2020

Health-aware Food Planner: A Personalized Recipe Generation Approach Based on GPT-2

Bushra Aljbawi
aljb1640@mylaurier.ca

Health-aware Food Planner: A Personalized Recipe Generation Approach Based on GPT-2

By:

Bushra Aljbawi

Master of Applied Computing, Wilfrid Laurier University, 2020

THESIS

Submitted to the Department of Physics and Computer Science

Faculty of Science

in partial fulfillment of the requirements for the

Master of Applied Computing

Wilfrid Laurier University

Abstract

"What to eat today?" With the flourish of Internet, more and more people nowadays are inclined to find an answer to this most problematic question online. The recent explosion of food networks; however, produces large volumes of recipes, making it even harder to make an informed decision. This yields the need for advanced decision-making algorithms and efficient recommendation systems. Conventional recommender systems are not feasible anymore as food is a complicated feature that presents unique challenges and is less studied. For example, it can be one of the main reasons for obesity and many other chronic diseases. Food recommender system has the potential to urge users to change their eating behaviors by adding a healthiness component as another factor in the recommendation procedure. Text generation, a hot area in machine learning, can be used as a part of a food recommender system to explore new recipes. However, existing works do not include the factors of users' preferences, nutritional needs, and knowledge of the ingredients. In this work, we tackle this issue by proposing a new task of healthy and personalized recipe generation given only a few ingredients. We also suggest personalizing the ingredient list by integrating the user profile extracted from the previous history. Specifically, our model consists of three main components: 1) completing the given initial ingredient list by predicting the most relevant, healthy, and personalized ingredients, 2) fine-tuning GPT-2 model to generate a new recipe given the ingredients, 3) finding and recommending the top similar recipes to the generated one. In contrast to other recipe generation models, we expand the final output to be the generated recipes in addition to the top-k similar recipes from the dataset. All the proposed solutions in this work have been evaluated separately to compare their evaluation

against their related works using suitable metrics. In addition to that, we did further analysis to study the hyperparameters and design options. By doing so, we intend to show our model's ability to recommend new yet logical recipes that balance the preferences with the healthiness.

Acknowledgments

First, I would like to express my genuine gratitude and sincere thanks to my advisor *Dr. Yang Liu* for the continuous support of my M.Sc. research, for her enthusiasm, guidance, and vast knowledge. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

Besides my advisor, I would like to deliver my appreciation to my committee members, *Dr. Chinh Hoang* and *Dr. Xu (Sunny) Wang*.

I wish to also thank the one who surrounds me with love. The one that I am just lost for words when it comes to him. To my source of strength, and lovely husband, *Munzer Alsallakh*. I can not thank you enough for anything from the fruitful discussions to the tasty dinners and happy distractions.

In addition, I want to thank *my mother and father* who were always there for me with their prayers and words of encouragement despite the distances, I owe you every single thing.

I want to also thank the ones who believe in me and support me all the way, *my father and mother in law and my siblings*.

Finally, I would like to acknowledge *Jussor and ISOW* for giving me this opportunity and make my dreams true. A special thanks to *Dr. Gavin Brockett and his wife Meg Brockett* for their priceless support and wise advice in every step of my journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Definition

Natural everyday action in human life is to cook and eat a meal as food is essential to the human being. We face the problematic issue of what to eat every single day especially with people's tendency to avoid repeating similar meals. Recipes sharing websites aim to help users address this issue. However, having a huge volume of varieties besides the multiple factors to choose based on them makes the decision making much harder. To save people's time and efforts, food recommender systems have emerged.

Basically, conventional food recommender systems focus on generating recipes that suit users' tastes [39, 40, 42-45]. Since food is one of the main factors in obesity and many chronic diseases, a healthiness component should be added to the selection criteria of a food recommender system. A survey provided in [48] proved that most of the people are aware of food's effects on health and prefer taking the healthiness into the consideration of a recommender system. People ignore healthiness due to their busy lives although they know its importance. A recommender system can urge users to change their eating behaviors by calculating or balancing the nutritional needs as the only factor [46] or just an additional one in recommendations [47-50].

To explore new or unseen recipes, a natural language processing task; text generation can be applied to recipes [63-66]. The recipe generation also tests how much a language model can be creative. As people may not have or know all the ingredients, some works suggest having only a few ingredients as input [56, 66] and a few other works [55-57, 60, 61] focus on completing the ingredient list as their primary goal where they explore different methods to achieve it. Personalizing the generated recipe is only suggested in a recent work to be a factor in recipe generation [66].

None of the existing works; however, take all the important factors of incomplete knowledge of the ingredients, users' preferences, and healthiness into consideration when building a recipe generation system. Therefore, there is a high need for an end-to-end system that helps people plan for their daily meals and save their time by including their existing ingredients and preferences while at the same time taking care of their healthiness.

1.2 Our Method and Contribution

The goal of this research is to meet the aforementioned challenges by proposing a new task of healthy and personalized recipe generation that fits in the intersection of two tasks: text generation from natural language processing, and personalized recommendation from recommender systems.

The input of our model is a few user-defined ingredients. We then suggest multiple novel methods to complete the given ingredients to a full list in three stages. The first stage aims to personalize the ingredient list by choosing the users' preferred ingredients. After extracting the users' preferences from their historical ratings, we build a recommender system using Neural Matrix Factorization (NeuMF) [67] model to select the users' liked ingredients. We argue that the

suggested method of personalizing the ingredient list before generation is simple yet effective to represent the users' tastes.

After that, the second stage creates a list of candidate ingredients that are compatible with the given and liked ingredients where we suggest three various models. The first model depends on association rules while the second is based on a deep neural network model. However, the final and adopted method fine-tunes BERT [8] model and then follows a new algorithm to predict the most related ingredients. The candidate list is further filtered based on the ingredients' healthiness level in the third stage. The healthiness in this work is represented in the form of calorie count. The filtering process follows two steps: the first predicts the ingredients' amounts using a neural network and the second is an iterative process that uses the candidate ingredients along with their predicted amounts to select the ingredients that result in lower calories.

In order to generate coherent directions of recipes, we pass the complete, personalized, and healthy ingredient list to GPT-2 model [10]. GPT-2 is the state-of-the-art language model that has the ability to generate reasonable and coherent natural language texts. A very important factor in GPT-2 model's good performance is the pre-training where GPT-2 is pre-trained on large corpora to obtain a language knowledge about the word sequences. Therefore, to use GPT-2 in our domain of recipes, we fine-tune the pre-trained parameters on our recipe dataset.

Moreover, we expand the output of similar works in recipe generation [63-66] to be the new generated recipe in addition to the top-k similar recipes from the dataset. We compute the similarity in terms of the ingredients, their amounts, and the recipe directions. This addition

guarantees to have plausible suggestions in the output and gives the user more freedom with the additional provided options to choose from.

Our proposed model's workflow can be summarized as follows:

- Completing the partial ingredient list to a complete one that is compatible with the given ingredients and balances the users' favorite ingredients and their nutritional values
- Generating a new recipe given the complete list of ingredients by using a fine-tuned GPT-2 model
- Searching the recipes dataset for the most similar recipes to the generated one

To the best of our knowledge, we are the first to include healthiness in a recipe generation task and to align users' preferences in completing the ingredient list before generation. Moreover, various novel machine learning methods are suggested to achieve our task's components and all the methods are then combined to construct the final system.

All the proposed solutions in this work are evaluated separately and thoroughly. We found that we make many contributions to the task of this work. The overall task in addition to some sub-tasks such as personalizing an ingredient list are completely new. On the other hand, the methods suggested in other parts demonstrate a significant improvement over related works such as the task of finding the most compatible ingredients to complete a list with them. In addition to that, we explore the applicability and efficiency of other existing algorithms to solve some sub-tasks such as using GPT-2 to generate new recipes. Overall, we end up building a new working system that combines many tasks successfully.

1.3 Outline of the Thesis

The thesis is organized as follows. In Chapter 2, we review notable works in Transformer-based language modeling, recommender systems, and food-related computational tasks. Chapter 3 extends the discussion of our suggested methods in the three phases of our system. The experiments and evaluation details are then elaborated in Chapter 4. Finally, we conclude the work along with future directions.

Chapter 2

Literature Review and Background

In this chapter, we discuss the most relevant background in the areas of language modeling, recommender systems, and food understanding. This is to provide a deeper comprehension of the methods used in this work and how they can improve the existing works.

2.1 Transformer-based language models

Humans communicate with each other using the natural language including speech and text. Natural Language Processing (NLP) is the technology used to give the machines the ability to handle natural languages including understanding, analyzing, and generating human language. The notable work that started the history of Natural language processing was the Turing test [1] in 1950. The early natural language processing systems that followed the Turing test were based on a set of designed rules. In the late 1980s, natural language processing systems started to depend on statistics and probabilities to learn the rules from large text corpora [2]. Nowadays, natural language processing is considered a subfield of artificial intelligence and machine learning. There are two main categories to implement natural language processing tasks. Namely, syntactic analysis and semantic analysis. While the syntax refers to the words' organization in a sentence, the semantics focuses on understanding the meaning of the sentences. Natural

language processing is a very broad area that has a wide range of tasks and applications. As we aim to generate natural language text of recipes' instructional steps in the second phase of our suggested framework, the most relevant NLP task is text generation. Thus, we focus on language modeling in what follows, which is the method used to generate texts.

2.1.1 Language Modeling

Language modeling is one of the most crucial components of modern natural language processing. While it is considered the base of many challenging tasks in NLP such as machine translation, it can also be used directly to generate text. Language models can be simply defined as a probability distribution over sentences in a language or corpora. Moreover, a language model can be used to predict the next word in a sequence by estimating the conditional probability for a word to follow the preceding words [3]. The more recent language models make use of neural networks in development; thus, they are called Neural Language Models (NLM in short).

A basic neural language modeling can be summarized into three major steps [4]:

- Mapping a real-valued feature vector to each word in the vocabulary
- Defining the joint probability function over words. The function outputs a vector in which the $i\text{-}th$ element represents the conditional probability of a word i given its context or previous words as:

$$P(w_t = i \mid w_1^{t-1}) \quad (1)$$

- Learning the word feature vectors and the parameters of the probability function at the same time.

For implementing the neural language models, feed-forward neural networks [4], recurrent neural networks (RNN) [5], and long short-term memory networks (LSTM) [6] are used. These neural models achieved a better performance than the classical methods due to their high ability of generalization. The most recent years; however, have been an inflection point for language models and many other NLP tasks. The Transformer architecture [7] was the inspiration source for many researchers to develop novel NLP language models [8, 9, 10]. These language models were able to achieve state-of-the-art results and break the records of many language-based benchmarks.

2.1.2 Transformer

The Transformer [7] is an encoder-decoder model that aimed primarily to be used in sequence modeling and transduction models like translation. It outperforms Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Long Short-Term Memory networks (LSTM), and others that used to be the best and most popular approaches for sequential modeling tasks. The Transformer structure depends solely and entirely on the attention mechanism without recurrence or convolution. This attention-based architecture proved its ability to overcome the main problems of sequential modeling. Unlike RNNs and other sequential models that require processing the data in order, it is fully parallelizable. Moreover, it reduces the problem of model forgetting in long sequences. And although it is memory intensive, it is scalable and relatively simple in computational complexity.

Before diving into the Transformer architecture, a background on sequence-to-sequence models and attention mechanism is given.

2.1.2.1 Background on Sequence to Sequence and Attention

Sequence-to-sequence (seq2seq) is a deep learning model that takes a sequence as an input, transforms it into another type of sequences, and outputs the transformed sequence [11, 12]. Although it is mainly used for NLP tasks, the items in a sequence can be words of any language, tokens, or features of images. A good and clear example of such a model is machine translation.

Seq2seq models are composed of two main components: Encoder and decoder. The encoder processes the items of the input and converts them to vectors while the decoder takes the encoder output vectors to produce the output item by item. The encoder and decoder can be simplified as two translators who have a mutual language other than the input/output language. Therefore, the input should be translated by the encoder into this medium language before feeding it to the decoder to produce the desired language output.

A novel advancement over the seq2seq models is the use of the “Attention” mechanism to improve the performance. The attention technique was initially defined in the field of neural science [13], motivated by the human visual system. As the visual system gives humans the ability to focus on the most important parts of an input image, the attention mechanism allows the models to focus on the most relevant or important parts of the input. The attention has been successfully integrated into the neural network models to enhance many computational tasks.

In the case of machine translation, the attention technique made a big improvement over the classical seq2seq models [14, 15]. Applying attention to a seq2seq model requires two major changes in the process. First, the encoder sends further data in the form of vectors to the decoder where each passed vector is associated with one token/word from the input. Second, the decoder makes an additional attention step before generating the output. As each forwarded vector from the encoder is related to one token or word from the input, the decoder makes use of them to apply the attention mechanism and focus on the most important tokens/words of the input at each decoding time step.

2.1.2.2 The Transformer Architecture

The attention mechanism started to be widely used in seq2seq models as in [14]. However, most of the works conducted before the Transformer use the attention mechanism but still rely on a recurrent network. The Transformer was the first work to propose relying entirely on attention in building a transduction model without any recurrence or convolution units. The main addition was enabling every token/word to attend to every other token/word by stacking attention layers.

