Automatic Extraction of Nutrition Information from Nutrition Strings

Anonymous ACL submission

Abstract

Finding the right food in a supermarket for someone's dietary needs is challenging due to the large variety of food products. One possible solution is to build a nutrition information dataset and a food search engine. This engine would allow consumers to find their desired food product by placing constraints on the nutrition factors and ranking the obtained results based on their criteria. However, collecting nutrition information for tens of thousands of food products is time-consuming, and an automatic method is desired. This paper investigates the problem of automatic extraction of nutrition information from nutrition strings. For this purpose, it introduces a dataset of nutrition strings collected from different websites and their corresponding nutrition information. The nutrition extraction problem can be viewed as a slot filling problem, and two transformer-based methods from the literature are evaluated. The paper also introduces a specialized algorithm based on dynamic programming, and evaluates it as well as the transformer-based methods and GPT-4 and GPT-4o, with encouraging results.

1 Introduction

The United States and the entire world are facing an obesity epidemic, with more than 42% of the adults in the USA being obese. Obesity is strongly associated with heart disease, diabetes, high blood pressure, and even certain forms of cancer.

Consumers have a hard time losing weight, in part because it is hard to find what foods fit their dietary needs from the thousands of food items from supermarkets by examining their nutrition information, one item at a time. It would be therefore desirable that the nutrition information of all products from supermarkets be organized into a searchable database where consumers could quickly search for products satisfying their particular criteria. However, building such a database requires the extraction of nutrition information from different websites containing the nutrition information of the food products, either as a text string or as a picture, or both.

This paper focuses on the problem of extracting the nutrition information values of food products from text strings. Given a string containing the nutrition information (a nutrition string), the problem is to automatically extract the values of the desired fields such as Calories, Total Fat, Saturated Fat, Sodium, Protein, Fiber, etc.

As such, this problem can be regarded as the intent classification task of a slot filling problem. In NLP, the classification task in slot filling is to extract the values of well-defined slot types of given queries. It is referred to as the process of retrieving information in order to convert the user intent into explicit instructions. This project can be seen as a slot filling problem where the slot types are the nutrition items.

For this purpose, two popular slot filling methods are investigated: JointBERT (Chen et al., 2019) and CTRAN (Rafiepour and Sartakhti, 2023), trained on our nutrition string data with slot labels. For instance, the text "Calories 90" is expected to be labeled as "Calories_item Calories_value," which offers a clear matching between the nutrition item "Calories" and its value "90.".

However, there are still several limitations of these slot filling labeling methods, such as increased training time for longer strings. For that reason, this paper also introduces in Section 4 a novel method based on dynamic programming to address the same problem.

In conclusion, this paper brings the following contributions:

- It proposes a new knowledge extraction problem, the extraction of nutrition information values from nutrition strings, which has not received much attention in the literature.
- It introduces a dataset of 26,165 nutrition strings collected from various websites, to-

gether with their associated ground truth nutrition values.

- It shows how this problem can be viewed as a slot filling problem, evaluating state-of-the-art slot filling transformers such as JointBERT (Chen et al., 2019) and CTRAN (Rafiepour and Sartakhti, 2023).
- It introduces a novel dynamic programming-based algorithm for addressing the nutrition extraction problem.
- It performs experiments on the newly introduced nutrition dataset, evaluating the proposed DP-based method, the JointBERT and CTRAN slot filling methods as well as GPT-4 and GPT-4o, which can also be used to extract information from nutrition strings.

The full dataset and the code used in this paper will be shared on GitHub.

1.1 Related Work

Related work can be divided into work on nutrition extraction datasets and slot filling methods.

Nutrition extraction datasets. To our knowledge, there exists no nutrition extraction dataset similar to the one introduced in this paper, containing nutrition strings and associated ground truth values. FoodDB (Harrington et al., 2019) introduces a database of UK foods and their associated nutrition information and chemical composition. The data is in the form of an online table containing the chemical compound and nutrition information of foods, without the nutrition strings. The data was collected from web pages and snapshots based on Optical Character Recognition (OCR). However, the dataset contains generic information such as protein content in breakfast cereal and pasta rather than on specific food products.

Slot filling methods. There has been quite a lot of research addressing the slot filling problem. Examples include recurrent neural network (RNN) based methods such as (Mesnil et al., 2015; Liu and Lane, 2016; Wang et al., 2018), convolutional neural network (CNN) based methods such as (Xu and Sarikaya, 2013; Vu, 2016), and also the Transformer encoder-based BERT models (Chen et al., 2019), and Transformer-based CNN methods (Rafiepour and Sartakhti, 2023).

