss module, which will provide a measure of learning.

$$L_I = -\langle S_I^+ - \log(e^{S_I^+} + \sum_{\Omega_I^-} e^{S_I^-} \rangle \quad (2)$$

Where the sum is taken over the set of negative samples Ω_I^- and the average $\langle . \rangle$ is taken over all

6.1.3.1.3 Entity recognition

Conditional Random Field (CRF): A Conditional Random Field (CRF) is a probabilistic model for sequence labeling tasks, such as named entity recognition (NER). The authors use a CRF layer on top of the transformer output to predict a sequence of entity labels for each input token. The CRF layer maximizes the likelihood of the correct label sequence given the input sequence. The CRF loss is the negative log-likelihood of the CRF layer, which is minimized during training. The CRF loss is one of the components of the total loss, along with the intent and mask losses. At inference time, the authors use the Viterbi algorithm to find the most probable label sequence given the input sequence and the CRF parameters.

Entity Loss: The entity loss module is provided the output of the CRF module described above, which will calculate the measure of how good the transformer is identifying entities in the sentence.

$$L_E = L_{CRF}(\mathbf{a}, \mathbf{y}_{\text{entity}}) \quad (3)$$

Where $L_{CRF}(.)$ denotes negative log-likelihood for a CRF

Similarity & Total loss Modules: There are 4 loss modules present in the architecture, the entity loss module, the mask loss module, the intent loss module and the total loss module. The total loss is calculated as:

$$L_T = L_E + L_I + L_M \quad (4)$$

The purpose of these modules is the same, compute the formula for each particular loss with its inputs and output the value of the loss. These modules are essential for training and provide one of the most important features of the DIET Model, the ability to infer intent and entity recognition at the same time.

The similarity module is just a particular case of a loss module, which is used for comparison of two different inputs, in one case, related to the mask, by comparing the output of the transformer with the output of the FFN of the masked token. Also, when comparing the CLS token for the whole sentence with the intent of the sentence. It's output provides a measure of how similar its inputs are, essential for computing the losses previously described.

6.1.3.1.4 Transformer

To capture the context of an entire sentence, we utilize a two-layer transformer model equipped with relative position attention. The transformer architecture mandates that the input dimension matches the transformer layers' dimensionality. Consequently, while ConveRT's sentence embeddings are 1024-dimensional and word embeddings are 512-dimensional, we resolve this discrepancy by expanding the word embeddings through a straightforward process of duplicating them, thereby adding an extra 512 dimensions to achieve 1024-dimensional word embeddings. This approach ensures consistency in the neural architecture across various pre-trained embeddings. Additionally, to harmonize the dimensions of the concatenated features with those of the transformer layers (256 in our experiments), we pass them through an additional fully connected layer with shared weights across all sequence steps [5].

6.1.3.2 Using DIET Classifier

DIET classifier is a Rasa tool that can identify the intent and the entities in a user message. It uses CountVectorsFeaturizer to create a sparse matrix of word and character features, which are then combined into a long binary vector and fed into a feed forward layer. Since there are no pre-trained word embeddings for Romanized Nepali, the pipeline does not use any embeddings like ConveRT, GloVe or BERT. Instead, the sparse matrix is passed to another feed forward layer, which then serves as the input for the transformer. The feed forward layer applies an activation function to the sparse

vector, which is the result of multiplying the vector by a weight matrix W and adding a bias term b. This is the softmax function used in the feed forward layer.

The transformer then predicts the entities using a CRF layer on top of it. For example, for the user input: “How to get nagarikta” The input is first split into tokens, then a sparse matrix is created at the word level and another sparse matrix is created for the characters within each word, choosing the correct tokens among the misspelled ones. The tokens are then sent to the DIET classifier, which determines the intent of the query and extracts the entities. The response selector then chooses the appropriate response based on the entities and the policies.

Parameters:

Epochs: 100 -Epochs represent the number of times the model goes through the entire training dataset. In this configuration, the DIET Classifier undergoes training for 100 epochs, allowing it to learn and adjust its internal parameters based on the provided training data.

constrain_similarities: true -This parameter, when set to true, applies constraints on similarities during training. It is a regularization technique that helps prevent the model from becoming overly specialized to the training data, promoting better generalization to new, unseen data.

Output of DIET Classifier for recognition of intent:

```
'text': 'how to get citizenship'
```

Table 6-3 DIET Classifier Intent Ranking Table for Valid Intent

Intent Ranking	**Name of the intent**	**Confidence**
1	citizenship_query	0.841346741
2	random_queries	0.023823019
3	citizenship_by_descent_query	0.017440192

Here, the confidence of the user input to be a citizenship_query is 0.84 which is higher than the fallback threshold (0.5). So, the response selector selects response for the citizenship_query.

Output of DIET Classifier for recognition of intent (for fallback case):

```
'text': 'nepal ko area kati ho'
```

Table 6-4 DIET Classifier Intent Ranking Table for Fallback Intent

Intent Ranking	**Name of the intent**	**Confidence**
1	nlu_fallback	0.5
2	death_certificate_query	0.13504600524902344
3	migration_steps_certificate_query_romanized	0.07748120278120041
4	birth_certificate_query	0.06386325508356094
5	birth_steps_certificate_query_romanized	0.06169731914997101
6	death_steps_certificate_query	0.05788872390985489
7	marriage_steps_certificate_query_romanized'	0.049536414444446564

If the confidence is lower than the fallback threshold like in the query “nepal ko area kati ho” then the Gemini API will be called and it will handle the required response.

6.1.4 EntitySynonymMapper

The configuration of the Entity Synonym Mapper is specified within the pipeline, indicating its place in the sequence of NLP components. Specifically, it is added to the

pipeline with the designated name "EntitySynonymMapper." When the Entity Synonym Mapper modifies an existing entity, it appends itself to the processor list associated with that entity, facilitating traceability and transparency in the processing flow. This capability makes the Entity Synonym Mapper an invaluable tool for handling diverse expressions of synonymous entities and ensuring a unified and accurate representation in NLP applications.

The EntitySynonymMapper's function manages synonyms associated with entities, creating a unified representation for synonymous values to enhance entity recognition. In scenarios involving location entities, for instance, it addresses variations by mapping them to a standardized form. Consider the examples "documents" and "kagajaat"; the Entity Synonym Mapper ensures both variations are recognized as the same entity, promoting consistency in understanding user inputs.

6.1.5 ResponseSelector

The ResponseSelector component in Rasa is a powerful tool for retrieving appropriate responses from a pool of candidates. It essentially acts as a "selector" that analyzes user messages and chooses the most relevant response based on its understanding of the conversation context.

The ResponseSelector requires dense or sparse features for both user messages and potential responses to function effectively. It then outputs a dictionary containing the predicted response key, its confidence score, and the associated responses with additional details like text and images. Additionally, it provides a ranking of the top candidate responses along with their confidence levels. This ranked list can be valuable for further decision-making within the dialogue management system.

Parameters:

- **learning_rate (0.1):** This value controls how drastically the model adjusts its internal settings during training. A learning rate of 0.1 signifies moderate adjustments, allowing the model to learn effectively without overfitting to the training data.