The Transformer structure is simply composed of a set of encoding layers and another set of decoding layers as shown in Figure 1. All encoder layers are identical and have two sub-layers. While the first sub-layer applies the attention mechanism, the second sub-layer is a simple feed-forward neural network. On the other hand, the decoder layers are also similar to each other but have one difference from the encoder layers which is the additional attention sub-layer to help the decoder attend to the encoder output. Each sub-layer in the encoder/decoder is followed by residual connection and layer normalization.

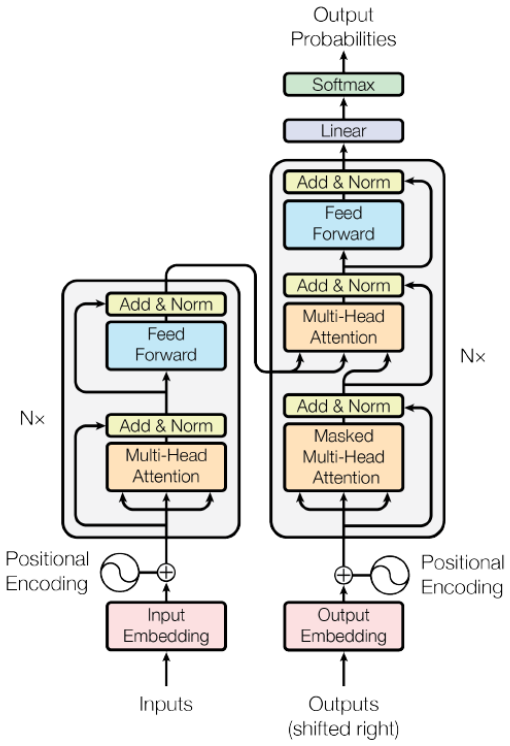

Figure 1. Transformer Architecture

Taken from: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. (2017). Attention is All you Need. Neural Information Processing Systems (NIPS), 6000–6010.

The input sequence can be split into parts called tokens or words. Each token is embedded using both an embedding algorithm to transform the token into a vector and a positional encoding algorithm to preserve the order of the token in the sequence. The vectors are then sent to the encoder self-attention layer, followed by its feed-forward network layer, then passed to the next encoder.

The first attention mechanism used is called “Scaled Dot-Product Attention” and is illustrated on the left side of Figure 2. This attention mechanism helps the encoder understand the relevant tokens from the input sequence by the following calculation steps:

1- For each input vector x_i, 3 vectors are created (query: q_i, value: v_i, key: k_i) by multiplying the input vector by the three matrices (W^Q, W^K, W^V). These matrices are randomly initialized and changed during the training process

2- Compute the dot product of the query vector and the key vector, divide the result by the square root of the key vector dimension, and apply a Softmax function to normalize the result. Finally, multiply the final score by the value vector.

In the real implementation, the above calculation is done for a set of input vectors ($x_1, \ldots, x_n$) in order to make it more efficient by matrix multiplication or vectorization. So rather than the three vectors of (query: q_i, value: v_i, key: k_i), the vectors of all input vectors are packed in three matrices (Q, V, K) to compute the attention as:

$$Attention\ (Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

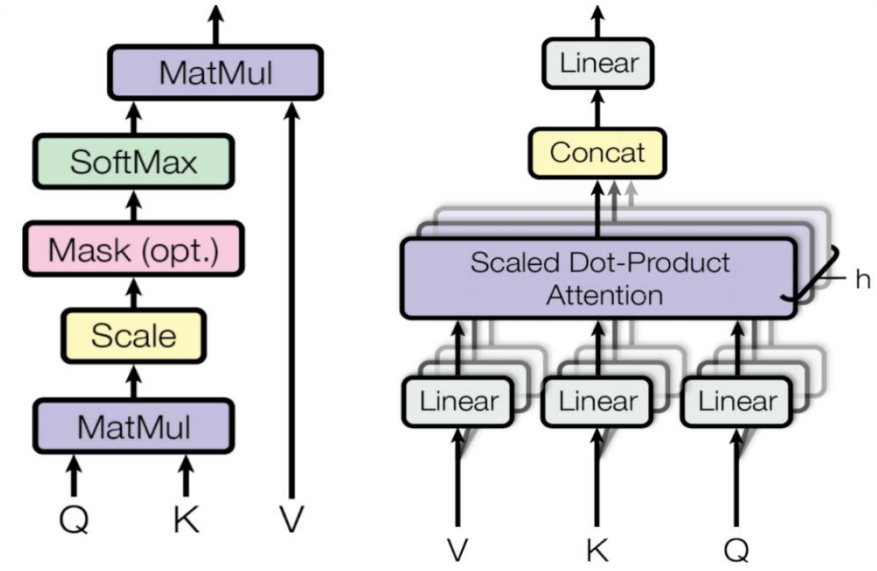

Figure 2. (Left): Scaled Dot-Product Attention. (Right): Multi-Head Attention

Taken from: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. (2017). Attention is All you Need. Neural Information Processing Systems (NIPS), 6000–6010.

The second attention mechanism used is called “Multi-Head attention” and is illustrated on the right side of Figure 2. This mechanism improved the performance by allowing the model to attend to different representations of (Q, V, K). In every attention head, the same self-attention calculation is computed in parallel but with different matrices of (Q, V, K), so we end up with different matrices. In order to find the final result of the attention to pass it to the feed-forward neural network, a concatenation of all the attention head outputs is done followed by multiplying it by the weights matrix W^O as:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O \quad (3)$$

Where $head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)$

2.1.3 BERT

The innovative idea of pre-training a language model or a language representation has seen rising success in improving many natural language processing tasks [8-10, 16-19]. The intuition behind this approach is to give the model the ability to get a general knowledge of the language and the use of its words. Pre-training a language model is also helpful when a big enough task-specific dataset is not available to develop a supervised learning model.

There are two major strategies of pre-training language models: feature-based and fine-tuning. In both strategies, the model is fed with huge unannotated corpora to learn the general language representations and another small but task-specific dataset to apply an NLP task. However, transferring the general model to a specific NLP task is different. In the feature-based approach [16, 17], a task-specific architecture is required, and only the weights of the added

layers change during the task-specific training phase. On the other hand, the fine-tuning approach [8-10, 18, 19] does not change the architecture, but all the model weights change when training on a new task.

Based on the two mentioned ideas of Transformer architecture and pre-training general language models, Bidirectional Encoder Representations from Transformers (BERT) [8]- has been proposed. BERT is a novel language representation model that is described as the start of a new era of NLP. It follows the fine-tuning approach from pre-training language modeling strategies. Therefore, BERT framework is composed of two main phases: pre-training where it is trained on huge unannotated datasets, and fine-tuning to use the pre-trained model parameters for a specific task.

BERT architecture is a stack of Transformer blocks that have the exact design of the Transformer encoder layers. Although BERT blocks are similar to the Transformer layers, the hyperparameters were increased to improve the performance such as the attention heads and the neural network hidden units. Moreover, different numbers of layers or blocks were used which yielded two main versions of BERT. Namely, $BERT_{BASE}$ and $BERT_{LARGE}$.

The principal contributions of BERT can be summarized in two points. First, the deep bidirectionality feature that enables the representation to include both left and right contexts. The bidirectionality feature is implemented by adopting the “Masked Language Model” (MLM) from the Cloze task [20] which is executed by randomly masking a fixed ratio of the tokens and training the model to predict these masked tokens. Second, the Next Sentence Prediction (NSP) task. NSP enhances BERT learning for the tasks that require processing multiple sentences such

as question answering. It can be explained as follows: given two sentences s1 and s2, what is the probability of the sentence s2 to follow the sentence s1.

The BERT-based research directions can be split into three different categories. Although BERT achieves new state-of-the-art results for eleven NLP tasks, it has been found to be undertrained in some cases. Moreover, it has some limitations such as using the masked language modeling that hinder the autoregression application and therefore generating texts. Thus, the first direction aims to further improve BERT results and solve its limitations. Robustly Optimized BERT Pre-training Approach (RoBERTa) [21] proposed a replication study of BERT hyperparameters and pre-training conditions. RoBERTa contributions include training on a cleaner and larger dataset, using bigger batches, and removing the next sentence prediction (NSP) objective. Another example of works in this direction is XLNet [22], a generalized autoregressive pre-training method that keeps the bidirectionality feature yet avoids the masking usage that causes a discrepancy problem in BERT. XLNet also suggests some new techniques such as the relative positional embedding and a new attention technique called “Two-Stream Self-Attention”.

Another research direction intends to incorporate other elements into BERT design such as ERNIE [23] which stands for Enhanced Language Representation with Informative Entities. ERNIE is a language representation that extends BERT model to integrate knowledge graphs. The architecture is identical to BERT architecture with the addition of a new encoder to capture the entities’ information.

The last branch of the BERT-related works is a direct fine-tuning of the pre-trained BERT to enhance the results of diverse NLP tasks. As an example, [24] fine-tunes BERT on Stanford Question Answering Dataset (SQuAD) to create a model they call it: “Reader”. For the purpose of building an end-to-end domain question answering system, they integrate the mentioned “Reader” with an informative retrieval tool called Serini. One more example is [25] that uses BERT for the task of Target Aspect-based Sentiment Analysis (TABSA) which is a subtask of sentiment analysis. TABSA task aims to identify fine-grained opinion polarity towards a specific aspect associated with a given target. The work proposes constructing auxiliary sentences from the aspect, converting the ABSA task to a sentence-pair classification task, and then fine-tuning the pre-trained model.

2.1.4 GPT/GPT-2

GPT [9] is another language model that depends on both ideas of pre-training language models and Transformer architecture. GPT-2 [10] is a direct scaleup version of its successor GPT with more parameters and a bigger dataset. The architecture of GPT-2 model is also composed of Transformer blocks stacked on top of each other. However, this model adopts the design of the Transformer decoder layers instead of the encoder in BERT. One difference from the original decoder blocks in Transformer is the removal of the second self-attention layer in GPT-2 blocks keeping only one masked self-attention layer and a feed-forward neural network. Four different versions of GPT-2 model were suggested with different numbers of parameters, layers, and dimensions in each version.

As the attention layer in GPT-2 blocks adopts the masked self-attention mechanism instead of the self-attention that BERT uses, the right tokens are masked in GPT-2 model. Thus, GPT-2 considers the left context only and does not have the bidirectionality feature. Losing the bidirectionality feature allows GPT-2 to use Auto-regression. Auto-regression means adding the predicted word to the input sequence in the next step of prediction. Auto-regression is definitely a more important feature when we need to predict the next word in a sequence given all the previous words. In other words, GPT-2 is more effective for generation tasks while BERT is better for Natural Language Understanding (NLU) tasks that require the model to predict one or a few missing words.

In our model, we use the pre-trained GPT-2 model to fine-tune its parameters on a recipes' dataset in order to use the fine-tuned model in recipes generation.

2.2 Recommender Systems

Recommender system is a subfield of machine learning that aims to facilitate the decision-making process for the user. The recommender system input is the historical user-item interactions and the main task in a general recommendation system is to predict the users' ratings of new items. Recommender systems can be categorized by many factors. The most common classification is based on the architecture type, where there are two types of recommender systems: collaborative filtering (CF) and content-based [26, 27]. Collaborative filtering methods recommend the items that similar users like. Content-based methods on the other hand focus on the items and recommend similar items to the ones the user liked before.

Other categorization factors include the input data type [27]. Models are considered explicit if the input data is numerical ratings and considered implicit if the input data is collected from the users' actions on websites such as the number of page visits or the clicking rates. Recommender systems can be further classified into general and sequential recommendation systems [34]. The task in sequential recommendation is to predict the next item given the user's history of interaction with items, rather than predicting items ratings in general recommendation.

In general, recommender systems suffer from two problems: cold-start and data sparsity. The cold-start problem occurs when there is no data to base the recommendation on it, which happens in many cases including the addition of a new item or the registration of a new user. The cause of the data sparsity problem; however, is the impossibility of having an item rated by all users or a user has rated all items in a real-life scenario.

2.2.1 Deep Recommender Systems

Deep learning (DL) is defined as a subfield of machine learning that can learn deep representations of data. Deep learning algorithms proved its ability to improve the performance of many fields in machine learning such as computer vision and natural language processing. Recently, many research works consider examining the use of deep learning techniques in recommender systems [27].

Adopting deep learning algorithms in recommender system is very appropriate and beneficial because of many reasons. Firstly, deep learning algorithms are able to learn non-linear

or non-trivial data patterns which makes them a good fit to learn the sophisticated user-item interactions. Moreover, in the case of sequential recommendation, using deep learning methods guarantees to have better outcomes after its promising results in similar sequential tasks. Lastly, the wide ubiquity of deep learning usage in academic and industry makes it flexible and easy to use them [27].

Deep learning-based recommender systems can be classified into two categories:

- *Recommender systems with neural building blocks.* In this category, recommender systems use one of the deep learning techniques either to get a better representation of users and items or to replace the conventional methods completely. Examples of the used methods are Multi-Layer Perceptron (MLP), Auto Encoders (AE), Convolutional Neural Network (CNN), and Recurrent Neural Networks (RNN). MLP is a very popular method that is simple yet able to capture the user-item interactions well [28]. CNN is known for its good results in image processing. Thus, it is used if the input data contains images [29] in addition to its adoption for text features extraction. RNN on the other side is good for sequential tasks. Therefore, it is used to capture items' sequential patterns [30].
- *Recommender systems with deep hybrid models.* Some recommender systems adopt a hybrid model that combines more than one deep learning algorithm. For instance, [31] suggests a combination of RNN and CNN.