There are several Spoken Language Understanding (SLU) datasets widely used for evaluating slot filling methods. A related dataset called SLURP (Bastianelli et al., 2020) is a collection of audio recordings. Anther dataset, Snips (Coucke et al., 2018), comes from a voice platform that includes queries with the intent outputs, such as "PlayMusics" and "GetWeather". The ATIS dataset (Hemphill et al., 1990) is a series of strings for a flight-booking system, which has higher similarity outputs than the other two datasets. Both the Snips and ATIS datasets are widely used in Slot Filling research.

Dataset	Size	Length Mean	Length Max	Slots	Slots per String
Snips	14484	9.0	35	72	4.6
ATIS	5871	11.1	46	120	4.1
NutriX (ours)	26164	244.0	1209	78	42.6

Table 1: A comparison of some existing Slot Filling datasets and our dataset.

A data format similar to (Goo et al., 2018) for the ATIS dataset was used in this paper to label the nutrition extraction dataset for slot filling. One of the challenges was the length difference between the strings of these datasets and the nutrition strings. The nutrition strings are much longer (most of them over 200 words per string), and there are more slot labels in each string, which may lead to much longer computational time for training.

In Table 1 are shown some statistics about two existing datasets, Snips (Coucke et al., 2018) and ATIS (Hemphill et al., 1990), in comparison with the proposed NutriX dataset. In this table, the size column represents the number of strings in the each dataset. The mean and maximum string length are shown in the Length Mean and Max columns respectively. Slots represents the total number of labeled categories in each dataset. Slots per String is the average number of slots in each string. From Table 1 one could see that the Nutrix dataset is quite challenging having longer strings and more slots per string than the other two datasets.

2 Constructing the Nutrition Extraction Dataset

The nutrition extraction dataset consists of a number of nutrition strings and their associated nutrition information. It is organized as a table with different food products as rows, and the nutrition string and corresponding nutrition items as columns. An example of a nutrition string is given in Figure 1.

The dataset construction can be divided into three main tasks:

1. Collection of nutrition strings from websites.
2. Semi-automatic construction of the ground truth table based on the nutrition strings.
3. Manual verification and correction of the constructed nutrition table.

Nutritional InfoIngredientsAllergensNutritional InformationServing Size: 125Calories: 90 Calories From Fat: 35Amount Per ServingPercentage Daily ValueCalcium8%Vitamin A15%Vitamin C30%Iron 4%Percentage Amount of Calories allergenno allergen information90calCalories4gTotal Fat 3g Fiber 3g Protein 410 mg Sodium

Figure 1: Nutrition string example.

These tasks will be discussed in the following subsections, together with the challenges induced by the variability of the collected nutrition strings.

2.1 Nutrition String Collection

The workflow for collecting nutrition strings from the web is illustrated in Figure 2.

Figure 2: Workflow for collecting nutrition strings.

The nutrition strings were collected from the manufacturers or retailers' websites. A web scraping technique was used to collect data from these websites with the Requests (Foundation, 2024) and Selenium (Muthukadan, 2024) Python libraries. As many websites were built using Asynchronous JavaScript and XML (AJAX) techniques, a web driver was used to control the browser in a Python environment and collect the information from the response sent by the server. Web pages written in AJAX can change only parts of the page without reloading the whole page as the page is scrolled down and new content is loaded from the website.

As shown in Table 2, four datasets were collected from different sources for a total of 26,165 strings.

The 'Manufacturers' dataset comes from more than 20 manufacturers, with 2,117 strings. Each manufacturer has a different string format, and for this reason, it is the most complex and challenging of the four datasets. The 'TraderJoes' dataset with 1,166 strings, the 'Publix' dataset with 16,848 strings, and the 'Target' dataset were acquired from the retailers' website by web scraping. The 'TraderJoes' dataset is collected from its own website (TraderJoes, 2025), while 'Publix' and 'Target' are from the Instacart website (Instacart, 2025). It was easier to extract the nutrition items and values from strings in these three datasets than from the 'Manufacturers' dataset. The 'Manufacturers' dataset contains food product information with strings without line endings between nutrition items, while the other three datasets have strings with new lines to separate the nutrition items, making it easier to label these datasets based on the line-ending information automatically.

dataset	size	length mean	length max	source
Manufacturers	2117	115	1194	Manufact
TraderJoes	1165	91	426	Retailer
Publix (Instacart)	16848	281	1209	Retailer
Target (Instacart)	6034	119	151	Retailer
Combined	24047	231	1209	Retailer
NutriX	26164	244	1209	all

Table 2: The nutrition datasets and their sources. The length mean and max in this table are the average and max word counts for the strings in each dataset.

