PLATICA: Personalized Language Acquisition Training & Instruction Chatbot Assistant

Nam Tran
Philip Cori
Andres Morales
Evan Johnson
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Nam Tran
Philip Cori
Andres Morales
Evan Johnson

ENTITLED

PLATICA: Personalized Language Acquisition Training & Instruction Chatbot Assistant

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Thesis Advisor

N. Ling
Department Chair
PLATICA: Personalized Language Acquisition Training & Instruction
Chatbot Assistant

by

Nam Tran
Philip Cori
Andres Morales
Evan Johnson

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 4, 2020
ABSTRACT

English is immensely important and useful in our society, however there are many people across the world who are learning English as a second language and have limited options to practice. Casual English conversations with native speakers is one of the most proven and immersive ways to practice a language. However, not everyone has those opportunities or the resources to attend ESL classes. We aim to solve this issue with our project PLATICA, a robust, low-cost mobile application that anyone can use to build experience conversing in English. PLATICA takes advantage of state-of-the-art deep learning and natural language processing techniques to emulate real conversations while providing real-time grammar feedback to assist the user in improving their English skills. PLATICA as an end-to-end learning pipeline could also be adapted to other languages in the future.
Acknowledgements

We would like to thank our advisor, Dr. Yi Fang, for his guidance and support throughout the course of our project. His dedication and expertise in the fields of machine learning and natural language processing were invaluable in creating a functional and usable application. Next, we would like to thank Álvaro Peris for developing the NMT-Keras library and helping us through challenges of using this library in developing our models. He provided invaluable advice and guidance in using the library in the form of published Github Issues and we thoroughly enjoyed the process of helping develop the library as a whole. We would also like to thank Travis Ebesu for providing us with helpful technical support on the SCU DeepInfo GPU server. Lastly, we would like to thank Dr. Ying Liu for taking the time to be our reader for this project as our Honors Thesis for the University Honors Program.
# Table of Contents

1 Introduction ........................................... 1
1.1 Problem ............................................. 1
1.2 Solution ............................................. 1
1.3 Motivation .......................................... 1

2 Related Work .......................................... 2
2.1 Chatbot ............................................. 2
  2.1.1 Mitsuku (2005) .................................. 2
  2.1.2 Microsoft XiaoIce (2014) ......................... 2
  2.1.3 Microsoft Tay (2016) ............................ 2
  2.1.4 Google Meena (2020) ......................... 3
  2.1.5 Facebook Blender (2020) ....................... 3
2.2 Language ............................................ 3
  2.2.1 Duolingo ....................................... 3
  2.2.2 ELSA ............................................ 3
  2.2.3 Andy ............................................ 4
2.3 Grammar ............................................ 4
  2.3.1 Grammarly ...................................... 4
2.4 Our Contribution ................................... 4

3 Requirements .......................................... 5
3.1 Functional .......................................... 5
  3.1.1 Grammar Checker ............................... 5
  3.1.2 Chatbot ........................................ 5
  3.1.3 Other ........................................... 5
3.2 Non-functional .................................... 5
  3.2.1 Ease of use .................................... 5
  3.2.2 Security ....................................... 6
  3.2.3 Compatibility .................................. 6

4 Diagrams .............................................. 7
4.1 Use Cases .......................................... 7
4.2 Activity Diagrams .................................. 8

5 Application ............................................ 9
5.1 Home Page ......................................... 9
5.2 Registration Page ................................... 10
5.3 Login Page ......................................... 11
5.4 Landing Page ....................................... 12
5.5 Chatbot Interface .................................. 13
5.6 Progress Page ...................................... 14
6 System Implementation
6.1 Front-End ........................................... 15
6.2 Back-End ........................................... 15
6.3 Speech-to-Text Capabilities ......................... 15
6.4 Login and Authentication .......................... 16
6.5 Data Storage ....................................... 16

7 Technologies Used & Hardware Setup .......... 17
7.1 Back-end ........................................... 17
  7.1.1 Mobile Application ............................ 17
7.2 Grammar Checker ................................. 17
7.3 Chatbot ............................................ 17
7.4 Hardware Setup .................................. 17

8 System Architecture .................................. 19
8.1 Data Flow Architecture ........................... 19
8.2 Human Operators .................................. 19
8.3 System ............................................. 19

9 Design Rationale .................................... 20
9.1 React Native ....................................... 20
9.2 Flask ............................................. 20
9.3 Keras ............................................. 21
9.4 NMT-Keras ....................................... 21

10 Conversational Chatbot ............................. 23
10.1 Implementation ................................... 23
  10.1.1 Using Sequence to Sequence models ........ 23
  10.1.2 Data Preparation ............................. 23
10.2 Experiments ...................................... 23
  10.2.1 Metric ......................................... 23
  10.2.2 Datasets ...................................... 24
  10.2.3 Architectures ................................ 26
  10.2.4 Model Parameters ........................... 26
  10.2.5 Context ....................................... 26
  10.2.6 Diversity Promoting Decoding Functions .... 26

11 Grammar Checker ................................... 28
11.1 Overview ......................................... 28
11.2 Grammar Checker Pipeline ....................... 28
11.3 Experiments ...................................... 30
11.4 Evaluation ....................................... 31

12 Obstacles Encountered & Lessons Learned .... 34
12.1 Challenges Unique to Conversational Chatbot . 34
12.2 Challenges Unique to Grammar Checker ........ 34
12.3 Challenges for Both .............................. 34

13 Testing Procedure .................................. 35
13.1 Front-End ......................................... 35
13.2 Back-End ......................................... 35
13.3 Machine Learning ................................ 35
14 Implications
14.1 Privacy 36
14.2 Societal 36
14.3 Safety 36