2.2.2 Attention-Based Recommender Systems

A more recent advancement in recommender systems is the use of attention mechanisms. In this section, we list some of the related works in recommender systems that use attention, or attention-based models such as Transformer [7] and BERT [8]. In addition to improving prediction accuracy, the following models provide solutions for the popular problems in recommender systems including cold-start, data sparsity, and explainability.

Two attention-based structures have been proposed in [32]. Attention-based systems attend to the most important parts of the input and accordingly lead to limited reasoning. Dual Attention Recommender with Items and Attributes (DARIA) was proposed as an attempt to overcome this problem by stacking two attention layers for items and their features. Self-Attention Recommender based on Attributes and History (SARAH), a variation of the same work, is one of the earliest attempts to include the self-attention approach in a recommender system, where SARAH uses two components of self-attention to provide a better representation for users and items. Besides improving the accuracy of the recommendations, this work applies the attention mechanism for providing explanations of the recommendations.

Combining users and items content information with their historical interactions to create a hybrid recommender system has been widely used to solve the problem of cold start. However, Self-Attentive Integration Network (SAIN) [33] suggests using attention to get a more efficient integration in the hybrid recommender system. SAIN structure is composed of three layers. The first layer uses multi-head self-attention to represent items' and users' features. The integration process is done in the second layer using an attention mechanism. For more reasonable results,

different weights are given to the feedback and content information depending on the number of users' interactions and if they are new to the system. Thus, a bigger weight is given to the feedback information if the user has a large number of historical interactions and a smaller weight to the feedback information if the user is new to the system. The third is an output layer that is responsible for predicting the items' ratings.

The Transformer architecture [7] that we explained previously inspired many works to enhance sequential recommendation. As discussed earlier, the task in sequential recommender systems is to predict the next item given the user's history of interaction with items, rather than predicting items ratings in general recommendation. Like Transformer, Self-Attentive Sequential Recommendation (SASRec) [34] stacks self-attention layers to build an efficient system that outperforms the most popular approaches in sequential recommender systems. Furthermore, SASRec system alleviates the data sparsity problem by using attention technique which enables the model to focus only on the few last items in the case of a sparse dataset and handle long sequences if the dataset is dense. Similarly, [35] followed SASRec but with special customization to be used in industry (Alibaba website). The system aims to predict the Click Through Rate (CTR) for a given set of candidate items instead of the next item.

In sequential recommender systems, the users' previous interactions follow a ''left to right" order. A novel work [36] argues that bidirectionality might be a good choice in sequential recommender systems to avoid the rigid order assumption. Inspired by BERT [8], Bidirectional Encoder Representations from Transformers for sequential Recommendation (BERT4Rec) [36] adds the bidirectionality feature to an attention-based architecture like SASRec. Using the Cloze

task, some items are randomly masked then predicted in pre-training. To address the mismatch between this task and the sequential recommender system task, a special token [mask] is appended and predicted as a fine-tuning stage. Another work [37] in recommender systems got inspiration from BERT [8] for the task of next basket recommendation where they assume that items are comparable to words in BERT while baskets are like sentences.

As recommender systems are problem-dependent and since part of our task is to build a recommender system for recipes, we decided to study the food recommender systems in detail. A summary of the related works is found in the next subsection.

2.3 Food Understanding

In recent years, there was a rising interest in exploring the culinary domain from a computational viewpoint. Many previous works studied the potential of building models to help in automating food-related tasks such as recommending [38-40, 41-51], completing [55-57, 60-61], or generating [62-66] cooking recipes. A typical recipe is a set of the following components: title, list of ingredients, and instructions or cooking steps. It might also contain additional information like cooking time, the number of servings, cuisine type, and nutrition values.

In this section, we discuss the most relevant techniques in the food understanding area. We focus on three areas which are: recipe recommender system, recipe completion, and recipe generation.

2.3.1 Food Recommender System

Food recommender system is a challenging domain for many reasons. The first reason returns to its complicated structure as each item/recipe is composed of many other items/ingredients that should be studied to generate suitable recommendations. Another reason for food recommender system complexity is the importance of food healthiness which adds another factor to users' preferences for the recommendation to be built upon. As mentioned in [38], food recommender systems can be categorized into four main types which we discuss in the following subsections.

2.3.1.1 Users Preferences-Based Food Recommender Systems

Like other recommender systems, the primary goal of this type of food recommender systems is to recommend recipes or foods suiting users' preferences only.

An earlier work [39] suggests a simple method for food recommendation. They first let users rate some food items to utilize these ratings in representing the users' preferences. The Term Frequency-Inverse Document Frequency (TF-IDF) method is used to create profiles for users from the rated items. To recommend food items, they then compute the similarity between the created users' profiles and food items, then filter the items that exceed a specific threshold. Instead of users' profiles, [40] uses both users' ratings and recipes' tags to represent the preferences. The adopted algorithm for getting recommendations is an updated version of the well-known Matrix Factorization (MF) method [41] that predicts the rating as:

$$\hat{r}_{uc} = \boldsymbol{p}_u{}^T \boldsymbol{q}_c \quad (4)$$

where u is a user, $\boldsymbol{p}_u$ is their parameter vector, and c is a recipe, $\boldsymbol{q}_c$ is its parameter vector. The used updated version of MF aims to include tags in the calculation as:

$$\hat{r}_{uc} = \left(\boldsymbol{p}_u + \frac{1}{|T_u|}\sum_{t \in T_u} x_t\right)^T \left(\boldsymbol{q}_c + \frac{1}{|T_c|}\sum_{s \in T_c} y_s\right) \quad (5)$$

where the set of tags assigned by a user u to any recipe is T_u and the set of tags of recipe c that are given by any user is T_c.

Rather than processing the whole recipe, other works break the recipe down into its ingredients to extract the preferences. [42] assigns ratings to each of the recipe ingredients using the user's ratings of the recipes that contain these ingredients as follows:

$$\hat{r}_{ui} = \frac{\sum_{m\ s.t\ i \in c_m} r_{u\,c_m}}{m} \quad (6)$$

where u represents the user, c is the recipe, i is the ingredient, and m is the number of recipes containing ingredient i. The suggested algorithm then predicts a rating for a recipe by averaging its ingredients' ratings as:

$$\hat{r}_{uc} = \frac{\sum_{j \in c} r_{u\,i_j}}{j} \quad (7)$$

After breaking the recipe down into its ingredients, [43] adds the quantities of ingredients into consideration before scoring the recipes to increase the score of a recipe that contains higher amounts of the user's favorite ingredients over another recipe that contains the same ingredients

but with fewer amounts of the favorite ones as the following example shows: given two recipes A and B.

- Recipe A ingredients are: 300g chicken, 60g cheese, 100g potatoes
- Recipe B ingredients are: 100g chicken, 120g cheese, 100g potatoes

Recipe A and B have equal ratings in a conventional system even if the user likes cheese and dislikes chicken. However, using the updated scoring method in [43] the rating of recipe A would be less than recipe B.

Breaking a recipe into its features in addition to its ingredients is proposed in [44]. The extracted ingredients and features are considered as content information which are added to the rating information for the purpose of building a hybrid recommender system. The work adopted two different methods to build the hybrid system where the first method is based on K-Nearest Neighbor (KNN) based, the second is based on Stochastic Gradient Descent (SGD).

Aside from the mentioned methods, [45] frames the recipe recommendation task as a *recipe pair prediction* where given two recipes, the model selects which recipe has a higher score than the other. Their main contribution; however, was in the modeling of the relationships between ingredients which is achieved by constructing two networks. The first is an ingredient complement network which uses Pointwise Mutual Information (PMI) to calculate the probability of the co-appearance of two ingredients i_1 and $i_2$as:

$$PMI(i_1, i_2) = log\ \frac{p(i_1, i_2)}{p(i_1)\,p(i_2)} \tag{8}$$

The second is an ingredient substitute network where they use the recipes' reviews to extract possible adjustments or modifications of ingredients.

2.3.1.2 Nutritional Needs-Based Food Recommender Systems

Food is one of the main reasons for obesity and other health issues. Thus, healthy diets are attracting attention in recent years. Some of the researchers claim that food recommender systems have the potential to incite people to have healthier food options; therefore, a new branch of food recommender systems has emerged where nutritional needs and health problems are the only criteria for getting recommendations. As an example, [46] lets users enter the main health concern they want to treat. Then, the system filters the recipes to recommend users the recipes that meet their nutrient needs to cure this issue.

2.3.1.3 Preferences and Nutritional Needs-Based Food Recommender Systems

Considering healthiness in food recommendations is vital for good eating habits, yet it is not enough as it might not be appealing to people. Other works in food recommender systems integrate users' nutritional needs with their preferences to generate balanced recommendations.

In [47], the prediction of the liked recipes is followed by calories and fats calculation to recommend the recipes with the least calories' or fats' values. The method presented in [48] extends the previously mentioned work [40] in the preferences-based recommender systems section. They simply add a healthiness component in the form of calorie balance to the rating

calculation as in Equation 9. Moreover, they weight the calculation by a healthiness factor w_h that can be adjusted by users. The final utility of a recipe c for a user u is given as:

$$util_{uc} = w_p * \hat{r}_{uc} + w_h * h_{uc} \quad (9)$$

The work just mentioned in [47] is also extended in [49] to recommend complete meal plans for a day instead of only recommending recipes. Similarly, [50] proposes a recommender for a complete food plan. However, this work considers more nutrients in planning, constructs a further detailed template for the daily meal plans, and avoids including the items consumed recently in the plan.

2.3.1.4 Group Food Recommender Systems

The target of most food recommender systems is individuals, but some works target groups instead [51]. This type of recommenders is essential for families and parties. Maximum satisfaction between the group members should be maintained. Mainly, there are two approaches used for group recommender systems which are aggregated models and aggregated prediction.

2.3.2 Recipe Completion

Recipe completion usually refers to the task of completing a partial list of ingredients. In other words, given an incomplete list of recipe ingredients, the recipe completion system tries to find the best fitting ingredients to be added to the list. This task is important because people consider leftover ingredients at home when deciding about their meals. Thus, they need to find

complement ingredients to add to their shopping lists. The nature of this problem makes it hard to be implemented as it depends on multiple factors and requires knowing the relations between ingredients. The problem is not extensively explored in literature. However, the few existing works model it differently and use it in diverse applications including recipe recommendation and recipe generation.

Initially, some works study the relations and combinations of ingredients to solve different problems. For instance, [52] proposes an iterative solution that aims to find alternatives for the ingredients listed in a cooking recipe if the user is not satisfied and wants to replace them. To do that, the work suggests a typicality measure that helps to recommend the best fitting ingredient to replace with. The same purpose of finding substitute ingredients is used in [53] but with a focus on Indian cuisine. The used method is also different as they adopt two machine learning models: vector space and Word2Vec. Similarly [54] aims to find alternatives to some ingredients in a recipe. However, they customize it for food allergen to help people who suffer from food allergies. In the previously mentioned work [45], the ingredients complement network can also be considered related to this type since the work extracts possible combinations of ingredients using PMI score Equation 8. Even though they originally aim to use this network to study the effect of complementary information on recipes' ratings, other works [55, 56] see it as a graph-based solution of recipe completion and implement it as a baseline to compare with their contributions.

The first and most popular work to intently deal with the recipe completion task is [57]. As the object is to transform a partial ingredients list $\boldsymbol{x}$ to a complete list $\boldsymbol{y}$, they suggest using a linear structure as:

$$\boldsymbol{y} = M\boldsymbol{x} \tag{10}$$

where M is a coefficient matrix. They then use two models to get the values of M matrix. The first is Non-Negative Matrix Factorization (NMF) [58] which is defined as a matrix decomposition method such as if a (n × m) matrix Y is given, NMF approximates it to two low-rank matrices containing k latent features: W, an (n × k) matrix and H, an (k × m) matrix as:

$$Y \approx WH \tag{11}$$

Y is the output of all recipes in training data while W and H are determined during training. After that, they use it for recipe completion as follows:

$$Y_{old\_recipes} \approx W_{old\_recipes}H^T = X_{old\_recipes}\beta \tag{12}$$

where $\hat{\beta}$ is the coefficient matrix M needed for completion and it is estimated as follows:

$$\hat{\beta} = (X^T_{old\_recipes}X_{old\_recipes})^{-1} X^T_{old\_recipes}W_{old\_recipes}H^T \tag{13}$$

The second used method is two-step regularization least squares [59] which is a method to make predictions on paired-comparison data. It mainly approximates the given matrix into two kernel matrices and a coefficient matrix as:

$$Y \approx K_u W K_v \tag{14}$$

The two kernel matrices K_u and K_v contain information to represent the data in matrix Y where the used information for recipe completion task are recipe data and flavor data. Although the method generates good results, we do not see it useful considering the difficulty of finding flavor data in most recipes datasets.

In [55], the authors compare three different models of ingredient prediction including the graph-based solution from [45] and the NMF-based solution from [57]. An additional method is based on a shallow neural network. The solution aimed originally to predict the amounts of ingredients, but they lend it easily to a recipe completion task by choosing the top-k ingredients that are not from the input set but result in high values.

