

# Recipe Intelligence

---

**Team 61 - ALPHA bet**

Gopendra Singh - 2022101003

Yash Nitin Dusane - 2022102078

9 May, 2025



# INTRODUCTION

---

- Goal: Analyze recipe text and generate a comprehensive Health Chart
- Output includes:
  - **Cuisine classification**
  - **Dietary category** (e.g., vegan, gluten-free)
  - **Recipe difficulty level**
  - **Nutritional value prediction** (calories, fat, carbs, etc.)



# KEY NLP TASK - INGREDIENT EXTRACTION

---

- Ingredient identification is the foundation for all downstream tasks
- Implemented using Named Entity Recognition (NER)
- Two approaches used:
  - Rule-based NER inspired by the "Foodie" paper
  - Machine Learning-based NER using spaCy, trained on annotated recipe data



# RULE-BASED NER APPROACH

---

- Inspired by the "**FoodIE**" paper, our rule-based NER:
  - Uses pattern-based matching, POS-tagging, and lexical cues
  - Identifies ingredients through predefined templates:
    - Example: [quantity] [unit] of [ingredient] → "2 cups of chopped onions"
  - Utilizes a domain-specific vocabulary of ingredients, measurement units, and cooking descriptors
- **Core Approach:**
  - Processes text to extract:  
Quantity, Unit, Modifiers, Food Name
  - Outputs clean strings like:  
"2 tablespoons chopped onion"



# RULE-BASED NER

- **Key Techniques Used:**

- **Preprocessing:**

- Uses spaCy to tokenize and remove non-essential stopwords (e.g., “of”, “each”)

- **Pattern Matching:**

- Extracts quantities, units, and modifiers by position and token type

- **Lexicon-Based Matching:**

- Matches food names using a domain-specific food lexicon

- Follow "**quantity-unit-ingredient**" rule for extracting food entities

# IMPROVEMENTS BASED ON FOODIE

- Improvements Based on FoodIE:
  - **Longest Match Strategy:**
    - Finds the longest possible food phrase (e.g., “red chili powder”) even if individual words aren't all in the lexicon
  - **Lemmatized Matching:**
    - Handles plural/singular forms (e.g., “tomatoes” → “tomato”)
  - **Most Complete Phrase Selection:**
    - Avoids duplicates by storing the longest matching phrase per food item

# ML-BASED NER WITH SPACY

---

- To overcome the limitations of rules, we built a machine learning-based NER:
  - **Trained a custom spaCy model** on annotated recipe text
  - Labels ingredient spans directly, accounting for variation in phrasing
    - Example: "a dash of cinnamon", "sliced tomatoes"
  - Captures contextual understanding better than rules
- To fine-tune the model using spaCy's NER pipeline, the annotated dataset needed for training the model was not available. So we used a regular recipe dataset that has the recipe instruction text and the ingredients column, and using a python script converted the data into the form required for training. The training data entry looks like :
- **Text : Mix brown sugar, butter, and milk in a saucepan.**
- **Entities: [(4, 15, 'INGREDIENT'), (17, 23, 'INGREDIENT'), (29, 33, 'INGREDIENT')]**



# GENERATING TRAIN DATA

Recipe  
text

Ingredients

For each ingredient **find the index at which it occur** in the recipe text. Use its start and end position and append the info as :

Repeat for each recipe text sample

Text : Mix brown sugar, butter, and milk in a saucepan.  
Entities: [(4, 15, 'INGREDIENT'), (17, 23, 'INGREDIENT'), (29, 33, 'INGREDIENT')]

# TRAINED IN 2 DIFFERENT WAYS

**Recipe text** = "to make the dosa recipe ,first add some **coconut oil** into the pan, then heat it for sometime then add **rice** and **potato**. then serve it nicely"

**Start with a blank pipeline**

```
spacy.blank("en")
```

**INGREDIENTS FOUND:**

- COCONUT OIL
- POTATO

**Start with a base model**

```
spacy.load("en_core_web_sm")
```

**INGREDIENTS FOUND:**

- COCONUT OIL
- POTATO
- RICE



## THE MODEL HANDLES MULTI-WORD AND CHAINING

**Recipe text** = "stir in 1/2 teaspoon of **turmeric powder**, 1 teaspoon of **red chili powder**"

### INGREDIENTS FOUND:

- RED CHILI POWDER
- TURMERIC POWDER

**BUT WAIT ..**

**Recipe text** = "Since this is a veg recipe, so do not add chicken.  
Add 1 cup sugar, 1 cup water, and mix with flour."