15 Development Timeline
15.1 Timeline 37

16 Future Work
16.1 Public Release & Expansion to More Languages 38
16.2 Multiplayer Mode 38
16.3 Conversational Chatbot Refinement 38
16.4 Grammar Checker Refinement 39
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Use Case Diagram</td>
<td>7</td>
</tr>
<tr>
<td>4.2</td>
<td>Activity Diagram</td>
<td>8</td>
</tr>
<tr>
<td>5.1</td>
<td>Home Page</td>
<td>9</td>
</tr>
<tr>
<td>5.2</td>
<td>Registration Page</td>
<td>10</td>
</tr>
<tr>
<td>5.3</td>
<td>Login Page</td>
<td>11</td>
</tr>
<tr>
<td>5.4</td>
<td>Landing Page</td>
<td>12</td>
</tr>
<tr>
<td>5.5</td>
<td>Example Conversation 1</td>
<td>13</td>
</tr>
<tr>
<td>5.6</td>
<td>Example Conversation 2</td>
<td>13</td>
</tr>
<tr>
<td>5.7</td>
<td>Progress over 5 week period</td>
<td>14</td>
</tr>
<tr>
<td>5.8</td>
<td>Progress over 5 month period</td>
<td>14</td>
</tr>
<tr>
<td>8.1</td>
<td>Data Flow Architecture</td>
<td>19</td>
</tr>
<tr>
<td>9.1</td>
<td>Sequence to Sequence Model with Attention (Image Credit: NMT-Keras)</td>
<td>21</td>
</tr>
<tr>
<td>10.1</td>
<td>Conversation with OpenSubtitles-trained chatbot</td>
<td>24</td>
</tr>
<tr>
<td>10.2</td>
<td>Good example</td>
<td>25</td>
</tr>
<tr>
<td>10.3</td>
<td>Bad example</td>
<td>25</td>
</tr>
<tr>
<td>11.1</td>
<td>Complete Grammar Checker Pipeline</td>
<td>29</td>
</tr>
<tr>
<td>11.2</td>
<td>Training Progress of LSTM 256 (Best Model)</td>
<td>32</td>
</tr>
<tr>
<td>15.1</td>
<td>Development Timeline</td>
<td>37</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Problem

Though speaking English is a crucial skill in today’s society, many people around the world and in our community lack the time and resources to learn this language. Adult ESL classes often fail to meet the needs of students who want to progress (11). Current applications such as Duolingo are often shallow in testing knowledge and do not offer sufficient learning capabilities. Another common problem with learning a new language is that people struggle to find other native speakers to practice with. This leaves many people unpracticed in English interactions and unprepared for more complicated real-life interactions, whether in their personal or professional lives.

1.2 Solution

Language development pedagogy can be greatly improved by taking advantage of modern technological advancements in areas such as machine learning and natural language processing. Using these technologies, we propose PLATICA (Personalized Language Acquisition Training & Instruction Chatbot Assistant), a mobile application to help people learn English on their own, with the app providing real-time feedback on grammar mistakes. The reason for this acronym is that in Spanish, platica means to chat. PLATICA will act as a friendly chatbot that holds conversations with users, giving them opportunities to practice their speaking and writing skills on their own time or on a commute and offering grammar feedback.

1.3 Motivation

Observing modern ESL classes’ use of antiquated technology and training videos with no curriculum motivated us to develop a tool to assist non-native speakers to learn English more efficiently and on their own schedule, while enjoying the experience at the same time. Existing language learning platforms such as Rosetta Stone Duolingo have their own shortcomings and are often boring and unengaging. We want to provide this application to offer a new, unique, and engaging platform to learn English.
Chapter 2

Related Work

2.1 Chatbot

2.1.1 Mitsuku (2005)

Mitsuku (2) is a five-time Loebner Prize Turing Test winning conversational chatbot. It is able to hold realistic conversations on a wide range of topics. Mitsuku does not give any advice and corrections to the user with respect to his or her skills. Mitsuku employs a rule based system, meaning it does not rely on deep learning networks but instead on a large set of predefined rules to generate responses. Nevertheless, its ability to communicate realistically resembling a human is a fantastic feat.

2.1.2 Microsoft XiaoIce (2014)

Microsoft XiaoIce (26) is a chatbot that has reached over 660 million online users worldwide. XiaoIce senses human emotional state and constructs responses in congruence with this sensed state. In doing so, the chatbot develops seemingly real emotional connections with its human chatter. Users have claimed that XiaoIce helped talk them through difficult times in their lives such as breakups, loneliness, etc. One novelty of XiaoIce is its vectorization of attributes. This allows the chatbot to store factual attributes in a vector and refer to these attributes to reduce the randomness of responses.

2.1.3 Microsoft Tay (2016)

Tay (14) was a chatbot aimed at mimicking the language patterns of a 19-year old American girl. The bot was programmed to learn from interacting with Twitter users. Unfortunately, Twitter users took advantage of this feature and trained Tay to respond users with vulgar, inappropriate tweets. The chatbot was shortly taken down by Microsoft. Tay was exemplary of the importance of NSFW filters. This is one reason why companies such as Google do not release their chatbots to the public until they have performed extensive testing.
2.1.4 Google Meena (2020)

Meena (6) is an open-domain chatbot developed by Google that was released in January 2020. At its time of release, it claimed to be the most sensible and specific chatbot yet, with an SSA score of 79% compared to Mitsuku’s 56%. A typical human conversation has an SSA score of 86%. It was trained for 30 days on 867 million conversational exchanges of open domain social media data, including 2.6 billion parameters contained within the model. Meena implements an Evolved Transformer seq2seq architecture designed to improve perplexity. The model utilizes a single encoder block with 13 decoder blocks, a hyper-parameter choice discovered to be the key to higher conversation quality. Meena is evaluated on Sensibleness and Specificity Average (SSA), a metric quantifying the sensibleness and specificity of chatbot responses (5).

