# Chef BiGRU: A Sequence-to-sequence Bidirectional GRU model with an Attention Mechanism for generating novel food recipes

Bungag, Viver Val P.
Panes, Julienne Andrea S.
Tan, Jay Innhahn Y.

ATENEO DE DAVAO UNIVERSITY
SCHOOL OF ARTS AND SCIENCES
DAVAO CITY

December 2023

# Abstract

This study delves into the challenges and advancements in automated recipe generation, a specific area of text generation within the field of Natural Language Processing (NLP). Recognizing the complexity involved in creating coherent and accurate recipes, which demands understanding the intricate relationships between ingredients and cooking steps, the research focuses on improving the quality and coherence of generated food recipes. Prior work in this domain, such as a study by Goel et al., utilized the Long Short-Term Memory (LSTM) architecture but encountered several limitations inherent to this model. To address these issues, this study introduces a novel approach: a sequence-to-sequence Bi-directional Gated Recurrent Units (BiGRU) model equipped with an attention mechanism. This proposed model is designed to enhance the quality and coherence of the food recipes generated by the system. The primary objective of the research is to assess the performance of the BiGRU model and compare the effectiveness of the newly proposed BiGRU model against the traditional LSTM model in the context of recipe generation. Through this comparison, the study aims to contribute significantly to the development of more advanced text generation models and explore their potential applications across various domains.

***Keywords*** – recipe generation, text generation, NLP, LSTM, BiGRU, compare, enhancement

# Table of Contents

**Abstract**.....1
**Table of Contents**..... 2
**Introduction**.....4
1.1 Background of the study.....4
1.2 Problem Statement.....4
1.3 Objectives of the Study.....5
1.4 Significance of the Study.....6
1.5 Scope and Limitations.....6
**Review of Related Literature**.....8
2.1 Natural Language Processing (NLP).....8
2.1.1 Recurrent Neural Networks (RNN) in NLP.....8
2.1.2 Long Short-Term Memory (LSTM) in NLP.....9
2.1.3 Gated Recurrent Units (GRU) in NLP.....10
2.1.3.1 Bi-directional GRU (BiGRU) in NLP.....12
2.1.4 Sequence-to-Sequence Models in NLP.....13
2.2 Tokenization in NLP.....14
2.2.1 Byte-Pair Encoding (BPE) Tokenization in NLP.....15
2.3 Word Embedding Layers.....15
2.4 Optimization Techniques.....16
2.4.1 Attention Mechanisms in NLP.....16
2.4.2 Regularization Techniques in NLP.....17
2.4.2.1 Layer Normalization.....18
2.5 Optimization Algorithms.....18
2.6 Recipe Generation.....19
**Technical Background**.....22
3.1 Theoretical Framework.....22
3.1.1 Natural Language Processing (NLP).....22
3.1.2 Tokenization.....22
3.1.2.1 Byte-level BPE Tokenizer.....23
3.1.3 Neural Networks.....24
3.1.3.1 Recurrent Neural Network (RNN).....24
3.1.3.1.2 Gated Recurrent Unit (GRU).....25
3.1.3.1.3 Bidirectional RNN.....26
3.1.3.1.4 Bidirectional Gated Recurrent Units (BiGRU).....27
3.1.3.2 Sequence-to-Sequence (Seq2Seq) Model.....27

3.1.3.3 Attention Mechanism....28
3.1.3.4 Embeddings....30
3.1.3.5 Layer Normalization....31
3.2 Conceptual Framework....32
3.3 Methodology....34
3.3.1 Gathering and Preprocessing of Data....34
3.3.1.1 Filtering the dataset through the recipe's rating....34
3.3.1.2 Filtering the dataset through the recipe's length, number of ingredients, and validity of NERs....37
3.3.1.3 Dataset Splitting....37
3.3.1.4 Adding Control Tokens....38
3.3.2 Training the Byte-level BPE Tokenizer....39
3.3.3 Tokenizing of data....40
3.3.4 Training the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism Model....40
3.3.5 Training the LSTM model....43
3.3.6 Evaluating the Model....43
3.3.6.1 BLEU score....43
3.3.6.2 Jaccard Similarity....44
3.3.6.3 Human Evaluation....45
3.3.7 Web Application....46
3.3.7 Timetable....47
**Results and Discussion....49**
4.1 Training Accuracy and Loss Comparison of LSTM and GRU....49
4.2 Training Time and Computing Time of LSTM and GRU....49
4.3 BLEU accuracy metric results....49
4.4 Jaccard Similarity Scores....49
4.5 Human Evaluation Results....49
**Conclusion and Recommendation....52**
**Glossary....53**
**Bilbiography....56**

# Chapter 1

# Introduction

## 1.1 Background of the study

Recipes are a specific genre of instructional language that guides food preparation. Creating a recipe can be tedious as it takes time, brain power, and a lot of creativity. Therefore, it would be better if there was a method to generate unique recipe ideas. The field of Natural Language Processing (NLP) has seen significant advances in recent years, particularly in the area of text generation. One area of text generation that has received attention is the generation of food recipes. Automated recipe generation has many potential applications, including meal planning and menu creation. However, generating coherent and accurate recipes is a challenging task, as it requires understanding the relationships between ingredients and the steps involved in cooking. A study by Goel et al. involves generating food recipes using the LSTM architecture, but this approach poses several problems in the model itself [1]. To address this, the study proposed a sequence-to-sequence Bidirectional GRU with an attention mechanism model that improves the quality and coherence of the generated food recipe. This study aims to use, assess, and compare the proposed and LSTM models. Overall, the study has the potential to contribute to developing more advanced text generation models and their applications in various domains.

## 1.2 Problem Statement

The current state-of-the-art methods for generating food recipes from a list of input ingredients using deep learning models such as LSTM have been proven possible. Yet, Goel et al. have made a point that using LSTM suffers from the problem of inconsistency and low coherence between ingredients and instructions in the generated food recipes [1]. This can be attributed to the limited ability of LSTM models to capture long-term dependencies and relationships between words in the input, which can lead to inconsistencies and incoherent recipes. The study's proposed model, Sequence-to-Sequence BiGRU with attention mechanism, can potentially solve these problems. However, how these models compare in terms of their

performances remains unclear, particularly concerning the accuracy, coherence, and diversity of the generated recipes.

Hence, this study aims to address these questions: "What is the efficacy of the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model on generating novel food recipes" and "How is the performance of the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model compared to the LSTM model in generating food recipes?

Specifically, the study aims to address the following sub-questions:

1. How accurate is the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model compared to the LSTM model in generating food recipes when using the BLEU score as the accuracy metric?

2. How do the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism and LSTM models compare their coherence between ingredients and instructions on generating food recipes measured by Jaccard Similarity?

3. How is the performance of the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model in generating novel food recipes according to human evaluators across the following criteria: clarity, creativity/novelity, suitability, doability/edibility, likelihood to try, and overall recipe quality?

## 1.3 Objectives of the Study

This research aims to determine if the proposed model outperforms the LSTM model in generating food recipes.

The specific objectives of the study are as follows:

(a) To compare the accuracy of the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model and LSTM model on generating food recipes using the BLEU accuracy metrics.

(b) Assess the coherence between ingredients and instructions of the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism and LSTM model on generating food recipes using the Jaccard Similarity.

(c) Evaluate the performance of the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model in generating novel food recipes by human evaluation across the following criteria: clarity, creativity/novelty, suitability, doability/edibility, likelihood to try, and overall recipe quality.

## 1.4 Significance of the Study

The study has the potential to significantly improve the quality and coherence of the generated food recipes, which could have practical applications in various domains such as meal planning, diet planning, and nutrition counseling. It can also have practical implications for the culinary industry, as generating novel recipes can create new dishes. Moreover, the study can contribute to the Natural Language Generation (NLG) field, which involves generating human-like text using machines. NLG can have applications beyond creating recipes, such as chatbots, virtual assistants, and automated news articles. Other researchers can use the comparison of BiGRU and LSTM in their study. Overall, this could result in a more capable model which is practical and scalable for real-world applications.

## 1.5 Scope and Limitations

The study will compare the BiGRU model to the LSTM model for generating food recipes. The dataset used for training and testing the models will be a food recipe dataset containing the most basic requirements, such as title, ingredients, and instructions. The research will also include the development of a web application for users to input their desired ingredients and receive the generated recipes. The list of ingredients on the website is only limited to the NERs of the trained recipes.

There are several limitations to this research. First, the generated recipes will be limited by the quality and size of the study's dataset. There is a possibility that the study's dataset will not capture all the possible variations of recipes that exist. Second, the quality of the generated recipes may be affected by the ingredients inputted by the user. If the input data is incomplete or

senseless, the generated recipes may not meet the user's expectations. Third, the research will focus on generating only English recipes. Fourth, the research will not incorporate the nutritional values of the generated recipes. Fifth, the number of users who rated the recipe and the users who rated the recipe was not taken into consideration when data scraping. Sixth, recipes that do not have ratings and links that cannot be webscraped were not included in the list of web-scraped recipes. Seventh, users of the web application will not be able to personalize their inputs, including providing measurements for each ingredient and specific cooking methods. Sixth, the researchers are bound by the hardware available in our hands. Lastly, the minimum number of ingredients required to be inputted is 3, and the maximum is 8.

# Chapter 2

# Review of Related Literature

This chapter explores Natural Language Processing (NLP) and its techniques. The use of language models like LSTM, GRU, BiGRU, RNN, and Sequence-to-Sequence Architecture, highlighting their significance, strengths, and areas for improvement in NLP tasks, was further discussed. Following this, the importance of Tokenization in the context of NLP tasks was emphasized. It was then discussed how word embedding layers are used to enhance the model's language understanding. Additionally, techniques were explored to refine the model's performance, including Attention Mechanisms and methods for regularization like Layer Normalization and Optimization Algorithms. Lastly, a comprehensive overview of various recipe generation studies, which form this study's foundational basis, was provided.

## 2.1 Natural Language Processing (NLP)

Naturalized Language Processing is the computerized analysis of text using theory and technology [2]. There are many forms of Natural Language Processing, such as Sentiment Analysis [3], Text Classification [4], and Text Generation [5].

Understanding the fundamentals of NLP is essential in this study as it is the foundation of the techniques and methods used to generate food recipes.

### 2.1.1 Recurrent Neural Networks (RNN) in NLP

Recurrent Neural Networks (RNNs) are neural network models that model sequence data. It transmits words to the network in an orderly manner and stores the previous word information [4]. It also captures the dynamics of sequences via cycles in the network of nodes [6]. That’s why it’s excellent for pattern classification tasks with sequence inputs and outputs. These include sequence data such as recipes. That’s why RNN was used in recipe generation studies such as Ingredient prediction [7], Novel Recipe Generation [1], and Personalized Recipe Generation [8].

However, RNNs have become difficult to train, especially on problems with long-term dependencies and vanishing/exploding gradients [9]. A study by Ni and Cao comparing Recurrent Neural Networks (RNNs) with their improved variants in sentiment analysis revealed a notable challenge with standard RNNs. As the number of iterations increased, the F1 score displayed significant fluctuations in a line graph. This variability underscored the difficulty of standard RNNs in effectively handling long-term dependencies [10].

RNN is a viable language model that would be great for NLP tasks such as generating recipes with its benefits in the context of text generation. However, it still has shortcomings regarding long-term dependencies and vanishing/exploding gradient problems.

### 2.1.2 Long Short-Term Memory (LSTM) in NLP

LSTM is an improvised version of the standardized RNN; it adds an input gate and a forget gate to solve the problem of gradient disappearance and gradient explosion so that long-term information can be captured and perform better in long-sequence text. [11]. This model has been applied effectively in NLP tasks like text generation [5], [12], text classification [13], and sentiment analysis [3].

Several studies that compare LSTM to other types of models and LSTM outperformed them in terms of accuracy: A Comparison of LSTM and BERT for Small Corpus [14], LSTM vs. GRU vs. Bidirectional RNN for script generation [5] LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example [11].

LSTMs could preserve history for long sentences and have been used extensively in NLP applications, from question-answering systems to machine translation. [12] Here are several studies that applied LSTM on NLP tasks that also lead to the conclusion that LSTM is excellent for long-term dependencies [5] [12] [11].

However, there are still things that could be improved in this model. Qian stated that the LSTM algorithm's main drawback is that it requires a significant sample of data and takes a long time to train the model [15]. Chen concluded in their study that transformer-based encoders are substantially faster than LSTM encoders. The LSTM encoder has a hidden state mechanism. Therefore, it must wait for the previous hidden state to finish before moving on to the next [16].

This shows that although LSTM outperforms other models in terms of accuracy, it still has gaps within the model; These studies compared LSTM with other models and showcased that using this particular model increased the training time and computational cost [11] [5].

A significant flaw found by Goel et al. in their study on novel recipe generation is that, generally, LSTM proved to be prone to overfitting. The LSTM model also did not do well in the context of their study as their model could only learn the structure of the recipe, not the relationship between ingredients and instructions [1].

This section proves the capabilities of LSTM when it comes to NLP tasks. It shows the techniques the other researchers used in their LSTM model to achieve their desired results. Moreover, it presents the shortcomings of LSTM, which validates the reason for enhancement.

### 2.1.3 Gated Recurrent Units (GRU) in NLP

A gated Recurrent Unit (GRU) is an improvised version of the standard RNN. Along with LSTM, they solve the vanishing/exploding gradients, long-term dependencies, and challenges of the standard RNN by gating mechanisms and cells. The main difference between GRU and LSTM is that GRU has a simpler architecture than LSTM since it only has two gates, the update and reset gates, instead of 3 from the LSTM, which allows for fewer parameters which can lead to lesser computation cost [17] [18] [19] [11].

Yang asserts in his study that GRU's ability to forget and select memory with a single gating mechanism and its fewer parameters than LSTM leads to significantly faster training speeds. His research demonstrated that the average running time for GRU was 129.17 seconds, whereas for LSTM, it was 167.00 seconds. This indicates that the GRU model's training speed is approximately 29.29% faster than the LSTM. These NLP studies also demonstrated that GRU is considerably more efficient in terms of training speed and computational cost: A Sequence-Based Machine Comprehension Modeling Using LSTM and GRU [20] LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example [11] Sentiment Analysis based on GloVe and LSTM-GRU. [10] Attention-based BiGRU-CNN for Chinese question classification [4] LSTM vs. GRU vs. Bidirectional RNN for script generation. [5] A Text Normalization Method for Speech Synthesis Based on Local Attention Mechanism [21]

Zulqarnain showed in his study that he tried comparing GRU, LSTM, RNN, and MV-RNN to determine the best model for text classification. The results showed that GRU outperformed all the other models regarding accuracy and error rate percentage [22]. With this, various NLP studies have also incorporated GRU to achieve more accurate results in each of their studies. Aspect-Level Sentiment Analysis Based on Bidirectional-GRU in SIoT [23] Attention-based BiGRU-CNN for Chinese question classification [4] Text mining-based construction site accident classification using hybrid supervised machine learning [24] Efficient processing of GRU based on word embedding for text classification [22] Translation for English to Hindi using gru. [25] A Text Normalization Method for Speech Synthesis Based on Local Attention Mechanism [21].

Yang's study also compared GRU on different parameters such as datasets and text lengths. The most favorable outcome was found in the shorter dataset, with the shorter text variant achieving an accuracy score of 0.7761 in contrast, the other variants achieved this score (0.6897) in long text and larger datasets (0.7699). Long text and small dataset and (0.673) short text and large dataset. GRU works best on shorter text and a shorter dataset [11].

Despite Yang's comparative study of the two variants of recurrent neural networks (RNNs), LSTM and GRU, highlighting that LSTM can be more accurate in certain aspects such as accuracy and metrics, it demonstrated that GRU excels in efficiency when considering the dual dimensions of performance and computing power cost. The performance-cost ratio of GRU surpasses that of LSTM, being 23.45%, 27.69%, and 26.95% higher in terms of accuracy ratio, recall ratio, and F1 ratio, respectively [11].

GRU is also used for text generation. These are studies that implemented GRU in their studies and improved their model: Research on text generation based on LSTM [26] R-AE: Differentially private text generation for authorship anonymization [27] A deep neural framework for image caption generation using gru-based attention mechanism [28] Content preserving text generation with attribute controls. [29] Generating personalized recipes from historical user preferences [8].

Although GRU has proved its potential in NLP studies, it also showcased its gaps or weaknesses. Zhang stated in his study that GRU is better than LSTM in many applications. Still,

there is a significant gap between long text generation's and LSTM's ability to capture complex semantics [26].

This section highlights GRU's capabilities in NLP tasks, emphasizing its efficiency in training time, computational costs, and accuracy compared to other models. Additionally, it explores GRU's applicability in text generation. However, it is essential to acknowledge one of GRU's identified weaknesses. While it excels in short sentences and smaller datasets, it may not perform optimally in tasks requiring learning complex long-distance relations in text generation.

#### 2.1.3.1 Bi-directional GRU (BiGRU) in NLP

Bi-directional Models are used to combine a word and its context, One word in a sentence can be learned from forward and backward twice in a bidirectional structure, which can get more word representation features and avoid some gradient vanishing problems in long sentences. [4] Bi-directional GRU (BiGRU) is the bidirectional model of the GRU architecture

A study by Liu et al. showed that Bi-GRU achieves better performance in both experiments when compared to LSTM and Attention-based LSTM. This proves that a bidirectional recurrent neural network also has advantages in foreign language classification. [4] These are also studies that incorporated BiGRU and led their model to outperform other baseline models: A Hierarchical Bidirectional GRU Model With Attention for EEG-Based Emotion Classification [30] Comment toxicity detection via a multichannel convolutional bidirectional gated recurrent unit [31].

Also stated in their study that while Bi-GRU and Bi-LSTM are much better than their unidirectional counterparts, using this type of architecture would lead to more training time and computational costs. [11] The computational cost of the bidirectional decoding algorithm presented in the previous section is, of course, larger than that because it enumerates all possible directions of the edges on top of the enumeration of possible tag sequences [32]

This section highlights that the BiGRU represents a significant improvement over the GRU. However, it's essential to be mindful of the potential increase in computational costs. Reading context from left to right can result in longer training times. This observation is crucial for the researchers to consider.

### 2.1.4 Sequence-to-Sequence Models in NLP

Sequence-to-sequence learning is a neural network-based approach aiming to map a data element sequence to another sequence. It employs an encoder to process the input sequence and create a fixed-size representation, which is then used as a context vector to initialize a decoder. The decoder generates the output sequence based on this context vector [33].

This approach can also be used on recurrent neural networks (RNN) and shows excellent capability for processing variable lengths of sequential data. In recent years, these structures have led to the noteworthy development of language models and have played an important role in the development of various tasks of natural language processing (NLP), such as text summarization [34], [35]. Text generation [35], [36], [37], relation extraction [38] text processing [16]

Some studies incorporated the GRU model along with the sequence-to-sequence framework that led to a great improvement in their nlp study: Transformer and seq2seq model for paraphrase generation [39] Applying a Hybrid Sequential Model to Chinese Sentence Correction [16] Aspect-Level Sentiment Analysis [23] Showing their compatibility with one another.