2.2 Challenges

Extracting nutrition information from the string data faces several challenges that need to be overcome. An example of a raw nutrition string was given in Figure 1. In this string, words are not always separated by space. For instance, the word "InfoIngredientsAllergensNutritional" should be separated into 4 words. In this case, words can still be split through the capital letters.

Generally, the nutrition values and the units of measure are placed after the nutrition item keyword in the nutrition string, but there are still some exceptions. For example, at the end of this text, the measure "90" is right before the item "Calories," and the measure "4 gram" is also located before the item "Total Fat".

Nutritional InfoIngredientsAllergensNutritionalInformation Serving Size: 125Calories: 90 Calories From Fat: 35Calcium 8%Vitamin A15%Vitamin C30%Iron 4% information90cal Calories4gTotal Fat 3g Fiber 3g Protein 410 mg Sodium

Figure 3: Nutrition string split example 1

In Figure 3 is shown an example of how a string is expected way to be split. Blue words are nutrition items such as Calories and Protein, red words are values, green words are the unit of measures, and black words should be ignored. Each nutrition item should be matched to one value combined with their unit, before or after the item.

Figure 4 is another nutrition string example. The measures are more complex than in Figure 3. For instance, the item "Fat" has the value "0.5g" and also "1 %" after that, which means 0.5 grams is 1% of the daily value of "Fat" given by the FDA. The value "2,000" here is not supposed to match any nutrition item even though it has the nutrition item keyword "calories" after it. The values and their percentage value should also be combined together and matched to a single item.

In summary, extracting nutrition information

Nutrition Facts 3.5 Servings Per Container Serving Size 1/2 cup(122g) Amount Per Serving Calories 45* Fat 0.5g 1% Saturated 0g 0% Trans 0g Polyunsaturated 0g Monounsaturated 0g Cholesterol 0mg 0% Sodium 5mg 0% Total Carb 10g 4% Fiber 3g 10% Total Sugars 5g Incl. 0g Added sugar Protein 1g Vitamin D 0mcg Calcium 20mg Iron 1mg Potassium 330mg Vitamin A 950mcg * The (DV) tells you how much a nutrient in a serving of food contributes to a daily diet. 2,000 calories a day is used for general nutrition advice.

Figure 4: Nutrition string split example 2

from nutrition strings faces several challenges:

1. The name of a nutrition item in the nutrition string differs from the column name in the database. For example, the name in the database is “total fat” but the corresponding name in the nutrition string is “fat”.
2. Different names of serving sizes such as pieces, tsp, and varying units of measure, e.g., mL, g, mg, oz, %, etc. Also different versions of the same unit of measure: oz, floz, fl oz, fl oz (US Ounce).
3. The placement of the associated value before or after the nutrition item name, e.g., “Calories: 90” or “Includes 0g Added Sugars”.
4. No space between nutrition items and numbers, e.g., “3gProtein410mgSodium”.
5. Nutrition values can be duplicated with percentage measures and should be combined together. For example, ”Carbohydrates 36 g 13 %” should be combined as a single item, which means the product contains 36 grams of carbohydrates, which is 13 % of the daily value suggested by the FDA.
6. False matching of some nutrition items with others, e.g., incorrectly matching ”Calories from Fat” with ”Calories” and Fat”.

2.3 Constructing the Ground Truth Table

For the TraderJoes, Publix, and Target datasets, the strings were collected in two ways: one in which the nutrition items are separated by the end-of-line symbol "\n" and one as strings without end-of-line. The end-of-line format is an aid to generate the ground truth table for training and evaluation because each nutrition item and their measures with units are located on a single line. This feature is specific to the TraderJoes, Publix, and Target datasets, but not to the Manufacturers dataset, and can be exploited to generate the ground truth for these three datasets. However, in Section 5, we will evaluate methods that are capable of processing all datasets, where each string is given as a single line.

The ’Manufacturers’ dataset is collected as strings without any line breaks between nutrition items. The ground truth table for this dataset was constructed manually with the help of ChatGPT.

Dataset Verification. The verification of the correspondence between the nutrition strings and the ground truth values for the four datasets was done manually, by visual inspection of the strings and their associated values.