- **constrain_similarities (true):** This setting enforces limitations on the similarities between training examples. It acts as a safeguard against the model becoming overly reliant on specific patterns and helps it generalize better to handle diverse user queries.

6.1.6 FallbackClassifier

The FallbackClassifier is a component designed to handle ambiguous cases in natural language understanding (NLU) by classifying a message with the intent nlu_fallback when the confidence scores from the previous intent classifier fall below a specified threshold.

The Fallback Classifier acts as a fail-safe, stepping in to offer a response when the model is unsure of the user's intention. When the confidence level of the main classifier drops below a certain threshold, the Fallback Classifier initiates a series of actions. These actions involve invoking the Gemini API, passing the user query to Gemini, and presenting the response generated by Gemini back to the user.

Parameters:

- **Threshold: 0.5** -Threshold is a confidence level set to trigger the Fallback Classifier. If the primary model's confidence in predicting the user's intent falls below this threshold (in this case, 0.5), the Fallback Classifier comes into play.

6.1.7 Gemini

We use Gemini, a large language model from Google DeepMind, to improve our Rasa chatbot's ability to handle situations where the user's intent is unclear or unknown. When the chatbot faces a query that does not match any of its predefined intents, it activates the fallback action, which connects to the Gemini API and produces a detailed and informative response.

Gemini has an outstanding capacity to process and comprehend natural language, which enables it to examine the user's query thoroughly. It takes into account the conversation context, accesses its extensive knowledge base, and applies its sophisticated reasoning

skills to generate a relevant and informative response. This integration allows our chatbot to go beyond pre-defined intents and deal with unexpected user queries with great flexibility and accuracy.

By using Gemini, we greatly improve the reliability and adaptability of our chatbot. Users get informative responses even for unanticipated queries, creating a more natural and engaging conversational experience. This integration shows the possibility of combining Rasa's structured approach with the versatility of large language models like Gemini, opening up new possibilities for chatbot development.

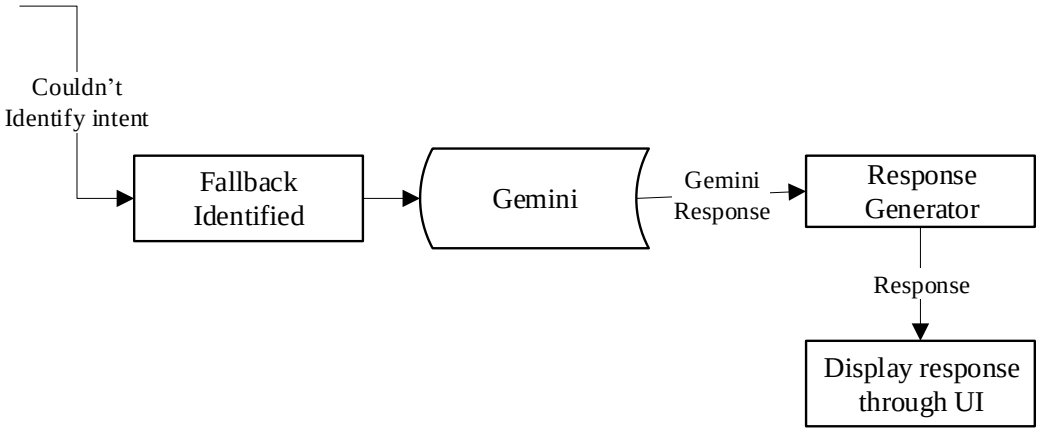


Figure 6-4 Gemini Response Generation

6.2 Policies

6.2.1 MemoizationPolicy

This policy is based on the idea of learning from past conversations and applying them to future ones. It keeps a record of successful conversation histories, which consist of user intents and actions. When the chatbot faces a similar situation in a new conversation, it looks up the stored history and suggests the next action based on it. This policy has some advantages, such as improving efficiency and consistency by reusing successful conversation patterns, and handling repetitive user queries without needing complex conversation flows. However, it also has some limitations, such as depending too much on past data and not being able to adapt well to unseen situations or variations in user queries.

6.2.2 RulePolicy

This policy is based on the idea of using explicit rules to control the chatbot's behavior in specific situations. These rules are usually manually created and have conditions based on user intents, entities, or conversation history. The chatbot triggers the action defined in the rule if the conditions are satisfied. This policy has some benefits, such as giving precise control over the chatbot's responses in specific scenarios, and being useful for handling edge cases, exceptions, or providing context-specific information. However, it also has some drawbacks, such as being time-consuming and complex to define and maintain a large number of rules, and not being able to generalize well to unseen situations or variations in user queries.

6.2.3 TEDPolicy

TED Policy (Transformer Embedding Dialogue Policy) is a policy that uses transformer models for next action prediction and entity recognition. It consists of several transformer encoders which are shard for both tasks. It's capable of understanding context and dependencies between user inputs.

TED Policy architecture comprises the following steps:

1. Concatenate features for
    - user input (user intent and entities) or user text processed through a user sequence transformer encoder,
    - previous system actions or bot utterances processed through a bot sequence transformer encoder,
    - slots and active forms
2. For each time step into an input vector to the embedding layer that precedes the dialogue transformer.
3. Feed the embedding of the input vector into the dialogue transformer encoder.
4. Apply a dense layer to the output of the dialogue transformer to get embeddings of the dialogue for each time step.
5. Apply a dense layer to create embeddings for system actions for each time step.
6. Calculate the similarity between the dialogue embedding and embedded system actions.

7. Concatenate the token-level output of the user sequence transformer encoder with the output of the dialogue transformer encoder for each time step.
8. Apply CRF algorithm to predict contextual entities for each user text input.

$$\hat{Y} = \arg\max_{Y} P(Y|X) \quad (5)$$

6.3 Training the model

- Training data (nlu.yml, domain.yml, stories.yml) were validated for consistency and completeness.
- Training data were splitted into training and validation sets.
- NLU pipeline was trained using the nlu.yml file, which contains the user intents and examples.
- Dialogue management policies were trained using the stories.yml file, which contains the conversation flows.
- Trained model was evaluated on the basis of validation set and reports the performance metrics.
- Trained model and the configuration files were packaged into a single archive file (model.tar.gz).

7 RESULTS AND ANALYSIS

7.1 User Interface

Our chatbot's UI is designed to be intuitive and user-friendly. It is a web page with a clean and organized layout using HTML. Users are welcomed by the chatbot when they enter. The HTML sets up the UI structure, with a chat container and input field for interaction. CSS shapes the UI appearance, with a sleek design for the chat container, a fixed width, subtle borders, and padding. These elements create a refined and positive look for users. The chat display area has a scrolling feature for easy conversation navigation. Interactive buttons are added for faster query resolution. Users can use predefined options for quicker and more efficient interactions, making the chatbot responsive and engaging. JavaScript enables dynamic interaction between the frontend and the Rasa backend. It displays user queries and bot responses in the chat window. It also sends user queries to the Rasa server for natural language processing. This is done through asynchronous communication using the Fetch API, which allows data exchange between the frontend and Rasa backend.