Liu et al. He concluded in his study that the performances of the seq2seq models are generally better than other baselines because they can identify overlapping relations more flexibly by decoding different triplets one by one. However, here is the main drawback compared to other baselines: the seq2seq models fail to precisely recognize the entity boundary. Thus, the longer the entity's length is, the worse these models perform. This showcased that although sequence-to-sequence models enhance and improve the studies on their own, it also showcased its main drawback, which is its inconsistency on longer length entities or long-term text [38].

These studies have demonstrated that incorporating the seq2seq architecture significantly enhances model performance compared to models without seq2seq architecture [36].

In this section, the researchers showcased the sequence-to-sequence architecture that changes the architecture of the said model and can help in the effectiveness of the NLP tasks. It is also compatible with the BiGRU model. This can help the researchers in their study since this architecture is also effective on NLP tasks.

## 2.2 Tokenization in NLP

In natural language processing (NLP), Tokenization or segmentation is a wide concept that covers simple processes, such as separating punctuation from words, or more sophisticated processes, such as applying morphological knowledge [40]. It's the first step of many natural language processing (NLP) tasks and plays an essential role in neural NLP models.

Domingo et al. stated in their study that although tokenization positively impacts the performance of a particular model, its main improvement differs from one another [40]. The main benefit of using tokenization differs from a different context.

In Goel et al.'s study on recipe generation, he evaluated two LSTM models to measure the overall quality of the tokenized LSTM. One consists of a character-level tokenized LSTM model, and the latter is a word-level tokenized LSTM model. He showed in his study that the Word Level Tokenized LSTM performed better when compared to the character-level tokenized LSTM. [1] This states that a word-level tokenized model is much better than a character-level tokenizer in recipe generation.

Here are text generation studies that used tokenization in their respective studies showcasing its importance in the specific field: Question Answering based University Chatbot using Sequence to Sequence Model. [36] Rare tokens degenerate all tokens: Improving neural text generation via adaptive gradient gating for rare token embeddings [41] Generating personalized recipes from historical user preferences [8] Dataset for automatic summarization of Russian news [42].

In this section, the researchers have discussed tokenization, highlighting its benefits and use in natural language processing. It also explores how tokenization is applied specifically in the context of recipe generation.

### 2.2.1 Byte-Pair Encoding (BPE) Tokenization in NLP

Byte Pair Encoding (BPE) is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. This algorithm can be adapted for word segmentation. Instead of merging frequent pairs of bytes, it can merge

characters or character sequences. [43] The main advantage of BPE is that it's better than word tokenization, as it enables rich morphology and decreases the number of unknown tokens [42].

These are all the nlp-related studies that showcased that using BPE made them outperform their respective baseline models: Dataset for automatic summarization of Russian news [42] Multi-task multi-resolution char-to-BPE cross-attention decoder for end-to-end speech recognition [44] Generating personalized recipes from historical user preferences [8].

These recipe generation studies used the byte-pair encoding (BPE) tokenizing method in their study RecipeGPT: Generative Pre-training Based Cooking Recipe Generation and Evaluation System. [45] Generating personalized recipes from historical user preferences [8].

This section discussed the potential of using a BPE tokenizer on natural language processing tasks. It also discusses its application in a recipe generation study.

## 2.3 Word Embedding Layers

A word embedding layer is a component within a neural network that maps discrete word indices to continuous vector representations, also known as word embeddings. Word embeddings is a powerful tool widely used in modern natural language processing tasks (NLP). They are fundamentally a form of word representation that links the human understanding of knowledge meaningfully to the understanding of a machine. It's a way for the model to understand the given text data input because specific models can only understand vectors [46].

It's one of the most common tools to use in NLP studies. These studies used embedding layers on their study to help improve their model: Ratatouille: A tool for Novel Recipe Generation [1] Generating Personalized Recipes [8] Question Answering based University Chatbot [36].

Altogether, Word embedding layers show great potential in producing coherent texts. Researchers can use this technique to improve the model's accuracy and the coherence of the text generated further.

# 2.4 Optimization Techniques

Optimization techniques are used to improve the model, further optimizing it would help avoid inaccuracies and loss. Several techniques can be used for these respective fields

## 2.4.1 Attention Mechanisms in NLP

Attention is a way of obtaining a weighted sum of the vector representations of a layer in a neural network model [47]. Attention mechanism has become a famous ingredient in NLP studies such as text classification [4], [48], [49], text generation [8], [37], and relation extraction [50].

Majumder et al [8] stated in their study that the accuracy of the attention model is better than that of the model without an attention mechanism. Still, the training time of the attention model is nearly six times longer than that of the without-attention model [21].

Multi Head Attention (MHAT) is the optimization of the traditional attention mechanism, and it is also superior in performance [51]. It allows the model to focus on information from different representation subspaces from different positions by stacking multiple self-attention layers [52].

The basic component of the MHAT mechanism is attention, but the difference is that the MHAT model performs multiple distributed calculations, which can process complex information [51].

In his study on Speaker Recognition [53], he compared the Multi-Head Attention Mechanism to other baseline pooling techniques such as i-vector with PLDA, statistical-based pooling layers, and vanilla self-attentive pooling. He found that MHAT outperformed them on specific metrics such as Equal Error Rate (EER) and the minimum Decision Cost Function (DCF). Self MultiHead attention has shown the best result of all the evaluated approaches. It outperforms both i-vector+PLDA and statistical-based pooling layers with a 58% EER and 18% EER improvement, respectively. This means that the model has a lesser error rate in identifying the speaker talking. These studies also showcased that multi-head attention improved the performance of their model in NLP studies [54] [55].

In this section, attention mechanisms and their potential for NLP tasks were discussed, proving helpful when used appropriately. This section also discusses Multi-Head Attention its benefits, and applications on Natural Language Processing studies.

### 2.4.2 Regularization Techniques in NLP

One of the most common problems researchers tend to avoid is overfitting. That is when the model performed exceptionally well on training data but could not predict test data to help solve this problem. Regularization is introduced. It's a set of techniques that can help avoid overfitting in neural networks, thereby improving the accuracy of deep learning models [56].

Salman & Liu showed in their study that using specific regularization techniques led to lesser loss and higher accuracy for their model. These are also NLP studies that implemented regularization technique to prevent overfitting from their data and to help lead to accurate results LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example [11] A Deep Learning Approach for Consumer Sentiment Analysis Using Qualitative User-Generated Contents [1], [3].

In this section, regularization techniques and their beneficial impact on NLP studies are discussed to prevent the overall model from encountering overfitting problems during training. This is advantageous as it helps the model become more accurate and better.

#### 2.4.2.1 Layer Normalization

Layer Normalization (LayerNorm) is designed to normalize intermediate layer distributions, leading to smoother gradients, faster training, and improved generalization accuracy. [48] Unlike standard Batch Normalization, Layer Normalization is adaptive to RNNs and self-attention-based models, [57] making it versatile across various architectures. Particularly effective in stabilizing hidden state dynamics in recurrent networks, Layer Normalization has proven empirically to reduce training time compared to previously published techniques significantly [58].

As highlighted in Liu's study, Layer Normalization outperforms Batch Normalization in addressing gradient malformation. The study by Shen further demonstrated the effectiveness of

Layer Normalization, showcasing its competitive performance against Batch Normalization, particularly in natural language processing tasks. This reinforces the broader applicability and advantages of LayerNorm, not only in stabilizing hidden state dynamics but also in yielding superior results across various machine learning tasks such as NLP tasks. [59] These studies are the studies that use Layer Normalization: Integrating multimodal information in large pre-trained transformers [60] Rethinking batch normalization in transformers [59].

This section delves into the significance of layer normalization in RNN studies as an effective technique for addressing challenges such as vanishing gradients in RNN models. Furthermore, the researchers explore its application in NLP studies, highlighting its relevance and impact across different contexts.

## 2.5 Optimization Algorithms

Optimization algorithms form the basis for machine learning through experience. They compute gradients and minimize the loss function [61]. It helps make the model better and optimizes it [10].

Optimization Algorithms have also been used in several studies, such as Text Generation [7], Recipe Generation [8], Sentiment Analysis [11], and Neural Machine Translation [40].

In his study on Optimization for deep learning algorithms, when comparing all the optimization algorithms, he found that Adam is one of the best optimization algorithms, outperforming other optimization algorithms like SGD, ADAGRAD, and RMSPROP [61].

Adam is a simple and computationally efficient algorithm for gradient-based optimization of stochastic objective functions; it only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients [62].

In the study on recipe generation by Lee et al., they found that choosing the correct learning rate and batch size was crucial. They decided to use a learning rate of 0.0001 based on their study into what worked best for their specific research. This choice was made by considering how it affected their model's performance in the context of their study [45]. Several

studies used a similar learning rate in text generation studies; in another recipe generation study by Majumder et al., they also used an Adam optimizer with a learning rate of 0.001. The use of the learning rate also depends on what works well with the model [8].

A study on optimization techniques for deep learning identifies Adam as a standout algorithm, surpassing alternatives like SGD, ADAGRAD, and RMSPROP. Adam's simplicity and computational efficiency make it a potent tool for optimizing stochastic objective functions based on gradients. This understanding lays the foundation for exploring the significance of optimization algorithms in language model investigations. It provides a solid basis for a forthcoming study on recipe generation that aims to utilize these algorithms for enhanced model performance [61].

This section underscores the significance of optimization algorithms in model development, shedding light on parameters such as learning rates that prove beneficial to researchers in their respective studies. The recognition of Adam as a standout algorithm lays the groundwork for further exploration of optimization algorithms in language model investigations, particularly in the upcoming study on recipe generation aiming to leverage these algorithms for enhanced model performance.

## 2.6 Recipe Generation

Several studies have already been conducted about generating food recipes [63] [64] [65] [66] [67] [68] based on food images [69] and generating recipes based on ingredients [1], [45].

A study by Bień et al. Used GPT-2 and fine-tuned it using the RecipeNLG dataset. The dataset contains over 2 million cooking recipes with ingredients, instructions, and food entities. [70]. The food entity, a Named Entity Recognition (NER) model, is instrumental in generating the recipe and determining the similarity between original and machine-generated recipes. In evaluating their model, they first used cosine similarity to measure the similarity between a generated recipe and its gold standard counterpart. Next, they used the Language Check spell and grammar checker to calculate the number of linguistic mistakes. Lastly, they used a variety of translation metrics such as the BLEU, GLEU, and WER.

Parvez et al. used LSTM as their primary approach in recipe generation. Similar to the study of Bień et al., the concept of Named entities was also applied. They got their recipes from "Now You're Cooking!" which has 150,000 filtered and preprocessed recipes. The configuration of their model consists of 20 instances of minibatch while training and 70 back-propagation through time value. They used a 3-layered LSTM architecture where the hidden layers are 1150 dimensional and have their own optimization and regularization mechanism. In evaluating their model, they used the perplexity metric [7].

Majumder et al. Implemented recipe generation using a sequential BiGRU model with an attention layer, The encoder portion of the model has 3 embedding layers, which are then passed on to the two-layered BiDirectional GRU encoder; Next is an attention layer implementation towards the ingredients. And lastly, their decoder is a two-layered GRU layer. They trained this model on a fine-tuned dataset from food.com, which amounts to about 180K+ recipes. To assess their generated recipes, they used BLEU scores to evaluate the overall text and cosine similarity to evaluate the coherence between recipes. Lastly, they implemented human evaluation by letting users choose between the generated recipe and a human-written one without knowing which one is which [8].

Lee et al. The GPT-2 model was trained on the Recipe 1M dataset. Their study also extracted root nouns the same as the Food Entities and Named Entities mentioned in the previous literature. For their evaluation section, they divided it into two parts: Ingredient generation and Between-field ingredient coherence. The ingredient generation was measured using BLEU and ROUGE, while the Between-field ingredient coherence was measured through Jaccard Similarity [45].

Same as Parvez et al., Goel et al. Constructed a study that uses LSTM as their model. The study extracted the named entities like the literature mentioned earlier in this section. The dataset they used is the RecipeDB, which contains 118,171 recipes from cuisines across the globe, 268 cooking processes, and 20,262 diverse ingredients further linked to their flavor molecules. Their evaluation section only used one method: the BLEU score. They experimented with two types of LSTMs: Character-level and Word-level LSTMs. The character-level LSTM has a BLEU score of 0.347, and the word-level LSTM has 0.412. Both of these scores are low compared to GPT-2 Medium, which has a BLEU score of 0.806 [1], [7], [70]. In their study, Goel et al. identified a

major drawback shared by Parvez et al.: the tendency of LSTM models to produce recipes with inconsistencies between ingredients and instructions [1]; as highlighted by Goel et al., The LSTM model tended to produce recipes with inconsistencies between ingredients and instructions, resulting in a lack of coherence.

This section is the main objective of the study. It contains the various methods that the other researchers used in their study of generating recipes. This is a significant factor in determining the best methods to generate novel food recipes.

# Chapter 3

# Technical Background

## 3.1 Theoretical Framework

This section delves into the theories and concepts employed in this study. These include Natural Language Processing (NLP), Byte-Level BPE Tokenization, Long Short-Term Memory (LSTM), Bidirectional Gated Recurrent Unit (BiGRU), Seq2Seq Architecture, Multi-Attention Mechanisms, Embeddings, and Layer Normalization.

### 3.1.1 Natural Language Processing (NLP)

NLP (Natural Language Processing) is a computational method for examining the text, combining theoretical concepts and technological advancements. It encompasses a collection of computational methods for analyzing and representing natural language texts, using various levels of linguistic analysis to replicate human language processing for multiple tasks or purposes [71]. Comprehension of the basics of NLP is crucial in this field of research as it underpins the approaches and methodologies utilized in recipe generation. Various computational methods essential in text analysis can be employed by using NLP, which allows for replicating how humans process language for different purposes. Thus, a thorough grasp of NLP is fundamental in developing innovative techniques for diverse fields of study, including text generation.

### 3.1.2 Tokenization

Tokenization is a crucial component of NLP that plays a critical role in breaking down text into its fundamental units. These units, known as tokens, are the building blocks for various NLP applications and text analysis. It is an essential step in NLP that enables the computer to understand and process natural language text more effectively [72]. This process involves dividing strings into words and punctuation [73], the most common type of tokenization [74]. Additionally, tokenization facilitates the application of machine learning algorithms to text data

by converting it into a format that algorithms can readily comprehend [75]. Several tokenization techniques are employed in NLP, each tailored to specific tasks and linguistic characteristics. In the context of recipe generation, which often includes a wide range of ingredients, some of which may have different forms, spellings, or variations, selecting the appropriate tokenization method that deals with these variations is crucial. For example, "tomatoes" and "tomato" in traditional tokenizing will be treated as two tokens. Therefore, selecting the proper tokenization technique tailored to the specific linguistic characteristics of recipe text is essential.

#### 3.1.2.1 Byte-level BPE Tokenizer

The general BPE algorithm is crucial in managing rare words and subword units. It repeatedly looks for the most common adjacent pair of bytes (or characters, in the case of standard BPE, which is character-level) and replaces them with a single, unused byte. This process is repeated iteratively. The merge function, responsible for selecting and combining these common pairs, underpins the algorithm's efficiency. By encoding infrequent and unknown words as sequences of subword units, this approach builds a vocabulary of subwords that may include individual bytes (representing characters in English text), common sequences of bytes (representing common words or word pieces), and less common sequences that might only appear in specific contexts. The merge function's role is to decide which adjacent pair of bytes to merge at each iteration, choosing the most frequent pair based on the training data. This choice allows BPE to manage rare words efficiently, optimize neural network models, and enhance text compression, making it a valuable technique in natural language processing. Moreover, BPE's parameters allow for flexibility regarding vocabulary size, number of merges, minimum frequency, etc [43].

In byte-level BPE, instead of the traditional character-level processing, it treats texts as a sequence of bytes, effectively allowing it to handle any possible character encoding, including UTF-8. This type of BPE allows for all characters and symbols, including those from different scripts or languages, to be represented consistently [76].

In the context of the previous example, the BPE tokenizer excels in addressing such linguistic variations. "Tomatoes" and "tomato" might be broken down into subwords like "toma" and "toes" depending on the most frequent subword in the training data, or "tomato" might

remain as one token if it's a common subword in the training corpus. Moreover, this tokenizing method can leverage neural network models more effectively in the context of recipe generation, as it aids in accurately processing and generating language by capturing finer linguistic subtleties.

### 3.1.3 Neural Networks

Neural networks are modeled according to the human brain's ability to process information. These models process and transmit information by weighted connections between interconnected nodes or neurons. The weights are changed during training to optimize the model's performance on a specific job, such as natural language processing. Moreover, neural networks excel in processing and analyzing large and complex data sets and can, to some extent, learn from the inputted data, allowing them to adapt over time to improve accuracy and performance [77]. The significance of neural networks in text generation lies in the fact that neural networks can emulate the natural learning processing capabilities of the human brain.

While there are many types of neural networks, one notable subtype that plays a crucial role in various tasks, including text generation, is the Recurrent Neural Network (RNN). RNNs are designed to address the unique challenges associated with sequential data, making them particularly well-suited for language modeling and text generation tasks.

#### 3.1.3.1 Recurrent Neural Network (RNN)

RNNs are deep learning models that use recurrent connections, which may be compared to cycles in a network of nodes, to capture the dynamics of sequences [78].

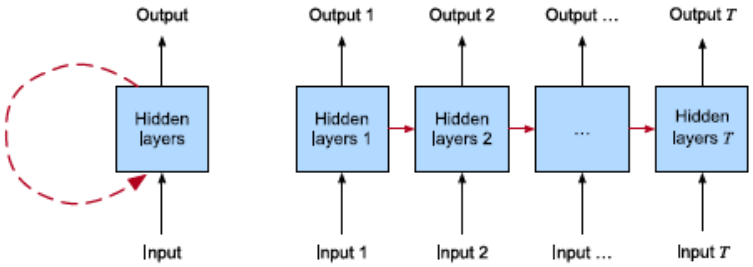



*Figure 3.1.3.1.1 General Concept of RNNs (Zhang et al., 2021)*

As seen in figure 3.1.3.1.1, the output of one node creates a cycle fed to another node's input. Having this kind of architecture made RNN perform better when posed with sequence-related tasks. Hence, RNN is a good model for generating texts. However, Sutskever et al. mentioned that although RNN may be a powerful tool because of their high-dimensional hidden state with non-linear dynamics that gives them the ability to recall and comprehend previous information and is quite expressive, a distinct kind of RNN that is more appropriate for text production had to be developed to compete in character-level language modeling [79]. Moreover, Pascanu et al. stated that an RNN model requires significant computational power and struggles with gradient vanishing/exploding problems and long-term dependencies [80].

### 3.1.3.1.2 Gated Recurrent Unit (GRU)

The GRU is a unidirectional variant of RNN that enhances the LSTM architecture by introducing a gating network that produces signals to manage the current input and the previous memory for updating the current activation and network state. It is a much more straightforward approach than LSTM, as it combines parameter updates for gates following a specific algorithm [81]. Moreover, it was introduced by Cho et al. as a neural network architecture designed for sequence-to-sequence tasks. The GRU addresses the vanishing gradient problem in traditional RNNs by incorporating update and reset gates [19].