3 Slot Filling Methods for Nutrition Information Extraction

This section poses the extraction of nutrition information as a slot filling problem and shows how to train deep learning models for this purpose.

Slot filling can be seen as the classification of an input string for subsequent extraction of the corresponding information. For example, from the string "Arrive in Atlanta on December 20th" for a ticket booking system, the keyword "Atlanta" should be placed in the predefined slot "Destination" and "December" and "20th" would be placed into the "Time of Arrival" slot by the algorithm. In this case, keywords "Arrive" and "on" will be put into the slots "Other" or "0".

Compared to standard slot filling problems, such as the ticket booking example above, the problem of nutrition information extraction has its own challenges because the input string is much longer (could be even 1000 words or longer) and the number of slots is quite large (78 slots).

The example above showed how the slot filling method was used to predict the intent of the input query. In this project, the intent is clear and the goal will be the prediction of slot labels.

The models used in this paper are JointBERT (Chen et al., 2019) and CTRAN (Rafiepour and Sartakhti, 2023), both having great slot label prediction performance on the Snips (Coucke et al., 2018) and ATIS (Hemphill et al., 1990) datasets.

3.1 Data Preprocessing

One major challenge for applying slot filling methods is the lack of spaces between words, nutrition items, and numbers. Capital letters are good references for inserting spaces between words. For instance, in Figure 1, "Nutritional InfoIngredientsAllergens" can be split into four words.

In this paper, spaces are inserted before capital letters and punctuations, e.g., ":", and before and after numbers, to split the strings into words. Other methods, such as tokenizers or Bayesian optimization (Moss et al., 2020) could be used to separate words in strings. A pretrained BERT tokenizer, such as (Moi and Patry, 2023), can also split words

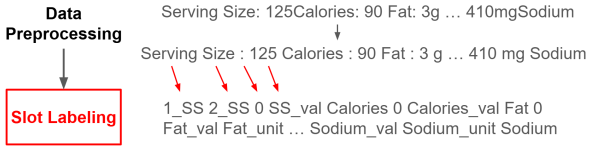

Figure 5: Labeling strings for Slot Filling training

based on spaces and capital letters.

3.2 Labeling Strings for Slot Filling Training

A special type of labeling needs to be obtained for training slot filling neural networks, in which each word needs to be labeled, as illustrated in Figure 5. The slot labels contain the nutrition items, values, and their units of measure. The irrelevant words are labeled with '0'. The keyword "Serving Size" is a nutrition item with two words; therefore it is labeled as (1_SS) and (2_SS). In the example string in Figure 4, the substring "Fat 0.5g 1%" contains the keyword 'Fat', measure '0.5', unit of measure 'g', and the percent Daily Value (%DV) '1%', which will be defined as four slot labels, 'Fat', 'Fat_val', 'Fat_unit', and 'Fat _dv'.

The slot labels for each nutrition string in TraderJoes, Publix, and Target datasets were generated from the string dataset that contained the nutrition items separated by the end-of-line symbol, as mentioned in Section 2.3.

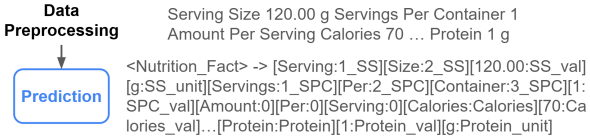

Figure 6: Prediction for Slot Filling models

3.3 Training details

In this paper, the pre-trained uncased BERT base model, (i.e. the bert-base-uncase model) (Devlin et al., 2018) was used for both the JointBERT and CTRAN methods. The number of training epoch were the default settings from the two papers, where JointBERT was trained with initial learning rate 5e-5 using a linear schedule for 10 epochs, and CTRAN was trained with learning rate 1e-3 in encoder, 1e-4 in decoder and 1e-4 in the BERT layer using StepLR scheduler for 50 epochs. The maximum length of the strings was set to 200, which contains almost all of the nutrition information in the nutrition datasets. Batch sizes were set to be 128 in JointBERT and 16 in CTRAN.

4 A Dynamic Programming Method for Nutrition Extraction

This section presents an unsupervised method for slot filling based on Dynamic Programming, that

Figure 7: Nutrition string splitting workflow

does not need any training. The whole method is described in Algorithm 1. Its steps are discussed in detail in the following subsections.