7.2 Testing the model

The test data were prepared by splitting the training data. The 80% of the data is used to train the model whereas 20% of the data is used to test the model. The trained model and test data were loaded and following tasks were performed:

- The NLU pipeline was tested using the test_nlu.yml file, which contains the user intents and examples.
- The dialogue management policies were tested using the test_stories.yml file, which contains the conversation flow.
- The test results were evaluated on various metrics such as accuracy, precision, recall and f1-score.

The test report were generated which contains evaluation results, confusion matrices, errors, and plots.

7.2.1 Intent Confusion

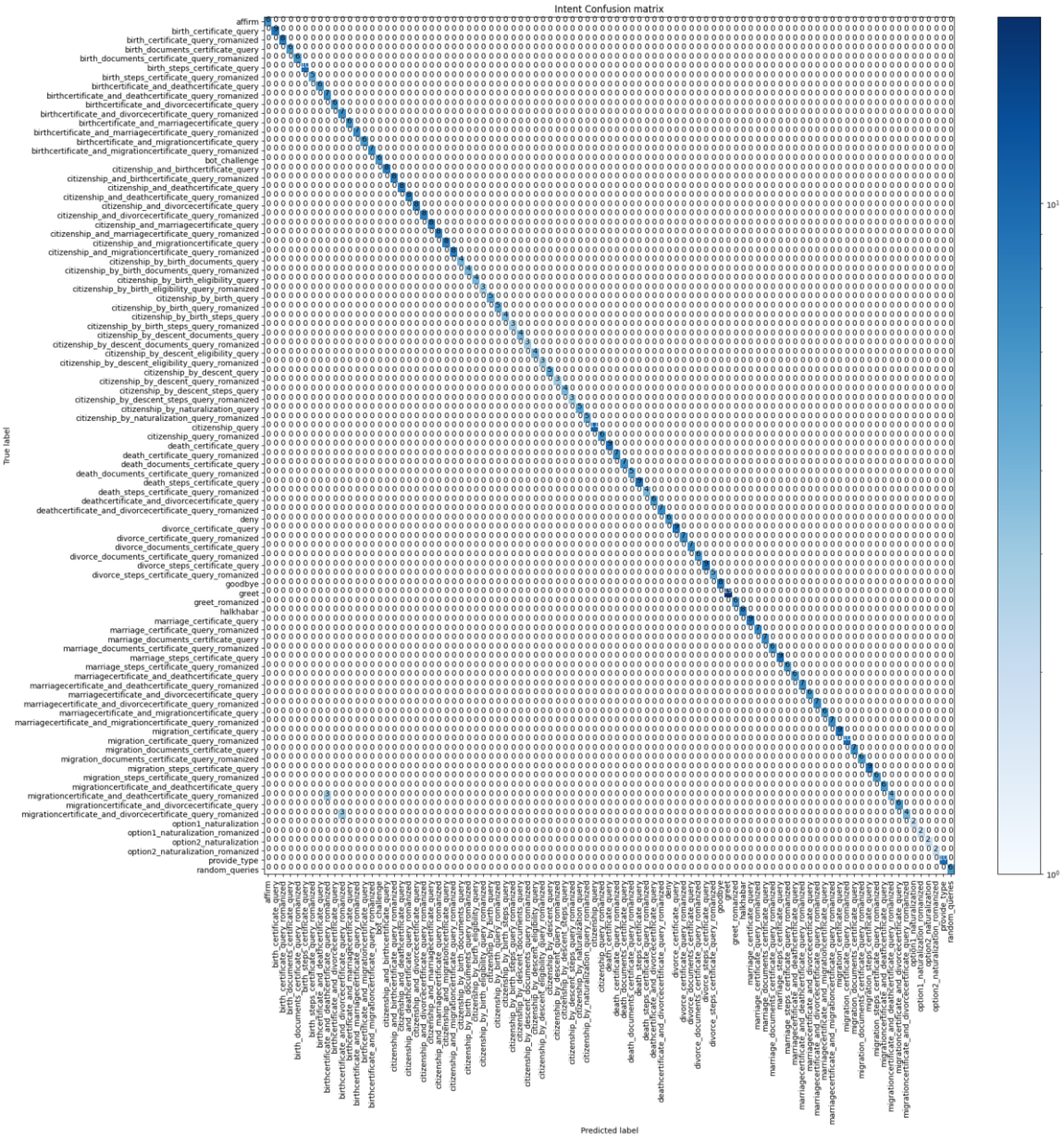


Figure 7-1 Intent Confusion Matrix

This is intent confusion matrix and here the rows represent true intents and the columns represent the intents that were predicted. The diagonal cells of the matrix show the number of times that the model correctly identified an intent. And the other cells represent the intents that the model misunderstood.

From the Figure 7-1, we can clearly see that the intents predicted is the true intent in large number of cases. So, the model can predict intent with a very high accuracy whenever a user input is given.

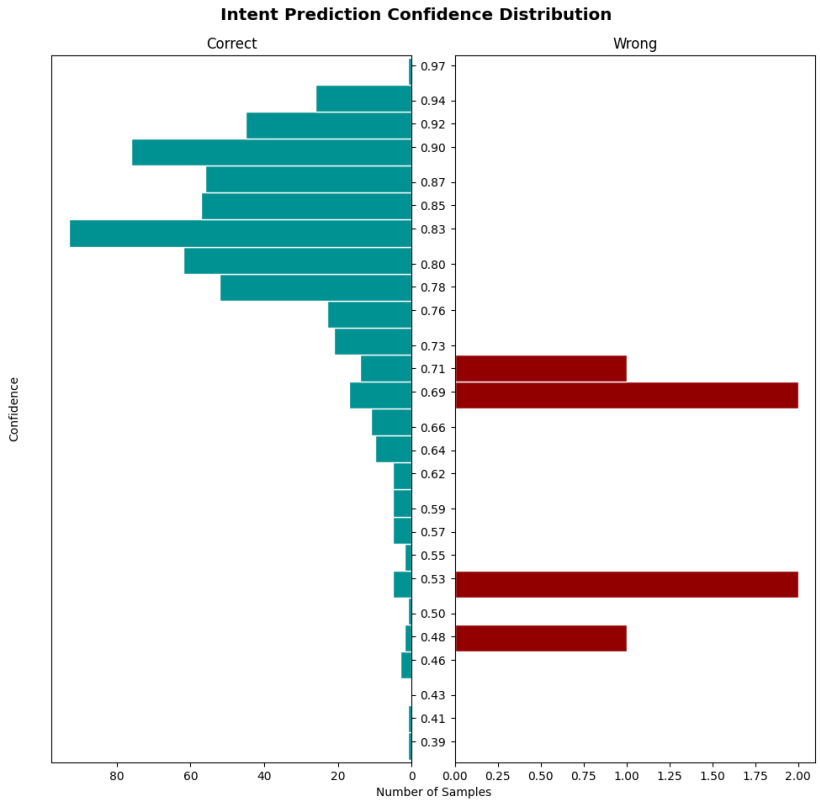


Figure 7-2 Intent Prediction Confidence Distribution

Figure 7-2 shows intent prediction confidence distribution. Here, it is clear that the confidence of predicting correct intents is much higher in comparison to incorrect prediction. The correct predictions are above 90% confidence with much greater number of samples.