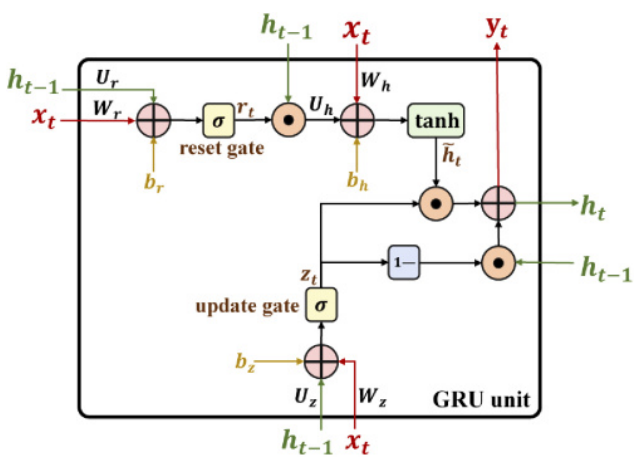



*Figure 3.1.3.1.2.1 GRU block Architecture (Liu et al., 2021)*

Figure 3.1.3.1.2.1 illustrates the different elements that make up the GRU architecture as well as how each element interacts with the other, as described by Cho et al. The update gate

controls the flow of information from the previous hidden state to the current hidden state, while the reset gate helps determine what to forget from the previous state. These gates, in combination with a candidate hidden state, enable the network to capture long-range dependencies and process sequential data more effectively. In the paper's context, GRUs are used in both the encoder, which encodes the input sequence, and the decoder, which generates the output sequence in tasks like machine translation. The GRU block contains two distinct gates: the update and reset gates [19]. With that said GRUs have a much simpler architecture than LSTM, which reduces the number of parameters and computations, making GRUs easier to train and potentially more computationally efficient. However, similar to the LSTM model, GRU's unidirectional characteristic allows it only to consider past contexts, which results in limited contextual understanding.

#### 3.1.3.1.3 Bidirectional RNN

BRNN is an extension of traditional RNN with the ability to process data in both forward and backward directions.

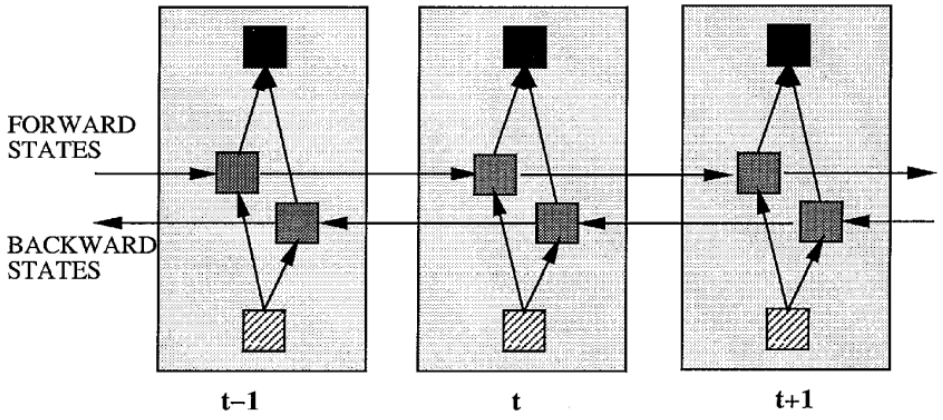



*Figure 3.1.3.1.3.1 General structure of the BRNN (Schuster & Paliwal, 2017)*

Figure 3.1.3.1.3.1 shows the general structure of a Bidirectional Recurrent Neural Network model. The BRNN structure serves as the foundation for bidirectional RNNs [82]. In the context of a bidirectional LSTM model, it may consist of multiple layers of RNN cells, where each cell has a forward and a backward RNN unit. The forward and backward RNN units receive inputs in opposite directions, allowing them to capture context from past and future time steps.

The forward and backward RNN unit outputs are concatenated and passed on to the next layer of BRNN cells. This allows the model to learn long-term dependencies in both directions of the input sequence. The final output of the BRNN architecture is the result of a fully connected layer, which maps the output of the last BRNN cell to the desired output.

Overall, BRNNs have been proven to be better in terms of accuracy and coherence in comparison to unidirectional RNN models due to their improved context understanding because of their mirrored RNN layers.

#### 3.1.3.1.4 Bidirectional Gated Recurrent Units (BiGRU)

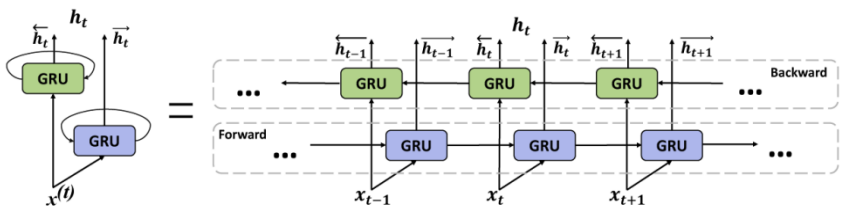



*Figure 3.1.3.1.4.1 BiGRU model structure (Liu et al., 2021)*

The BiGRU architecture is established by utilizing the states of two unidirectional GRUs that operate in opposite directions. One GRU layer progresses forward, commencing from the beginning of the data sequence, while the other GRU layer moves backward, starting from the end of the data sequence. This configuration enables the integration of information from past and future contexts to influence the current state, improving the model's contextual understanding.

### 3.1.3.2 Sequence-to-Sequence (Seq2Seq) Model

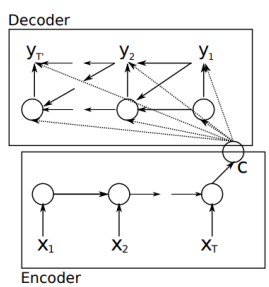



*Figure 3.1.3.2.1 Encoder-Decoder Seq2Seq Architecture (Sutskever et al., 2014)*

A Seq2Seq model comprises two main components: an encoder and a decoder, as shown in figure 3.1.3.2.1. The encoder processes the input sequence and summarizes it into a fixed-length vector, while the decoder generates the output sequence from this vector. The encoder and decoder are connected by a neural network, and they can have different lengths [33]. With this, Seq2Seq allows neural networks to have improved handling of long-range dependencies and a better contextual understanding of inputs. However, Seq2Seq models can be further improved by applying different components, including attention mechanism, embedding layer, and layer normalization [57].

### 3.1.3.3 Attention Mechanism

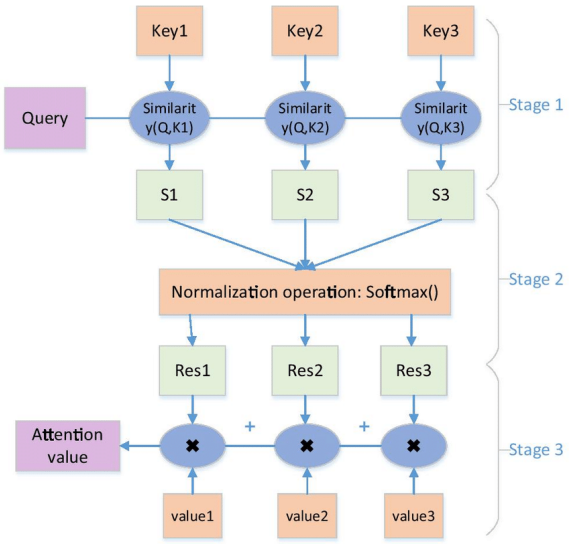



*Figure 3.1.3.3.1 General process of attention mechanism (H. Liu et al., 2021)*

The attention mechanism is used to weigh the importance of different parts of an input sequence when processing a particular sequence element. This allows the model to focus on relevant information. Moreover, the core concept of the attention mechanism involves calculating a weighted sum of the input elements, as shown in figure 3.1.3.3.1, with the weights being adaptively determined by the resemblance between the elements and the currently processed element. It enables the model to attend to different parts of the input sequence, emphasizing relevant information while de-emphasizing less relevant or irrelevant information [57].

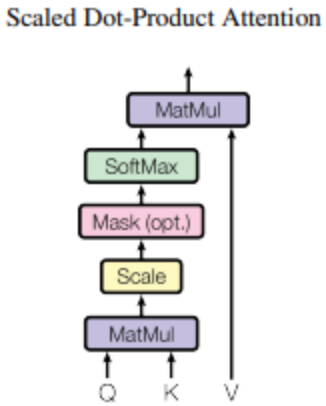



*Figure 3.1.3.3.2 Scaled Dot-Product Attention Block (Vaswani et al., 2017)*

Vaswani et al. described the attention mechanism with three sets of vectors: query (Q), key (K), and value (V). The query vector represents the element of the input sequence for which the model is trying to compute the attention weights. It's like the focus of the attention. The key vector is associated with each element in the input sequence and helps determine the relevance of that element to the query [57]. The dot product between the query and key vectors is a measure of similarity or importance. The value vector contains information about the content of the input sequence element. The values are weighted by the attention scores and summed up to produce the final output of the attention mechanism.

These vectors are employed to compute attention scores by taking the query and key vectors' dot product, which measure each element's relevance in the input sequence to a given query. The resulting scores are scaled and passed through a softmax activation, creating a probability distribution that determines the weights for a weighted sum of the corresponding value vectors. This modular approach allows different components of the model to focus on distinct aspects of the input sequence, contributing to the transformer's effectiveness in capturing complex dependencies.

#### 3.1.3.3.1 Multi-Head Attention Mechanism

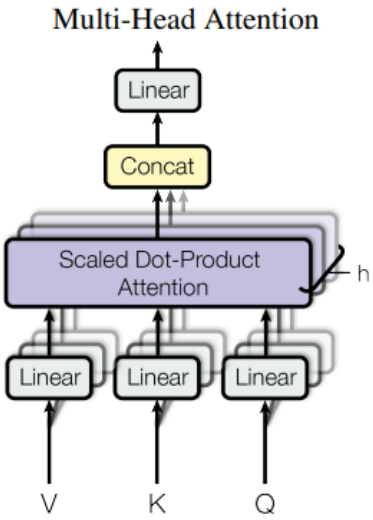



*Figure 3.1.3.3.1.1 Multi-Head Attention Block (Vaswani et al., 2017)*

The multi-head attention mechanism is an extension of the basic attention mechanism, which uses parallel sets of attention weights [57]. Moreover, it conducts multiple attention calculations using distinct sets of adaptable parameters called "heads." Each head concentrates on distinct aspects or patterns within the input data. The results from these multiple attention heads are commonly combined through concatenation or linear operations to create a richer and more expressive representation.

### 3.1.3.4 Embeddings

Embeddings refer to continuous and compact vector portrayals of words or phrases. These vector representations are crafted through diverse methods, intending to grasp words' semantic and contextual significance, making them beneficial for tasks in natural language processing (NLP). Rather than depicting words as separate symbols or one-hot encodings, embeddings offer a numerical, uninterrupted representation that encodes connections between words, rooted in how they are used in extensive text collections [83].

Moreover, these embeddings are proven to help reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space. Embeddings serve three primary functions: they enable the discovery of nearest neighbors in the embedding space,

which can be utilized for recommendations and categorization. Additionally, they function as inputs for supervised machine-learning tasks and facilitate the visualization of relationships and concepts between categories [84].

### 3.1.3.5 Layer Normalization

Layer normalization is a crucial deep-learning technique that focuses on normalizing the activations of neurons within a neural network layer. Its primary function is to ensure that the inputs to a layer maintain a consistent scale and mean, ultimately improving the stability and efficiency of training deep neural networks.

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$

*Figure 3.1.3.5.1 Layer Normalization by Ba et al*

The Layer Normalization first takes the mean of each layer's activation neuron. Then, use the calculated mean to get the standard deviation of each layer. Each activation neuron will then be subtracted from the mean and divided by the standard deviation, outputting a new sequence. This normalizes data for each layer, making it adaptable to scenarios with varying or impractical batch sizes. One of its key advantages lies in mitigating the issues of vanishing or exploding gradients, enhancing the training process for deep neural networks [58].

## 3.2 Conceptual Framework

This chapter presents the study's proposed model which is a sequence-to-sequence BiGRU model to address the problems of an LSTM model in generating food recipes. This section discusses how the Inputs from the encoder and decoder produce the output.

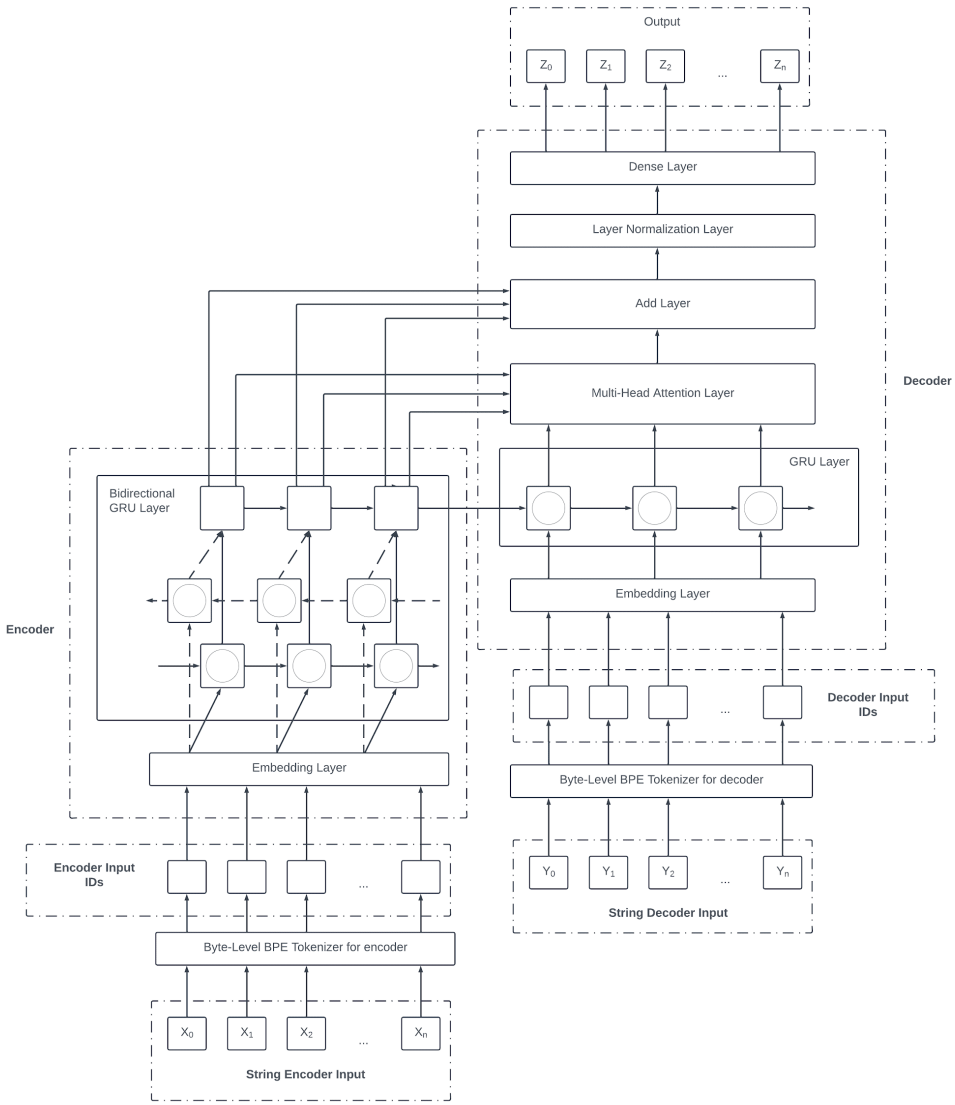



*Figure 3.2.1 Conceptual Framework*

The architecture of the study's model has a sequence-to-sequence architecture similar to [8] but instead of having 2 encoders, the study's model only consists of 1 encoder. The attention mechanism used will also be altered by the researchers to fit the context of the study. The layers inside the encoder and decoder will be further discussed in the next paragraphs.

The encoder of the architecture starts with the String Encoder Input which consists of NERs of each recipe. Using the Byte-level BPE Tokenizer, the NERs will be converted to IDs which will serve as the input of the Embedding Layer. The inclusion of an Embedding layer was also inspired by [8]. The Embedding Layer will then output a dense vector for each token. Each of these dense vectors will be processed by the bidirectional GRU layer. The outputs from the forward and backward GRU layers will be concatenated and passed to the decoder.

The decoder of the architecture starts with the String Decoder Input which consists of Ingredients, Instructions, and Title. Same as the encoder, the String Inputs will be converted to IDs using the Byte-level BPE Tokenizer which will serve as the input of the Embedding Layer. The Embedding Layer will then output a dense vector for each token. Each of the dense vectors will be processed by a Unidirectional GRU layer. Using [85] as the basis, the unidirectional GRU will get its initial states from the hidden state of the Bidirectional GRU from the encoder. The output of the Unidirectional GRU of the Decoder, together with the output of the Bi-Directional GRU of the encoder, will be passed to the Multi-Head attention layer. A Multi-Head Attention layer consists of a Query, Value, and Key. The query in this context will be coming from the encoder while the value and key will be from the Decoder. The next layer will be the Add layer, this layer is getting the sum of the output of Multi-Head Attention and the Encoder BiGRU. The output sum of the Add layer will be used as the input in the layer normalization. The combination of the Multi-Head attention layer, Add layer, and Layer Normalization is inspired by [57]. The final layer which is the dense layer is a fundamental component of a neural network that transforms the representations of the data.

# 3.3 Methodology

The methodology section involves gathering and preprocessing the RecipeNLG dataset, training the Byte-level BPE Tokenizer, tokenizing, and training the Sequence-to-Sequence BiGRU Model. Loss function, BLEU scores, Jaccard Similarity, and Human Evaluation are used to evaluate the quality of the generated recipes The results are then discussed to identify opportunities for improvement.

## 3.3.1 Gathering and Preprocessing of Data

Datasets are one of the big factors that affect the process of training a neural network model. It must be ensured that the data used is accurate and has the correct representation of what the context is all about. The dataset needed to generate a recipe should have a title, a list of ingredients with measurements, and step-by-step instructions. RecipeNLG [52] satisfies this condition and was used in training the model. In addition, the dataset contains a link column which would be used later for web scraping. The RecipeNLG dataset offers over a million new and preprocessed recipes on top of the Recipe1M+ dataset, specifically, 2,231,142 recipes. One of the deciding factors for using this dataset is the Named Entity Recognizer (NER). NER allows pairwise comparison of the original and generated recipe which would be important in testing and measuring the relevance of the recipe generated.

To further improve the quality of the recipes that would be used to train the model, the researchers decided to use filtering methods. In addition, the study has hardware limitations which hindered the researchers from using the whole dataset. The methods used to filter the dataset will be discussed in the following sections.