2.1.5 Facebook Blender (2020)

Facebook Blender (20) was released very recently in April 2020. Coming at the heels of Google Meena, Facebook now claims this to be the best conversational chatbot developed so far. They utilize an ensemble approach, taking the predictions of 3 different models. These models are trained on Wikipedia, EmpathyDialogue, and PersonaChat datasets respectively, blending together aspects of personality, empathy, and knowledge. The models were initially trained on a Reddit corpus containing 1.5 billion comments. Subsequently, the model was further fine tuned by training it on various corpuses representing friendly, emotional, and topical sentences. Overall this creates a very robust chatbot that can have very natural conversation with users.

2.2 Language

2.2.1 Duolingo

Basic language learning applications have existed for a long time beginning with Rosetta Stone in 1992. Duolingo is a popular mobile application that helps users learn a language of their choice. Nevertheless, the app is limited because it focuses on memorization of words and phrases rather than inducing natural conversations in the target language.

2.2.2 ELSA

ELSA is a mobile app that helps users adopt an American accent. Using AI, it pinpoints the users’ pronunciation mistakes and focuses mainly on learning and pronouncing correctly useful and common words, phrases, and sentences. A user is provided with a wide range of pronunciation lessons tailored to his or her level, determined by an initial test. The service does not include grammatical checking nor a writing component. ELSA also does not include any form of a conversational chatbot. This means that it is not possible to practice realistic speaking nor writing in topics that are beyond those that are pre-installed with the app.
2.2.3 Andy

Andy is a mobile app that enables users to learn and practice English by communicating with a chatbot. The user can practice casual day-to-day interactions, ask questions, and discuss various topics that they like. The app also includes grammar lessons and vocabulary lessons. Nevertheless, grammar checking capabilities are not integrated into the chatbot module itself.

2.3 Grammar

There are currently dozens of approaches to automated grammar error checking. The first began to be developed in the early 1980s, utilizing a rule-based approach. However, these did not perform well as they could not capture nuances of the English language and also required the maintenance of hundreds of grammar rules by English experts. Machine learning approaches first emerged in the 2000s, followed by hybrid systems that combine rule-based and machine learning approaches, which have shown to produce the best results.

2.3.1 Grammarly

Grammarly is a popular Google Chrome extension that offers real-time grammar corrections on users’ screens in written form on websites. Once installed and active on a site, Grammarly underlines identified grammar mistakes or overused words and offers replacements after interacting with it.

2.4 Our Contribution

Finally, our contribution to this field is combining both a conversational chatbot and a grammar-correction system in an attempt to provide natural conversation with helpful feedback in one full package. While noise insertion is not novel (it has been done in (24) and (13)), to our best knowledge it has not been attempted in combination with deep learning to provide an end-to-end learning experience. We do this through a simple to use mobile application.
Chapter 3

Requirements

We split the requirements for our project into three parts: functional requirements, non-functional requirements, and constraints. For the requirements, we also fulfilled them in order of most to least important.

3.1 Functional

3.1.1 Grammar Checker

Firstly, the grammar checking functionality of our project was of utmost importance as the project was based around it. We specifically made it a requirement that PLATICA would be able to identify grammar mistakes made by the users and communicate those mistakes and the corrections like an ESL teacher might.

3.1.2 Chatbot

The requirement of the chatbot was also crucial to the project. For an English learner, a stimulating English conversation is the best way to learn a language quickly, and so it was critical for the chatbot to be sensible and specific and keep the user engaged with the conversation to enhance learning.

3.1.3 Other

Other functional requirements were the compatibility of the mobile application to be able to run on both iOS and Android. Lastly, we wanted to make sure that the app could be used hands-free so that a user could use it during a commute.

3.2 Non-functional

3.2.1 Ease of use

The app would have to be easy to navigate, especially for a user who is learning English. The chatbot would need to provide a friendly, encouraging, and respectful conversation in order to maintain the users’ attention and interest.
Grammar corrections would need to be made quickly and in an understandable format.

3.2.2 Security

Our system had to be secure with the user’s personal data that we ask for and ensure the security of our login system, keeping credentials encrypted. Any user conversations could be sensitive, so the personal chatbot conversations also needed to be secure.

3.2.3 Compatibility

One design constraint that we had for this project is that it had to be a mobile application, downloadable from the Google Play Store or Apple App Store. We also needed to take this into consideration when storing data.
Chapter 4

Diagrams

4.1 Use Cases

The actor in the context of our application is a user who wants to improve their conversational English. The user have the ability to register, login, start a conversation with the chatbot, view progress, and logout. Furthermore, the user will have the ability to speak to PLATICA hands free. Through speech to text capabilities, our app can capture user voice input and generate responses while they are performing various hands free activities such as commuting to work. These tasks will be offered to the user through a simple, easy to navigate interface. Figure 4.1 highlights our user and his or her multiple options for interacting with the system.

![Use Case Diagram](image)

Figure 4.1: Use Case Diagram
4.2 Activity Diagrams

Our application will have a number of activities contributing to its functionality. Firstly, the user will have the option to register for an account with our service, assuming they do not currently have one. Once registered, their information is persisted in a database, through which all their progress and additional information will be stored. After registering, users are redirected to our landing page, from which will they have the option to either view their progress or begin chatting with PLATICA. While chatting, the user may type or speak to the chatbot (assuming they enabled speech to text capabilities) and receive responses. After chatting with PLATICA, the user may end chatting and return to the landing page. From the landing page, the user may toggle settings, view his or her progress, or logout.

![Activity Diagram](image.png)

Figure 4.2: Activity Diagram
Chapter 5

Application

5.1 Home Page

Our home page is the first interface that users will see when they start up our application. From here, they will have the choice to either login to an existing PLATICA account or register for a new one.