Overall, the test seems to show that the model performs well in correctly identifying intents, with a majority of predictions clustered around higher confidence levels (0.8 and above) within the green zone. This indicates a strong ability to accurately understand user intent. However, there's also a notable spread of incorrect predictions (red bars) across the confidence range.

7.2.2 Entity Confusion Matrix

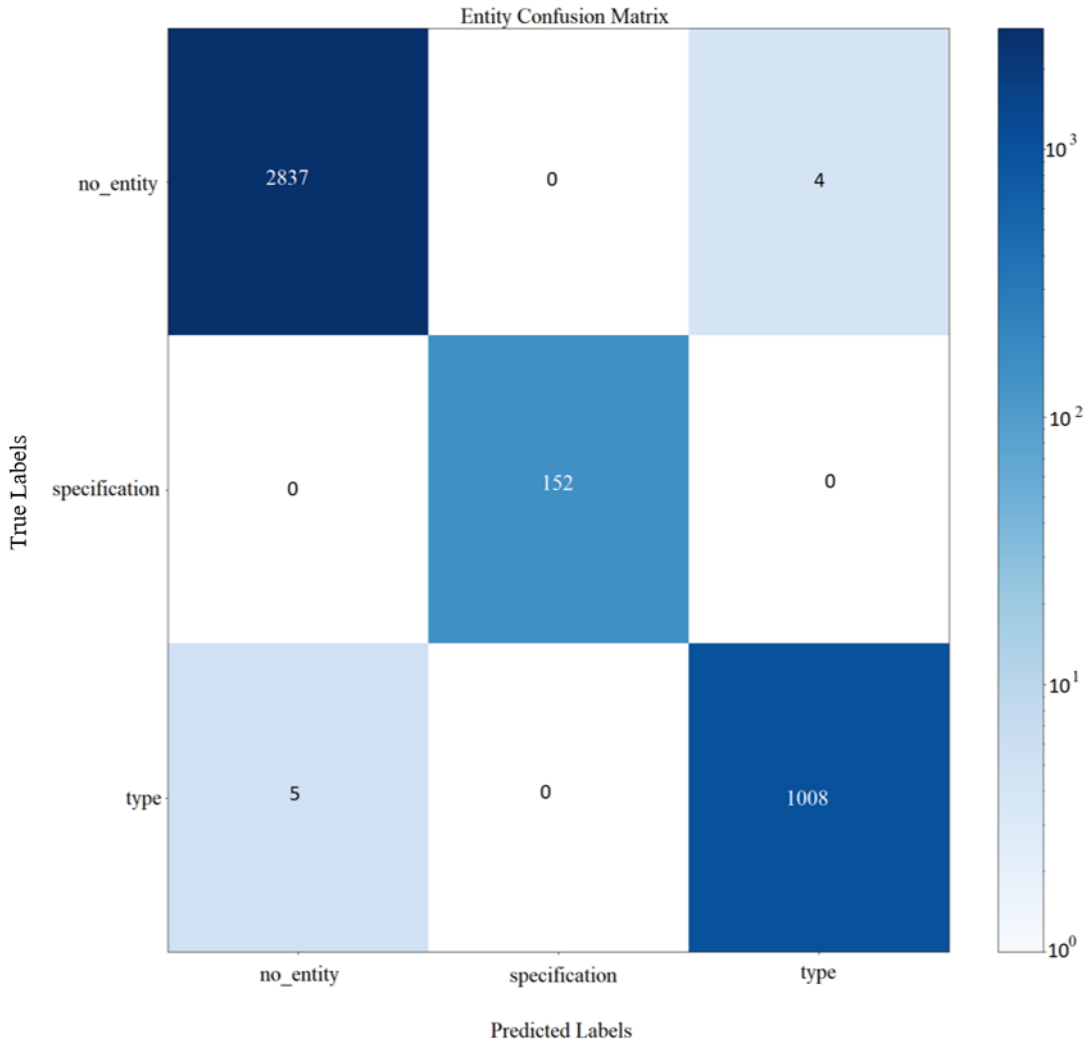


Figure 7-3 Entity Confusion Matrix

The rows of the matrix represent the actual labels of the named entities in the text, and the columns represent the labels that the model predicted. Each cell in the matrix shows the number of times that the model predicted a particular label for a named entity that actually had a different label. For example, the cell in the row labeled “specification” and the column labeled “no_entity” shows that the model predicted that the word “specification” was not a named entity, when it actually was.

The diagonal cells of the matrix show the number of times that the model correctly identified a named entity. In this image, the model correctly identified 2837 tokens that are not labelled as entities, 152 specifications, and 1008 types.

The other cells in the matrix show the number of times that the model made mistakes. Here, the model predicted a token to be ‘type’ entity for 5 times, and it predicted that the entity ‘type’ was not an entity for 4 times.

The confusion matrix shows the distribution of correct and incorrect predictions made by the NER model. The model achieved an overall accuracy of 98.4%, with 96.3% precision, and F1-score values also 97.1% for most named entity categories.

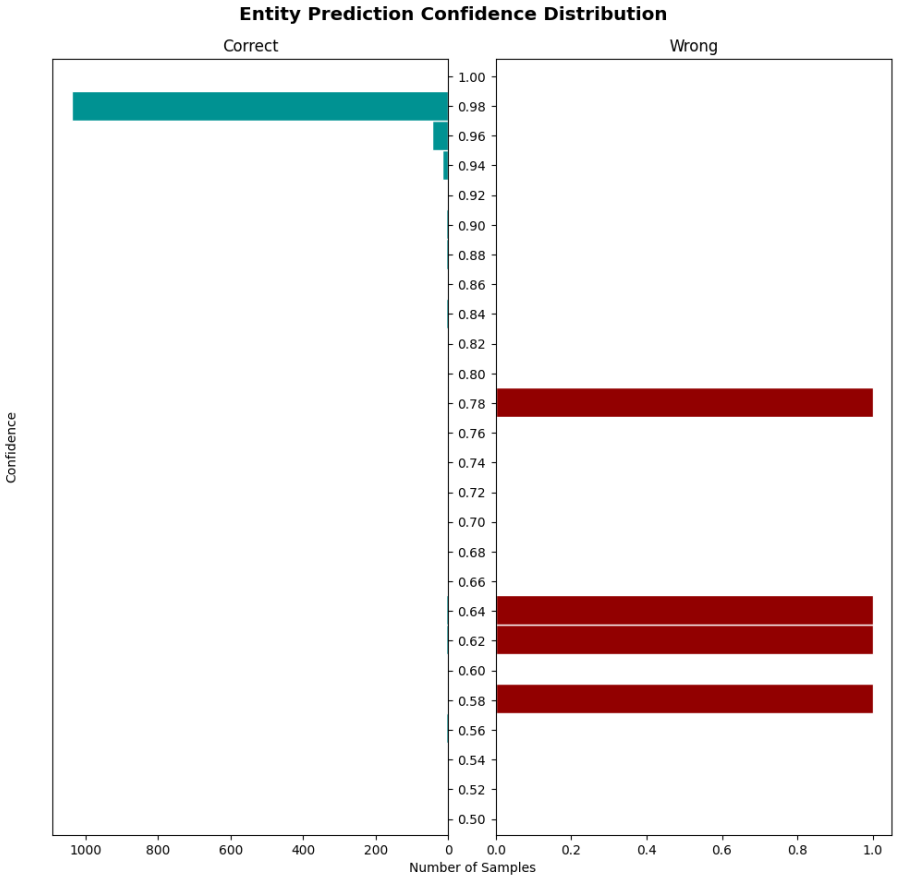


Figure 7-4 Entity Prediction Confidence Distribution

The graph shows that the model is most confident in its predictions when the confidence level is high, between 0.94 and 0.99. There are also a significant number of correct predictions when the confidence level is between 0.55 and 0.8. There are very few incorrect predictions when the confidence level is high, and the number of incorrect predictions increases as the confidence level decreases.