### 3.3.1.1 Filtering the dataset through the recipe's rating

The first method to lessen the quantity of the dataset is to filter each recipe through its ratings. The recipeNLG dataset itself does not have a rating column, hence, the researchers decided to web-scrape the ratings for each recipe by accessing the link of each recipe through BeautifulSoup and Selenium WebDriver. The researchers used Beautiful Soup for websites that don't require Javascript to load the data since it is faster compared to tools that use WebDriver. However, one of the drawbacks of BeautifulSoup is the inability to data-scrape Javascript-based

websites which is what Selenium WebDriver can do. Selenium WebDriver enables the users to simulate actions just as a person would. This feature enables JavaScript to be loaded before scraping the data [86].

| **Base links** | **Number of recipes** | **Base links** | **Number of recipes** |
|---|---|---|---|
| www.cookbooks.com | 896341 | www.food.com | 499616 |
| www.epicurious.com | 129444 | tastykitchen.com | 78768 |
| www.myrecipes.com | 64895 | www.allrecipes.com | 61398 |
| cookpad.com | 61020 | cookeatshare.com | 59307 |
| www.yummly.com | 51963 | www.tasteofhome.com | 51594 |
| www.foodnetwork.com | 49443 | food52.com | 48501 |
| www.kraftrecipes.com | 42010 | recipeland.com | 24418 |
| recipes-plus.com | 20524 | cooking.nytimes.com | 16367 |
| www.foodandwine.com | 15436 | www.seriouseats.com | 12632 |
| www.foodgeeks.com | 8963 | www.cookstr.com | 8797 |
| online-cookbook.com | 5691 | www.chowhound.com | 5671 |
| www.vegetariantimes.com | 4578 | www.delish.com | 3880 |

| | | | |
|---|---|---|---|
| allrecipes.com | 3204 | www.landolakes.com | 2492 |
| www.foodrepublic.com | 2259 | www.lovefood.com | 1930 |

*Table 3.3.1.2.1 The number of recipes for each base links*

Table 3.3.1.2.1 is a sorted table that refers to the number of recipes associated with each of the base links. In the process of web-scraping, the researchers did not include www.epicurious.com as recipes are inaccessible, cookpad.com as it doesn't have ratings, and www.yummily.com as it needs a subscription. As for www.cookbooks.com, the researchers only included the first 260000 recipes due to time constraints.

| Base links | The tool used to data scrape |
|---|---|
| www.cookbooks.com | BeautifulSoup |
| www.food.com | BeautifulSoup |
| tastykitchen.com | BeautifulSoup |
| www.myrecipes.com | BeautifulSoup |
| www.allrecipes.com | BeautifulSoup |
| cookeatshare.com | BeautifulSoup |
| www.tasteofhome.com | Selenium Webdriver |
| www.foodnetwork.com | Selenium Webdriver |

*Table 3.3.1.2.2 Accepted base links with their respective tools*

In consideration of the accessibility of the base links, the quality of the recipes, and the study's limitations on time and hardware, the base links in Table 3.3.1.2.2 are what was used by the researchers. After data scraping, the dataset will then be filtered to only accept recipes that have 4 or above ratings out of 5 which is inspired by [8]. This filter method resulted in 292,831 recipes.

#### 3.3.1.2 Filtering the dataset through the recipe's length, number of ingredients, and validity of NERs.

There were 3 more methods that the researchers used for filtering the dataset. First is the length of the recipe, which is a crucial factor for the study's model. As was discussed in [11], GRU is better when the sequence is short than when the sequence is long. The dataset was then filtered to have at most 200 words per recipe which resulted in 204,833 recipes. Second is the number of ingredients, the researchers observed that some recipes had only 1 ingredient while some had 36 ingredients. The number of ingredients affects the quality of recipes. A recipe with 1 ingredient has either wrong ingredients or the recipe has low quality and A recipe with 36 ingredients has either wrong ingredients or the instructions are very short. This led the researchers to limit the number of ingredients to 3 as the minimum and 15 as the maximum which was estimated through the number of recipes it resulted. This filter method resulted in 198,933 recipes. Third is the validity of NERs, there are some recipes in the dataset in which the length of the NERs doesn't match the length of the ingredients, this means that the NERs extracted were wrong. Hence, the researchers also checked if the number of NERs are equal to the number of ingredients to accept the recipe which resulted in 165,498 recipes.

#### 3.3.1.3 Dataset Splitting

The dataset was splitted into 3 sets: training, validation, and testing sets. The training and validation sets were both used for the duration of the training, the difference is that the training set is used to train the model, while the validation set is used to evaluate the performance of the model during training. The testing set on the other hand will be used for the BLEU metrics and

Jaccard Similarity. The split ratio is 85% for the training set, 10% for the validation set, and 5% for the testing set. To be exact, the training set consists of 140,673 recipes, the validation set consists of 16,550, and the testing set consists of 8,275 recipes.

#### 3.3.1.4 Adding Control Tokens

[START] butter [NER_NEXT] chicken breasts [NER_NEXT] cream of chicken soup [NER_NEXT] swiss cheese [NER_NEXT] ham [NER_NEXT] white wine [END]

*Figure 3.3.1.4.1 Preprocessing of encoder through adding control tokens*

Control Tokens are used to structure the recipe into different parts. Refer to Figure 3.3.1.4.1, there are 3 different control tokens that were used in the encoder: [START], [NER_NEXT], and [END].

[START] [INGREDIENTS_START] 1 tablespoon butter [INGREDIENTS_NEXT] 4 boneless skinless chicken breasts ... [INGREDIENTS_END] [INSTRUCTIONS_START] Melt butter in a large frying pan. [INSTRUCTIONS_NEXT] Add chicken and cook until browned, about 10 minutes... [INSTRUCTIONS_END] [TITLE_START] Chicken With Ham And Cheese [TITLE_END] [END]

*Figure 3.3.1.4.2 Preprocessing of decoder through adding control tokens*

For the decoder, refer to Figure 3.3.1.4.2, there are 10 different control tokens: [START], [INGREDIENTS_START], [INGREDIENTS_NEXT], [INGREDIENTS_END], [INSTRUCTIONS_START], [INSTRUCTIONS_NEXT], [INSTRUCTIONS_END], [TITLE_START], [TITLE_END], and [END].

### 3.3.2 Training the Byte-level BPE Tokenizer

```
encodedTokenizer.train(
    files=f"data/filtered_texts/encoded_string_filtered_for_vocab.txt",
    vocab_size=10000,
    special_tokens=special_tokens,
)
decodedTokenizer.train(
    files=f"data/filtered_texts/decoded_string_filtered_for_vocab.txt",
    vocab_size=30000,
    special_tokens=special_tokens,
)
```

*Figure 3.3.2.1 Code for training the tokenizer*

The first step for training the Byte-level BPE tokenizer is to generate a text file that contains all of the words that would be trained for the tokenizer. Refer to Figure 3.3.2.1 for the code used to train the Byte-level BPE tokenizer. Since the study's model is a sequence-to-sequence model, two byte-level BPE tokenizer was made: one for the encoder, and one for the decoder. The encoder vocabulary text file consists of NERs joined together while the decoder consists of ingredients, instructions, and titles that were also joined together. These vocabulary text files were used to train the encoder and decoder byte-level BPE tokenizer. The library used for creating the Byte-level BPE tokenizer is tokenizers (https://huggingface.co/docs/tokenizers/index). Creating this kind of tokenizer requires a vocabulary size, hence, the vocabulary size for the encoder is 10000, while for the decoder is 30000. The vocabulary size for the encoder and decoder was decided by considering the hardware and time constraint limitations since the size of the vocabulary will affect the resources needed and the training time of the model. After training the two tokenizers, they were saved and will be then used for tokenizing.

### 3.3.3 Tokenizing of data

```
context_raw_ids_train = list(
    map(lambda id: encodedTokenizer.encode(id.numpy().decode()).ids, context_raw_train)
)
context_raw_ids_val = list(
    map(lambda id: encodedTokenizer.encode(id.numpy().decode()).ids, context_raw_val)
)
target_raw_ids_train = list(
    map(lambda id: decodedTokenizer.encode(id.numpy().decode()).ids, target_raw_train)
)
target_raw_ids_val = list(
    map(lambda id: decodedTokenizer.encode(id.numpy().decode()).ids, target_raw_val)
)
padded_context_raw_train = pad_sequences(context_raw_ids_train, padding="post", value=0)
padded_context_raw_val = pad_sequences(context_raw_ids_val, padding="post", value=0)
padded_target_raw_train = pad_sequences(target_raw_ids_train, padding="post", value=0)
padded_target_raw_val = pad_sequences(target_raw_ids_val, padding="post", value=0)
```

*Figure 3.3.3.1 Code for tokenizing the data*

In order to convert the text recipes into data that can be used to train the model, a tokenization process must take place. Text recipes will be tokenized using the byte-level BPE tokenizer that was created in the previous process. Refer to Figure 3.3.3.1 for the code for tokenizing the data. After tokenization, each sequence will be padded by 0. The length of the padding is equal to the maximum length of the sequences since they must be equal in length. The tokenized data were then saved using pickle.

**Recipe Text:** [START] oyster [NER_NEXT] milk [NER_NEXT] butter [NER_NEXT] salt [NER_NEXT] seafood seasoning [END]

**Converted:** [3000, 1699, 226, 3004, 349, 226, 3004, 313, 226, 3004, 289, 226, 3004, 2413, 642, 226, 3001]

*Figure 3.3.3.1 Example of a tokenized data*

### 3.3.4 Training the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism Model

The tokenized 140,673 recipes will be the data to be used in training the sequence-to-sequence BiGRU model. The dataset was divided to have 64 sequences for each batch, which maximizes the capacity of the machine. The platform used by the researchers is

Google Colab with the Nvidia Tesla V100 GPU. As for the tools, the researchers used Tensorflow for the deep learning framework, Keras for the building and training of the BiGRU model, and Tensorboard for the visualization of the model.

```
BATCH_SIZE = 32
BUFFER_SIZE_TRAIN = len(padded_context_raw_train)
BUFFER_SIZE_VAL = len(padded_context_raw_val)

train_raw = (
    tf.data.Dataset
    .from_tensor_slices((padded_context_raw_train, padded_target_raw_train))
    .shuffle(BUFFER_SIZE_TRAIN)
    .batch(BATCH_SIZE))

val_raw = (
    tf.data.Dataset
    .from_tensor_slices((padded_context_raw_val, padded_target_raw_val))
    .shuffle(BUFFER_SIZE_VAL)
    .batch(BATCH_SIZE))
```

*Figure 3.3.4.1 Code for Converting data to tensors*

The first step for training the model was to convert the tokenized recipes to tensors as this data type works well when training the model. Tensors are n-dimensional arrays that can be run in GPU [87]. For the optimizer, the researchers used Adam with a learning rate of 0.0001 which is inspired by [45]. The researchers tried other learning rates like 0.001 and 0.01 but the 0.0001 produced more steady loss values. A Checkpoint callback was also added to save the weights and load them later if the model needs additional training.

```python
class Encoder(tf.keras.layers.Layer):
    def __init__(self, units, encodedTokenizer):
        super(Encoder, self).__init__()
        self.vocab_size = encodedTokenizer.get_vocab_size()
        self.units = units
        self.encodedTokenizer = encodedTokenizer
        self.embedding = tf.keras.layers.Embedding(
            self.vocab_size, units, mask_zero=True
        )

        self.rnn = tf.keras.layers.Bidirectional(
            merge_mode="concat",
            layer=tf.keras.layers.GRU(
                units, return_sequences=True, recurrent_initializer="glorot_uniform"
            ),
        )

    def call(self, x):
        x = self.embedding(x)
        x = self.rnn(x)
        return x
```

*Figure 3.3.4.2 Code for the encoder of the model*

```python
class Decoder(tf.keras.layers.Layer):
    def __init__(self, units, decodedTokenizer):
        super(Decoder, self).__init__()
        self.decodedTokenizer = decodedTokenizer
        self.vocab_size = decodedTokenizer.get_vocab_size()
        self.start_token = decodedTokenizer.encode("[START]").ids
        self.end_token = decodedTokenizer.encode("[END]").ids
        self.units = units
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, units, mask_zero=True)
        self.rnn = tf.keras.layers.GRU(units, return_sequences=True, return_state=True, recurrent_initializer="glorot_uniform")

        self.attention = CrossAttention(units)
        self.output_layer = tf.keras.layers.Dense(self.vocab_size)

    def call(self, context, target, state=None, return_state=False):
        target = self.embedding(target)
        target, state = self.rnn(target, initial_state=state)

        target = self.attention(target, context)
        self.last_attention_weights = self.attention.last_attention_weights
        logits = self.output_layer(target)

        if return_state:
            return logits, state
        else:
            return logits
```

*Figure 3.3.4.3 Code for the decoder of the model*

The encoder and decoder of the model are then created, refer to Figure 3.3.4.3 and 3.3.4.4. The training of the model consists of 30 epochs and 2199 steps per epoch. The steps per epoch were calculated by the number of sequences divided by the batch size. The training of the sequence-to-sequence BiGRU with an Attention Mechanism model took 5 hours and 38 minutes to complete.

## 3.3.5 Training the LSTM model

```
def lstm_model():
    model = models.Sequential()
    model.add(layers.Embedding(vocab_size, UNITS))
    model.add(layers.LSTM(UNITS))
    model.add(layers.Dropout(0.1))
    model.add(layers.Dense(vocab_size, activation='softmax'))
    return model
```

*Figure 3.3.5.1 Code for LSTM model*

The aim of this study is to compare the study's model and Goel et al.'s model [1]. The LSTM model cannot be replicated completely because there is a lack of details on how it was constructed, therefore the researchers assume the layers of the model based on vague descriptions. Refer to Figure 3.3.5.1 for the code of the LSTM model. The model comprises 4 layers: Embedding layer, LSTM layer, Dropout layer, and Dense layer. The preparation of the dataset, tools used, optimizer, and number of epochs is similar to the BiGRU model. The dropout rate of the dropout layer is 0.1, and the dense layer has a softmax activation. The training of the LSTM model took 52 minutes to complete.

## 3.3.6 Evaluating the Model

There are three areas of the research that need to be evaluated: The accuracy, coherence, creativity, novelty, and doability of the generated food recipe. The accuracy was gauged by the BLEU score, coherence was measured through Jaccard Similarity, and creativity, novelty, and doability were assessed by Human Evaluation.

### 3.3.6.1 BLEU score

In relation to the study of Goel et al, the researchers used BLEU as the accuracy metric for generating recipes. The BLEU (Bilingual Evaluation Understudy) score is commonly used in natural language processing to evaluate the quality of a machine-generated text. It is a metric that measures the similarity between a generated text and one or more reference texts that are considered to be the ground truth [88]. In the context of the study, the BLEU score was used to evaluate how closely the generated recipes match the human-written recipes in the newly preprocessed RecipeNLG dataset.

```
counter = 0
list_of_original_and_generated_recipes = []
for recipe in test_list:
    ## Get the NERs from the recipe
    original_and_generated_recipe = {}
    original_ners = extract_ners_from_recipe(recipe)
    original_instructions = extract_instructions_from_recipe(recipe)
    original_ingredients = extract_ingredients_from_recipe(recipe)
    original_title = extract_title_from_recipe(recipe)

    if (model_name == 'gru'):
        formatted_ner = f'[START] {" [NER_NEXT] ".join(original_ners)} [END]'
    else if (model_name == 'lstm'):
        formatted_ner = f'[START] [NER_START] {" [NER_NEXT] ".join(original_ners)} [NER_END] [INGREDIENTS_START]'


    result = model.translate(formatted_ner)
    generated_recipe = result[0].numpy().decode()

    generated_instructions = extract_instructions_from_recipe(generated_recipe)
    generated_ingredients = extract_ingredients_from_recipe(generated_recipe)
    generated_title = extract_title_from_recipe(generated_recipe)
```

*Figure 3.3.6.1.1 Code for generating the recipes using a BiGRU and LSTM in test dataset*

In order to evaluate the BLEU score of the BiGRU and LSTM model, the researchers first generated the recipes using the two models, refer to Figure 3.3.6.1.1. The NERs used as prompts came from the test dataset. Moreover, since the BiGRU model has a sequence-to-sequence structure, the prompt is different from the LSTM model.

```
def get_reference_recipes(recipe, list_generated_ners):
    original_recipe = recipe["original"]
    reference_recipes = []
    reference_recipes.append(tokenize_text(convert_recipe_to_string(original_recipe)))
    for recipe_train in train_data:
        extracted_ners = extract_ners_from_recipe(recipe_train)
        if len(set(extracted_ners).intersection(set(list_generated_ners))) == len(set(extracted_ners)):
            reference_recipes.append(tokenize_text(recipe_train))
    return reference_recipes
```

*Figure 3.3.6.1.2 Code for acquiring the reference recipes*

Since it will take too much time to compare each of the recipes in the test data set, the researchers decided to only get the relevant reference recipes. The reference recipes were made by first adding the test recipe, and then comparing the NERs of the generated recipe and training data set, refer to Figure 3.3.6.1.1. If the NERs of the test recipe match the train recipe, the train recipe will be added to the reference recipes. In addition, each of the recipes added will be tokenized.

```
def calculate_bleu_score(generated_recipe, reference_recipes):
    tokenized_generated_recipe = tokenize_text(convert_recipe_to_string(generated_recipe))
    bleu_score = sentence_bleu(reference_recipes, tokenized_generated_recipe)
    recipe_bleu.append(bleu_score)
    return bleu_score
```

*Figure 3.3.6.1.2 Code for calculating the BLEU score*

In evaluating the quality of the generated recipes, the BLEU score was computed on the test dataset. This involves comparing each generated recipe with the corresponding reference recipes in the dataset and calculating a similarity score. The researchers decided to calculate the BLEU score in 3 areas; ingredients, instructions, and the whole recipe. The researchers used the sentence_bleu library from NLTK, refer to Figure 3.3.6.1.2. The sentence_bleu function accepts two parameters, first is the reference recipes which were already generated, and the recipe generated by the machine. The BLEU score was calculated as the geometric mean of the n-gram precisions, where n ranges from 1 to 4. The score ranges from 0 to 1, with higher scores indicating better similarity between the generated and reference recipes. This helped the researchers in determining whether the BiGRU model is more capable to generate recipes that closely match the human-written recipes in the RecipeNLG dataset than the LSTM model.

### 3.3.6.2 Jaccard Similarity

```
total_similarity = 0
upper_limit = len(data)
limit = 0

for row in data[:upper_limit]:
    original_ners = row['original']['ners']
    ingredients_array  = extract_ners(row['generated']['ingredients'],original_ners)
    instructions_array  = extract_ners(row['generated']['instructions'],original_ners)

    if row['generated']['ingredients'] != None and row['generated']['instructions'] != None:
        if (len(ingredients_array) > 0 and len(instructions_array) > 0):
            jaccard_similarity_ner = 1 - jaccard_distance(ingredients_array, instructions_array)
            total_similarity += jaccard_similarity_ner
        limit += 1
    else:
        print(row)

average_similarity = total_similarity / limit

print(average_similarity)
```

*Figure 3.3.6.2.1 Code for calculating Jaccard Similarity*

Similar to how Lee et al. [45] measured the Between-field ingredient coherence, the researchers will use the same method in measuring the coherence of ingredients and instructions. Same as the BLEU score metric, the first step is to generate the recipe using the two models.

```
def findWholeWord(w):
    pattern = r"\b({0})\b".format(re.escape(w))
    return re.compile(pattern, flags=re.IGNORECASE).search

def extract_ners(item_list,original_ners):
    extracted_ners = set()

    if item_list is not None:
        for ner in original_ners:
            for item in item_list:
                if findWholeWord(ner)(item):
                    extracted_ners.add(ner)
                    break

    return extracted_ners
```

*Figure 3.3.6.2.2 Code for extracting NERs in the Jaccard Similarity process*

As shown in figure 3.3.6.2.2, the food entities were extracted by comparing the generated ingredients and instructions to the original food entities, which would yield a result of two lists containing the extracted food entities from the generated ingredients and instructions. These 2 lists would then be evaluated using Jaccard Similarity.

$$JS = \frac{|A \cap B|}{|A \cup B|}$$

*Figure 3.3.6.2.2 Jaccard Similarity*

As per [89], the Jaccard Similarity is a method used to assess the similarity between two sets in Set Theory. It measures how alike two sets are, so the greater the value of Jaccard Similarity, the more similar the two sets are. This was used to measure the similarity of extracted food entities of ingredients and instructions. would mean that the ingredients and instructions have high coherence. The Jaccard similarity coefficient ranges from 0 to 1, where 0 indicates no similarity, and 1 indicates complete similarity.