Figure 5.1: Home Page
5.2 Registration Page

Through our registration page, users will be able to choose a unique username and create an account with an associated password.

Figure 5.2: Registration Page
5.3 Login Page

PLATICA users can login to an existing account using the username and password they created during registration. After doing so, they will be redirected to our landing page, shown in Figure 5.4.

![Login Page](image)

Figure 5.3: Login Page
5.4 Landing Page

Our landing page will be the gateway through which users can engage in their learning experience. From this interface, users will be able to either start chatting with PLATICA or view their updated progress. In the upper right is a settings button which allows the user to toggle the hands-free mode.

Figure 5.4: Landing Page
5.5 Chatbot Interface

Our chat page provides a simple interface for users to interact with our PLATICA chatbot. As users chat with PLATICA, they will receive real-time grammar corrections and chat responses. Through speech-to-text capabilities, they will also be able to interact with PLATICA hands-free.

Figure 5.5: Example Conversation 1

Figure 5.6: Example Conversation 2
5.6 Progress Page

The progress modal provides a visually appealing interface through which users can view their progress as they interact with PLATICA. The modal will tell them the number of messages sent, words per message, interaction streak with the application, and their overall grammatical score. They can also view a graph of their grammatical score over the past 5 days, months, or years.

Figure 5.7: Progress over 5 week period

Figure 5.8: Progress over 5 month period
Chapter 6

System Implementation

Links to GitHub repositories:

- Back-End: https://github.com/evanj354/Platica_Backend
- Chat UI: https://github.com/evanj354/ChatUI
- Conversational Chatbot: https://github.com/philipcori/PLATICA_Chatbot
- Grammar Checker: https://github.com/NamTran838P/PLATICA_Grammar_Checker

6.1 Front-End

We built the front-end of our application in React Native. This allowed for cross platform development and deployment. We built a minimalist UI to make it easier for users to navigate. React Native charts allowed us to easily create visual representations of the users’ progress.

6.2 Back-End

Our back-end code was written in python using the Flask micro-web framework. User information is stored in an SQLite database and queries using the Python SQLAlchemy toolkit. Our chatbot models are stored on the back-end and invoked by our server. The Flask server is hosted on an AWS EC2 instance.

6.3 Speech-to-Text Capabilities

The user has the ability to toggle hands free speech capabilities while using our app. For speech-to-text capabilities, we used Expo Audio to capture the user’s voice and convert it to the appropriate audio file format given the use device. Our back-end then receives the audio file via a POST from our front-end and converts it to plain text by making an API call to the Google Cloud platform. The text is then relayed back to our front-end via a JSON payload. After receiving the payload, our app reads the text aloud to the user using Expo Speech.
6.4 Login and Authentication

Upon entering one's credentials on our login page, the data is sent in a JSON payload to our back-end. After receiving the credentials, the back-end first checks that the user is registered for an account and ensures that their password hash matches that hash associated with the registered username. After confirming this, our server sends a JSON payload to the front-end containing an authentication confirmation. Upon receiving this confirmation, the user is now authenticated and given permission to navigate the various pages of our application. They may continue to do so until the app session terminates or they choose to logout.

6.5 Data Storage

Data is stored in a SQLite database on our server. Each user will have their username and hashed password stored in our database as well as their conversations, progress history, and statistics.
Chapter 7

Technologies Used & Hardware Setup

7.1 Back-end

For the back-end of our application we used Flask, a micro-web framework written in Python. Data is stored in an SQLite relational database, using SQLAlchemy to abstract interactions. The server was deployed on an Amazon AWS EC2 instance.

7.1.1 Mobile Application

For the front-end of our application we used React Native with the Expo SDK. We used typescript to statically type our JSX code as well as various React Native and SDK libraries.

7.2 Grammar Checker

For the grammar checking functionality, we used NMT-Keras, a tool that provides Neural Machine Translation with Keras. We used Keras because it is a deep learning framework that abstracts TensorFlow operations and models.

7.3 Chatbot

For the chatbot, we similarly used NMT-Keras and Keras.

7.4 Hardware Setup

We used two NVIDIA GPU servers to train our deep learning models: Deep Info and WAVE. The configurations are as follows:

- Deep Info:
  - 4 x GTX TITAN X
  - Each has 16 GB of Graphics Memory
– Each use session is not limited in terms of time

• WAVE:

  – 2 x TESLA V100

  – Each has 32 GB of Graphics Memory

  – Each use session is limited to 48 hours
Chapter 8

System Architecture

8.1 Data Flow Architecture

Because our system is centered around large data processing, a data flow architectural diagram is an effective way to visualize our architecture. 8.1 below shows the data flow of our system. In the first step, user login information is persisted in the database. As the user converses with the chatbot, our system will use the speech to text API to generate textual data and generate appropriate responses to this text using natural language processing. It will then display this generated response on the UI.

![Data Flow Architecture Diagram]

Figure 8.1: Data Flow Architecture

8.2 Human Operators

There are no human operators required to keep the system running. As long as users are conversing with the chatbot, responses will be generated automatically.

8.3 System

We will store user information such as credential and learning progress in relational database. Some considerations for our database software include MongoDB and PostgreSQL.
Chapter 9

Design Rationale

9.1 React Native

React Native is a mobile application framework developed by Facebook. It is a cross-platform framework, meaning it can be used to develop apps with cross platform capabilities. This contributes to flexibility and usability. Additionally, React is rapidly rising in popularity within the software industry, seeing continuous updates and extensive online resources. React is unique in that it completely manages the DOM independently using a Virtual DOM. It utilizes React components to store application states for quick and responsive interfaces.