An embedding-based method presented in [56] considers all the existing ingredients in the pre-set as the context and retrieve the most relevant ingredients to complete the initial list with it. The top relevant ingredients are those with the highest probabilities given their contexts as:

$$Pr\left(i_a \middle| context(i_a)\right) \tag{15}$$

Each ingredient i_a is embedded into a vector $\vec{\boldsymbol{v}}_{\boldsymbol{i_a}}$ and the context vector is calculated by averaging all the embedding vectors of contextual ingredients -which are all the other ingredients in the recipe-:

$$\boldsymbol{c}_{\boldsymbol{i_a}} = \frac{1}{N_r - 1} \sum_{t=1, t\neq a}^{N_r} \boldsymbol{v}'_{\boldsymbol{i_t}} \tag{16}$$

A Softmax function is then applied to compute the conditional probability from Equation 15 as follows:

$$Pr\left(i_a \middle| context(i_a)\right) = \frac{exp(\boldsymbol{c_{i_a}}^{\boldsymbol{T}}.\boldsymbol{v_{i_a}})}{\sum_{i \in I} exp(\boldsymbol{c_{i_a}}^{\boldsymbol{T}}.\boldsymbol{v_i})} \quad (17)$$

where I represents the ingredients set.

Other examples of research attempts to present a solution for recipe completion problem include the following works. In [60], the authors frame the problem differently where they see it as a recommendation problem in which they assume that recipes are comparable to users in conventional recommendation systems while ingredients are like items. The ratings given from a specific recipe to a specific ingredient is a binary variable which value is 1 if the recipe contains the ingredient and 0 otherwise. Each ingredient is then represented using a vector of its ratings. To use this representation for recipe completion, two steps are required: First, a similarity measure is computed between all ingredients to find the k nearest neighbors of each one. Second, using the found neighbors from the first step, the system determines how much an ingredient i fits to a recipe c as:

$$P(c, i) = \frac{\sum_{j \in N_i} s_{ij}.r_{cj}}{\sum_{j \in N_i} |s_{ij}|} \quad (18)$$

where N_i is the set of the ingredient i nearest neighbors, r_{cj} is the binary rating of recipe c to ingredient i,or in other words a binary value that determines whether ingredient $i$is one of recipe $c$ingredients. After that, the system selects the ingredients that achieved the higher fitting scores

based on Equation 18 to complete the recipe with. Another research attempt is presented in [61] where they suggest using a deep neural network with four hidden layers to complete the ingredients of a recipe. However, the used setting for the data and the network makes it applicable to the case when only one ingredient is missing and not applicable to the opposite case when only a few ingredients are given.

2.3.2 Recipe Generation

Recipe generation is a non-trivial task that can fit into two research fields. Namely: natural language processing and recommender system. The goal of generating a recipe differs from a work to another. Generally, there are two main research directions. The first aims to generate a recipe text given an image claiming that people might be interested in knowing the recipe behind a dish photo on social media. As an example of such a work, [62] introduces an inverse cooking system. The used method is based on the Transformer where image features and ingredients' embedding are extracted, then fused together to create the decoder input. The decoder afterward outputs the instructions sequence. The second direction; however, is to generate the recipe text given a partial or full list of ingredients. As the first direction is far from our model, we focus on the second one in the following works.

The work in [63] presents an unsupervised method to interpret recipes from unannotated data. Two models are designed in this work: a segmentation model and a graph model. The segmentation model is responsible for segmenting the recipe into a structured collection of *actions*. Each action is a tuple of a verb and a list of arguments where each argument is a string span that can represent an ingredient, a location, or others. The segmented recipe from the first

model constructs the input of the graph model that outputs a directed graph after selecting the most likely connections between the recipe actions.

To avoid the customized segmentation presented in [63], a general RNN-based encoder-decoder can be proposed. However, such a model can cause coherence problems where some information might be missed. To solve this problem, [64] suggests a goal-oriented and agenda-driven text generation system with an application on cooking recipes. The model adds two attention models to an RNN-based encoder-decoder to maintain a good coverage of the agenda items and therefore a coherent output. The first attention tracks the used items while the second attention is responsible for tracking the remaining items that need to be covered. The recipe ingredients construct the agenda items in the case of the model's application on recipes.

Similarly, [65] presents a recipe generation model that depends on the encoder-decoder structure in addition to attention models. However, this model adds the amounts of the ingredients into consideration while generating a recipe. The suggested solution to map the ingredients into a recipe is split into two steps rather than a direct mapping. After encoding the ingredients into a vector v, the vector is fed into two decoders. The first decoder interprets the ingredients into amounts and units whereas the second one decodes the ingredients into events using an attention model. Events are defined as a low-level description of the actual cooking steps such as the following example:

- Text of a cooking step: "cook ½kg of elbow macaroni in the boiling water"
- The extracted event: cook (macaroni, in boiling water)

After getting the events, the second step starts to encode the events and then decode them into the final cooking steps with the use of a second attention model.

Another work that utilizes a middle representation of recipes is the previously mentioned work [56]. Since the main object of this work is to recommend recipes and not to generate them, no further steps are taken to generate the actual text. Rather, this middle representation that is called “pseudo recipe” is used to search the datasets for similar recipes. Moreover, the suggested solution requires only a few ingredients to be given to the system as it implements a recipe completion method that we discussed in the previous section. Another contribution is adding the healthiness component by calculating the macro-nutrients values of the ingredients.

None of the works yet takes the personalization into consideration during generation except for a more recent work [66] that can generate a personalized recipe given a partial ingredient list and a recipe name. They claim that people might know the name of a dish and a few basic ingredients and wish to know the complete list of ingredients and cooking steps to prepare it. In general, the used method is an encoder-decoder model that combines additional layers to the structure including an ingredient attention layer and a personalization layer. In the personalization layer, two approaches are used to represent the preferences. Namely, prior recipe attention and prior technique attention.

To summarize, the area of food understanding is not fully explored in previous works. As an example, food recommender system is more complicated than other recommender systems because of its special characteristics and therefore a lot of novel recommender systems algorithms are not implemented and evaluated on food datasets. Additionally, the task of

completing an ingredient list is not studied except in a limited number of works as mentioned before. Moreover, recipe generation is the most recent task in the food-related area which is worth examining the state-of-the-art text generation frameworks in. What is more important is that the research studies that combine multiple novel solutions to create an end-to-end system are barely found in literature.

Unlike previously described models, in this work, we introduce a novel system that generates a *healthy* and *personalized* recipe given only a partial list of ingredients. The model completes the partial list and then filters the candidate list of ingredients by balancing both users' preferences and healthiness. The final list is fed into a fine-tuned GPT-2 model to generate new recipes. Moreover, we expand the final output to be the new generated recipe but in addition to the top-k similar recipes from the dataset.

Chapter 3

Health-aware Food Planner: A Personalized Recipe Generation Approach Based on GPT-2

In this chapter, we propose our novel framework that is composed of multiple machine learning methods. Our suggested methods can not only generate a recipe from a few ingredients but also maintain the healthiness and personalization features in the generated recipes. The model's input is a partial list of ingredients that can be either entered by the users or selected from their top favorite ingredients. The final output of the model is the instructions of the generated recipe in addition to a few other similar recipes from the dataset to recommend to the user.

Our general framework consists of three main phases: 1) completing the ingredient list; 2) generating a recipe; 3) searching for similar recipes in the dataset. A visualization of the model workflow is given in Figure 3.

- **Completing the ingredient list**: The users' preferred ingredients are formed as a partial ingredient list which can be either passed to the framework or extracted from the users' historical interactions with recipes. Given this partial list of ingredients, we then suggest three different methods to find the most compatible ingredients to complete the list. The

candidate ingredients are then filtered based on their calorie balance to output the healthier options.

- **Generating a recipe**: The complete list of ingredients is then passed to a fine-tuned text generation model to generate the recipe instructions.
- **Searching for similar recipes in the dataset**: The final phase is a search process in the recipes dataset to find the top-k similar recipes to the generated one. This guarantees to have plausible alternatives as output and to give the user more options to choose from.

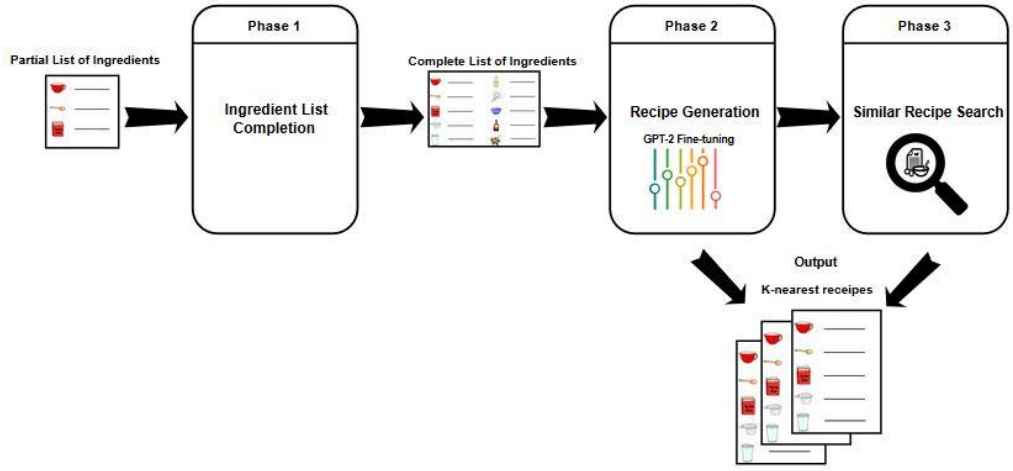

Figure 3. The proposed model workflow

3.1 Completing the Ingredient List

3.1.1 Overview

The method we suggest to create a complete, personalized, and healthy ingredient list is composed of three stages. The first stage personalizes the ingredient list and reflects the user's

own preferences. Users can choose their preferred ingredients by entering a few ones to the system. We set the maximum number of entered ingredients to be 3. However, if the users prefer not to enter any ingredients or just have one or two in mind, we create or complete the partial list to a three-ingredient list by building a recommender system that analyses their historical interactions with recipes and then selects the top liked ingredients.

In the second stage, the proposed algorithms return the most related or compatible ingredients with the ingredients in the partial list. The output of this stage is the list of candidate ingredients that are then filtered based on their healthiness level in the third stage. The healthiness is defined in this work as the calorie balance. Since the ingredient list alone is not sufficient to calculate the calories and other nutrients, we propose a method to predict the quantities of the ingredients before the filtering. Then, we add ingredients from the candidate list to the final ingredient list until no more calories are allowed in a healthy meal.

Detailed visualization of this phase with its three stages is given in Figure 4.

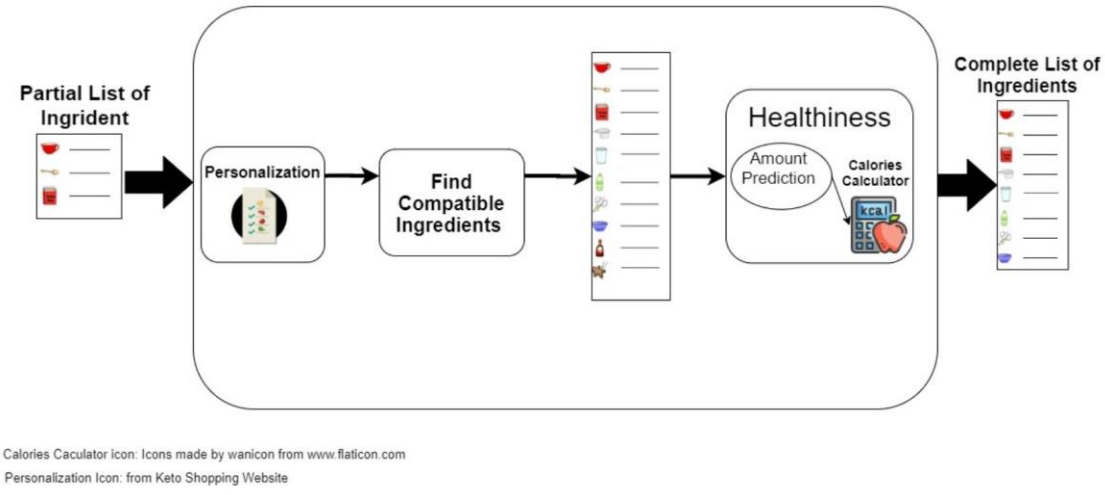

Figure 4. Completing the ingredient list visualization

3.1.2 Personalization: Find the Most Preferred Ingredients

This stage is responsible for finding the users' favorite ingredients in case they did not select enough number of their preferred ingredients as an input as explained in the overview above. In order to achieve that, we build a recommender system of ingredients which can order ingredients based on each user's taste by analyzing their past interactions with ingredients. However, this task is challenging as there is no clear interactions' information between users and ingredients in the dataset. For that reason, we had two main steps in the personalization phase: the first is to create the required data of user-ingredient interactions while the second is to build the actual recommender system.

Using existing user-recipe interactions, we create the required user-ingredient interactions data as follows. First, we map each user to a list of ingredients lists for each recipe they interact with. Then, inspired by [42] we compute the rating of each ingredient the user has used in any recipe by averaging the ratings given to each recipe that contains this ingredient as:

$$rating(u_i, ing_j) = \frac{\sum_{m\ s.t.\ ing_i \in recipe_m} rating(u_i, recipe_m)}{m} \quad (19)$$

Where m is the number of the recipes that contain ing_j and all the variables are as explained in Equation 6. After ratings calculation, tuples of (user, ingredient, rating) are listed to create the required interactions dataset.