#### 3.3.6.3 Human Evaluation

The human evaluators are provided with an evaluation form to assess the quality of a generated recipe based on several key criteria, scored on a scale from 0 to 5. The first criterion, clarity, quantifies the ease with which the recipe can be followed. Creativity and novelty measure the inventive use of user-inputted ingredients. Suitability evaluates how well the generated recipe aligns with the provided ingredients, ensuring effective utilization. Doability and edibility assess the practicality of preparation and the dish's overall appeal. Likelihood to try combines various factors to gauge the inclination to experiment with and taste the generated recipe. The overall recipe quality rating synthesizes these dimensions, offering a comprehensive evaluation of the generated recipe's excellence.

The questions provided are as follows:

1. How easy was the recipe to follow? (Clarity)

2. How creative or inventive is the recipe generated from the user-inputted ingredients? (Creativity/Novelty)

3. How well does the generated recipe suit your provided ingredients? (Suitability)

4. How doable and edible does the recipe appear to be based on your inputted ingredients and the generated recipe? (Doability and Edibility)

5. How likely are you to try the recipe generated from your inputted ingredients? (Likelihood to Try)

6. How would you rate the overall quality of the recipe generated from your inputted ingredients? (Overall Recipe Quality)

The results are then filtered to only contain evaluation information from respondents that have at least 1 year of cooking experience. A table will then be generated to contain the visualization of the distribution of the respondents' scores, as well as the average score of each citeria.

| Kappa | Agreement |
|---|---|
| < 0 | Less than chance agreement |
| 0.01–0.20 | Slight agreement |
| 0.21– 0.40 | Fair agreement |
| 0.41–0.60 | Moderate agreement |
| 0.61–0.80 | Substantial agreement |
| 0.81–0.99 | Almost perfect agreement |

*Figure 3.3.6.3.1 Fleiss' Kappa Interpretation Chart (Hartling, 2012)*

```
from statsmodels.stats.inter_rater import fleiss_kappa,aggregate_raters

formatted_table = []
formatted_table = aggregate_raters(all_scores, 6)

kappa = fleiss_kappa(formatted_table[0], method='randolph')
```

*Figure 3.3.6.3.2 Fleiss' Kappa Code*

Moreover, results from this evaluation are used to calculate Fleiss' Kappa score to measure overall agreements between raters with results closest to -1 being lowest and results

closest to 1 being the highest, as shown in figure 3.3.3.6.3.1 . Figure 3.3.6.3.2 displays the code used by the researchers to calculate the Fleiss' Kappa score. Furthermore, the researchers used the statsmodels.stats.inter_rater library. The all_scores contains a 2-dimensional array holding the evaluators' score, which will then be prepared and transformed into a shape that the method fleiss_kappa can process.

### 3.3.7 Web Application

The web application serves as the host for the recipe-generating model. Users are provided with a selection of ingredients from the preprocessed recipeNLG dataset. Moreover, users are provided with an evaluation page that allows them to rate the output of the model, which will then be stored in a database.

Given that the model is integrated in Python, the researchers chose to employ Flask, a lightweight framework renowned for its scalability, ease of use, flexibility, and shallow learning curve [90].

For the database, the researchers opted to use SQLite due to it being serverless, having a lightweight nature, and its inclusion in Python's standard library [91]. This database is used to store user evaluation scores, along with optional information such as name, email, contact number, and profession.

The back-end's API endpoints facilitate the execution and retrieval of recipe outputs generated by the training model and the storage of evaluator data in the database. These endpoints are then called in the front end. The researchers chose to utilize Next.js for the front end due to its strong developers' support, good user experience, and great SEO [92]

The deployment strategy for the web application involves the utilization of two distinct platforms. The front end of the application is hosted on Render, as it is known for its user-friendly and scalable hosting services, effectively supporting the front end's requirements. In contrast, due to the model's substantial data consumption that exceeds Render's limit for free-tier users, the back-end is deployed through the utilization of ngrok. This choice enables the backend to be accessible via a public URL, ensuring the seamless interaction of the frontend and backend components despite the resource limitations imposed by the hosting platform.

### 3.3.7 Timetable

The study's timetable will be presented using the Gantt Chart. The study was conducted for a total of 4 months. The preprocessing of the data was done for 3 months. The 3rd month consisted of training the Byte-level BPE tokenizer, tokenizing of data, and training of the BiGRU model. In the last month, the training of the sequence-to-sequence BiGRU model was continued, and started the training of the LSTM model, evaluating the model, and Web application.

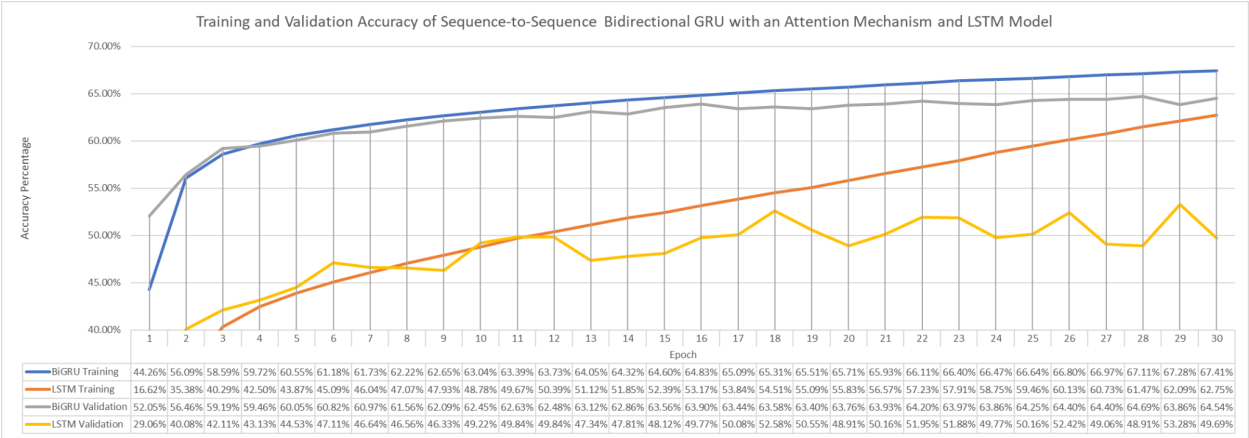
| Tasks | Aug. | Sept. | Oct. | Nov. |
|---|---|---|---|---|
| Preprocessing the data | | | | |
| Training the Byte-level BPE Tokenizer | | | | |
| Tokenizing of data | | | | |
| Training the Sequence-to-Sequence BiGRU Model | | | | |
| Training the LSTM model | | | | |
| Evaluating the Model | | | | |

| | | | | |
|---|---|---|---|---|
| Web Application | | | | |

# Chapter 4

# Results and Discussion

## 4.1 Training Accuracy and Loss comparison of Sequence-to-sequence Bidirectional GRU with an Attention Mechanism and LSTM Model.

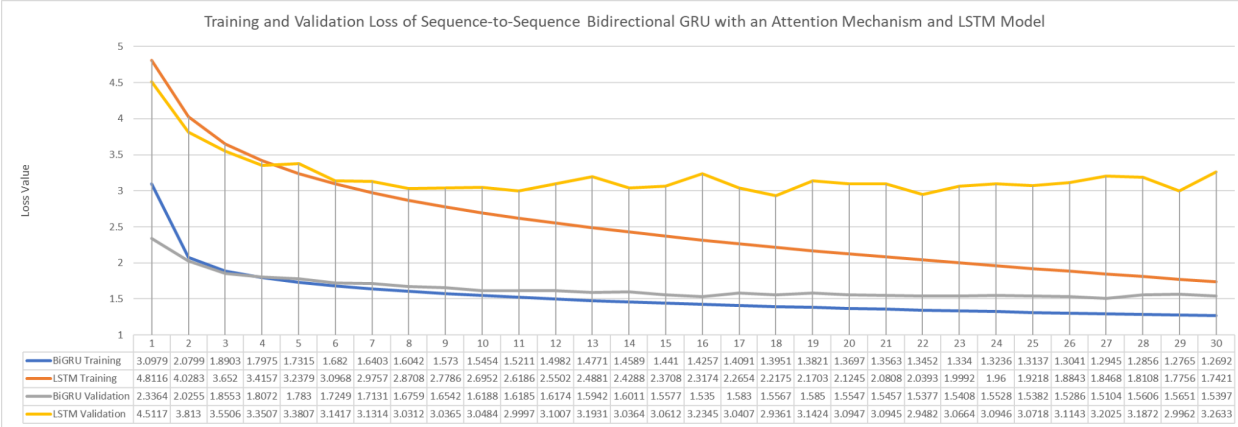



| Epoch | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BiGRU Training | 44.26% | 56.09% | 58.59% | 59.72% | 60.55% | 61.18% | 61.73% | 62.22% | 62.65% | 63.04% | 63.39% | 63.73% | 64.05% | 64.32% | 64.60% |
| LSTM Training | 16.62% | 35.38% | 40.29% | 42.50% | 43.87% | 45.09% | 46.04% | 47.07% | 47.93% | 48.78% | 49.67% | 50.39% | 51.12% | 51.85% | 52.39% |
| BiGRU Validation | 52.05% | 56.46% | 59.19% | 59.46% | 60.05% | 60.82% | 60.97% | 61.56% | 62.09% | 62.45% | 62.63% | 62.48% | 63.12% | 62.86% | 63.56% |
| LSTM Validation | 29.06% | 40.08% | 42.11% | 43.13% | 44.53% | 47.11% | 46.64% | 46.56% | 46.33% | 49.22% | 49.84% | 49.84% | 47.34% | 47.81% | 48.12% |

| Epoch | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BiGRU Training | 64.83% | 65.09% | 65.31% | 65.51% | 65.71% | 65.93% | 66.11% | 66.40% | 66.47% | 66.64% | 66.80% | 66.97% | 67.11% | 67.28% | 67.41% |
| LSTM Training | 53.17% | 53.84% | 54.51% | 55.09% | 55.83% | 56.57% | 57.23% | 57.91% | 58.75% | 59.46% | 60.13% | 60.73% | 61.47% | 62.09% | 62.75% |
| BiGRU Validation | 63.90% | 63.44% | 63.58% | 63.40% | 63.76% | 63.93% | 64.20% | 63.97% | 63.86% | 64.25% | 64.40% | 64.40% | 64.69% | 63.86% | 64.54% |
| LSTM Validation | 49.77% | 50.08% | 52.58% | 50.55% | 48.91% | 50.16% | 51.95% | 51.88% | 49.77% | 50.16% | 52.42% | 49.06% | 48.91% | 53.28% | 49.69% |

*Figure 4.1.1 Graph for the Training and Validation Accuracy of LSTM and BiGRU models*

Training accuracy is a measure of how well a model is performing on the training data during the training process while the Validation accuracy is a measure of how well the model generalizes the data that was not seen in the training. Figure 4.1.1 shows that the training accuracy for the BiGRU and LSTM models has a consistent upward trend, indicating that these two models are learning and improving over time. The training accuracy for BiGRU starts at 44.26%, while the LSTM starts at 22.63%. The final training accuracy for the BiGRU model is 67.41%, while the LSTM model is 60.94%. As for the validation accuracy, there are some fluctuations over epochs. The validation accuracy for BiGRU starts at 52.05%, while the LSTM starts at 29.06%. The final validation accuracy for BiGRU model is 64.54%, while for the LSTM is 53.28%.




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BiGRU Training | 3.0979 | 2.0799 | 1.8903 | 1.7975 | 1.7315 | 1.682 | 1.6403 | 1.6042 | 1.573 | 1.5454 | 1.5211 | 1.4982 | 1.4771 | 1.4589 | 1.441 |
| LSTM Training | 4.8116 | 4.0283 | 3.652 | 3.4157 | 3.2379 | 3.0968 | 2.9757 | 2.8708 | 2.7786 | 2.6952 | 2.6186 | 2.5502 | 2.4881 | 2.4288 | 2.3708 |
| BiGRU Validation | 2.3364 | 2.0255 | 1.8553 | 1.8072 | 1.783 | 1.7249 | 1.7131 | 1.6759 | 1.6542 | 1.6188 | 1.6185 | 1.6174 | 1.5942 | 1.6011 | 1.5577 |
| LSTM Validation | 4.5117 | 3.813 | 3.5506 | 3.3507 | 3.3807 | 3.1417 | 3.1314 | 3.0312 | 3.0365 | 3.0484 | 2.9997 | 3.1007 | 3.1931 | 3.0364 | 3.0612 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BiGRU Training | 1.4257 | 1.4091 | 1.3951 | 1.3821 | 1.3697 | 1.3563 | 1.3452 | 1.334 | 1.3236 | 1.3137 | 1.3041 | 1.2945 | 1.2856 | 1.2765 | 1.2692 |
| LSTM Training | 2.3174 | 2.2654 | 2.2175 | 2.1703 | 2.1245 | 2.0808 | 2.0393 | 1.9992 | 1.96 | 1.9218 | 1.8843 | 1.8468 | 1.8108 | 1.7756 | 1.7421 |
| BiGRU Validation | 1.535 | 1.583 | 1.5567 | 1.585 | 1.5547 | 1.5457 | 1.5377 | 1.5408 | 1.5528 | 1.5382 | 1.5286 | 1.5104 | 1.5606 | 1.5651 | 1.5397 |
| LSTM Validation | 3.2345 | 3.0407 | 2.9361 | 3.1424 | 3.0947 | 3.0945 | 2.9482 | 3.0664 | 3.0946 | 3.0718 | 3.1143 | 3.2025 | 3.1872 | 2.9962 | 3.2633 |

*Figure 4.1.2 Graph for the Training and Validation loss of LSTM and BiGRU models*

Training loss is a measure of how well the model is able to minimize the error between its predicted values and the actual values in the training data. Referring to Figure 4.1.2, the trend is decreasing, which indicates that the model is learning and improving its ability to make predictions on the training data. The training loss for the BiGRU model starts at 3.0979, while the LSTM model starts at 4.8116. The final training loss for the BiGRU model is 1.2692, while the LSTM model is 1.7421. The validation loss for the BiGRU model starts at 2.3364, while the LSTM model starts at 4.5117. The final validation loss for the BiGRU model is 1.5397, while the LSTM model is 3.2633.

Overall, the BiGRU model is higher in training accuracy, validation accuracy, training loss, and validation loss when compared to LSTM. This denotes that it is better on learning patterns and relationships within the train data, predicts more accurately, and more effective in generalizing unseen data. Moreover, the values of the training and validation accuracy/loss of the BiGRU model is close to each other compared to the LSTM model. This signifies that the LSTM overfits compared to BiGRU.

# 4.2 BLEU Accuracy Metric Results

| Model | BLEU Score |
|---|---|
| **Sequence-to-sequence BiGRU model with an attention mechanism** | **0.5313** |

| | |
|---|---|
| LSTM | 0.3759 |

*Figure 4.2.1 BLEU Score results*

Figure 4.2.1 showcases the BLEU Score of the BiGRU model when compared to the LSTM model. It was shown on the table that the BLEU score of the BiGRU model is higher than the LSTM model, hence, this suggests that the BiGRU model outperformed the LSTM model in terms of accuracy. Moreover, this also means that the generated recipes of the BiGRU model matches the human-written recipes more closely than the LSTM model.

| **Model** | **BLEU score (Instructions)** | **BLEU score (Ingredients)** |
|---|---|---|
| **Sequence-to-sequence BiGRU model with an attention mechanism** | **0.1762** | **0.5086** |
| LSTM | 0.0812 | 0.1639 |

*Figure 4.2.2 BLEU Score results in regards to ingredients and instruction*

Figure 4.2.2 showcases the different BLEU score of the ingredients and instructions of the recipe. The results indicates that the BiGRU model outperformed the LSTM model. The ingredients shows a drastic improvement on BLEU score when compared to the instructions. The reason for the big difference in BLEU scores between ingredients and instructions lies in how recipes are structured. Ingredients are usually more straightforward and consistent, leading to fewer variations in different recipes. On the other hand, there are more room for changes in instructions since various techniques are included here. This leads to a lower BLEU score for instructions, highlighting their greater diversity and complexity, making them difficult to match precisely.

Although the BLEU score assess the quality of the machine generated text by comparing it to the reference data found in the dataset and considers the words used in the reference text as basis to score the generated text, this does not consider slight variations in the wording of the generated text leaving a room for error in terms of the creativity of the recipe. This is an issue when evaluating recipes since certain recipes have different techniques and words they might use that may be synonymical to each other.

Additionally, achieving a higher BLEU score does not always pertain to achieving a better generated recipe It pertains that the recipe generated was closer to their respective reference recipes which is an conflict of interest since the primary function of the research is to generate novel recipes that are something new that the model generated not a recipe found in the dataset.

# 4.3 Jaccard Similarity Scores

| Model | Jaccard Similarity Score |
|---|---|
| **Sequence-to-sequence BiGRU model with an attention mechanism** | **0.5860** |
| LSTM model | 0.0723 |

*Figure 4.3.1 Jaccard Similarity*

Figure 4.4.1 suggests that the BiGRU model demonstrated a higher level of coherence in the generated recipes compared to the LSTM model. The higher Jaccard Similarity Score of the BiGRU model indicates that it was more effective in creating recipes where the ingredients and instructions were more closely aligned or similar, reflecting better internal consistency in the generated recipes.

# 4.4 Human Evaluation Results

| | **0** | **1** | **2** | **3** | **4** | **5** | **Average Score** |
|---|---|---|---|---|---|---|---|
| **Clarity** | 0 | 0 | 0 | 3 | 6 | **19** | **4.57** |
| **Creativity/ Novelty** | 0 | 0 | 0 | 3 | 8 | **17** | **4.5** |
| **Suitability** | 0 | 0 | 0 | 2 | 4 | **22** | **4.71** |
| **Doability and Edibility** | 0 | 1 | 0 | 2 | 4 | **21** | **4.57** |
| **Likelihood to try** | 0 | 0 | 0 | 6 | 3 | **19** | **4.46** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Overall Quality** | 0 | 0 | 0 | 2 | 8 | **18** | **4.57** |

*Figure 4.4.1 Human Evaluation Results*

Figure 4.4.3 displays the results of the human evaluators' response to the study's recipe generating model involving 28 participants. Originally, the study received responses from 49 participants however, it was filtered to only include respondents that have at least 1 year of cooking experience. The human evaluators rated the model's outputs as clear (4.57), suggesting that the recipes were easy to follow and aligned well with the provided ingredients. The scores for creativity/novelty (4.5) and doability/edibility (4.57) indicate a slightly lower, yet still strong performance in these areas. The overall quality of the recipes was rated highly as well, with a score of around 4.57.