React Native comes with several other benefits:

- Optimal Performance
- Large Community
- Modular Architecture
- Simple UI
- Stable and Reliable

9.2 Flask

Flask is a Python-based micro-web framework. Since our chatbot and grammar correction models were written entirely in python using keras, it was easy to integrate them into our back-end. Flask supports quick and easy development with its build-in development server and offers easy-to-use middle ware for routing requests to RESTful API endpoints. Furthermore, because it is a micro-web framework, Flask did not restrict the libraries we used on top of it.
9.3 Keras

Keras is a powerful machine learning library built on top of the Tensorflow back-end. It provides a host of useful machine learning tools that are critical for robust model generation. Additionally, built in models and open source code facilitated our process of generating a natural language processor. Keras also supports distributed GPU training and a wide range of deployment options, which were useful once our models were incorporated into our back-end service.

9.4 NMT-Keras

We used a library called NMT-Keras, developed by Álvaro Peris and Francisco Casacuberta (7). The library is built on top of Keras, providing the building blocks to seamless neural translation models. It allowed us to save a significant amount of time in our experiments. NMT keras also made it easy to adjust parameters such as encoding and decoding layers, hidden size, batch size, learning rate, and beam size for predictions. Additionally, it supports LSTM, GRU, as well as other Transformer based seq2seq models. The library supports beam search decoding and a multitude of metrics to evaluate models. Lastly, NMT-Keras helps take care of many of the preprocessing tasks required for neural machine translation, such as word tokenization, capitalization elimination, sentence padding, and vocabulary limitation. Below we provide a diagram of the architecture of an attention-based sequence to sequence model, which we used for both our chatbot and grammar checker.

![Sequence to Sequence Model with Attention](Image Credit: NMT-Keras)

Sequence to sequence (seq2seq) models were first introduced by Google in 2014 to provide a mechanism for predicting sequences given sequences as input (22). In our case, these sequences are English sentences. They have proven useful for a multitude of applications, including language translation, speech recognition, and image captioning. At
a high level, these models consist of an encoder and decoder. The encoder consists of a sequence of recurrent neural networks (RNNs) which are trained to find patterns in the input sequences, producing a hidden state for each token. Once the encoder reads the entire input, it produces an output state that is then fed to the decoder to produce the first word of the output sequence. This in turn produces a new hidden state, which is used to predict the next word, and so on. This process repeats until the decoder predicts the end of the sequence. To produce sequences, we use an algorithm called beam search that finds the n most likely sequences and returns the best one.
Chapter 10

Conversational Chatbot

10.1 Implementation

Implementing the conversational chatbot required performing a significant amount of research on modern work done in this field, some of the most recent being released in April 2020. It was essential to use these papers’ approaches in developing a functional chatbot that engages users in conversations.

10.1.1 Using Sequence to Sequence models

As mentioned, we use the NMT-Keras library to implement our chatbot. At its core, the conversational chatbot can be framed as a neural machine translation problem. Except rather than predicting how to produce an English sentence given a French sentence, we train a model to produce a response to a given statement.

10.1.2 Data Preparation

The chatbot requires using a dataset of conversational data. The input sequences are sentences in the conversation, while the output sequences are the responses that were said to those statements. Thus, our dataset becomes a series of these input-output pairs, which we created using a Python script. In preparing our data, we filter out very long sentences of length over 24 tokens, eliminate special characters other than regular punctuation, and ensure no profane language is included. Lastly, we shuffle our samples.

10.2 Experiments

We ran several experiments to develop the best conversational chatbot we could given our constraints in time, computing resources, and access to data.

10.2.1 Metric

Google proposes a general metric for evaluating conversational chatbots: the SSA score, which stands for Sensibleness and Specificity Average (6). However, this can only be judged by humans. To automate this metric, Google proposes
the perplexity metric, which measures how "uncertain" or "perplexed" the model is in predicting the words of the output sequence. They claim to find a strong correlation between perplexity and SSA score. Thus, we used this perplexity when evaluating the chatbot model. After testing with several models, however, we found that this metric is only a rough guideline for how specific and sensible the chatbot is. It can give insight on how well a model trains over time, but does not ultimately correctly determine the best chatbot. This was done instead by manual testing, giving various types of statements such as personal questions, knowledge-based questions, statements that evoke emotional responses, and greetings and salutations.

10.2.2 Datasets

- OpenSubtitles

This is a 13GB open source dataset of movie subtitles taken from hundreds of films (3). Although the data consists of interpersonal dialogue, we found this dataset not to give the best results. This is likely due to a few reasons. Firstly, the speakers are not labeled. This makes it impossible to determine which statements were said in response to which other statements, since there can be many speakers in a single scene. Additionally, the domain of the dialogue did not fit our use case. The context and types of statements in movie subtitles are very different in nature compared to the types of conversations we try to evoke in our application. We trained a seq2seq model on 60MB of this data for 7 epochs, which took a total of around 3 days. Figure 10.1 shows an example conversation with a chatbot trained on this dataset.

![Conversation with OpenSubtitles-trained chatbot](image_url)

Figure 10.1: Conversation with OpenSubtitles-trained chatbot

24
The chatbot relies heavily on over-represented statements such as "I don't know." It struggles to give responses that either sensible or specific. It is clear that such a chatbot is not realistically capable of holding a conversation.

- **PersonaChat**

  This is a dialog dataset of 162,000 utterances between crowd workers who were instructed to portray a given persona (25). An example of a persona might be, "I love the beach. My dad has a car dealership. I just got my nails done. I am on a diet now. I like horses." It was originally intended for the purpose of creating a conversational chatbot, so this clearly serves as a much higher quality dataset. Conversations consist of two individuals getting to know each other which is much closer to what we seek to emulate in our application.

- **EmpatheticDialogues**

  Lastly, we experimented with the EmpatheticDialogues dataset, a conversational dataset of 60,000 utterances between individuals having conversations grounded in emotional situations (19). This is another high quality dataset that provides another dimension to our chatbot: empathy.