7.2.3 Stories Confusion/Action Confusion

The confusion matrix was generated from ‘rasa test core’ operation to evaluate the performance of a NER model. The NER model was trained on a dataset of text that was annotated with named entities. The confusion matrix shows how often the model correctly identified named entities in the text, and how often it made mistakes.

The rows of the matrix represent the actual labels of the named entities in the text, and the columns represent the labels that the model predicted. Each cell in the matrix shows the number of times that the model predicted a particular label for a named entity that actually had a different label.

The diagonal cells of the matrix show the number of times that the model correctly identified a named entity. The other cells in the matrix show the number of times that the model made mistakes.

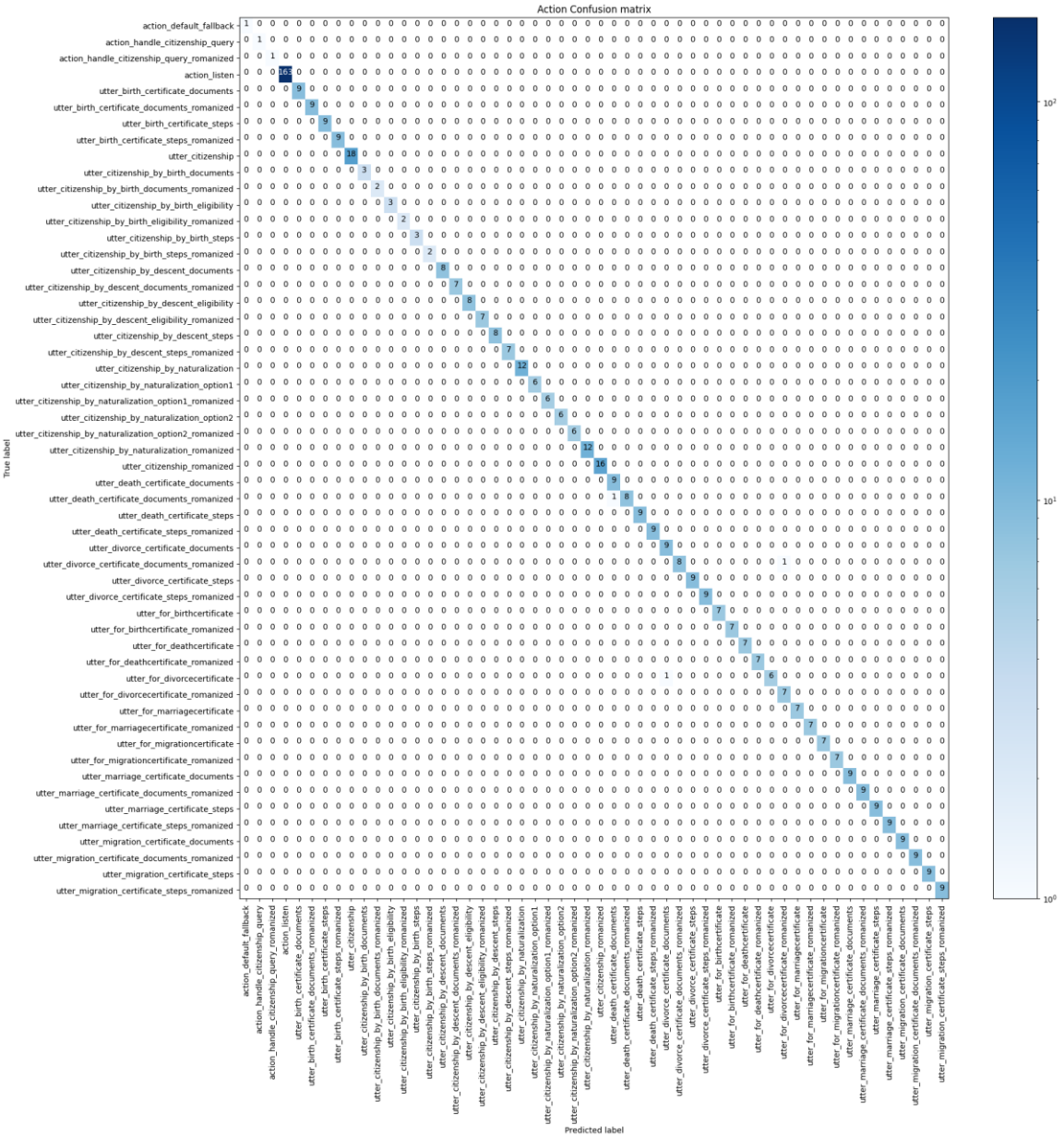


Figure 7-5 Action Confusion Matrix

7.3 Performance Evaluation Metrices

7.3.1 Calculating the loss function

The default loss function used for intent classification was the Categorical Cross-Entropy Loss function. This loss function was used in multiclass classification tasks, such as intent classification, where each inputs (user message) were associated with one of several possible classes (intents). Categorical Cross-Entropy Loss measures the discrepancy between the predicted probability distribution over intents and the true

probability distribution by computing the cross-entropy between them. During training, the model's parameters were optimized to minimize this loss, effectively learning to accurately classify user messages into the correct intents.

Intent 1: marriage_steps_certificate_query

Intent 2: marriage_steps_certificate_query_romanized

For example, consider a batch of user messages and their true intents:

User message: "how to make marriage certificate?"

True intent: marriage_steps_certificate_query

User message: "bibah darta kasari banaune?"

True intent: marriage_steps_certificate_query_romanized

7.3.1.1 Softmax Function

The intent classification component of the DIETClassifier uses a softmax function. The softmax function is a type of activation function that is commonly employed in neural networks for multi-class classification problems.

Given an input vector of raw scores (logits) for each intent class, the softmax function computes the probabilities for each class. The probability for each class is calculated as the exponentiation of the corresponding raw score divided by the sum of exponentiated scores across all classes.

Mathematically, for a class i,

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad (6)$$

; Where z_i is the raw score for class i and N is the total number of classes.