The highest average score in the evaluation was for suitability, with an average score of 4.71, indicating that the generated recipes align very well with the provided ingredients. This suggests that the algorithm generating the recipes is effective at incorporating user-inputted ingredients into the recipes in a way that makes sense to the evaluators.

```
User input: cocoa powder,milk,butter,egg,vanilla,sugar,flour,baking powder
Recipe Output: {
    "ingredients": [
        "1/2 cup cocoa powder",
        "1/2 cup milk",
        "1/2 cup butter",
        "1 egg",
        "1 teaspoon vanilla extract",
        "1 cup sugar",
        "1 cup all-purpose flour",
        "1 teaspoon baking powder"
    ],
    "instructions": [
        "Preheat oven to 350 degrees F (175 degrees C). Grease a 9x13-inch baking dish.",
        "Mix cocoa powder, milk, butter, egg, and vanilla extract together in a bowl; stir in flour until smooth. Pour batter into prepared baking dish.",
        "Bake in the preheated oven until a toothpick inserted into the center comes out clean, about 1 hour. Cool in the pan for 10 minutes before removing to cool
        completely on a wire rack."
    ],
    "ner_tags": [],
    "title": "Chocolate-Chocolate Cake"
}
Comments: I got CHOCOLATE-CHOCOLATE CAKE, two chocolates means double chocolatey goodness for me!!^_^ Number of instructions are quite limited (and may need to improve
to consider preparation details) but they are straightforward to follow though. But anyways, this is a cool recipe generator to use if one has limited ingredients and
don't have any idea what recipe to follow. All in all, a very good project.
```

*Figure 4.4.2 Lowest Rater response with comments for Likelihood to Try criterion*

On the other hand, the lowest average score, albeit still quite high, is for the evaluators' likelihood to try at 4.46. Although this score indicates a positive reception, it is marginally lower than the other criteria. To further explore this, figure 4.4.2 displays the ingredients inputted and recipe output received by one of the lowest raters of said criterion. As provided by the study's limitations, the recipe output is greatly impacted by the quality of the ingredients inputted by the

user. If the combination of ingredients does not make sense, there is a great chance that the model will produce non-sensical recipe outputs as the model will try to incorporate all given ingredients in the output.

Furthermore, the concentration of scores between 4 and 5 for all criteria suggests that the generated recipes were well-received, and participants found them to be clear, suitable, and appealing enough to consider trying.

### 4.4.1 Fleiss' Kappa Score

The human evaluation data returned with a Fleiss' Kappa score of 0.4253 which indicates that there is a moderate agreement between the evaluators. This degree of agreement implies a fair degree of consistency in the evaluators' assessments, highlighting both areas of concordance and potential areas for improvement.

## 4.5 BiGRU and LSTM output recipes

In this section, the researchers showcased certain examples of the recipe generated by the BiGRU compared to the recipe generated by the LSTM model. To give a clear indication of the test results. The specific ners were chosen at random and were inputted in both of the models. Below is the result of those generated recipes with those NER's.

| **NERs:** cream cheese, vanilla, butter, icing sugar | |
|---|---|
| **LSTM MODEL** | **BiGRU MODEL** |
| Title: Bacon and beans And Cilantro And Sh Salad<br><br>INGREDIENTS:<br>● 1/4 cup butter,<br>● melted butter<br>● 1 1/2 cups ice cream<br><br>INSTRUCTIONS: | Title: Cream Cheese Frosting<br><br>INGREDIENTS:<br>● 8 ounces cream cheese, softened<br>● 1 teaspoon vanilla extract<br>● 1/2 cup butter, softened<br>● 1/2 cup icing sugar<br><br>INSTRUCTIONS: |

| | |
|---|---|
| 1. Preheat oven to 350\\u00b0F '<br>2. In a mixing boil and cook oven to low heat for 15 minutes<br>3. until done<br>4. about 1/2 more lbs should be round<br>5. Serve with pineapple: during Pecan/4 cupfulsgue' | 1. Beat cream cheese and vanilla together in a bowl until smooth.<br>2. Gradually add icing sugar, beating until smooth.<br>3. Add icing sugar and beat until smooth.<br>4. Add icing sugar and beat until smooth.<br>5. Serve with a dollop of whipped cream. |

*Figure 4.5.1 Sample Recipe Results #1*

| **NERs:** chicken broth, chicken breasts, butter, flour, mustard, tarragon, salt | |
|---|---|
| **LSTM MODEL** | **BiGRU MODEL** |
| Title: Curried Chickenoni Cheese<br><br>INGREDIENTS:<br>● 2 tablespoons cornmeal<br>● pickled stick noodles (optional)<br>● 1 pint extract<br>● 1 (2 ounce) can crushed<br>● 1/4 cup gorgonzola or your frozen best<br><br>INSTRUCTIONS:<br>1. Melt butter on a mixing bowl or wok butter in skillet in a large bowl<br>2. mix fat, feta cheese, salt, garlic, and salt<br>3. Beat onion, chili powder, oregano and salt and return to a boil.<br>4. Stir constantly, dry together and repeat together and mixing paste mixture until batter for 30 minutes.<br>5. Stir in all of of batter. Sprinkle milk evenly over 1/4 hours within hot<br>6. Spoon potato onto pan on center.<br>7. You potatoes on top. | Title: Tarragon Chicken<br><br>INGREDIENTS:<br>● 1/2 cup chicken broth<br>● 1/4 cup flour<br>● 1/4 cup butter<br>● 1/4 cup Dijon mustard<br>● 2 boneless skinless chicken breasts<br>● 1/2 teaspoon dried tarragon<br>● salt and pepper<br><br>INSTRUCTIONS:<br>1. In a large saucepan, combine chicken broth, flour, butter, and mustard.<br>2. Bring to a boil, then reduce heat and simmer for 10 minutes.<br>3. Add chicken and tarragon and cook for 1 minute.<br>4. Season with salt and pepper.<br>5. Serve chicken and sauce over chicken. |

| | |
|---|---|
| 8. Bake at least full hours and place until absorbed is bubbly and almost criss. | |

*Figure 4.5.2 Sample Recipe Results #2*

| **NERs:** honey, mint, basil | |
|---|---|
| **LSTM MODEL** | **BiGRU MODEL** |
| Title: Jo's peeled<br><br>INGREDIENTS:<br>● 3 cups light bacon<br>● melted grill in a<br>● blender and broil 30-35* added<br>● or frequently and cut until tender to the paper<br>● about 3 medium - 5 minutes<br>● nectarines lightly browned<br>● stirring, orange<br>● (about 4 roll wedges as apple recipe you -- fine dough as.)<br><br>INSTRUCTIONS:<br>1. None | Title: Honey Mint Pesto<br><br>INGREDIENTS:<br>● 1/2 cup honey<br>● 1/4 cup of fresh mint, chopped<br>● 1/4 cup basil leaves, chopped<br><br>INSTRUCTIONS:<br>1. Mix honey, mint, and basil together in a bowl.<br>2. Pour into a small bowl and stir in basil.<br>3. Serve with a straw. |

*Figure 4.5.3 Sample Recipe Results #3*

In the figures shown above, the LSTM model produced unreliable recipe results. First is it contains incoherent relationship between ingredients and instructions as there are ingredients can't be found in the instructions. Second, it did not correctly include the ner's that were supposed to be included in the ingredients. Third, the instructions has grammar and spelling errors. Fourth, it also showcased a misleading title that does not correlate with the recipe's context. Lastly, there are certain instances where the LSTM model would just output none on a certain area or include the instruction on the ingredients column, as can be seen in Figure 4.5.3.

Though the BiGRU model showed a much better overall recipe structure, including a great showcase of coherent ingredients and instruction relationship, there are still factors that can be improved such as repeating instructions, as shown in figure 4.5.1.

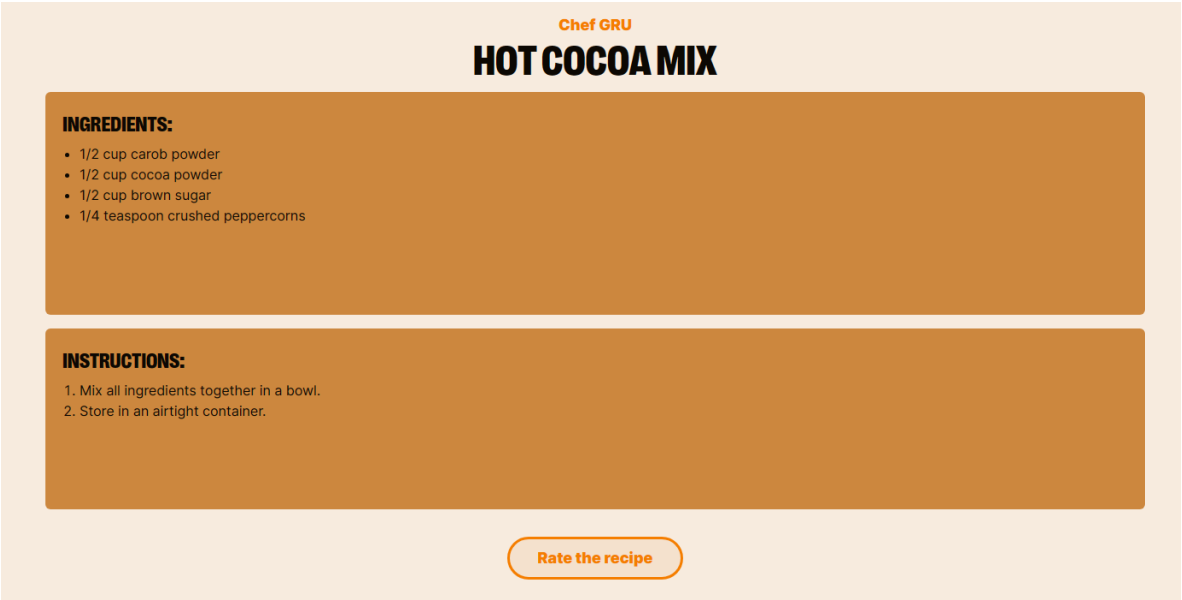



*Figure 4.5.4 BiGRU Recipe Result (Website) #1*

*Figure 4.5.5 BiGRU Recipe Result (Website) #2*

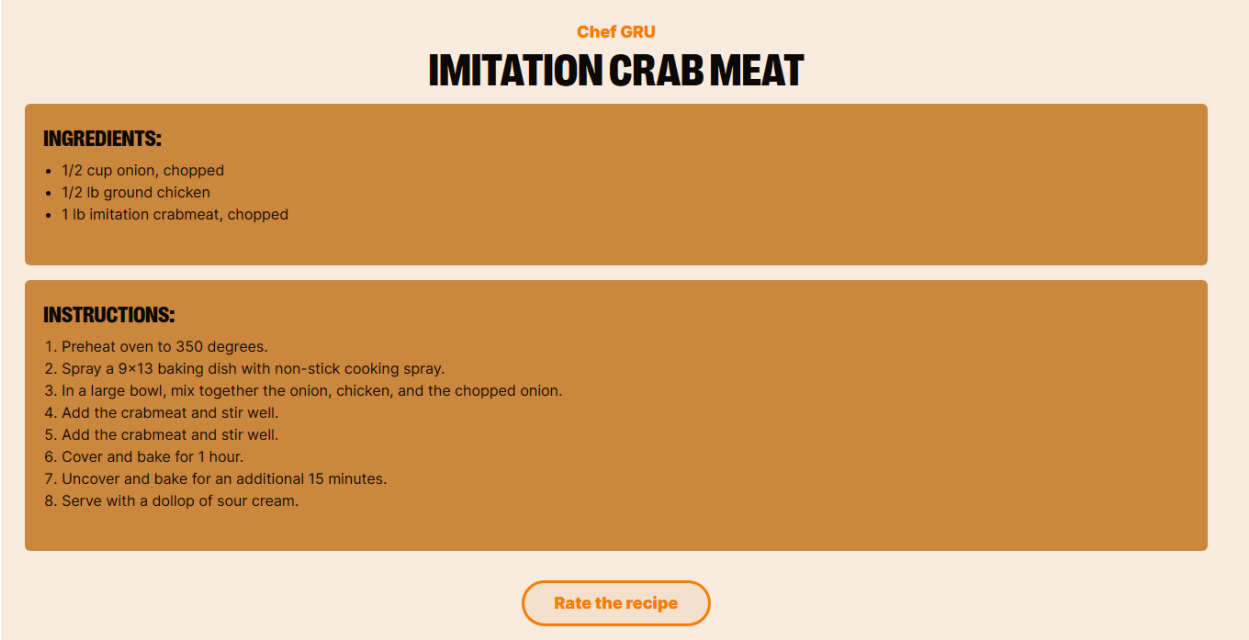



*Figure 4.5.6 BiGRU Recipe Result (Website) #3*

In the figures shown above, it can be seen that the recipes generated have certain flaws within them. This includes certain recipes not having a title, refer to figure 4.6.2. Specific instructions being duplicated jeopardizing the recipe's overall structure which is found in Figure 4.6.3 as the "Add the crabmeat and stir well" was repeated. In figure 4.6.3, there's an ingredient such as the sour cream that is present on the instructions but is not on the ingredients section. This indicates that there are some inconsistencies in the instructions and ingredients.

# Chapter 5

# Conclusion and Recommendation

## 5.1 Conclusion

The Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model is better than the LSTM model in terms of accuracy when measured by the BLEU accuracy metric. In terms of the coherence between the ingredients and instructions measured by Jaccard Similarity, the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism model outperforms the LSTM model. As for the quality of the recipe generated by the BiGRU model, it was concluded that most of the recipes were coherent, creative, novel, suitable, doable, edible, and most likely to try. This proves that the Sequence-to-Sequence Bidirectional GRU with an Attention Mechanism is a suitable model for generating novel food recipes. Although the BiGRU model performed great, there were still problems that were discovered while conducting the study. First is that it was discovered that the BLEU accuracy metric may not be the most suitable metric to use in measuring the accuracy of a generated recipe as it primarily looks at word overlap and doesn't consider subtle phrasing differences. Second are the problems on the recipes generated by the BiGRU model, such as, random ingredients appearing on the instructions, repeating instructions, and inconsistent title. Lastly, the computational resource used by the BiGRU is high considering that only a portion of the RecipeNLG dataset was used.

## 5.2 Recommendation

.As for the future researchers, it is advised to use a transformer-based models as it might have the capability to outperform RNN models. Next is to utilize the whole RecipeNLG dataset. Lastly, the use of a better evaluation metrics like Word2Vec which takes the word similarties into account.

# Appendix A

# Libraries

- **TensorFlow** - is an open-source machine learning framework developed by Google. It provides a comprehensive set of tools for building and deploying machine learning models, especially deep learning models.
- **TensorFlow-GPU** - is a version of TensorFlow optimized for GPU (Graphics Processing Unit) acceleration. It leverages the parallel processing capabilities of GPUs to speed up the training and execution of machine learning models.
- **Matplotlib** - is a 2D plotting library for creating static, animated, and interactive visualizations in Python. It is widely used for generating charts, graphs, histograms, and other types of plots.
- **Keras** - is a high-level neural networks API written in Python. It serves as an interface for building, training, and deploying machine learning models, particularly neural networks.
- **Tokenizers** - is a library that offers efficient tools for text data tokenization, a critical process in breaking down text into meaningful units.
- **Pickle** - is a module in Python used for serializing and deserializing Python objects. It allows objects to be saved to a file and loaded later, preserving their structure and state.
- **Pandas** - is a data manipulation library for Python. It provides data structures such as Series and DataFrame, along with functions for data cleaning, exploration, and analysis.
- **NumPy** - is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.

- **Scikit-learn** - is a machine learning library in Python that provides simple and efficient tools for data mining and data analysis. It includes various algorithms for classification, regression, clustering, and more.

- **BeautifulSoup** - is a library for pulling data out of HTML and XML files. It provides Pythonic idioms for iterating, searching, and modifying the parse tree, making it easy to scrape data from web pages.

- **Selenium** - is a web testing library that allows automating browser actions. It is often used for web scraping and testing web applications by simulating user interactions with a web page.

- **NLTK** - is a powerful library for working with human language data. It provides tools that can be used on processes such as text processing, classification, tokenization. But it's most important use is that it consists of libraries that could help in evaluation metrics such as BLEU and Jaccard Similarity

# Appendix B

# Dataset

The dataset we used for this study is the dataset made by Bien et al on their study on recipe generation, they also called this dataset RecipeNLG. The dataset amounts to about 2 Million Recipes the dataset consists of different columns title, ingredients, directions, link, source, NER. After preprocessing this dataset it led us to around 165,498 recipes.