We found a combination of the PersonaChat and EmpatheticDialogues datasets yielded the best chatbot. Training a seq2seq model with ideal parameters gave us a chatbot that was able to portray a specific persona, while also being capable of expressing empathy. The PersonaChat dataset made the bot capable of answering specific, personal questions, while the EmpatheticDialogues dataset allowed it to respond with empathy to statements that call for such a response. Figure 10.2 demonstrates a conversation where PLATICA is able to answer specific personal questions sensibly and specifically. Additionally, it is capable of responding with empathy to statements like "I am so tired today...". However, as can be seen in Figure 10.3, the chatbot still struggles at times by generating overly generic responses, or making no sense at all. This is a common problem even in cutting edge research.

![Figure 10.2: Good example](image1.png)

![Figure 10.3: Bad example](image2.png)
10.2.3 Architectures

Next, we experimented with two different sequence to sequence architectures: LSTM-based versus Transformer-based. The LSTM based architecture was first developed in 2014 by Google to provide a model that can predict a sequence of data given an input sequence of data, as mentioned in section 10.4. Transformers were introduced in 2017 by Google and were shown to perform better for certain machine translation problems. They are simpler than the traditional model in that they eliminate all recurrent neural networks and convolutions in the model, instead using purely an attention mechanism for decoding. Google Meena and Facebook Blender both use Transformer models. We found that the traditional LSTM-based seq2seq model performed better. This can be attributed to our limited dataset, as Transformers were shown to perform better on extremely large datasets.

10.2.4 Model Parameters

There are several model hyperparameters that determine how well it fits the training data. The ones we experimented with include: number of encoding and decoding layers, hidden size, batch size, learning rate, and beam size. We used Tensorboard to analyze our training sessions to determine whether our model was improving over time, using the training and validation perplexity curves. We found the following parameters to yield the best results:

- Number of encoding layers = 2
- Number of decoding layers = 2
- Number of hidden layers = 100
- Batch size = 128
- Beam size = 20
- Learning rate = 0.001

10.2.5 Context

We also experimented with using more than the last statement as context for the chatbot’s response. We trained a few models using 3 statements as context. However, this showed to greatly deteriorate our results, as the chatbot began to generate sentences that were grammatically incorrect or nonsensical.

10.2.6 Diversity Promoting Decoding Functions

Lastly, we attempted to solve the problem of generating overly generic sentences. We implemented Google Meena’s sample-and-rank algorithm that samples using a certain temperature when decoding (6). This essentially levels out the probabilities of the next predicted word, making it easier to predict more unique words rather than the ones that
commonly have a high probability. It then selects the sentence with highest probability, normalized by its length. Unfortunately, however, for this method to be successful, the model must be very well trained and achieve a perplexity comparable to Google Meena’s which is 10. As we could not achieve this given our limited time and resources, this algorithm proved to give unique but grammatically incorrect sentences.
Chapter 11

Grammar Checker

11.1 Overview

The grammar checker was trained on a large dataset of famous novels and grammatically correct conversations. It used the same seq2seq model with attention mechanism that the conversational chatbot used. The novels chosen were well-written English classics:

- *Wuthering Heights* by Emily Brontë
- *War and Peace* by Leo Tolstoy
- *Jane Eyre* by Charlotte Brontë
- *The Scarlet Letter* by Nathaniel Hawthorne
- *The Great Gatsby* by F. Scott Fitzgerald
- *Pride and Prejudice* by Jane Austen
- *The Picture of Dorian Gray* by Oscar Wilde
- *The Communist Manifesto* by Karl Marx
- *Great Expectations* by Charles Dickens

The short conversations were compiled using various bilingual datasets retrieved from the Tatoeba Project (1; 4).

11.2 Grammar Checker Pipeline

Fig. 11.1 depicts the complete grammar checker pipeline. As shown by the figure, we created a dataset for the grammar checker by stochastically adding various types of noise (grammatical mistakes) to well-written and grammatically correct text. Specifically, for each clean sample from the originally clean dataset, we created multiple variants of that
We then had two sets of text: one with noise and one without noise. Table 11.1 shows an overview of the final processed dataset. A validation split of 15% is applied to the dataset to minimize overfitting. The number of samples and vocabulary size were sufficiently large to ensure decent generalization. A neural machine translation model was then trained to correct the set with noise, using the set without noise as ground truth. Specifically, the stochastic error generator added the following common ESL mistakes:

- Preposition: randomly swap a particular adjective with an adjective from a hard-coded list.
- Comparative: randomly introduce mistakes with comparison structures (more, less, the most, the least).
- Modal Verb: randomly switch to different modal verbs (can, could, may, might, will, would).
- Verb Tense: randomly switch the sentence’s verb tense.
- Pluralization: randomly switch between singular and plural forms for a particular noun.

- Misuse and Misspelling: a hard-coded list of words that are commonly misused and misspelled, even among native speakers (12; 16).

- Dropout: randomly drop a particular token from the sample.

Each error type had a 30% of being added to a sample. We wanted to ensure a realistic amount of errors being added to simulate English sentences generated by ESL learners. As can be seen in Fig. 11.1, duplication was used to ensure a balanced mixture of correct and incorrect English sentences. This taught the model to only correct wrong sentences and to let pass sentences that were already correct. Duplicating each sample means that the model was able to see different possible mistakes with any given sentence structure.