The ingredient rating data we created in the first step is the only resource of information about users' preferences since there is no additional content information. Therefore, we chose to adopt the Neural Matrix Factorization (NeuMF) [67] method, the state-of-the-art framework

for recommendation using only past feedbacks. NeuMF upgrades the conventional Matrix Factorization (MF) method using the following components:

- Generalizing MF: As a result of this generalization, MF is considered a special case of NeuMF. To create the generic MF, the activation function can be changed from a simple identity function. Moreover, the weights of the output layer can be learnable parameters rather than a constant number.
- Better modeling of user-item interactions: which is done using an MLP to learn the interactions instead of the fixed inner product that MF utilized.

After implementing these two components, their outputs are fused to be fed into the output layer -or the NeuMF layer- that outputs the predicted ratings for user-item tuples.

We train a NeuMF model on our user-item interactions dataset. The model can be then used to predict a user's rating of all the ingredients and get the top-k ingredients as their favorite ones to complete the recipe based on them.

3.1.3 Compatibility: Find the Most Compatible Ingredients

After getting the user's preferred ingredients, this phase seeks to find the most compatible ingredients that can be good candidates to complete the ingredient list with them. In this work, we explore three novel methods to implement this task: association rules, deep neural network, and fine-tuned BERT. The following subsections explain these methods in detail.

3.1.3.1 Association Rules

Association rules is a rule-based machine learning method that is used for exploring large datasets and discovering the relations between its items. In other words, the method determines how likely two items in a collection to co-occur together. Therefore, it was our first suggestion to find the most compatible ingredients based on their co-occurrences in the recipe dataset. Moreover, the method is considered very easy to understand and implement.

The usual usage of association rules is for analyzing the sales of a supermarket in order to develop suitable marketing strategies where each entry in the dataset is a single transaction that contains a set of items or products. Association rules method does not consider the order of items in a set which makes it appropriate for use in this case as the order of the products in a transaction does not really matter. Similarly, we consider each recipe in our dataset as a transaction while each ingredient is an item or product where the order of the ingredients in a recipe's ingredient list is not important.

In general, there are two main steps in the process of selecting the association rules between items. The first is to extract the frequent itemsets in the dataset while the second is to form the rules using the extracted itemsets. Thus, our method can be summarized as follows:

1- **Extracting the frequent itemsets**: An itemset is any set of single or multiple items from the dataset. Selecting all the frequent itemsets in a large dataset is infeasible since it requires finding all the possible combinations of items. Therefore, we first determine the maximum length of the generated itemsets instead of finding itemsets of all lengths. The *support* measure determines the frequency of an itemset X in a transaction dataset T as:

$$Support\ (X)\ =\ \frac{|t\ \in\ T;\ X\ \subseteq\ T|}{|T|} \tag{20}$$

Using this measure, we define a minimum support threshold of the frequent itemsets. Apriori algorithm is one of most the efficient algorithms to find frequent itemsets. The algorithm relies on the property of "anti-monotonicity": all the subsets of a frequent itemset are also frequent and as a result, there is no need to generate an itemset out of any infrequent subset. We then adopt Apriori algorithm to generate all the frequent itemsets which their support exceeds the minimum threshold and their length is less than the defined maximum length.

2- **Generating the association rules:** As mentioned, an association rule determines the correlation between its left-hand-side which is called antecedents X and its right-hand-side or consequent Y. The rule is defined as: $X\ \rightarrow Y$. Using the frequent itemsets extracted in the previous step, we then form the association rules. To define the strength of a rule or how often it is true, various measures are used. We adopt the confidence metric which is given as:

$$Confidence\ (X\ \rightarrow\ Y)\ =\ \frac{Support\ (X\ \cup\ Y)}{Support\ (X)} \tag{21}$$

The confidence metric can also be defined as a conditional probability which makes it suitable for our problem as we need to know the possible ingredients given a partial set of ingredients. After setting a minimum threshold of confidence, we extract and save all the possible association rules between ingredients.

In order to use the association rules for finding the most compatible ingredients given a partial set of ingredients, we follow these steps. First, we find all the possible combinations of the given

ingredients. We then select all the rules that have any of the found combinations in their antecedent side. The consequent side of the selected rules constructs the candidate ingredients to be added to the list. We sort the candidate ingredients to return the top-k compatible ingredients. The sorting is done by a determined weight w_i for each ingredient i over all the rules that contain this ingredient as a part of its consequent part. The weight w_i is calculated as the sum of the confidence values multiplied by the number of the items in its antecedents' side as:

$$w_i = \sum_{rule\ (X \rightarrow Y) \in all\_rules;\ i \in Y} confidence\ (rule) * |x \in X| \quad (22)$$

The reason behind getting the number of the items as a part of the weight is that an ingredient that results from a rule that has all the given ingredients in its antecedent side is more compatible than an ingredient that results from a rule where there is only a subset of the given ingredients in its antecedents side. Since we first select only the rules that have combinations of the given ingredients, a larger set of antecedents is always an indicator of involving more items from the given ingredients.

3.1.3.2 Deep Neural Networks

Neural networks and deep learning algorithms are used nowadays as a solution to various problems due to their ability to recognize patterns with high accuracy. Thus, we suggest building a neural network model as a second method for ingredients completion. A few related works used the neural network to complete the ingredient lists. However, none of them used a deep structure of the network. Moreover, they used a manner that omits only one ingredient from the list and trains the neural network to predict it as a label which we claim that it is not enough to

complete a list of ingredients starting from only a few ingredients. In our solution, we form the problem as a multi-label classification instead and explore if a deep neural network is efficient and applicable to this case that has a huge number of labels.

We implemented a neural network model that takes a partial list of ingredients and outputs compatibility scores of each ingredient with the given ones. To unify the number of input and output nodes for all training examples, we hot-encode all the ingredient lists and use the number of unique ingredients in the dataset (over 14K) as the input and output nodes number.

The adopted design of the neural networks uses 3 hidden layers with 100 nodes in each layer. The utilized activation function is ReLU while the optimizer is Adam. Choosing the loss function; however, was not a straightforward process as the task is multi-label classification where each example can belong to multiple labels rather than one class only. For more numerical stability, we use a linear function for the output layer and adopt Binary Cross Entropy with Logits Loss that is the same as Binary Cross-Entropy Loss but with applying the sigmoid function internally. The loss between input x and target y for a batch size of N can be given as follows:

$$l\,(x, y) \;=\; mean\,(L)$$

$$where\; L = l_1, \dots, l_N, l_n = \; - w_n\,[y_n\,.\,log\;\sigma(x_n) \;+\; (1 \;-\; y_n)\,.\,log(1 \;-\; \sigma(x_n))] \quad (23)$$

A graphic illustration of the neural network design is given in Figure 5.

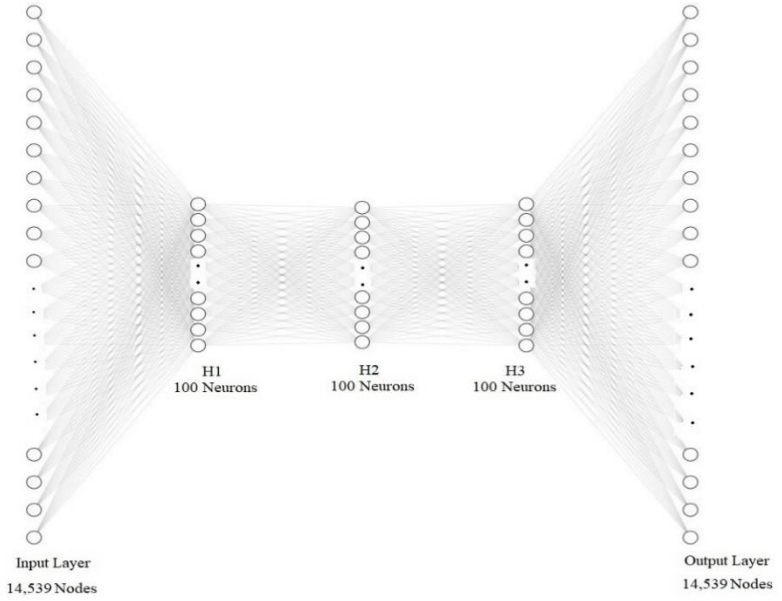

Figure 5. The neural network structure.

To use the neural network model for completing an ingredient list, we do the following. First, we convert the given partial list to a hot-encoded vector and pass it as an input of the model. After applying the model, we get a vector of scores of all the ingredients. We apply the sigmoid function on the resulting scores to get the probabilities. After sorting the probabilities vector, we choose the top-k ingredients excluding the given ones.

3.1.3.3 BERT

As the ingredient lists are given originally in the form of raw texts, we suggest considering the ingredient list information as a corpus where each ingredient list is a sentence. With this forming of data, the task of completing an ingredient list becomes the same as predicting missing words in a sentence which is a problem we can use BERT for implementing it. Choosing BERT as a solution to this task comes from the fact that one of the main objectives BERT is pre-trained on

is the Masked Language Model (MLM) which is exactly designed for predicting a few missing words in a sentence.

Although BERT can be used directly to predict missing words in a sentence, we propose fine-tuning the model on our ingredient lists data to gain a better performance. However, the task of the masked language model differs from our task of completing an ingredient list in two points. First, we need to predict most of the ingredients as the given part contains only a few ingredients rather than predicting only a few words in the original task of MLM. Second, the order of the words matters in an ordinary sentence, unlike the ingredient list where the order of ingredients does not really matter. Therefore, we did multiple BERT fine-tuning experiments with higher than default rates of masked words in an attempt to make the model able to predict more words in an ingredient list. Moreover, we explored the performance of not considering the order of ingredients in a list by fine-tuning BERT without positional encoding embedding. Surprisingly, the model's ability to retrieve a missing ingredient dropped after these modifications as we explain in the experiments section.

After fine-tuning the original BERT with the default positional encoding and MLM probability, we propose overcoming the above differences and problems in the prediction phase instead of the fine-tuning phase by following the steps presented in Algorithm 1. As BERT is not trained to predict a lot of words in a sentence, we use an iterative process to predict one ingredient at a time rather than predicting all the missing ingredients at once. We start from the partial list of ingredients, append a [MASK] token to its end (line 3), use the model to predict the masked ingredient (line 4), and then add it to the partial list (line 11) and repeat the process k times to find the top-k compatible ingredients.

Algorithm 1 Complete a partial list of ingredients using BERT

Input: s: initial ingredient-set

k: no. ingredients to return

Output: c: final list of candidate ingredients

```
c = empty list
for i =0 to k do
    input = s + [MASK]
    candidateIngredients = Bert (input)
    ingredient = CheckIngredient(candidateIngredients, 0)
    j = 0
    while j < 10 and (ingredient = NONE or ingredient in s) do
        ingredient = CheckIngredient(candidateIngredient, j)
        j += 1
    end while
    s += ingredient
    c += ingredient
    added += 1
end for
return c
```

While testing BERT model on this task, we found other problems including repeating the same ingredient in the candidate ingredient list, predicting one word from the ingredient name instead of the whole ingredient name such as predicting “baking” instead of “baking soda”, and rarely predicting words other than the ingredient names. In order to avoid these problems, we update the prediction method by adding these constraints: 1) make sure the predicted word is an ingredient which is implemented in *CheckIngredient* method; 2) if the predicted word is a part of an ingredient name, we complete it by searching the dataset for the most frequent ingredient

that contains this word which is also implemented in *CheckIngredient* method; 3) check if the ingredient is already added to the list before (line 7). If the predicted word is already added to the list or it is not an ingredient, neither a part of an ingredient, we check the next predicted word as we got the top-10 model's predictions for each masked word.

3.1.4 Healthiness: Find the Healthiest Ingredients

Maintaining the healthiness level of the suggested recipes is one of the main objectives of this work. Thus, the final filtering to form the ingredient list is a selection of the healthier options from the candidate list of ingredients. The healthiness in this work is defined as the calorie balance of the ingredients where the calories in a meal should not exceed a specific threshold. However, knowing the ingredients alone without their corresponding amounts is not sufficient to calculate calories or any kind of nutrition values. Unfortunately, the dataset we use does not contain any amounts or quantities information of its recipes. This led us to re-crawl the recipes' website to extract the amounts of each recipe's ingredient list.

After mapping each recipe with its ingredients' amounts, we utilize this information to learn predicting the amounts of the ingredients. To achieve that, we follow [56] in training a dense one hidden layer neural network. The neural network input is a hot-encoded vector of all the unique ingredients length with a value of 1 to the ingredients in a recipe and 0 to the others. On the other side, the neural network output is a hot-encoded vector of the same input length with 0 values except for the included ingredients in a set where their values are their amounts in grams. However, the results were not really promising so we did extensive experiments on the neural network settings to select its best structure and hyperparameters that minimize the error.

The final adopted neural network is a shallow network with one hidden layer of 128 neurons where the activation function is Leaky ReLU to alleviate the dead neurons problem.

As a following step after predicting the amounts of the ingredients,, we use the Canadian Nutrient File (CNF) [1] to extract the calories in each 100g of a food item and use them in calculating the calories of each ingredient in the list. To choose the healthiest options, we then suggest a process to add ingredients to the final ingredient list based on their calories' values. The suggested process is first to add the first three ingredients in the candidate list since we need to keep the user's preferred ingredients. Then, we iteratively add the most compatible ingredient from the candidates and repeat the process until reaching a pre-set calories maximum limit. This method guarantees to have the most relevant options first and to not exceed the calories limit except for the case when the first three ingredients are of high calorie levels, amounts, or both. As a second variation, we implement another adding algorithm that is inspired by [56] as well but with some changes. In this second process, we also add the first three recipes and then iterate over the candidates to add the ingredient with the lowest number of calories regardless of its compatibility or the resulting calorie count. However, the iterations to add an ingredient are limited to a specific number. The output of these two methods is the final personalized, complete, and healthy ingredient list that can be directly passed to the generation system.