Algorithm 1 Nutrition Information Extraction by Dynamic Programming

Input: Nutrition string S
Output: Nutrition items and their values

1: Preprocess string S by adding spaces before capital letters and special characters
2: Detect nutrition values
3: Detect units of measure
4: Detect nutrition items
5: Preprocess for DP
6: Apply DP

Preprocessing uses the approach discussed in Section 3.1, based on adding spaces before capital letters and certain special characters.

4.1 Detecting Nutrition Values and Units of Measure

The first step in the DP approach is to detect the numbers and their locations in the string. The numbers are defined as contiguous sequences of characters from $\{0, 1, .., 9, '.', '-', '/', ' ', ', '\}$. These numbers include integers, decimals, numbers with thousand separators, fractions and mix-fractions such as '10', '1.5', '2,000', '3/4' and '1-2/5'. They are detected together with the start and end locations in the string using regular expressions in Python. Then the string is split into substrings based on the locations of the numbers, and hence each substring contains only words and symbols but no numbers.

After that, units of measure are detected whether they are right at the beginning of substrings. The units of measure are taken from a dictionary of 116 units containing 'g','mg','oz','floz', etc.

After the units of measure are detected, the numbers are combined with the units of measure into a single substring using "_". For example, the value "1" combined with the unit of measure "mg" will be "1_mg". In some cases after the numbers+unit of measure there is another number+%, the percent Daily Value (%DV), such as "0.5 g 1 %". In such

cases, all four items (two numbers and two units of measure) are combined, obtaining "0.5_g__1_%". This whole conglomerate represents a candidate nutrition value, with its unit of measure and possibly its %DV.

4.2 Nutrition Item Detection

The next step is to detect whether the substrings contain any keywords for the nutrition items we are trying to extract. The nutrition items are detected using a list of 19 possible keywords, 'Servings Per Container', 'Serving Size', 'Calories', 'Fat', 'Saturated Fat', 'Trans Fat', 'Cholesterol', 'Sodium', 'Carbohydrate', 'Fiber', 'Sugars', 'Added Sugars', 'Protein', 'Vitamin A', 'Vitamin C', 'Vitamin D', 'Calcium', 'Iron', 'Potassium', and their variations, such as 'fat', 'total fat', 'saturated', 'saturates', etc. For each detected item, the start and end location in the string and the corresponding nutrition item are recorded. In the case of the ambiguous nutrition items such as 'Fat' and 'Saturated Fat', the items with more words will be detected before to those with fewer words. Once any of the items is detected, the detected nutrition item will be removed from the substring and the next possible nutrition item will be searched in the current substring.

For example, in the string "Total Fat 0.5 g Total Saturated Fat 0 g Total Trans Fat 0g", the substring "Total Saturated Fat" will be tested if 'Saturated Fat' is contained. Since 'Saturated Fat' is detected, it will be removed from the substring and the program will detect whether the next item 'Fat' is contained in the substring "Total". This method also works in the case of the substring "Servings Per Container Serving Size" in Figure 4, which contains more than one nutrition item. In this case, both 'servings per container' and 'serving size' will be detected.

The substrings that are left at the end of nutrition item detection are text to be ignored. These ignored texts will be joined with an adjacent nutrition item and will be considered as a single node for Dynamic Programming in Section 4.3.

4.3 Overview of Dynamic Programming

Dynamic programming is an efficient globally optimal algorithm for minimizing an additive cost on a chain of nodes $V_1, ..., V_n$, each having a set of possible labels (values) $l_i \in L$.

Let $\theta_k(i)$ be the unary cost for node V_k taking value i, and $\theta_k(i, j)$ be the binary cost for node V_k taking value i and node V_{k+1} taking value j.

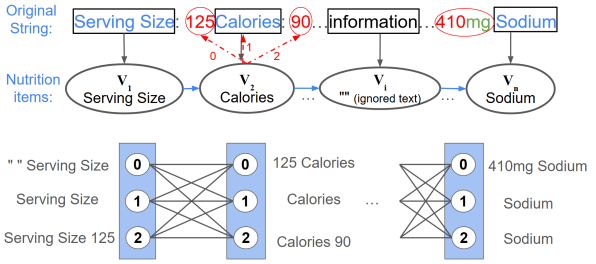

Figure 8: Dynamic Programming

Under these assumptions, dynamic programming obtains the global optimum of the minimization problem:

$$\min_{\mathbf{l}\in L^n}[\sum_{i=1}^{n}\theta_i(l_i) + \sum_{i=1}^{n-1}\theta_i(l_i, l_{i+1})]. \quad (1)$$