### 11.3 Experiments

We trained our seq2seq model and tuned the hyperparameters using best practice, as described in (17). The encoder RNN type was LSTM, while the decoder RNN type was Conditional LSTM. This was because LSTMs yielded better predictive performance than Transformers did, based on our testing. The following hyperparameters were considered:

- Batch size: 32, 64, 128

- Number of encoder layers: 128, 256, 512

- Number of decoder layers: 128, 256, 512

- Source text embedding size: 128, 256, 512

- Target text embedding size: 128, 256, 512

- Skip vector hidden size: 128, 256, 512

- Attention size: 128, 256, 512

- Encoder hidden size: 128, 256, 512

- Decoder hidden size: 128, 256, 512

- Beam size: 1, 2, 3, 4, 5

- Length norm factor: 0.5, 1.0

After many trials, it was found that the configuration values listed in red provided a model with the best predictive performance.
11.4 Evaluation

In addition to manual testing with arbitrary sentences, we evaluated our grammar checker using a wide variety of metrics, all of which are listed in Table 11.2. The metrics listed in Table 11.2 are out of 1, with the exception of CIDEr, which is out of 100. Also, for all of the metrics used, higher is better, with the exception of TER, for which lower is better. The leftmost column of Table 11.2 shows the various LSTM models that were tested, with 128, 256 and 512 referring to the size of the model’s hidden layer. The most important metric in this table is the Bleu 4 score, which was used in both validation and training. Overall, the LSTM 256 model yielded the best predictive performance, with the highest scores across all evaluation metrics.

Our evaluation metrics were implemented by the pycoco package, which was integrated with the NMT-Keras library. The pycoco package included a wide variety of metrics for various natural language processing (NLP) uses. Although not all of these were directly related to grammar checking, we did use all of them during validation as they exhibited a strong correlation with each other. A well-trained model would perform well across all the metrics and vice versa.

Bleu (Bilingual Evaluation Understudy) is a popular metric for comparing a candidate translation of text to one or more reference translations (18). Although originally developed for translation, Bleu can also be used for other NLP tasks. This metric compares n-grams of the candidate sample with the n-grams of the reference translation and counts the number of matches (18). As Table 11.2 shows, we considered only 4-gram and fewer when using the Bleu score. We specifically used the Bleu_4 not only for validation but also for training, as it is a versatile and powerful metric in NLP. It is known that higher order n-grams of the Bleu metric do reflect a sample’s grammatical well-formedness (8). Using this popular metric enabled us to make comparisons between our NLP model and others. As such, this is our most important metric.

CIDEr (Consensus-based Image Description Evaluation) is a metric used to evaluate image descriptions using human consensus, a task which remains challenging in computer vision and NLP (23). Although this metric is not typically used for grammar checking and language translation, we did notice a clear correlation between this metric and Bleu_4 when performing grammar checking validation. As such, we decided to include CIDEr in our list of evaluation metrics.

We also used METEOR, a metric for translation evaluation based on unigram matching between the candidate
Figure 11.2: Training Progress of LSTM 256 (Best Model)

translation and reference translations (8). The METEOR metric was designed to address the weaknesses of the Bleu score. Nevertheless, we still noticed a strong correlation between the Bleu score and the METEOR score.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a metric for evaluating how close a summary is by comparing it to other ground-truth summaries created by humans (15). The ROUGE_L variant that was used in validating our grammar checker emphasizes the longest common subsequence when comparing two summaries. This metric is not very relevant to the grammar checking use case. However, we did include it as a reference and found that it still correlated well with Bleu_4.

TER (Translation Edit Rate) is an intuitive NLP metric for evaluating translations that is able to solve two problems: knowledge intensiveness of more meaning-based approaches and labor-intensiveness of human judgments (21). TER measures how much editing one would need to apply to a candidate translation to make it completely match its ground truth. TER has also been shown to correlate more strongly to human judgements than Bleu ((21). In the case of our
grammar checker, the TER metric seems to correlate strongly with the Bleu metric.

Although several of the aforementioned metrics were not directly relevant to the task of grammar checking, none of them showed contradictory behavior. Table 11.2 shows how strongly correlated all of them are, unanimously agreeing if a certain model performs better than another model on the validation set.

Fig. 11.2 shows the full training progress in terms of validation performance using all the metrics for the LSTM 256 model, which was our best performing model for grammar checking. Each epoch took about 8 to 12 hours including both training and validation. Consequently, we had to stop after epoch 6, due to limited time and computing resources. From Fig. 11.2, it seems that epoch 3 was the best, with the best scores across all listed metrics. Obviously, this might not be the actual best model in the long run, as if we had more time and computing resources, we would have trained the model longer and used early stopping to ensure sufficient training. However, given our limited time and computing resources, we selected the LSTM 256 model yielded from epoch 3 for our final system.
Chapter 12

Obstacles Encountered & Lessons Learned

12.1 Challenges Unique to Conversational Chatbot

A significant challenge was producing unique and specific responses. However this is a common problem seen in modern research literature in this field. To put this in perspective, Google Meena was trained on 341 GB of text compared to our dataset of only 14 MB. In addition, it was trained for 30 days using a set of TPUs, or GPUs optimized for machine learning applications. Despite all this, they still face similar issues. Lastly for the chatbot, as mentioned, the perplexity metric was often inconsistent, which made evaluating our model tedious and time consuming.

12.2 Challenges Unique to Grammar Checker

The model was highly dependent on the kinds of noise generated in the training data. This means that the model may not be able to recognize errors that were not present in the training dataset. Obviously, we were not able to generate all possible types of noise in the training dataset, due to limited time and computing resources. Creating a grammar checker that is able to robustly recognize and correct grammatical errors like an educated native speaker remains a challenging problem in neural machine translation.

12.3 Challenges for Both

Even using top-of-the-line GPUs and a significantly smaller dataset compared to modern research in the field, each model took 2 to 3 days to train. This made it difficult and time consuming to find the ideal parameters and configurations for our models.

Regarding deployment, it required a series of correspondence with the creator of NMT Keras, Álvaro Peris, to deploy our GPU-trained models to our AWS Linux backend, as the library was primarily intended for research purposes only. We worked continuously with Alvaro throughout our project to help refine NMT-Keras library.
Chapter 13

Testing Procedure

13.1 Front-End

We tested the React Native application manually with an Expo development server and the Expo app to allow us to run the app natively on our personal mobile devices. This allowed us to continuously run and test our application and make small tweaks to the look and feel of the front-end. This was followed by manual app use and regression testing to ensure that the entire application and user interface worked together as a pipeline and provided a satisfying user experience.