3.2 Generating a Recipe

After forming the complete, healthy, and personalized ingredient list in the first phase, we aim to generate the recipe instructions in this second phase. The recipe generation process is

[1] https://food-nutrition.canada.ca/cnf-fce/index-eng.jsp

represented as a natural text generation task. Language models are the usual choice for generation texts as they create a probability distribution over sentences in a language which makes them able to predict the next word in a sentence given its preceding words.

For generation tasks, the bidirectionality feature is not quite useful while the auto-regression is essential to generate reasonable outputs by making use of the already predicted texts to generate the next words. Thus, we adopt the state-of-the-art GPT-2 [10] model in our work to generate recipes.

GPT-2 is a large casual language model that is pre-trained on large text corpora to build a probability distribution over sentences in a language. In other words, its task is to predict the next word in a sentence given its previous context and therefore generate coherent and realistic texts. The model is described as a Transformer-based but a more specific description would be decoder-only Transformer-based. The architecture of GPT-2 is a stack of semi-decoder blocks of Transformer that is illustrated on the right side of Figure1. The only difference between GPT-2 blocks and the original Transformer decoder blocks is removing the encoder-decoder self-attention layer that was responsible for paying attention to specific parts from the encoder.

As can be noted from Figure 1, the utilized attention in the decoder block after omitting the encoder-decoder layer is the masked self-attention only. Unlike the self-attention, the masked self-attention masks all the tokens to the right and thus allows the model to attend to the previous tokens only. This is the reason behind classifying GPT-2 as a unidirectional model as it is shown in Figure 6.

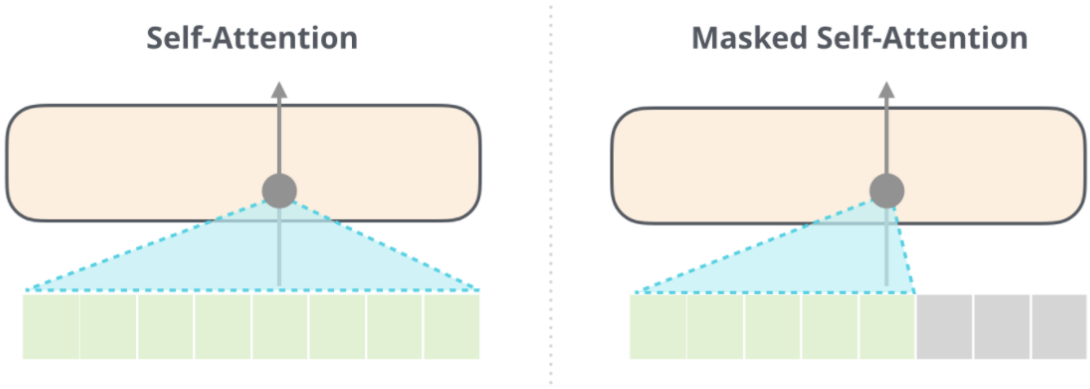

Figure 6. Comparison between self-attention and masked self-attention.

Taken from: http://jalammar.github.io/illustrated-gpt2/

A summarization of the GPT-2 model's workflow is the following. If one or more words are given to the system, embeddings of the words are passed throughout the decoder blocks where each block forwards its calculated output to the next one above. The final output is a vector of all the words the model knows which is 50K words. The values in the vector are the probabilities scores against each word in the model's vocabulary and the word with the highest probability is selected. After that, the output word is added to the input in order to better select the next word and avoid repetition.

The number of the stacked blocks varies between different versions of GPT-2 from 12 layers in the small version to 48 layers in the x-large version. In the medium version we adopted in this work, there are 24-layers, and the parameters total for 345M.

Considering the massive number of model parameters, re-training it from scratch is very costly since it requires huge amounts of power and data. Moreover, re-training the model means not taking advantage of the knowledge GPT-2 is pre-trained on with enormous text data.

Therefore, fine-tuning the model is not only feasible but a better option as well. We fine-tune different versions of GPT-2 model on our recipes dataset where each entry contains the ingredients and instructions of a single recipe.

To make use of the fine-tuned model and generate a recipe in our system, we pass the complete ingredient list form the first phase output to the fine-tuned model alongside with the average length of recipes. We then truncate the output to start from the directions section of the recipe and end with the set special token <|endoftext|>. The final output is then the new recipe instructions.

3.3 Searching for Similar Recipes in the Dataset

The final phase is a search process in the recipes dataset to find the top-k similar recipes to the generated one. Defining the similarity between recipes is not straightforward giving the large volume of the dataset and the complex structure of the recipe as a single recipe contains the ingredients names, the ingredients' amounts, and the recipe instructional steps. Therefore, using the conventional similarity metrics on the whole recipe is not a meaningful way to find the most similar recipes.

As a better solution, we define the similarity between each part separately and then combine them in a weighted average measure by following these steps:

1- We select the most similar recipes in terms of the ingredient list. The similarity between two ingredients lists is defined using Jaccard index since the order of the ingredients does not really matter. Jaccard index is defined as the rate of the mutual tokens in the two texts to the total number of unique tokens as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (24)$$

As Jaccard similarity is applied on the tokens lists of the texts, its performance depends a lot on the tokenization process. Thus, we adopt the state-of-the-art BERT tokenizer [8] to tokenize the ingredients sets of the two compared recipes before computing their similarity. As the ingredients are the most important part in defining two recipes' similarity and to reduce the time of the overall time of finding the similar recipes, we only select the top-k1 similar recipes in terms of their ingredients sets to be passed to the next steps.

2- In this second step, we define the similarity of the amounts in the compared recipes as it is the second most important part to maintain the healthiness of the suggested recipes. To define the amounts' similarities, we use the Euclidean distance between the mapped amounts of the two ingredients sets where we consider the non-existing ingredients in the target to 0. Instead of the direct Euclidean distance, we use the distance-based similarity that is given as follows:

$$\frac{1}{1 + d(A,B)} \quad (25)$$

where $\mathrm{d}(A,B)$ is the distance that is defined between two vectors as:

$$d(A,B) = \sqrt{\sum_{i \in item} (a_i - b_i)^2} \quad (26)$$

And similar to the ingredients' similarity, we select the top-k2 recipes that are the most similar ones in term of their amounts to forward them to the third step.

3- The final step in retrieving the most similar recipes is to compute the similarity of the instructional steps in recipes. To do that, we use the Cosine similarity as the sequence of the words matter in the directions, unlike the ingredients sets. A pre-step of embedding the words is required to find the Cosine similarity where we also use BERT to embed texts by first tokenizing them and then converting the tokens to IDs or numbers and then pad tokens to the right to have vectors of equal lengths. BERT embedding yields better and more efficient results in computing the similarities as it combines context information in embedding words. Cosine similarity is then defined between the resulting vectors as:

$$similarity = cos(\theta) = \frac{A.B}{||A||\,||B||} \tag{27}$$

After computing the three similarities, we combine them together to select the output recipes using a weighted average of the three similarities as follows:

$$Similarity = w_{ingredients}\, sim_{Jaccard} + w_{amounts} sim_{euclidean} + w_{steps}\, sim_{cosine} \tag{28}$$

We suggest using the 0.5, 0.3, 0.2 for weighting the similarities of ingredients, amounts, and recipes' steps, respectively. As we need to filter k=3 recipes, we also propose setting k1 as 20 and k2 as 10 to filter k1 recipes based on their ingredients' similarity and k2 recipes based on their amounts' similarity.

Chapter 4

Experiments and Evaluation

4.1 Dataset

After deeply searching for most of the available recipes' datasets, we found that they mostly do not include users' personal information or any entries about their ratings and preferences. Some works in the field collect their datasets manually or crawl the popular recipes websites. However, the resulting datasets are either less informative or not large enough to study the recipes and users well because crawling a good dataset requires a lot of time and resources. As a result, we chose to adopt the dataset provided by [66] to evaluate our proposed framework. To the best of our knowledge, this is the only suitable dataset that covers the users' historical interactions in addition to the recipe data.

The dataset has been collected from Food.com[2] and features more than 230K recipe texts and over 1M+ user interactions. We pre-filter the dataset to include only recipes with at least 3 instructional steps and ingredients between 4 and 20. We also discard users with less than 5 reviews. This pre-filtering results in 213K+ recipes and 15K+ users. There are about 14K+ unique ingredients across the filtered dataset. The average number of words in recipe steps is 102 words.

[2] https://www.food.com/

For a better evaluation, we split the dataset into training with 80% of the data and testing with the remaining 20%. However, different ways to pre-process and use the dataset are used for each sub-task in the framework. Thus, more details about each are provided in the subsections below.

4.2 Evaluation on Completing the Ingredient List

In this section, we provide the experiments' details and evaluation results on the three phases of completing the ingredient list.

4.2.1 Evaluation on the Preferences Component

Data Pre-Processing: To build the recommender system on ingredients, we utilize the user-ingredient interactions data we built as explained in the methodology. The resulting data contains over 2M+ interactions.

Python Library: We use Microsoft's Recommenders[3] library for NeuMF implementation.

Metrics: Three different metrics are used to evaluate the recommender system where all of the metrics are based on a defined rank threshold of k = 10:

- Precision: which computes the percentage of relevant items in the top k items
- Mean Average Precision (MAP): this metric computes the precision@K for each rank position k less than the threshold. After that, it returns the average of them
- Normalized Discounted Cumulative Gain (NDCG): which measures the ranking quality by assigning weights based on items' positions within the top k suggestions

[3] https://github.com/microsoft/recommenders

Experimental details and Results:

We train NeuMF model on our user-item interactions dataset with 4 latent factors and 3 layers in its MLP for 100 epochs. Evaluating the implemented NeuMF recommender system on our dataset returns the following values:

- Precision@K: 0.43
- MAP@K: 0.15
- NDCG@K: 0.50

4.2.2 Evaluation on the Compatibility Component

This sub-section provides a detailed explanation of the experiments in addition to the results of the three suggested methods for completing the ingredient list.

4.2.2.1 Association Rules Evaluation

Data Pre-Processing: Prior to applying the association rules solution, the dataset had to be formed as a list of hot-encoded transactions. Thus, we transform each recipe's ingredient list to a hot-encoded vector of the unique ingredients' length where a value of 1 represents the existing ingredients and a value of 0 represents any other ingredient.

Python Library: We use Mlxtend[4] library for Apriori implementation and extracting association rules.

Thresholds Setting: The maximum length of the generated itemsets is 4 because 3 is chosen as the number of the given ingredients and rules with no more than 3 ingredients in the antecedent

[4] http://rasbt.github.io/mlxtend/

side is needed and finding itemsets of a larger length is very difficult given the large dataset and limited resources. We also did extensive experiments on generating association rules to set suitable thresholds of itemsets support and rules confidence. As we need to guarantee to generate a list of compatible ingredients even when an infrequent ingredient is given, we set a low support threshold to cover most of the ingredients in the extracted itemsets. After some experiments, we chose the value 0.000004 as a minimum support threshold. In regard to the association rules metric, we use the confidence measure. To set a confidence threshold that results in meaningful yet enough ingredients, we generate the rules of confidence thresholds between 0.6 and 0.2. Table 1 is an example of finding the compatible ingredients with {eggs, chocolate, flour} using the rules extracted with each threshold. Since we need to get enough ingredients to complete an ingredient list with, we define the confidence threshold as 0.2. The resulting rules are still meaningful as the used support threshold is very low.

Table 1. An example of completing ingredients using association rules method

Confidence threshold	**Resulting ingredients given {eggs, chocolate, flour}**
0.6	{salt, butter, sugar}
0.5	{salt, butter, sugar}
0.4	{salt, butter, sugar, baking powder}
0.3	{salt, butter, sugar, baking powder, vanilla}
0.2	{salt, butter, sugar, baking powder, vanilla, baking soda, milk, brown sugar, cinnamon, all-purpose flour, vanilla extract}

Discussion: Although the association rules method is slow and computationally expensive on a large dataset, it is a straightforward and interpretable method. Moreover, it yields interesting analytical information about the dataset such as the top frequent and infrequent ingredients which is given in Table 2.

Table 2. The top-10 frequent and top-10 infrequent ingredients

Top-10 frequent ingredients	Salt, butter, sugar, onion, eggs, water, olive oil, flour, garlic cloves, milk
Top-10 infrequent ingredients	Land o lakes roasted garlic butter with oil, kala namak, kala jeera, kahlua-flavored syrup, jumbo male blue crabs, jonathan apple, Jonagold apples, jolly rancher candies, join of beef, Johnsonville hot Italian sausage links

4.2.2.2 Deep Neural Network Evaluation

Data Pre-Processing: We also use the same hot-encoded vectors of ingredient lists as described in association rules evaluation to train and evaluate the neural network model. To create the input-output tuples, we randomly pick three ingredients of each ingredient list and create a hot-encoded vector to be the input, while the output is the original hot-encoded vector with all the ingredients in the list activated. We split the dataset into training with 80% of the data and testing with the remaining 20%.

Python Library: We use PyTorch[5] library for building the neural network model and scikit-learn [6] for one of the evaluation metrics.