**RecipeNLG Dataset:**

*Figure A.B.1 RecipeNLG Dataset*

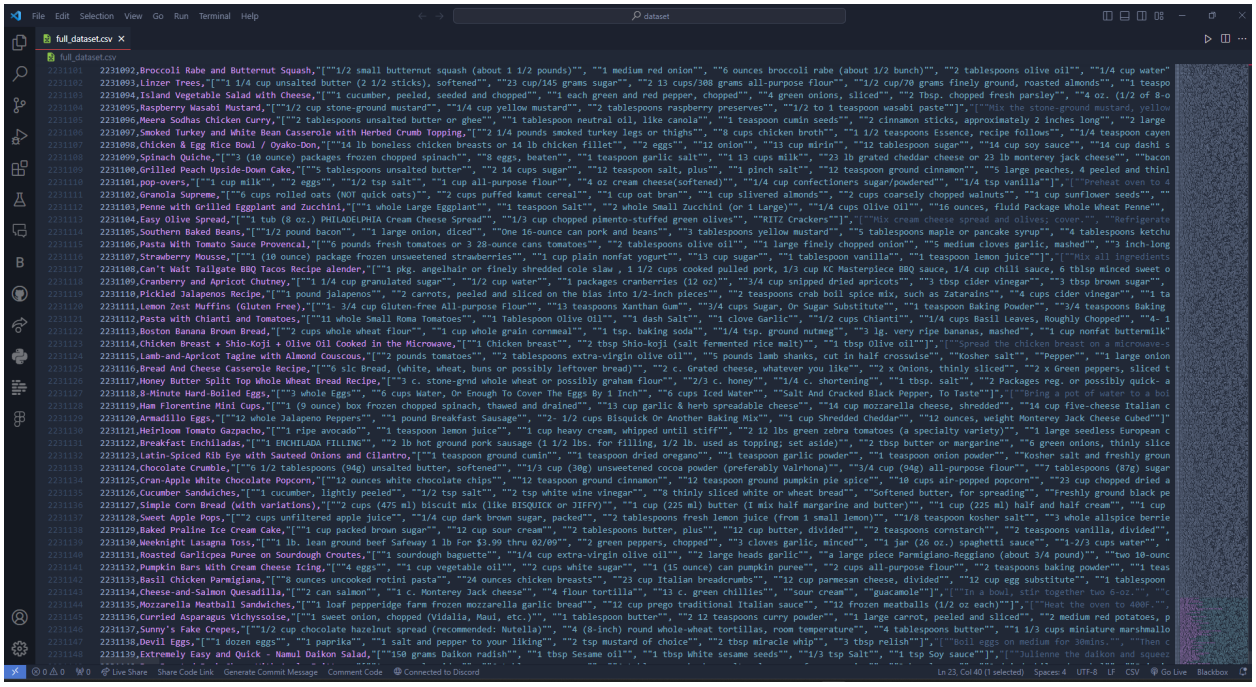


*Figure A.B.2 RecipeNLG Dataset #2*

# Appendix C

# Website

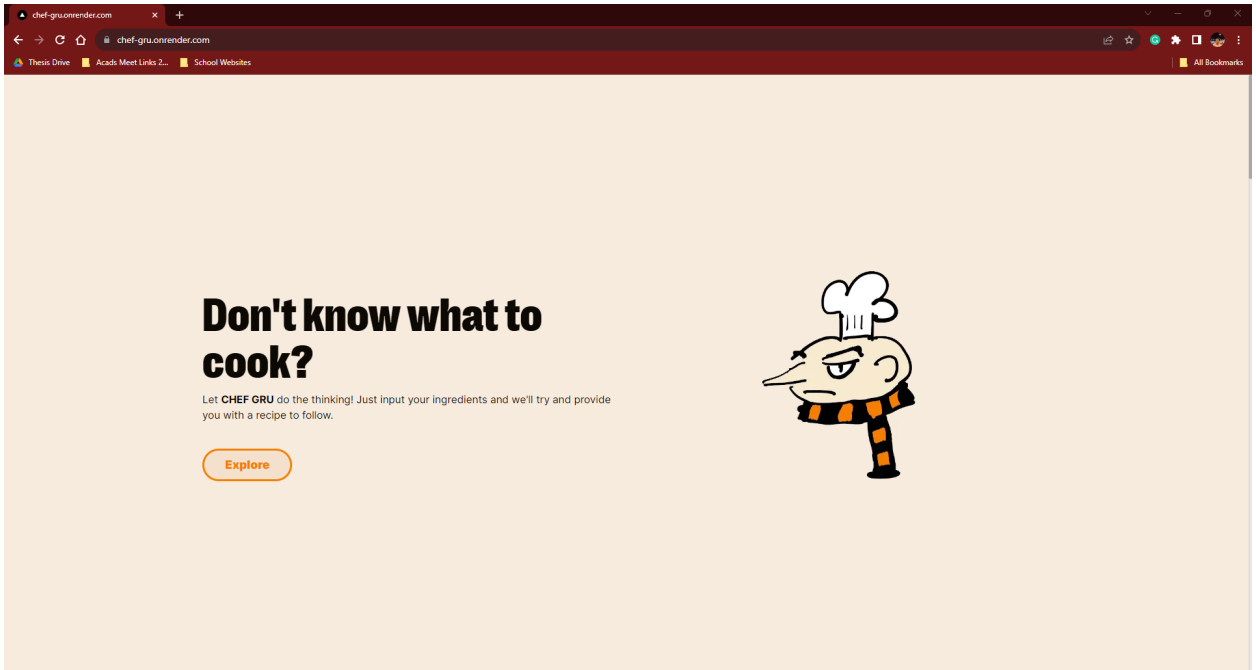



*Figure A.C.1 Landing Page Section*

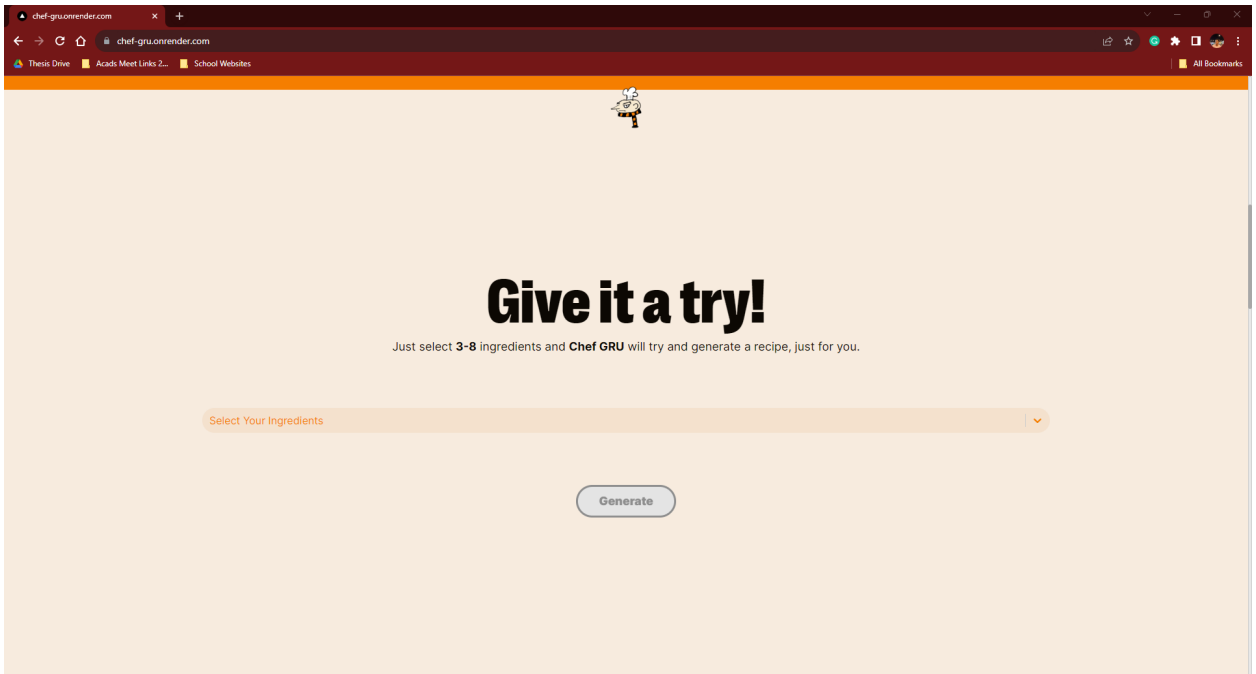



*Figure A.C.2 NERs selection section*

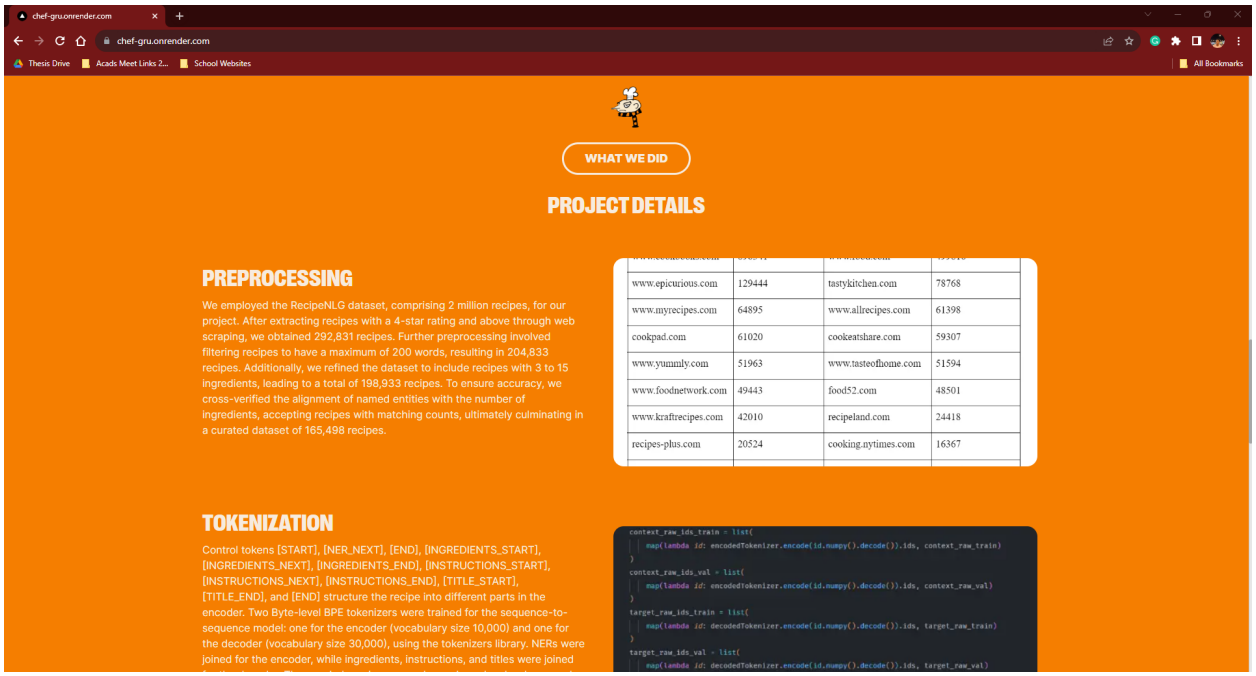



*Figure A.C.3 Project Details Section*

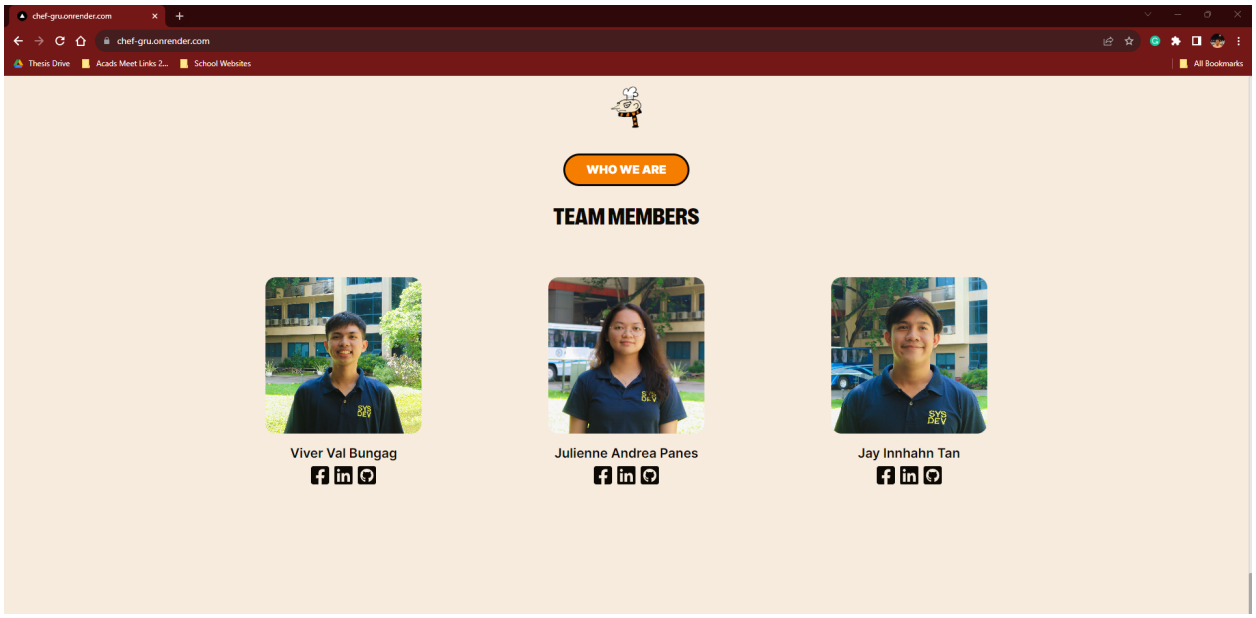



*Figure A.C.4 Project Members Section*

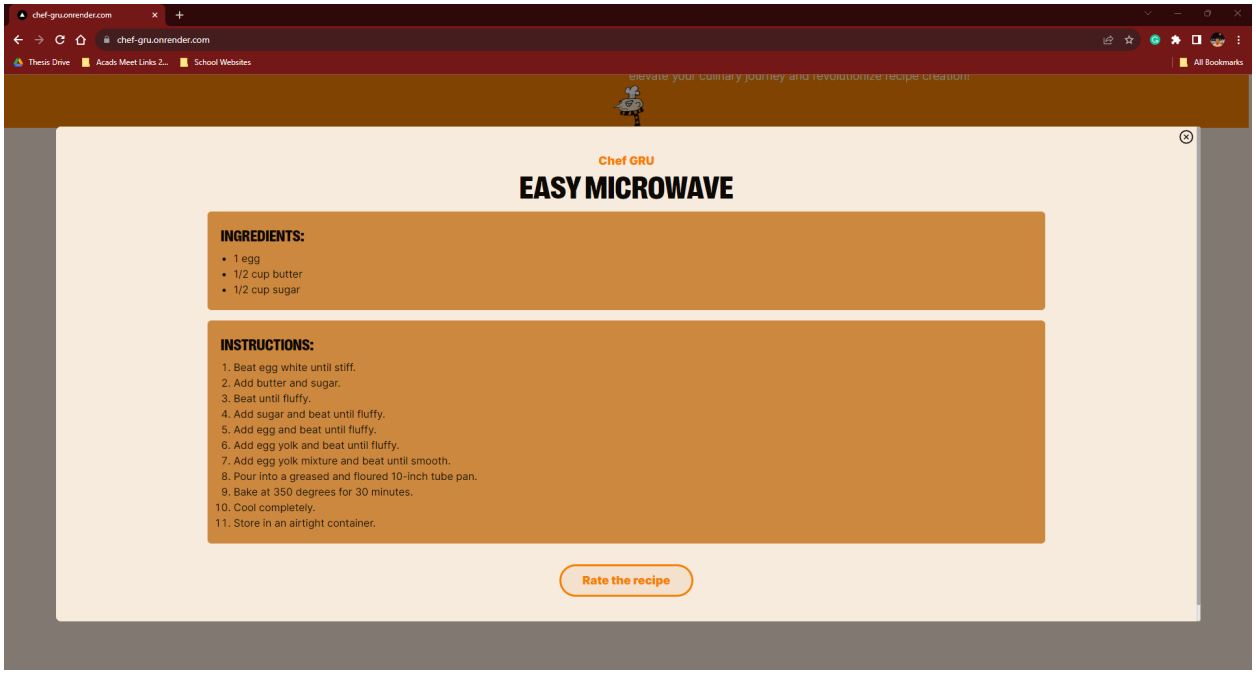



*Figure A.C.5 Recipe Generation Pop Up Modal*

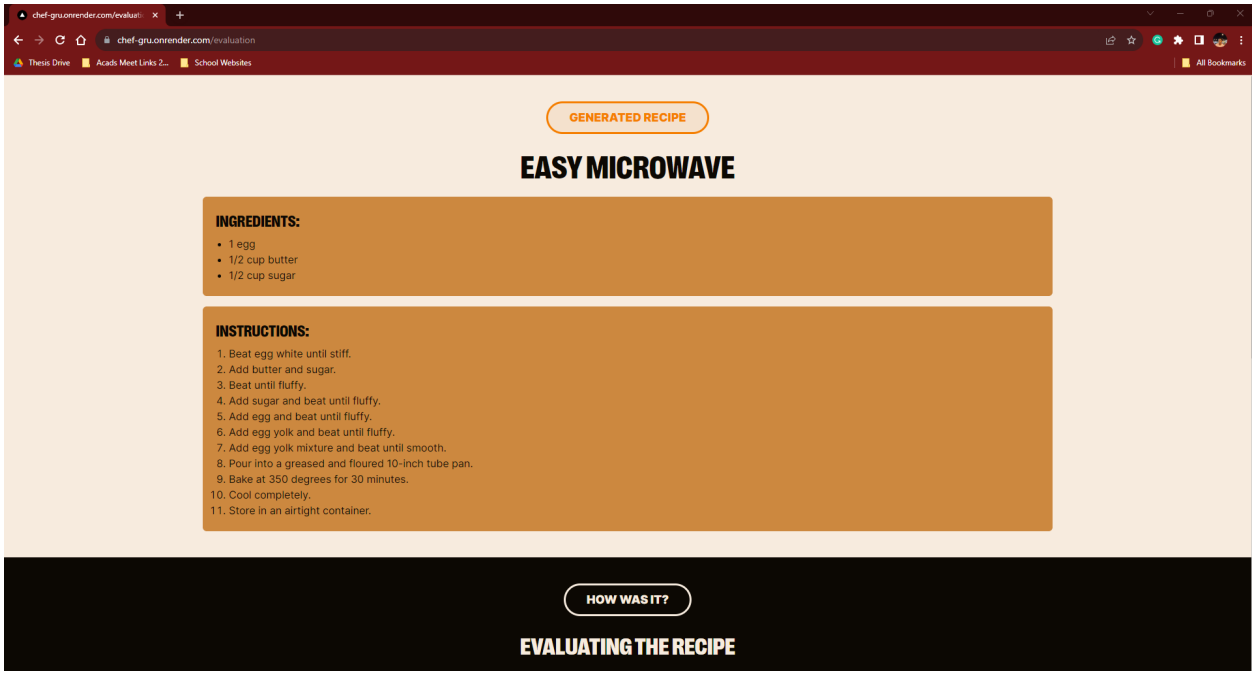



*Figure A.C.6 Generated Recipe Section*

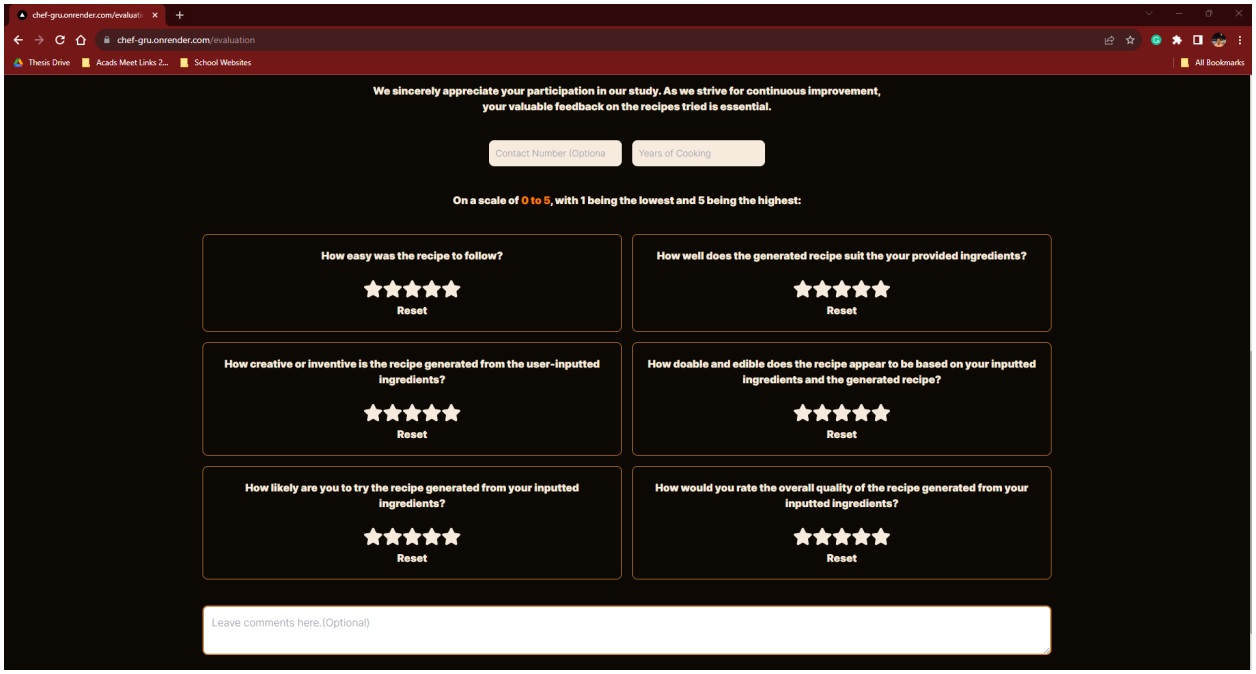



*Figure A.C.7 Evaluation Section*

# Glossary

**Named Entity Recognition (NER)** serves as the fundamental tool for identifying and classifying entities within a text in the context of the Recipe Generation it's the most basic form of an ingredient.