Computing Predicted Probabilities:

Our model predicted the following probabilities for each intent:

Predicted probabilities for Intent 1: [0.8, 0.1, 0.05, 0.05, ..., 0.01]

Predicted probabilities for Intent 2: [0.1, 0.8, 0.05, 0.05, ..., 0.01]

Computing True Label Probability Distribution:

The true intents were one-hot encoded, so the true label probability distributions were as follows:

True label probability distribution for Intent 1: [1, 0, 0, 0, ..., 0]

True label probability distribution for Intent 2: [0, 1, 0, 0, ..., 0]

7.3.1.2 Cross Entropy Loss

Cross-entropy loss is commonly used as the loss function during training for intent classification tasks. It measures the difference between the predicted probability distribution and the true distribution (one-hot encoded label).

Given the predicted probability distribution $P(y_i)$ for class i and the true distribution $Q(y_i)$ (one-hot encoded), the cross-entropy loss is calculated as:

$$\mathrm{H(y, q)} = -\sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{q_i} \cdot \log(\mathrm{p_i}) \qquad (7)$$

;Where N is the number of classes, q_i is the true probability for class i, and q_i is the predicted probability for class$i$.

Computing Cross-Entropy Loss for Each User Message:

For the first user message,

$$\text{Cross} - \text{entropy loss} = -(1 * \log(0.8) + 0 * \log(0.1) + 0 * \log(0.05) + \ldots)$$

For the second user message,

$$\text{Cross} - \text{entropy loss} = -(0 * \log(0.1) + 1 * \log(0.8) + 0 * \log(0.05) + \ldots)$$

Averaging the Loss across the Batch:

Finally, the cross-entropy losses for all user messages in the batch were averaged to get the overall loss for that batch. The graph between the training loss and epoch is obtained as:

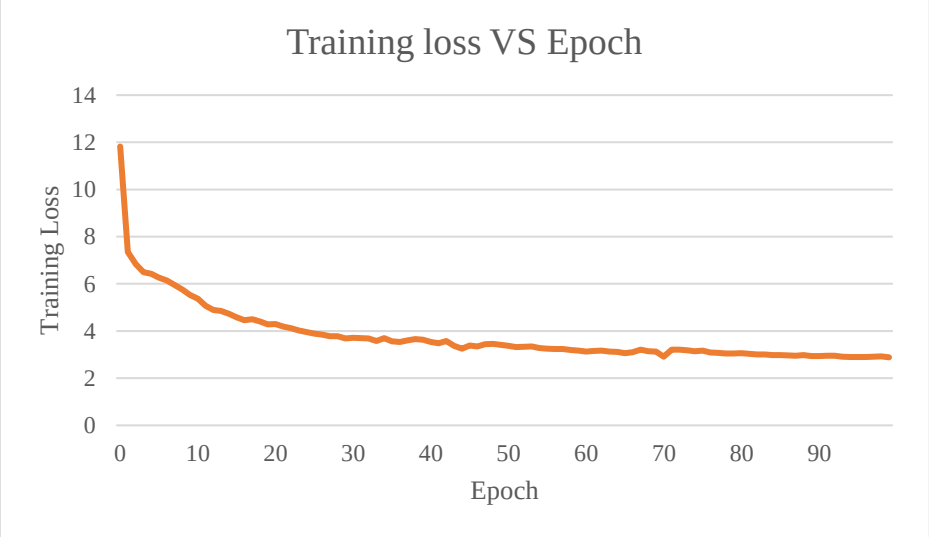


Figure 7-6 Training loss VS Epoch

7.3.2 Precision

In the context of intent classification, precision refers to the accuracy of the positive predictions made by the model. It measures the proportion of correctly predicted instances of a specific intent (true positives) out of all instances predicted as that intent (true positives + false positives).

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \quad (8)$$

Intent: marriage_steps_certificate_query

Examples: 9

True Positives (TP): 9 (All examples matched this intent)

False Positives (FP): 0 (No examples are falsely predicted as this intent)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 9 / (9 + 0) = 1$$

Precision for this intent was 1, this indicated that all predicted instances of this intent were correct.

Intent: marriage_steps_certificate_query_romanized

Examples: 6

True Positives (TP): 0 (No examples matched this intent)

False Positives (FP): 0 (No examples were falsely predicted as this intent)

Precision = TP / (TP + FP) = 0 / (0 + 0) = 0

Precision for this intent is undefined since there were no positive predictions.

7.3.3 Recall

Recall measures the proportion of true positive predictions out of all actual positive cases in the dataset. Recall in the context of intent classification measures the model's ability to identify all relevant instances of a particular intent among all the instances that actually belong to that intent.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \quad (9)$$

Intent: marriage_steps_certificate_query

Examples: 9

True Positives (TP): 9 (All examples match this intent)

False Negatives (FN): 0 (No examples are missed)

Recall = TP / (TP + FN) = 9 / (9 + 0) = 1

Recall for this intent is 1, indicating that all instances of this intent were correctly identified by the model.

Intent: divorce_certificate_query_romanized

Examples: 7

True Positives (TP): 0 (No examples matched this intent)

False Negatives (FN): 7 (All examples were missed)

Recall = TP / (TP + FN) = 0 / (0+7) = 0

Recall for this intent was 0, indicating that none of the instances of this intent were correctly identified by the model.

7.3.4 F1 Score

F1 score is the harmonic mean of precision and recall. It provides a single metric to assess the balance between precision (the ability of the classifier not to label a negative sample as positive) and recall (the ability of the classifier to find all positive samples).

Computing F1 Score for Each Intent:

$$\text{F1 Score} = \frac{2 * (\text{Precision} * \text{Recall})}{\text{Precision} + \text{Recall}} \quad (10)$$

Example:

True Positives (TP) = 5

False Positives (FP) = 1

False Negatives (FN) = 0

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})} = \frac{5}{(5 + 1)} = \frac{5}{6}$$

$$\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})} = \frac{5}{(5 + 0)} = 1$$

$$\text{F1 Score} = \frac{2 * (\text{Precision} * \text{Recall})}{\text{Precision} + \text{Recall}} = \frac{10}{11}$$

So, the F1 score for the intent marriage_steps_certificate_query was calculated as 10/11.

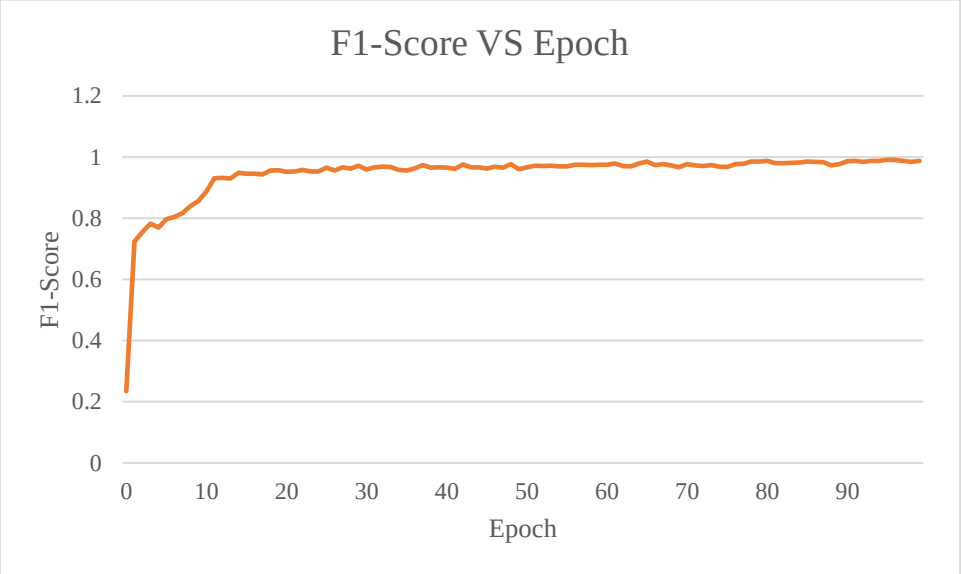


Figure 7-7 F1-Score VS Epoch

7.3.5 Confidence

Confidence represents the level of certainty or probability associated with a model's prediction. It indicates how confident the model is in its predictions.the confidence score for an intent prediction is obtained from the softmax output of the model, which represents the probability distribution over all possible intents.