Metrics: To evaluate the neural network evaluation and compare the different structures and parameters, we use a metric called Average precision (AP) that computes the average precision from prediction scores and outputs a value between 0 and 1 where a higher value is better. The metric is proved to not be overly optimistic with the evaluation and to be good for evaluating multi-label classification problems.

In addition to the AP score, we follow the previous works in examining the ingredients' completion method as followed. First, we remove one of the ingredients and use test the model's ability to retrieve it using:

- Rank <=10: which is the percent of predicting the removed ingredient in the top-10 predictions
- Mean rank: which is the average of the removed ingredients ranks in predictions
- Median rank: which is the median of the removed ingredients ranks in predictions

Experiments: We did extensive experiments to explore the best model structure and hyperparameters. For all the experiments, we use 100 epochs and avoid the random initialization of parameters by using a uniform distribution known as He initialization instead.

We explored a series of hidden layer structures including their number and their nodes number and the results are listed in Table 3.

[5] https://pytorch.org/
[6] https://scikit-learn.org/stable/

The 3 layers with 100 in each is the best structure as the values of mean and median are less than the others while the rank <=10 is the same as the 2 layers structure and the average precision is very close.

Table 3. Comparison of hidden layer structures in neural network model evaluation

Hidden Layer Structure	**Mean**	**Median**	**Rank <=10**	**Average Precision (AP)**
2 Layers: 100 nodes in each	3935.63	135	**0.24**	**0.41**
2 Layers: 1024 nodes in each	7561.37	9852.5	0.09	0.36
3 Layers: 100 nodes in each	**3056.16**	**72**	**0.24**	0.39

After choosing the structure, we did another set of experiments to choose the suitable activation function. Table 4 records the results to compare.

Table 4. Comparison of activation functions in neural network model evaluation

Activation Function	**Mean**	**Median**	**Rank <=10**	**Average Precision (AP)**
ReLU	3935.63	**135**	**0.24**	**0.41**
Sigmoid	**2876.36**	188	0.22	**0.41**

Although the results are very close in general, ReLU is better in terms of median and rank <=10 so we choose it to complete our experiments.

In order to choose the loss function, we compare Binary Cross Entropy with Logit Loss and another function called Multi-Label Soft Margin Loss. The formulas of them and our results state that they return almost the same results in a general setting. Thus, we complete the experiments with Binary Cross Entropy with Logit Loss.

To choose the optimizer, we compare Adam with Stochastic Gradient Descent (SGD) and the results are listed in Table 5.

Table 5. Comparison of optimizers in neural network model evaluation

Optimizer	**Mean**	**Median**	**Rank <=10**	**Average Precision (AP)**
SGD	5094.74	4343	0.02	0.14
Adam	**3056.16**	**72**	**0.24**	**0.39**

As shown in the results above, there is a big drop in performance when using SGD over Adam. We even did further experiments by changing the momentum values and learning rates in SGD but the results were always worse than Adam.

The final experiment we did on the neural network was to decide about the learning rate and the usefulness of dropout regularization. We find out that the learning rate of 0.001 results in much better performance over bigger values. Regarding the dropout regularization, we also

found that using a 0.2 dropout value is much better than removing the dropout component from the neural network.

4.2.2.3 BERT Evaluation

Data Pre-Processing: For this method, we use raw text data. We list the ingredient lists for all the recipes to create our corpus. Similar to neural network model evaluation, we then split the corpus into training with 80% of the data and testing with the remaining 20%.

Python Library: We use HuggingFace's Transformers[7] library to fine-tune and use BERT model. As one of PyTorch or TensorFlow libraries should be chosen as a base to Transformers library, we choose PyTorch.

Metrics: As BERT is a language model, we use the most frequently used metric for evaluating a language model which is perplexity. Perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It is mathematically defined for a sequence $X = (x_0, \ldots, x_t)$ as:

$$PPL(X) = exp\{-\frac{1}{t}\sum_{i}^{t} log\ p_\theta(x_i|x < i)\} \quad (29)$$

where $log\ p_\theta(x_i|x < i)$ is the language model's log-likelihood of the i[th] token conditioned on the preceding tokens $x < i$. A lower perplexity score indicates better performance.

Additionally, we plan to use the same metrics of rank <=10, mean, and median in order to compare the model with others. However, the mean and median rank are not meaningful

[7] https://huggingface.co/transformers/

metrics in this case as the length of the predicted words from a BERT model does not equal the unique ingredients but the model vocabulary list. To recapitulate, we use Perplexity and rank <= 10 metrics to evaluate BERT model performance.

Experiments: As mentioned in the methodology section, we did multiple experiments to explore different rates of masked words and the importance of positional encoding embeddings. For all the experiments, we use 3 training epochs.

We compare the default value of masked language model probability with larger values to train the model mask and predict more messing words. However, the default value gives the best performance in terms of perplexity although the rank<=10 does not differ at all as illustrated in Table 6.

Table 6. Comparison of different mlm probabilities in BERT evaluation

MLM Probability	**Perplexity**	**Rank <=10**
default = 0.15	**12**	**0.22**
0.225	15.06	**0.22**
0.30	17.88	**0.22**

Moreover, we compare the model's performance with positional encoding as in default setting with its performance with removing this part of embedding. From the results listed in Table 7, we found a large decrease in the model's performance when removing the positional encoding which is unlike what we expected.

Table 7. Comparison of including/excluding the positional encoding in BERT evaluation

Positional encoding	Perplexity	Rank <=10
With positional encoding as default	**1.60**	**0.26**
Without positional encoding	12	0.22

4.2.2.4 Comparison

In this section, we answer the following question: "*How do the suggested methods for completing ingredients lists compare with the most popular baseline and with each other*?".

The baseline we adopt is the Non-negative matrix factorization-based method described in [57] where they first decompose the recipe-ingredient matrices into two low-rank matrices containing k latent features and then use the two matrices to construct the coefficient matrix needed for completing a partial ingredient list into a complete one. This method is the first and the most popular one to compare with in ingredient completion works.

We planned to use the metrics of rank <=10, mean, and median to compare all methods. However, we could not evaluate the association rules this way due to the model's slowness although its effectiveness in resulting in reasonable outputs. In addition to that, the mean and median are not meaningful in the case of BERT model as we have mentioned before. Thus, we listed all other possible results in Table 8. Unlike the related previous works, we tested all the suggested model on a test split of the dataset that differs from the part we train the model on to evaluate the models' generalization abilities.

Table 8. Comparison of all the suggested methods for ingredients completion

Model	**Mean**	**Median**	**Rank <=10**
Baseline (NMF)	7335.18	9296	0.14
Neural Network	**3065.16**	**72**	0.24
BERT	--	--	**0.26**

As seen from the results, BERT gains the best performance over all other methods. Not to mention that BERT model's ability to retrieve missing words is enhanced using the predicting method we used. Neural networks solution is still than the baseline with a large margin between them. The performance of the third suggested solution of using association rules is also good and guarantees to not produce random results in spite of its slowness. The performance of all the methods can be further improved if a smaller dataset is used or if additional pre-processing is done before testing.

A couple of examples of finding the top-10 compatible ingredients with a given list using the three suggested method are listed in Table 9 below.

Table 9: Examples of completing ingredients using the three suggested methods

Partial List	Candidates from AR	Candidates from NN	Candidates from BERT
eggs, chocolate, flour	butter, sugar, salt, baking powder, vanilla, baking soda, milk, vanilla extract, brown sugar, egg	oil, vanilla, butter, powdered sugar, baking powder, whipping cream, zucchini, heavy cream, rum, sour cream	butter, sugar, vanilla, salt, milk, nuts, cinnamon, coconut, bananas, dates
Corn, cheese, mayonnaise	onion, salt, sour cream, water, butter, milk, black beans, tomatoes, chili powder, ground beef	cheddar cheese, red onion, tomatoes, plain yogurt, lemon juice, egg, ground turkey, fresh basil	bacon, onion, salt, pepper, eggs, milk, bread, butter, ham, tomatoes

4.2.3 Evaluation on the Healthiness Component

Data Pre-Processing: To learn predicting the amounts of the ingredients in the first step in this phase, we re-crawl Food.com website and parse their web pages to extract the corresponding amounts. We then construct an input-output tuple for each recipe in the dataset in a similar manner to the one used in the neural network solution for finding compatible ingredients. Similar to other tasks, we split the data to train and test with 8:2 ratio.

For extracting the nutritional values of each ingredient, we use the Canadian Nutrient File (CNF) as mentioned before. Although the dataset contains information about a lot of food items, some ingredients could not be found by a simple string matching. Thus, we use Jaccard index to define the similarity between each ingredient and all the food items in the dataset and map the ingredients with the most similar food item to easily calculate its calories.

Python Library: We also use Pytorch library to build and train the amounts neural network model and Beautifulsoup[8] to parse the web pages.

Metrics: To evaluate the neural network performance, we use the Mean Squared Error (MSE) metric that is given as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2 \tag{30}$$

Experiments: To evaluate and compare the performance of the neural network in predicting the amounts, we did many experiments on the structure and the hyperparameters. We explore different structures including one, two, and three hidden layers with different neurons numbers {64, 100, 128, 512}. Another set of experiments are conducted on the activation functions of ReLU and LeakyReLU and on the optimizers including RMSprop, Adam, and SGD with learning rates {0.1, 0.01, 0.001}. Since the experiments are very similar to the ones of the neural network solution for completing the ingredient list, we avoid adding more details. The lowest train MSE score is 4.68 while the lowest test MSE score is 5.70.

[8] https://www.crummy.com/software/BeautifulSoup/

Regarding the candidates adding methods, we compare the final calories sum of the selected ingredients in both methods. In over 40% of the cases, both models generate the same calories' count. However, about 50% of the time, the second method generates a lower number of calories. We claim that although that, the first method would result in more reasonable outputs as the second method tends to choose the ones with the smaller amounts while the first focuses on the most compatible ingredients.

4.3 Evaluation of Recipe Generation

Data Pre-Processing: In this phase, we evaluate the recipe generation using the raw text data where all the recipes are listed together. Each recipe contains the ingredient list followed by the recipe's directions and ends with the special token <|endoftext|>. Similar to previous methods, we split the corpus with a ratio of 20% of training to test data.

Python Library: Like BERT evaluation, we use HuggingFace's Transformers library to fine-tune different versions of GPT-2 and other models.

Metrics: Text generation evaluation is deemed one of the hardest problems that is itself another research area. Therefore, we use the most popular metrics in related works. Namely, perplexity, Bilingual Evaluation Understudy (BLEU) [68], and ROUGE [69]. In addition to them, we use the newly suggested metric of BLEURT [70] for a better evaluation of text generation:

- Perplexity: Perplexity is the measurement we previously mentioned in BERT evaluation. It is a metric of how well a probability distribution predicts a sample

- BLEU: BLEU measures the closeness of a generated sentence to a reference sentence using a modified version of precision. In other words, it counts the matching tokens between the generated and the reference texts. BLEU values are in the range of 0 and 1 where a larger value means the model's outputs are closer to the reference and hence better
- Recall-Oriented Understudy for Gisting Evaluation (ROUGE): ROUGE is a set of metrics that initially meant for evaluating translation and summarization models. However, they can be also used for generation evaluation as it mainly compares a produced text against a reference text
- BLEURT: BLEURT is a novel, learned, and BERT-based metric for natural language generation evaluation. Like other metrics, its input contains two sentences: the generated and the reference one

Baselines: We compare the adopted GPT-2 medium against other versions of GPT-2. Besides GPT-2, we fine-tuned two other models that can be used for text generation:

- GPT [9]: which is the GPT-2 model's precedent that has fewer layers and parameters and pre-trained on a smaller corpus
- XLNet [22]: a generalized autoregressive and bidirectional language model

All the models are trained for 3 epochs and the results are listed in Table 10. GPT-2 medium gains the best performance overall. To enhance the model further, we fine-tune GPT-2 medium for two additional epochs.

Table 10. Comparisons of recipe generation models

Model / metric	Perplexity	BLEU-1	ROUGE-L	BLEURT
GPT	8.15	0.02	0.10	**-0.59**
XLNet	6.73	0.02	0.07	-0.67
GPT-2 small	6.12	**0.06**	0.21	-0.67
GPT-2 medium	5.10	**0.06**	**0.23**	-0.65
GPT-2 medium with 5 epochs	**4.87**	**0.06**	**0.23**	-0.63

A couple of examples of recipe generation using the fine-tuned GPT-2 medium model are listed in Table 11.

Table 11. Examples of recipe generation

Given Ingredient List	cake flour, unsweetened cocoa powder, cornstarch, baking soda, baking powder, salt, eggs, granulated sugar, mini chocolate chip, whipped topping.	salmon fillet, butter, orange zest, lemon zest, cayenne pepper, chives, oregano, salt, pepper.
Generated Recipe	preheat oven to 350 degrees mix dry ingredients in bowl in medium bowl beat eggs with mini chocolate chips on mixer at high speed add sugar and beat well add wet ingredients and beat until well mixed	preheat the oven to 400 degrees combine all the ingredients in a bowl, blending thoroughly if the dressing seems too thin, place just enough oil or water in the bowl for the desired thickness

	pour batter into greased or lined 11 x 7 -inch pan add whipped topping and let set about half way until all is spread sprinkle with remaining cup mini chocolate chips when ready to serve	pour about 1 tablespoon or so of olive oil onto the bottom of a shallow baking pan sprinkle salmon fillets evenly, making an even layer in the pan place one piece of lemon in each filet spoon an equal portion of the dressing over the fish, dividing it evenly finish with about 1 tablespoon of zest roast for 4 to 5 minutes until the fish is just cooked through and the lemon is starting to turn white slice the fish, serve with white or lemon dressing delicious!