13.2 Back-End

The Flask back-end was tested with direct API calls via Postman. This allowed us to test the API endpoints, ensuring predictable behavior and a robust implementation to handle various types of inputs. After the back-end was developed, we ran manual integration tests, tying the back-end and front-end together by running both servers simultaneously.

13.3 Machine Learning

Every time we trained a machine learning model, we always tested it to verify that there was no overfitting. When the model overfitted, it meant that our model conformed to the training data too closely and was unable to generalize for the testing data. This meant that our machine learning models were unable to perform predictions and we had to manually change parameters for our model. In our case it meant that the app was unable to accurately provide grammatical corrections or it may respond unnaturally during conversations.

To test our models, we fed them new data that they had not seen before in order to evaluate their predictive performance. It was understandable that the testing performance was not as accurate as the training data. However, we tested different parameters until the testing behavior was not significantly inferior to the training behavior.
Chapter 14

Implications

14.1 Privacy

We are transparent with the user regarding our use policy. We do not attempt to confuse them with long-winded documents. Instead, we use concise and understandable language to describe the ground rules set forth for this app.

Regarding usage, as the conversations that take place between the user and the app can be of any topic, they are considered private. This means we store these conversations securely in a SQLite database. Later on, we would likely scale up to a more robust database such as PostgreSQL to maintain security, as the quantity of data increases. Also, if a user agrees to share the conversations with the developers, then the conversations will be stored for analysis and app improvement.

14.2 Societal

Learning English has become more important than ever in today’s interconnected world. Although there are many apps available that teach English skills, many of them address only certain aspects of the learning process.

With this project, we anticipate a large number of non-native English speakers signing up as users. This app will save them not only money but also time as they work on their English skills. Learning English will be less and less daunting with the advent of PLATICA.

14.3 Safety

One of our use cases is using the application during a user’s commute. Hence, we implemented the hands-free functionality to ensure the user can safely drive while using the application. We developed a UI that is clean and easy to use to ensure the user is not distracted while driving. In the future, we may include more safety features such as restricting actions within the application when the user is driving, which could be detected using their device’s motion sensor.
Chapter 15

Development Timeline

15.1 Timeline

One of the constraints for our projects was the limited timeline we had to come up with a fully functional system. To prepare for this, we created the following development timeline to help us to stay on track.

<table>
<thead>
<tr>
<th>Timeline (week of)</th>
<th>DECEMBER</th>
<th>JANUARY</th>
<th>FEBRUARY</th>
<th>MARCH</th>
<th>APRIL</th>
<th>MAY</th>
<th>JUNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>React front-end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grammar checking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suggestions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contextual Responses</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliverables</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 15.1: Development Timeline
Chapter 16

Future Work

16.1 Public Release & Expansion to More Languages

Given that we already achieved compatibility with iOS and Android, we plan to publish the application on both platforms’ app store. In order to prepare the application for distribution, we need to make the application itself production ready. Our back-end must be scalable and secure. To prepare for potentially thousands of users concurrently interacting with our application during deployment, we must ensure our production server incorporates proper load balancing techniques to handle these requests. Additionally, we must research Keras model deployment methods to enable concurrent predictions. We would likely have to scale up our deployment resources to services such as AWS EC2 Auto Scaling. Furthermore, we must ensure our front-end is secure and fast. Users should only have access to authorized pages. Progress data and previous conversations need to be loaded quickly to ensure a smooth user experience. Once we have a solid number of users on the app stores, we can apply the same methodology on other languages to serve even more learners.

16.2 Multiplayer Mode

To make the app more interesting, we can also create a multiplayer mode where users can compete to see who can maintain the conversation with the highest grammatical rate. This means users can engage with their friends and family to learn English together. Also, in a classroom setting, an instructor can also have competitions using this mode to encourage learning in a fun and collaborative manner. Daily, weekly and monthly leaderboards will inspire friendly competition and effective learning for all.

16.3 Conversational Chatbot Refinement

For the chatbot, there are a number of ways to improve. For one, training on a larger dataset is crucial and would create a more robust model. Reddit offers copious amounts of conversational data, but requires a significant amount of preprocessing to prepare the data. Additionally, as done by Facebook Blender, using an ensemble of multiple
models could be beneficial: one for expressing a persona, one for expressing empathy, and potentially one to express knowledge. Lastly, we would ideally like to make the chatbot speak at a level that makes sense for the user. In the future we may incorporate age, gender, and English proficiency as parameters to the model.

16.4 Grammar Checker Refinement

It has been shown for various machine learning applications that combining multiple systems can substantially improve performance (10). By training multiple grammar checking models using different configurations and ensemble them during testing, we should be able to see noticeable improvements in terms of predictive performance. As each model has its own strengths and weaknesses, when working together, the models are able to collectively address each one’s shortcomings.

It follows that training on a larger dataset also helps the performance. At the moment, the training dataset is primarily dominated by novels instead of standard conversations. As PLATICA is more geared towards conversations, we also plan to incorporate more conversational data in our training set. A healthy mixture of short sentences and long sentences in the training set should yield even more improvements.

Last but not least, we could also make use of other advanced neural machine translation architectures. For example, BERT (Bidirectional Encoder Representations from Transformers), one of the latest advances in neural machine translation, is empirically powerful, yet requires a large amount of computing resources to use (9). As such, this is left for future work, when we have more time and computing resources.
Bibliography

[4] Tab-delimited bilingual sentence pairs these are selected sentence pairs from the tatoeba project. http://www.manythings.org/anki/.