**INGREDIENTS FOUND:**

**SUGAR**

**FLOUR**

**CHICKEN**

The chicken is actually not a part of ingredient, but it is being extracted as ingredient.

# INGREDIENT FILTERING USING NEGATION AND SUBSTITUTION

---

- To ensure only actually used ingredients are considered, **we implemented a rule-based filter using spaCy.**
  - **Negation Detection:** Identifies ingredients that are not used, e.g.,  
"This recipe doesn't use chicken."
  - **Substitution Detection:** Detects if an ingredient is mentioned only as a replacement, e.g.,  
"Used almond milk instead of cow milk."
  - Uses **dependency parsing and pattern matching** for phrases like:  
"instead of", "replace with", "without" etc.

This step improves classification and nutrient prediction accuracy by excluding irrelevant ingredients.



```

used = False
for sent in doc.sents:
    sent_doc = nlp(sent.text)
    sentence_lemmas = [token.lemma_.lower() for token in sent_doc]

    # Check for ingredient presence
    for i in range(len(sentence_lemmas) - ingredient_length + 1):
        if sentence_lemmas[i:i+ingredient_length] == ingredient_lemmas:
            start_idx = i
            end_idx = i + ingredient_length
            ingredient_span = sent_doc[start_idx:end_idx]

            # Check for negation
            negated = any(is_negated(token) for token in ingredient_span)

            # Check for substitution
            substituted = False
            matches = matcher(sent_doc)
            for match_id, start, end in matches:
                pattern = substitution_patterns[match_id]
                replaced_pos = pattern['replaced_pos']

                if replaced_pos == 'after':
                    replaced_part = sent_doc[end:]
                else:
                    replaced_part = sent_doc[:start]

                if (ingredient_span.start_char >= replaced_part.start_char and
                    ingredient_span.end_char <= replaced_part.end_char):
                    substituted = True
                    break

            if not negated and not substituted:
                used = True

return used

```

## AFTER FILTERING

**Recipe text** = "Since this is a veg recipe, so do not add **chicken**.  
Add 1 cup **sugar**, 1 cup **water**, and mix with **flour**."

### INGREDIENTS FOUND:

SUGAR  
FLOUR  
WATER  
CHICKEN 



Filtering using Negation and Substitution

### INGREDIENTS FOUND:

SUGAR  
WATER  
FLOUR

## USING BOTH RULE BASED AND ML BASED NER

- Since rule-based NER relies on a predefined food ingredient dictionary to identify whether a token is an ingredient, it may fail to recognize unseen or uncommon ingredients.
- On the other hand, training a machine learning-based NER model to extract entities like “quantity” and “unit” requires manually annotated data for each, which is a significant challenge.
- A hybrid approach can be used to address these limitations:
  - First, **use the ML-based NER model to extract the list of ingredients** from the text.
  - Then, **based on this ingredient list, apply rule-based NER to identify the corresponding quantities and units.**

This hybrid method significantly improves the overall performance of the named entity recognition task by combining the strengths of both approaches.

# CLASSIFICATION TASKS

---

- Using extracted ingredients and recipe text, we perform:
  - Cuisine Classification  
Indian, Mexican, Chinese, etc.
  - Dietary Category Classification  
Vegan, Vegetarian, Gluten-free, etc.
  - Difficulty Classification  
Easy, Medium, Hard



## CLASSIFICATION – MODEL DETAILS

---

- We use **BERT embeddings** to represent the recipe text and ingredients
- For each classification task, a separate supervised model is trained on curated recipe datasets
- Ingredients extracted from NER are optionally used as additional input features
- Training data includes recipes labeled for each class type to fine-tune models.