The confidence score was derived from the model's output probabilities for each intent class. After processing an input text, the model computed the probability distribution over all possible intents based on the learned patterns and features in the training data. The intent with the highest probability was selected as the predicted intent, and its corresponding probability score was served as the confidence score for that prediction.

For the intent "marriage_steps_certificate_query", the confidence score for each prediction would be a value between 0 and 1, where higher values indicated greater confidence in the prediction. These confidence scores were used to assess the reliability of the model's predictions and to set thresholds for decision-making. When the model predicted an intent for a user message, it also provided a confidence score indicating how confident the model was in its prediction. For example, if the model predicted the "migration_certificate_query_romanized" intent with a confidence score of 0.85, it meant that the model was 85% confident that the user's message was about query on migration certificate in romanized form.

7.3.6 Accuracy

Accuracy measures the proportion of correctly classified cases (true positives and true negatives) out of all cases.Accuracy in the context of intent classification measures the overall correctness of predictions made by the model across all intents.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positive} + \text{False Negative}} \quad (11)$$

Intent: divorce_certificate_query_romanized

Examples: 7

True Positives (TP): 0 (No examples matched this intent)

False Positives (FP): 0 (No examples were falsely predicted as this intent)

True Negatives (TN): 28 (All other intents were correctly predicted as not being this intent)

False Negatives (FN): 7 (All examples of this intent were missed)

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})} = \frac{(0 + 28)}{(0 + 28 + 0 + 7)} = \frac{28}{35}$$

Accuracy for this intent was approximately 0.8, indicating that around 80% of predictions were correct. The graph between the intent accuracy and epoch is plotted as:

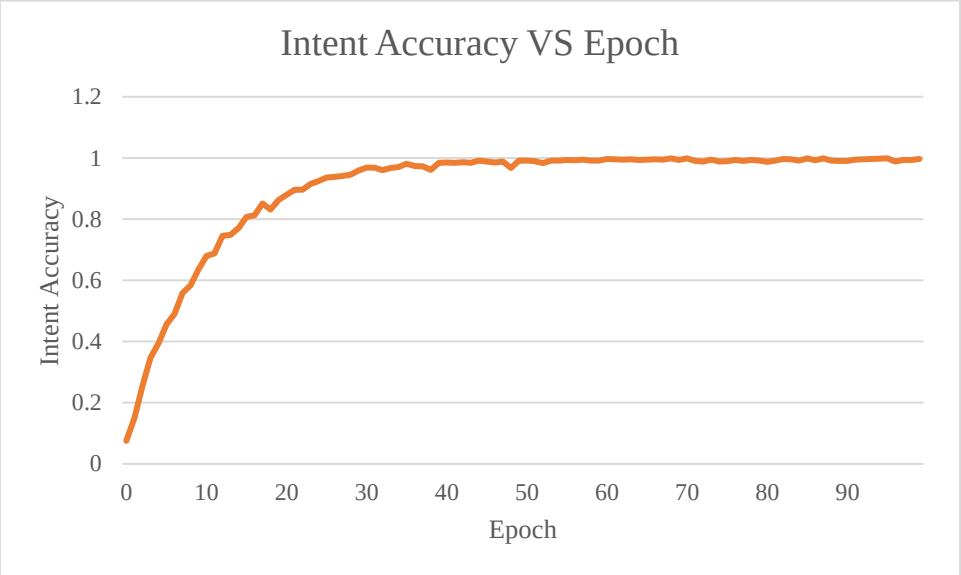


Figure 7-8 Intent Accuracy VS Epoch

7.4 Comparison of evaluation metrics

7.4.1 Precision and Recall

Precision and recall have an inverse relationship: increasing precision often leads to a decrease in recall and vice versa. This trade-off occurs because adjusting the model to be more precise may result in it being less inclusive, thereby missing some relevant instances (lower recall), and vice versa. Therefore, changes in precision can impact recall and vice versa.

7.4.2 F1 Score

The F1 score balances precision and recall, so changes in either metric affect the F1 score. If precision increases while recall decreases (or vice versa), the F1 score may

remain unchanged, increase, or decrease depending on the magnitude of change in each metric.

As the F1 score is the harmonic mean of precision and recall, it is more sensitive to smaller values, meaning it's heavily influenced by whichever metric is lower.

7.4.3 Accuracy

Accuracy measures the overall correctness of the model's predictions. It is influenced by changes in both precision and recall. If precision or recall increases without a corresponding change in the other, accuracy may increase or decrease accordingly. However, if precision and recall change in opposite directions, their impact on accuracy depends on the relative importance of false positives and false negatives in the context of the application.

7.4.4 Confidence

Confidence reflects the model's certainty in its predictions. Higher confidence generally leads to higher precision, as the model is more confident in its correct predictions. However, overly high confidence scores may also lead to false positives if the model becomes overconfident.

Adjusting the confidence threshold can affect both precision and recall. Increasing the threshold may improve precision but decrease recall, as the model becomes more selective in its predictions.

7.4.5 Loss Function

Loss functions guide the model's learning process during training. Changes in the loss function can affect how the model optimizes its parameters, leading to alterations in both precision and recall. For instance, using a loss function that penalizes false positives more heavily may improve precision but potentially at the expense of recall, and vice versa.

8 FUTURE ENHANCEMENTS

8.1 Web application deployment

As a future enhancement, we will deploy our web application using a suitable hosting platform, such as Heroku, AWS, or Azure, that supports the Rasa framework and the socketio or REST channel. Our web application will have a user-friendly and secure interface.

8.2 Covering more services

Our project aims to provide a chatbot that can solve queries related to government services in Nepal. Currently, our chatbot can handle queries related to citizenship, birth, death, marriage, divorce, and migration certificates. However, we plan to expand our chatbot's capabilities to cover more services in the future. Some of the services that we intend to include are taxation, health insurance, driving license, passport, PAN card, etc. By adding these services, we hope to make our chatbot more useful and convenient for the citizens of Nepal.

8.3 Enhanced Chatbot Contextual Understanding

Our chatbot will be able to remember what it said before and use that to give better answers to related questions. The user can ask questions about the chatbot's previous answer. For example: The user wants to know how to get citizenship by descent and gets stuck on the second step of the chatbot's answer. The user asks a question about that step and the chatbot answers it correctly.