It does so in a recursive manner by memorizing $M_k(j)$, the minimum cost of the partial optimization problem on $V_1, ..., V_k$ with the label of V_k being j:

$$M_k(j) = \min_{\mathbf{l}\in L^k, l_k=j}\sum_{i=1}^{k-1}[\theta_i(l_i) + \theta_i(l_i, l_{i+1})]. \quad (2)$$

The vectors $M_k(j)$ are computed recursively, with the first one being $M_1(j) = 0$, and the following ones as:

$$M_{k+1}(j) = \min_{l\in L}[M_k(l) + \theta_k(l) + \theta_k(l, j)]. \quad (3)$$

Finally, the optimum is obtained by minimizing:

$$\min_{l\in L}[M_n(l) + \theta_n(l)], \quad (4)$$

and the optimal solution can be traced back by memorizing the argmin for Eq. (4) and each step of Eq. (3), and tracing back the optimum starting from n down to 1.

4.4 Dynamic Programming for Matching Nutrition Values to Items

At this step, the nutrition string has already been split into nutrition values and some nutrition items and ignored texts between them.

The problem is now how to match the nutrition items with the nutrition values, assuming that the value could be before or after the nutrition item. For that purpose, dynamic programming (DP) is used to find the matches.

To use DP to match the nutrition items to values, the DP nodes $V_1, ..., V_n$ are the detected nutrition items in the order they are in the string. The labels $L = \{0, 1, 2\}$ take three values, defining whether the corresponding nutrition value is before the item ($l = 0$), after the item ($l = 2$) and does not have a match ($l = 1$).

The unary cost $\theta_k(j)$ is the cost that a certain nutrition item matches the values before or after in the string based on the units of measure of those values. For example, the nutrition item "Protein" is

Unit type\j	0	1	2
Preferred	1	999	1
Other	10	999	10

a) $\theta_k(j)$

$i\backslash j$	0	1	2
0	1	1	1
1	1	1	1
2	10	1	1

b) $\theta_k(i,j)$

Table 3: Unary costs $\theta_k(j)$ and binary costs $\theta_k(i,j)$ used in dynamic programming.

more likely to be matched to a value with unit "g" but not "mg", and hence the cost of "g" in this case will be smaller than "mg". The dimension of the unary cost $\theta_k(j)$ for each nutrition item is 3.

The cost values for each nutrition item that is matched to nutrition values having different units are defined based on matching to a preferred unit of measure or not, as shown in Table 3, a).

The preferred units of measure are:

1. No unit for: 'Servings Per Container', 'Calories' and ignored text.
2. Any unit except no unit for 'Serving Size'.
3. 'g' for: 'Total Fat', 'Saturated Fat', 'Trans Fat', 'Carbohydrate', 'Dietary Fiber', 'Total Sugars', 'Added Sugars', and 'Protein'.
4. 'mg' for: 'Cholesterol' and 'Sodium'.
5. 'mg' or '%' for: 'Vitamin A', 'Vitamin C', 'Vitamin D', 'Calcium', 'Iron' and 'Potassium'.

The pairwise cost $\theta_k(i,j)$ is the cost of having node k with label i and node $k+1$ with label j. If the label of V_k is 2 (i.e., "after"), there should be a high cost to have the label of V_{k+1} be 0 (i.e., "before") because that would assign the same nutrition value to two items. Thus, the cost should be larger in this case than in other combinations. The values of $\theta_k(i,j)$ in our experiments are given in Table 3, b).

5 Experiments

Experiments were performed on a Windows 11 computer with a 13th Gen Intel(R) Core(TM) i7-13700HX 2.10 GHz processor with 32GB RAM and an RTX 4060 GPU with 8GB memory.

Experiments were performed on the nutrition extraction dataset introduced in Section 2, with its parts shown in Table 2.

Because the 'Manufacturers' dataset contains food product information from strings without line breaks between nutrition items, the 'Manufacturers' dataset could not be labeled for slot filling, so it was only used as a test set for all methods.

The other three parts: TraderJoes, Publix and Target were merged into a 'Combined' dataset containing 24047 observations, shown as 'Combined' in Table 2.

The methods that were evaluated are: JointBERT (Chen et al., 2019), CTRAN (Rafiepour and Sartakhti, 2023), GPT-4 (OpenAI, 2024a), GPT-4o (OpenAI, 2024b), and the proposed dynamic programming (DP) method described in Section 4.