Chapter 5

Conclusion and Future Work

In this thesis, we present a framework that combines various machine learning methods to solve the new task we propose: to generate personalized and healthy recipes given only a few key ingredients. The suggested work is implemented in three phases. The first phase completes the given partial list of ingredients in a manner that achieves the personalization, compatibility, and healthiness features in the resulting ingredients. Next, the complete ingredient list is passed to a GPT-2 based recipe generation system where the directions of an unexplored recipe are generated. Finally, the system searches the dataset for the top similar recipes in terms of the completed list of ingredients, their predicted amounts, and generated directions which guarantees to give users more plausible options to choose from.

As evident from the described experiments on a large dataset, each component in the system achieves its task successfully to build up an end-to-end system that is able to generate personalized, healthy, and reasonable recipes and therefore contributes solving a real-world problem.

In the future, we would like to advance our study on the following topics. As we struggled to find suitable recipes dataset, we think that building a clean dataset that contains all required information about recipes including their ingredients, amounts, and steps in addition to users' information that include their personal information and ratings would be beneficial for exploring and evaluating the task in a better way. Another data-related future work is to pre-process the dataset. Although we filter some recipes and users based on multiple conditions, there is a big need for a further pre-processing to filter and classify the ingredients.

In addition to the data-related improvements, we suggest extending the healthiness feature by including more nutrients in the selection. Also, adding some health rules would allow users with specific allergies and diseases to make use of such a system. Finally, including more evaluation techniques is essential in text generation tasks such as including human-based evaluation.

References

[1] A. Turing, " Computing Machinery and Intelligence," *Mind,* pp. 433-460, 1950.

[2] M. Johnson, "How the statistical revolution changes (computational) linguistics," *Association for Computational Linguistics,* pp. 3-11, 2009.

[3] Y. Goldberg, Neural Network Methods in Natural Language Processing, 2017.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research,* p. 1137–1155, 2003.

[5] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, Sanjeev Khudanpur, "Extensions of recurrent neural network language model," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[6] Martin Sundermeyer, Ralf Schlüter, Hermann Ney, "LSTM Neural Networks for Language Modeling," in *Conference of the International Speech Communication Association*, Portland, 2012.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. (2017). Attention is All you Need. *Neural Information Processing Systems (NIPS)*, 6000–6010.

[8] Jacob Devlin, Ming-Wei Chang ,Kenton Lee ,KristinaToutanova. (2019). BERT: Pre-training of Deep Bidirectional Transformers for. *Proceedings of NAACL-HLT*.

[9] Radford, Alec, Karthik Narasimhan, Time Salimans, Ilya Sutskever. (2018). *Improving Language Understanding by Generative Pre-Training.* Technical report, OpenAI.

[10] Alec Radford, Jeffrey Wu , Rewon Child, David Luan, Dario Amodei , Ilya Sutskeve. (2019). *Language Models are Unsupervised Multitask Learners.* Technical report, OpenAI.

[11] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, "Sequence to Sequence Learning with Neural Networks," in *Neural Information Processing Systems (NIPS)*, 2014.

[12] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Conference on Empirical Methods in Natural Language Processing*, Doha, 2014.

[13] Laurent Itti, Christof Koch, Ernst Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 20, no. 11, pp. 1254-1259, 1998.

[14] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *The International Conference on Learning Representations (ICLR)*, San Diego, 2015.

[15] Minh-Thang Luong, Hieu Pham, Christopher D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Lisbon, 2015.

[16] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer, "Deep Contextualized Word Representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, New Orleans, 2018.

[17] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, Russell Power, "Semi-supervised sequence tagging with bidirectional language models," in *Association for Computational Linguistics(ACL)*, Vancouver, 2017.

[18] Andrew M. Dai, Quoc V. Le, "Semi-supervised Sequence Learning," in *Advances in Neural Information Processing Systems (NIPS)*, Montreal, 2015.

[19] Jeremy Howard, Sebastian Ruder, "Universal Language Model Fine-tuning for Text Classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, Melbourne, 2018.

[20] W. L. Taylor, "Cloze procedure: A new tool for measuring readability.," *Journalism & Mass Communication Quarterly,* vol. 30, pp. 415-433, 1953.

[21] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692* .

[22] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *NIPS*.

[23] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, Qun Liu. (2019). ERNIE: Enhanced Language Representation with Informative Entities. *ACL*.

[24] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, Jimmy Lin. (2019). End-to-End Open-Domain Question Answering with BERTserini. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 72-77.

[25] Chi Sun, Luyao Huang, Xipeng Qiu. (2019). Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence. *NAACL*.

[26] Gediminas Adomavicius. Alexander Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering (TKDE),* vol. 17, no. 6, pp. 734-749, 2005.

[27] Shuai Zhang, Lina Yao, Aixin Sun, Yi Tay, "Deep Learning Based Recommender System: A Survey and New Perspectives," *Association for Computing Machinery,* vol. 52, no. 1, 2019.

[28] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al., "Wide & Deep Learning for Recommender Systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, Boston, 2016.

[29] Wei-Ta Chu, Ya-Lun Tsai, "A hybrid recommendation system considering visual information for predicting favorite restaurant," *World Wide Web (WWW),* vol. 20, p. 1313–1331, 2017.

[30] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, How Jing, "Recurrent Recommender Networks," in *Proceedings of the tenth ACM international conference on web*, 2017.

[31] Hanbit Lee, Yeonchan Ahn, Haejun Lee, Seungdo Ha, Sang-goo Lee, "Quote recommendation in dialogue," in *Proceedings of the SIGIR*, 2016.

[32] Omer Tal, Yang Liu, Jimmy Huang, Xiaohui Yu, Bushra Aljbawi. (2019). Neural Attention Frameworks for Explainable Recommendation. *IEEE Transactions on Knowledge and Data Engineering* .

[33] Seongjun Yun, Raehyun Kim, Miyoung Ko, Jaewoo Kang. (2019). SAIN: Self-Attentive Integration Network for Recommendation. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1205–1208

[34] Wang-Cheng Kang, Julian McAuley. (2018). Self-Attentive Sequential Recommendation. *IEEE International Conference on Data Mining (ICDM)*.

[35] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, Wenwu Ou. (2019). Behavior sequence transformer for e-commerce recommendation in Alibaba. *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, 1-4.

[36] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, Peng Jiang. (2019). BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* , 1441–1450.

[37] Jingxuan Yang, Jun Xu, Jianzhuo Tong, Sheng Gao, Jun Guo, Jirong Wen. (2019). Pre-training of Context-aware Item Representation for Next Basket Recommendation. *arXiv:1904.12604* .

[38] Thi Ngoc Trang Tran, Müslüm Atas, Alexander Felfernig, Martin Stettinger, "An overview of recommender systems in the healthy food domain," *Journal of Intelligent Information Systems,* vol. 50, p. 501–526, 2017.

[39] El-Dosuky, M.A., Rashad, M.Z., Hamza, T.T., & El-Bassiouny, A.H. (2012). Food recommendation using ontology and heuristics. *AMLTA, Springer, communications in computer and information science, Vol. 322*, 423–429

[40] Mouzhi Ge, Mehdi Elahi, Ignacio Fernández-Tobías, Francesco Ricci, David Massimo. (2015). Using Tags and Latent Factors in a Food Recommender. Proceedings of the 5th International Conference on Digital Health, 105- 112.

[41] Yehuda Koren, Robert Bell, Chris Volinsky, "Matrix factorization techniques for recommender systems," *COMPUTER,* vol. 42, pp. 30-37, 2009.

[42] Freyne, J., Berkovsky, S. (2010). Intelligent food planning: personalized recipe recommendation. *Proceedings of the 15th international conference on Intelligent user interfaces, ACM*, 321–324

[43] Mayumi Ueda, Syungo Asanuma, Yusuke Miyawaki, and Shinsuke Nakajima. (2014). Recipe Recommendation Method by Considering the User's Preference and Ingredient Quantity of Target Recipe. *Proceedings of the International MultiConference of Engineers and Computer Scientists* , 519-523.

[44] Lipi Shah, Hetal Gaudani, Prem Balani. (2016). Personalized Recipe Recommendation System using Hybrid Approach. *International Journal of Advanced Research in Computer and Communication Engineering*, 192-197.

[45] Chun-Yuen Teng, Yu-Ru Lin, Lada A. Adamic. (2012). Recipe recommendation using ingredient networks. *Proceedings of the 4th Annual ACM Web Science Conference*, 298–307.

[46] Tsuguya Ueta, Masashi Iwakami, Takayuki Ito. (2011). A recipe recommendation system based on automatic

nutrition information extraction. Proceedings of the 5th international conference on knowledge science,

engineering and management, Springer-Verlag, 79-90.

[47] David Elsweiler, Morgan Harvey, Bernd Ludwig, Alan Said. (2015). Bringing the “healthy” into Food Recommenders. *Ge, M., & Ricci, F. (Eds.) DMRS, CEUR-WS.org, CEUR workshop proceedings, 1533*, 33-36.

[48] Mouzhi Ge, Francesco Ricci, David Massimo. (2015). Health-aware Food Recommender System. *Proceedings of the 9th ACM Conference on Recommender Systems*, 333–334.

[49] Morgan Harvey, David Elsweiler. (2015). Automated recommendation of healthy, personalised meal plans. *Proceedings of the 9th ACM conference on recommender systems, ACM*, 327–328.

[50] Raciel Yera Toledo, Ahmad A. Alzahrani, Luis Martínez. (2019). A Food Recommender System Considering Nutritional Information and User Preferences. *IEEE Access, 7*, 96695-96711.

[51] Mehdi Elahi, Mouzhi Ge, Francesco Ricci, David Massimo, Shlomo Berkovsky. (2014). Interactive Food Recommendation for Groups. *RecSys*.

[52] Satoshi Yokoi ; Keisuke Doman ; Takatsugu Hirayama ; Ichiro Ide ; Daisuke Deguchi ; Hiroshi Murase, "Typicality analysis of the combination of ingredients in a cooking recipe for assisting the arrangement of ingredients," in *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, Turin, 2015.

[53] Suyash Maheshwari, Manas Chourey, "Recipe Recommendation System using Machine Learning Models," *International Research Journal of Engineering and Technology (IRJET),* vol. 6, no. 9, pp. 366-369, 2019.

[54] Marlies De Clercq, Floris Ramon, Bernard De Baets, Willem Waegeman, "Replacing a food allergen in a recipe using data-driven methods," Department of Mathematical Modelling, Statistics and Bioinformatics, Ghent University, Ghent, Belgium, 2016.

[55] Elizabeth Gorbonos , Yang Liu , Chính T. Hoàng, "NutRec: Nutrition Oriented Online Recipe Recommender," in *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, Santiago, 2018.

[56] Meng Chen, Xiaoyi Jia, Elizabeth Gorbonos, Xiaohui Yu, Yang Liu. (2019). Eating healthier: Exploring nutrition information for healthier recipe recommendation. *Information Processing & Management*.

[57] Marlies De Clercq, Michiel Stock, Bernard De Baets, Willem Waegeman, "Data-driven recipe completion using machine learning methods," *Trends in Food Science & Technology,* vol. 24, pp. 1-13, 2016.

[58] Daniel D. Lee, H. Sebastian Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature,* vol. 401, p. 788–791, 1999.

[59] Toon Calders, Floriana Esposito, Eyke Hüllermeier, Rosa Meo, "Machine Learning and Knowledge Discovery in Databases," in *European Conference, Machine Learning and Knowledge Discovery in Databases (ECML)* , Nancy, 2014.

[60] Paula Fermín Cueto, Meeke Roet, Agnieszka Słowik, "Completing partial recipes using item-based collaborative filtering to recommend ingredients," arXiv preprint arXiv:1907.12380

[61] Muhammad Shihab Rashid. Quazi Mishkatul Alam, Marc Giannuzzi, Quentin Lacroix, Kristrian Tram, "Health Based Ingredient Recommender System for Recipes," Department of Computer Science and Engineering, University of California Riverside.

[62] Amaia Salvador, Michal Drozdzal, Xavier Giro-i-Nieto, Adriana Romero. (2019). Inverse Cooking: Recipe Generation From Food Images. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 10453-1046

[63] Chloe Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, Yejin Choi. (2015). Mise en Place: Unsupervised Interpretation of Instructional Recipes. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 982–992.

[64] Chloe Kiddon, Luke Zettlemoyer, Yejin Choi. (2016). Globally Coherent Text Generation with Neural Checklist Models. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 329–339.

[65] Bostan, L. A. (2017). *Ingredient-driven Recipe Generation Using Neural and Distributional Models.*

[66] Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, Julian McAuley. (2019). Generating Personalized Recipes from Historical User Preferences. *EMNLP*]

[67] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, Tat-Seng Chua, "Neural Collaborative Filtering," in *Proceedings of the 26th International Conference on World Wide Web*, Perth, 2017.

[68] Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu, "Bleu: a Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, 2002.

[69] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Text Summarization Branches Out, Association for Computational Linguistics (ACL)*, Barcelona, 2004.

[70] Thibault Sellam, Dipanjan Das, Ankur Parikh, "BLEURT: Learning Robust Metrics for Text Generation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics (ACL)*, Online, 2020.