9 CONCLUSION

To sum up, Samadhan is a revolution in enhancing the delivery of government services in Nepal. With its easy-to-use interface, Samadhan makes navigating government procedures straightforward for everyone.

The Rasa framework uses Natural Language Processing to provide quick and insightful user interactions. Samadhan also promotes user empowerment by supporting Romanized Nepali and English, making government services more accessible and fair. It changes the complex task of dealing with government processes into a simple and inclusive experience.

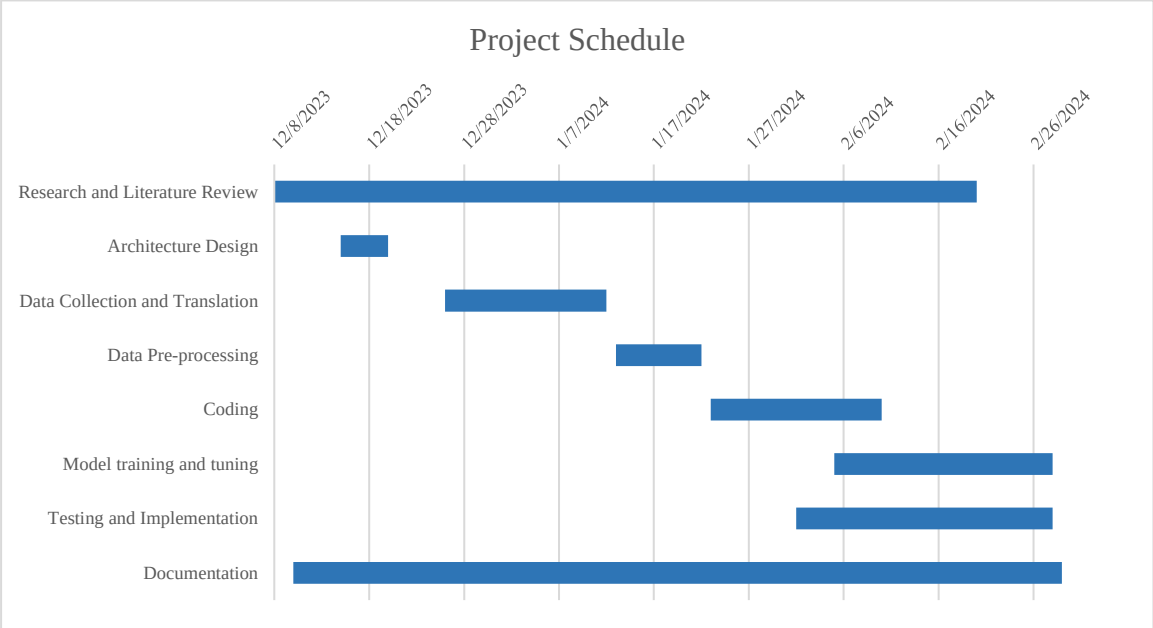
The future plans for Samadhan include increasing the range of services and creating a fully functional web application to bring everything together. The project team is planning on adding user feedback to continually enhance the chatbot, making sure it meets the users' needs.

In essence, Samadhan is more than just a chatbot; it's a forward-looking tool that connects and streamlines Nepal's government services. It represents technological advancement, easy access to services, and a commitment to user empowerment. As Samadhan continues to grow, it confirms the role of technology in improving society's overall well-being.

10 APPENDICES

Appendix A: Project Timeline

Table 10-1 Gantt chart Showing Project Schedule

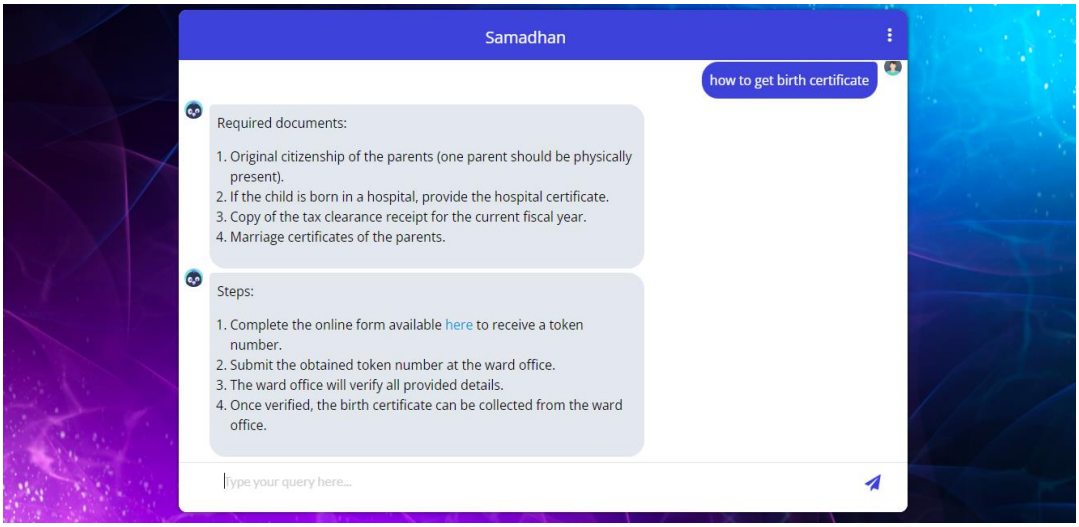


Appendix B: Project Budget

Table 10-2 Project Budget

S.N.	Particulars	Price (Rs.)	Quantity	Total Price (Rs.)
1	Printing	300	10	3000/-
2	Miscellaneous	1000	1	1000/-
TOTAL				4000/-

Appendix C: User Interface



REFERENCES

[1] E. Adamopoulou and L. Moussiades, "An Overview of Chatbot Technology," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*, 2020.

[2] H. Bansal and R. Khan, "A Review Paper on Human Computer Interaction," *nternational Journals of Advanced Research in Computer Science and Software Engineeing*, vol. 8, no. 4, 2018.

[3] L. Klopfenstein, S. Delpriori, S. Malatini and A. Bogliolo, "The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms," in *The 2017 Conference*, Urbino, 2017.

[4] R. K. Sharma, "An Analytical Study and Review of open source Chatbot framework, Rasa," *International Journal of Engineering Research & Technology (IJERT)*, vol. 9, no. 6, 2020.

[5] T. Bunk, D. Varshneya, V. Vlasov and A. Nichol, "DIET: Lightweight Language Understanding for Dialogue Systems," arXiv, 11 May 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2004.09936. [Accessed 25 January 2024].

[6] A. Jiao, "An Intelligent Chatbot System Based on Entity Extraction Using RASA NLU and Neural Network," *Journal of Physics: Conference Series*, vol. 1487, no. 012014, 2020.

[7] Diyo.ai, "Localizing AI | DIYO.AI," [Online]. Available: https://diyo.ai/muna. [Accessed 15 12 2023].

[8] S. Government, "AskGov | Q&A: Your questions answered," Open Government Products, [Online]. Available: https://ask.gov.sg/. [Accessed 15 12 2023].