GPT-4 and GPT-4o were queried using Python and their respective API from the OpenAI library (https://openai.com/api/).Each prompt in the GPT-4 and GPT-4o API includes the model to be used, i.e., "gpt-4", "gpt-4o" and "gpt-4o-mini", and the message part with the nutrition strings and the request to extract the nutrition items and their values. The response of the prompts were collected and evaluated using the GT table.

The other three methods were implemented in Python. For JointBERT and CTRAN, their GitHub implementations from https://github.com/monologg/JointBERT and https://github.com/rafiepour/CTRAN were used.

JointBERT and CTRAN were trained as described in Section 3. Training each fold of the JointBERT took an average of 1.1 hours for 10 epochs and for CTRAN took 16.5 hours for 50 epochs.

The methods that require training (JointBERT and CTRAN) were evaluated with four-fold cross-validation on the 'Combined' dataset. The model trained on each one of the folds was also used to evaluate the 'Manufacturers' dataset.

The unsupervised methods (GPT-4, GPT-4o and DP) were evaluated on the entire dataset.

The DP algorithm took around 49 msec per query while the JointBERT took 13 msec and CTRAN took 396 msec per query. Querying GPT-4o using the API took about 4097 msec per query.

5.1 Evaluation Measure

The ground truth (GT) table of each dataset was used to create the baseline for evaluation and to calculate the precision and recall of each food product (row) by matching the predicted and GT values for each nutrition item.

The overall precision and recall are computed by averaging the precision and recall in each row, and the F1-score is calculated based on the overall precision and recall.

The precision is defined as the ratio $p = M/N$, where N is the number of nutrition items that are output by the method, and M is the number of nutrition items from the output that are matched to the GT table items.

The recall is defined as the ratio $r = M/A$,

Method	Prec(std.)	Recall(std.)	F_1(std.)
Combined dataset, N=24047			
JointBERT	98.6(0.01)	89.9(0.05)	94.0(0.02)
CTRAN	**99.9(0.01)**	**99.9(0.01)**	**99.9(0.00)**
GPT-4	95.5	83.1	88.9
GPT-4o	95.4	90.0	92.6
DP	98.9	95.4	97.1
Manufacturers dataset, N=2117			
JointBERT	83.2(0.03)	78.6(0.06)	80.8(0.03)
CTRAN	93.9(0.02)	91.4(0.02)	92.6(0.01)
GPT-4	90.3	80.9	85.4
GPT-4o	92.2	85.7	88.8
DP	**98.1**	**99.4**	**98.7**

Table 4: Nutrition extraction evaluation. Shown are the precision, recall, and F_1-scores of the five methods, with best results in bold.

where A is the total number of nutrition items that are in the GT table, and M is the number of items from the GT table that are found in the output.

The F_1 measure has the usual definition in terms of the precision p and recall r, $F_1 = 2pr/(p+r)$.

5.2 Results

The results are shown in Table 4, with the methods that require training having mean (and std.) test values on the four cross-validation folds.

From Table 4 one can see that the F_1 scores on the Combined dataset are almost consistently better than those on the Manufacturers dataset because the Combined dataset strings are more homogeneous, hence easier to process. GPT and GPT-4o have good performance on the Combined dataset, and both JointBERT and CTRAN have similar precision results, while CTRAN has better recall on both the Combined dataset and the Manufacturers dataset. The proposed DP method has a stable performance on both datasets and outperforms the other methods on the more challenging 'Manufacturers' dataset.

GPT-4 and GPT-4o provide a convenient Python API environment for sending input prompts containing the nutrition strings and the request. But one disadvantage of GPT-4 is that each prompt is irrelevant to others, which means that the user has to give very specific prompts such as: "Extract the nutrition item Protein from this string." for each string; otherwise they cannot expect stable results. Moreover, although it is not necessary to train a model on GPT-4 and GPT-4o, it takes a much longer time querying strings than the other methods. ChatGPT does have a good performance on extracting the nutrition items from strings, but users have to type in the strings repeatedly in the message bar manually unless using a web driver.

Method	Prec	Recall	F_1
DP	98.1	99.4	98.7
No preprocessing	95.2	93.1	94.1
No binary cost	76.8	72.9	74.8

Table 5: Ablation study on the Manufacturers dataset.

Ablation study. An ablation study of the DP method introduced in this paper is shown in Table 5. It evaluates the importance of the preprocessing step and the binary cost in the obtained result. The performance is poor without the binary cost from Table 3. In this case, nutrition items may be matched to same values in the strings. The preprocessing step is less important than the binary cost. However, this step was also necessary for slot filling methods, i.e., JointBERT and CTRAN.