**GRU (Gated Recurrent Unit):** A type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem, allowing for more effective modeling of sequential data.

**LSTM (Long Short-Term Memory):** Another type of RNN architecture with enhanced memory cells, facilitating the learning of long-term dependencies in sequential data.

**Adam:** An optimization algorithm used for training deep neural networks,

**Learning Rate:** A hyperparameter that determines the step size during the training of a neural network, influencing the convergence and performance of the model.

**Sequence-to-sequence architecture:** A neural network architecture designed for tasks involving input and output sequences, commonly used in machine translation and text summarization.

**BLEU Score:** A metric for evaluating the quality of machine-generated text by comparing it to one or more reference texts, measuring the precision of n-grams.

**Jaccard Similarity:** A measure of similarity between two sets, calculated as the size of their intersection divided by the size of their union, often used in recipe generation tasks to asses the coherence between ingredients and instructions.

**Attention Model:** A mechanism in neural networks that focuses on specific parts of input data when making predictions, allowing the model to weigh different elements differently during processing.

**BPE Tokenizer (Byte Pair Encoding Tokenizer):** A tokenization method that iteratively merges the most frequent pairs of consecutive tokens, effectively capturing both common and rare patterns in the data.

# Bibliography


[1] M. Goel *et al.*, "Ratatouille: A tool for Novel Recipe Generation," in *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*, IEEE, 2022, pp. 107–110.

[2] E. D. Liddy, "Natural language processing," 2001.

[3] P. K. Jain, V. Saravanan, and R. Pamula, "A hybrid CNN-LSTM: A deep learning approach for consumer sentiment analysis using qualitative user-generated contents," *Trans. Asian Low-Resour. Lang. Inf. Process.*, vol. 20, no. 5, pp. 1–15, 2021.

[4] J. Liu, Y. Yang, S. Lv, J. Wang, and H. Chen, "Attention-based BiGRU-CNN for Chinese question classification," *J. Ambient Intell. Humaniz. Comput.*, pp. 1–12, 2019.

[5] S. Mangal, P. Joshi, and R. Modak, "LSTM vs. GRU vs. Bidirectional RNN for script generation," *ArXiv Prepr. ArXiv190804332*, 2019.

[6] S. Zargar, "Introduction to sequence learning models: RNN, LSTM, GRU," *Dep. Mech. Aerosp. Eng. N. C. State Univ. Raleigh N. C.*, vol. 27606, 2021.

[7] M. R. Parvez, S. Chakraborty, B. Ray, and K.-W. Chang, "Building Language Models for Text with Named Entities." 2018.

[8] B. P. Majumder, S. Li, J. Ni, and J. McAuley, "Generating personalized recipes from historical user preferences," *ArXiv Prepr. ArXiv190900105*, 2019.

[9] S. Yan, J. Chai, and L. Wu, "Bidirectional GRU with multi-head attention for Chinese NER," in *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, IEEE, 2020, pp. 1160–1164.

[10] R. Ni and H. Cao, "Sentiment Analysis based on GloVe and LSTM-GRU," in *2020 39th Chinese control conference (CCC)*, IEEE, 2020, pp. 7492–7497.

[11] S. Yang, X. Yu, and Y. Zhou, "Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example," in *2020 International workshop on electronic communication and artificial intelligence (IWECAI)*, IEEE, 2020, pp. 98–101.

[12] S. Santhanam, "Context based text-generation using lstm networks," *ArXiv Prepr. ArXiv200500048*, 2020.

[13] J. Dai, C. Chen, and Y. Li, "A backdoor attack against lstm-based text classification systems," *IEEE Access*, vol. 7, pp. 138872–138878, 2019.

[14] A. Ezen-Can, "A Comparison of LSTM and BERT for Small Corpus," *ArXiv Prepr. ArXiv200905451*, 2020.

[15] F. Qian and X. Chen, "Stock prediction based on LSTM under different stability," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, IEEE, 2019, pp. 483–486.

[16] J. W. Chen, X. K. Sigalingging, J.-S. Leu, and J.-I. Takada, "Applying a hybrid sequential model to Chinese sentence correction," *Symmetry*, vol. 12, no. 12, p. 1939, 2020.

[17] T. Iqbal and S. Qureshi, "The survey: Text generation models in deep learning," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 6, pp. 2515–2528, 2022.

[18] Y. Deng, L. Wang, H. Jia, X. Tong, and F. Li, "A sequence-to-sequence deep learning architecture based on bidirectional GRU for type recognition and time location of combined power quality disturbance," *IEEE Trans. Ind. Inform.*, vol. 15, no. 8, pp. 4481–4493, 2019.

[19] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *ArXiv Prepr. ArXiv14061078*, 2014.

[20] S. Viswanathan, M. Anand Kumar, and K. Soman, "A sequence-based machine comprehension modeling using LSTM and GRU," in *Emerging Research in Electronics, Computer Science and Technology: Proceedings of International Conference, ICERECT 2018*, Springer, 2019, pp. 47–55.

[21] L. Huang, S. Zhuang, and K. Wang, "A Text Normalization Method for Speech Synthesis Based on Local Attention Mechanism," *IEEE Access*, vol. 8, pp. 36202–36209, 2020, doi: 10.1109/ACCESS.2020.2974674.

[22] M. Zulqarnain, R. Ghazali, M. G. Ghouse, and M. F. Mushtaq, "Efficient processing of GRU based on word embedding for text classification," *JOIV Int. J. Inform. Vis.*, vol. 3, no. 4, pp. 377–383, 2019.

[23] W. Ali, Y. Yang, X. Qiu, Y. Ke, and Y. Wang, "Aspect-level sentiment analysis based on bidirectional-GRU in SIoT," *IEEE Access*, vol. 9, pp. 69938–69950, 2021.

[24] M.-Y. Cheng, D. Kusoemo, and R. A. Gosno, "Text mining-based construction site accident classification using hybrid supervised machine learning," *Autom. Constr.*, vol. 118, p. 103265, 2020.

[25] P. Shalu and M. Meera, "Neural machine translation for english to hindi using gru," *Available SSRN 3851323*, 2021.

[26] L. Li and T. Zhang, "Research on text generation based on LSTM," *Int Core J Eng*, vol. 7, no. 5, pp. 525–535, 2021.

[27] H. Bo, S. H. Ding, B. Fung, and F. Iqbal, "ER-AE: Differentially private text generation for authorship anonymization," *ArXiv Prepr. ArXiv190708736*, 2019.

[28] R. Khan, M. S. Islam, K. Kanwal, M. Iqbal, M. I. Hossain, and Z. Ye, "A deep neural framework for image caption generation using gru-based attention mechanism," *ArXiv Prepr. ArXiv220301594*, 2022.

[29] L. Logeswaran, H. Lee, and S. Bengio, "Content preserving text generation with attribute controls," *Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[30] J. Chen, D. Jiang, and Y. Zhang, "A hierarchical bidirectional GRU model with attention for EEG-based emotion classification," *IEEE Access*, vol. 7, pp. 118530–118540, 2019.

[31] A. Kumar, S. Abirami, T. E. Trueman, and E. Cambria, "Comment toxicity detection via a multichannel convolutional bidirectional gated recurrent unit," *Neurocomputing*, vol. 441, pp. 272–278, 2021.

[32] S. Singh and A. Mahmood, "The NLP cookbook: modern recipes for transformer based deep learning architectures," *IEEE Access*, vol. 9, pp. 68675–68702, 2021.

[33] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 27, 2014.

[34] A. K. M. Masum, S. Abujar, M. A. I. Talukder, A. S. A. Rabby, and S. A. Hossain, "Abstractive method of text summarization with sequence to sequence RNNs," in *2019 10th international conference on computing, communication and networking technologies (ICCCNT)*, IEEE, 2019, pp. 1–5.

[35] X. Zhang, M. H. Chen, and Y. Qin, "Nlp-qa framework based on lstm-rnn," in *2018 2nd International Conference on Data Science and Business Analytics (ICDSBA)*, IEEE, 2018, pp. 307–311.

[36] N. N. Khin and K. M. Soe, "Question answering based university chatbot using sequence to sequence model," in *2020 23rd Conference of the Oriental COCOSDA International Committee for the Co-ordination and Standardisation of Speech Databases and Assessment Techniques (O-COCOSDA)*, IEEE, 2020, pp. 55–59.

[37] R. Koncel-Kedziorski, D. Bekal, Y. Luan, M. Lapata, and H. Hajishirzi, "Text generation from knowledge graphs with graph transformers," *ArXiv Prepr. ArXiv190402342*, 2019.

[38] J. Liu, S. Chen, B. Wang, J. Zhang, N. Li, and T. Xu, "Attention as relation: learning supervised multi-head self-attention for relation extraction," in *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, 2021, pp. 3787–3793.

[39] E. Egonmwan and Y. Chali, "Transformer and seq2seq model for paraphrase generation," in *Proceedings of the 3rd Workshop on Neural Generation and Translation*, 2019, pp. 249–255.

[40] M. Domingo, M. García-Martínez, A. Helle, F. Casacuberta, and M. Herranz, "How much does tokenization affect neural machine translation?," in *International Conference on Computational Linguistics and Intelligent Text Processing*, Springer, 2019, pp. 545–554.

[41] S. Yu, J. Song, H. Kim, S. Lee, W.-J. Ryu, and S. Yoon, "Rare tokens degenerate all tokens: Improving neural text generation via adaptive gradient gating for rare token embeddings," *ArXiv Prepr. ArXiv210903127*, 2021.

[42] I. Gusev, "Dataset for automatic summarization of Russian news," in *Artificial Intelligence and Natural Language: 9th Conference, AINL 2020, Helsinki, Finland, October 7–9, 2020, Proceedings 9*, Springer, 2020, pp. 122–134.

[43] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *ArXiv Prepr. ArXiv150807909*, 2015.

[44] D. Gowda, A. Garg, K. Kim, M. Kumar, and C. Kim, "Multi-Task Multi-Resolution Char-to-BPE Cross-Attention Decoder for End-to-End Speech Recognition.," in *Interspeech*, 2019, pp. 2783–2787.

[45] H. H. Lee *et al.*, "RecipeGPT: Generative pre-training based cooking recipe generation and evaluation system," in *Companion Proceedings of the Web Conference 2020*, 2020, pp. 181–184.

[46] B. Wang, A. Wang, F. Chen, Y. Wang, and C.-C. J. Kuo, "Evaluating word embedding models: Methods and experimental results," *APSIPA Trans. Signal Inf. Process.*, vol. 8, p. e19, 2019.

[47] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv Prepr. ArXiv14090473*, 2014.

[48] Y. Li, L. Yang, B. Xu, J. Wang, and H. Lin, "Improving user attribute classification with text and social network attention," *Cogn. Comput.*, vol. 11, pp. 459–468, 2019.

[49] G. Liu and J. Guo, "Bidirectional LSTM with attention mechanism and convolutional layer for text classification," *Neurocomputing*, vol. 337, pp. 325–338, 2019.

[50] Y. Li, T. Shen, G. Long, J. Jiang, T. Zhou, and C. Zhang, "Improving long-tail relation extraction with collaborating relation-augmented attention," *ArXiv Prepr. ArXiv201003773*, 2020.

[51] F. Long, K. Zhou, and W. Ou, "Sentiment Analysis of Text Based on Bidirectional LSTM With Multi-Head Attention," *IEEE Access*, vol. 7, pp. 141960–141969, 2019, doi: 10.1109/ACCESS.2019.2942614.

[52] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021, doi: https://doi.org/10.1016/j.neucom.2021.03.091.

[53] M. India, P. Safari, and J. Hernando, "Self Multi-Head Attention for Speaker Recognition." 2019.

[54] Y. Cheng *et al.*, "Sentiment Analysis Using Multi-Head Attention Capsules With Multi-Channel CNN and Bidirectional GRU," *IEEE Access*, vol. 9, pp. 60383–60395, 2021, doi: 10.1109/ACCESS.2021.3073988.

[55] R. Yi and W. Hu, "Pre-Trained BERT-GRU Model for Relation Extraction," in *Proceedings of the 2019 8th International Conference on Computing and Pattern Recognition*, in ICCPR '19. New York, NY, USA: Association for Computing Machinery, 2020, pp. 453–457. doi: 10.1145/3373509.3373533.

[56] A. Oppermann, "Regularization in deep learning—L1, L2, and dropout," *Data Sci.*, vol. 19, 2020.

[57] A. Vaswani *et al.*, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[58] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *ArXiv Prepr. ArXiv160706450*, 2016.

[59] S. Shen, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, "Powernorm: Rethinking batch normalization in transformers," in *International Conference on Machine Learning*, PMLR, 2020, pp. 8741–8751.

[60] W. Rahman *et al.*, "Integrating multimodal information in large pretrained transformers," in *Proceedings of the conference. Association for Computational Linguistics. Meeting*, NIH Public Access, 2020, p. 2359.

[61] R. Zaheer and H. Shaziya, "A study of the optimization algorithms in deep learning," in *2019 third international conference on inventive systems and control (ICISC)*, IEEE, 2019, pp. 536–539.

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ArXiv Prepr. ArXiv14126980*, 2014.

[63] A. Salvador, M. Drozdzal, X. Giró-i-Nieto, and A. Romero, "Inverse cooking: Recipe generation from food images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10453–10462.

[64] D. P. Papadopoulos, E. Mora, N. Chepurko, K. W. Huang, F. Ofli, and A. Torralba, "Learning program representations for food images and cooking recipes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16559–16569.

[65] H. Wang, G. Lin, S. C. Hoi, and C. Miao, "Structure-aware generation network for recipe generation from images," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII 16*, Springer, 2020, pp. 359–374.

[66] A. R. Desai, S. Goel, T. Karennavar, and P. Kanwal, "Instant Recipe Generation from Food Images," in *Computational Vision and Bio-Inspired Computing: Proceedings of ICCVBIC 2021*, Springer, 2022, pp. 721–735.

[67] A. Channam, B. R. Swarup, and S. G. Rao, "Extraction of Recipes from Food Images by Using CNN Algorithm," in *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, IEEE, 2021, pp. 1308–1315.

[68] A. Mane, M. Bawankar, M. Goradia, and S. Chaudhari, "Image to Recipe and Nutritional Value Generator Using Deep Learning," in *Inventive Communication and Computational Technologies: Proceedings of ICICCT 2022*, Springer, 2022, pp. 731–748.

[69] M. De Clercq, M. Stock, B. De Baets, and W. Waegeman, "Data-driven recipe completion using machine learning methods," *Trends Food Sci. Technol.*, vol. 49, pp. 1–13,

2016.

[70] M. Bień, M. Gilski, M. Maciejewska, W. Taisner, D. Wisniewski, and A. Lawrynowicz, "RecipeNLG: A Cooking Recipes Dataset for Semi-Structured Text Generation," in *Proceedings of the 13th International Conference on Natural Language Generation*, B. Davis, Y. Graham, J. Kelleher, and Y. Sripada, Eds., Dublin, Ireland: Association for Computational Linguistics, Dec. 2020, pp. 22–28. [Online]. Available: https://aclanthology.org/2020.inlg-1.4

[71] J. D. McDonald and M. Levine-Clark, *Encyclopedia of library and information sciences*. CRC Press, 2017.

[72] J. J. Webster and C. Kit, "Tokenization as the initial phase in NLP," in *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.

[73] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

[74] C. Khanna, "Word, Subword, and Character-Based Tokenization: Know the Difference — towardsdatascience.com." Jan. 06, 2022. [Online]. Available: https://towardsdatascience.com/word-subword-and-character-based-tokenization-know-the-difference-ea0976b64e17

[75] C. D. Manning, *An introduction to information retrieval*. Cambridge university press, 2009.

[76] C. Wang, K. Cho, and J. Gu, "Neural machine translation with byte-level subwords," in *Proceedings of the AAAI conference on artificial intelligence*, 2020, pp. 9154–9160.

[77] B. Mehlig, *Machine learning with neural networks: an introduction for scientists and engineers*. Cambridge University Press, 2021.

[78] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into deep learning*. Cambridge University Press, 2021.

[79] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 1017–1024.

[80] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, Pmlr, 2013, pp. 1310–1318.

[81] D. Pawade, A. Sakhapara, M. Jain, N. Jain, and K. Gada, "Story scrambler-automatic text generation using word level RNN-LSTM," *Int. J. Inf. Technol. Comput. Sci. IJITCS*, vol. 10, no. 6, pp. 44–53, 2018.

[82] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997.

[83] F. Almeida and G. Xexéo, "Word embeddings: A survey," *ArXiv Prepr. ArXiv190109069*, 2019.

[84] W. Koehrsen, "Neural Network Embeddings Explained — towardsdatascience.com." 2018. [Online]. Available: https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

[85] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *ArXiv Prepr. ArXiv150804025*, 2015.

[86] G. Pornaras, "Selenium vs. Beautiful Soup: A Full Comparison | BlazeMeter by Perforce — blazemeter.com." Oct. 07, 2021. [Online]. Available: https://www.blazemeter.com/blog/selenium-vs-beautiful-soup-python

[87] M. Labonne, "What is a Tensor in Machine Learning? — towardsdatascience.com." Mar.

29, 2022. [Online]. Available: https://towardsdatascience.com/what-is-a-tensor-in-deep-learning-6dedd95d6507

[88] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[89] K. Joshi, "Jaccard Similarity and Jaccard Index using Python and NLP - AskPython — askpython.com." Feb. 23, 2023. [Online]. Available: https://www.askpython.com/python/set/jaccard-similarity-python

[90] H. Shah, "6 Reasons Why Flask is Better Framework for Web Application Development — able.bio." Dec. 01, 2021.

[91] A. Hassan, "What Is SQLite? — builtin.com." Dec. 21, 2022. [Online]. Available: https://builtin.com/data-science/sqlite

[92] R. Kurtzman, "Advantages and disadvantages of Next.js — dev.to." Sep. 09, 2022. [Online]. Available: https://dev.to/richkurtzman/advantages-and-disadvantages-of-nextjs-5hg6