6 Conclusion

This paper introduced the NLP problem of extracting nutrition information from nutrition strings, which is a slot-filling problem that has not received attention in the NLP literature. In this regard, the paper provides a nutrition extraction dataset called NutriX containing more than 26,000 nutrition strings and their associated ground truth nutritional information. The paper shows how the dataset was collected and annotated, and the challenges facing this problem compared to other slot filling problems, such as long strings and a large number of slots per string.

The paper also shows how to view the nutrition extraction problem as a slot filling problem and how to train deep learning models on this data. Moreover, the paper introduces a novel method based on dynamic programming for the same task.

Finally, experiments on the provided dataset show that the problem is not trivial, and one part of the dataset called the 'Manufacturers' dataset is quite challenging where the proposed DP method works best among the five methods evaluated, including two GPT-4 versions.

Future work includes collecting more data and training a classifier that will predict the food category (e.g. pizza, hot dogs, ice cream, etc.) from the nutrition string. The ultimate goal is to build a food search engine that anybody can use to search food products that fit their specific dietary needs.

Limitations

There are some limitations for the methods used in this paper:

1. Training data from various resources is important for the supervised learning methods to have better prediction in complex nutrition strings dataset.

2. Data preprocessing is also necessary for both JointBERT and CTRAN (Slot-Filling methods) since the slot labels should be separated by space.

3. Manual verification and correction of the constructed ground truth table is needed.

4. GPT-4 and GPT-4o are convenient inputting queries, but both of them take longer time to generate the results than other methods.

5. The DP method performs well on both the Combined and Manufactures datasets in this paper. However, in the nutrition item detection step, a prebuilt dictionary to convert ambiguous nutrition items is needed. For instance, both of the items 'Potas.' and 'Potassium' represent the same nutrition item. Therefore, once the new dataset is collected, the prebuilt dictionary should bee updated manually.

References

Emanuele Bastianelli, Andrea Vanzo, Pawel Swietojanski, and Verena Rieser. 2020. SLURP: A spoken language understanding resource package. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7252–7262, Online. Association for Computational Linguistics.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. BERT for joint intent classification and slot filling. *CoRR*, abs/1902.10909.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *Preprint*, arXiv:1805.10190.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Python Software Foundation. 2024. Requests: Http for humans. https://pypi.org/project/requests/. [Online; accessed 15-May-2025].

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757, New Orleans, Louisiana. Association for Computational Linguistics.

Richard Andrew Harrington, Vyas Adhikari, Mike Rayner, and Peter Scarborough. 2019. Nutrient composition databases in the age of big data: fooddb, a comprehensive, real-time database infrastructure. *BMJ Open*, 9(6).

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Instacart. 2025. Instacart retailer website. https://www.instacart.com/. [Online; accessed 15-May-2025].

Bing Liu and Ian R. Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *CoRR*, abs/1609.01454.

Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, and Geoffrey Zweig. 2015. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 23(3):530–539.

Anthony Moi and Nicolas Patry. 2023. HuggingFace's Tokenizers.

Henry B. Moss, Daniel Beck, Javier Gonzalez, David S. Leslie, and Paul Rayson. 2020. BOSS: bayesian optimization over string spaces. *CoRR*, abs/2010.00979.

Baiju Muthukadan. 2024. Selenium with python. https://selenium-python.readthedocs.io/. [Online; accessed 15-May-2025].

OpenAI. 2024a. Models of gpt-4. https://platform.openai.com/docs/models/gpt-4. [Online; accessed 15-May-2025].

OpenAI. 2024b. Models of gpt-4 omni. https://platform.openai.com/docs/models/gpt-4o. [Online; accessed 15-May-2025].

Mehrdad Rafiepour and Javad Salimi Sartakhti. 2023. Ctran: Cnn-transformer-based network for natural language understanding. *Engineering Applications of Artificial Intelligence*, 126:107013.

TraderJoes. 2025. Trader joe's retailer website. https://www.traderjoes.com/. [Online; accessed 15-May-2025].

Ngoc Thang Vu. 2016. Sequential convolutional neural networks for slot filling in spoken language understanding. In *Interspeech 2016*, pages 3250–3254.

Yu Wang, Yilin Shen, and Hongxia Jin. 2018. A bi-model based rnn semantic frame parsing model for intent detection and slot filling. *Preprint*, arXiv:1812.10235.

Puyang Xu and Ruhi Sarikaya. 2013. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 78–83.