# CLASSIFICATION USING BERT

---

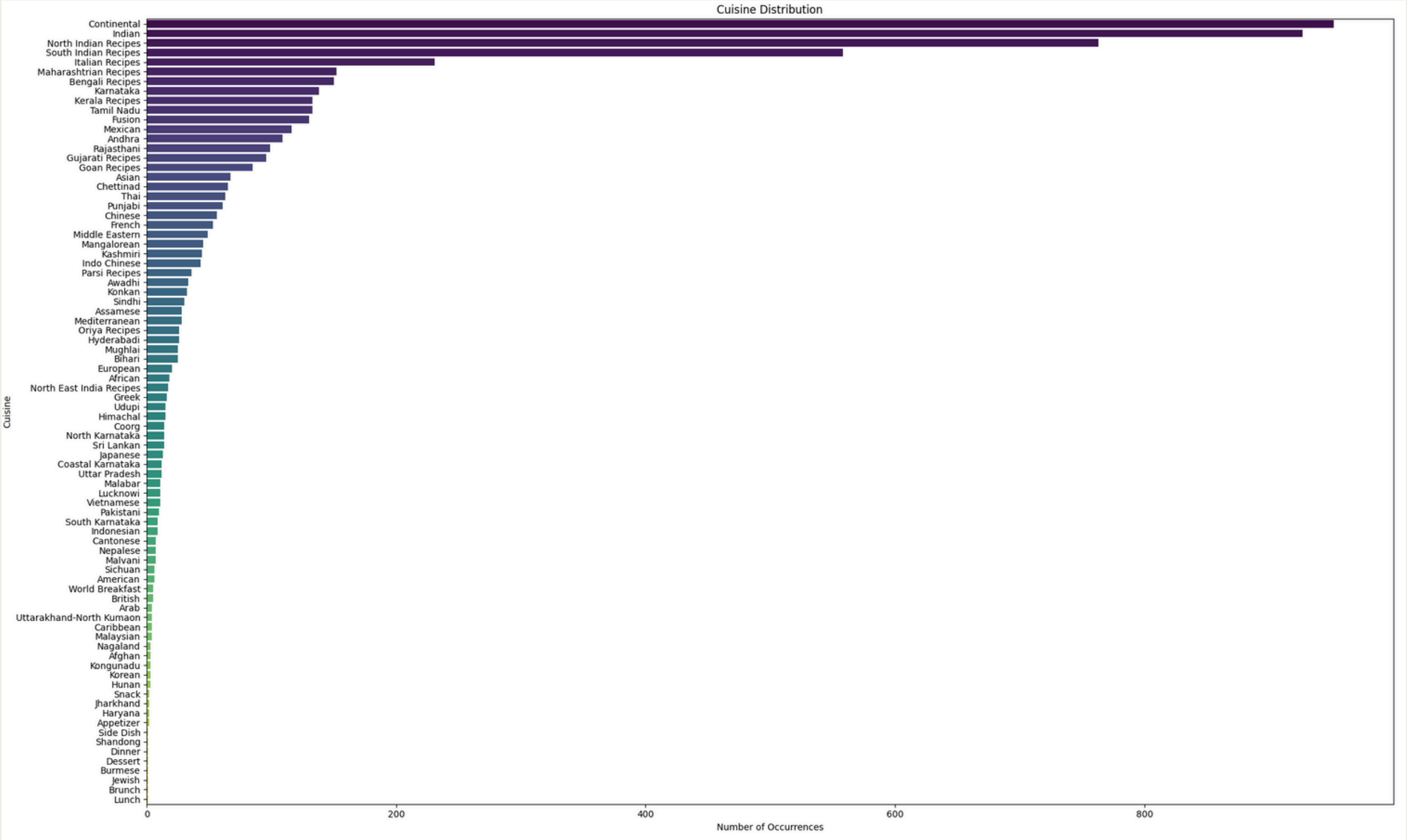
We classified recipes based on the instructions using a fine-tuned BERT model.

- Preprocessing:
  - Encoded cuisine labels using LabelEncoder.
  - Tokenized instruction texts using [BERT tokenizer](#).
- Model Architecture:
  - [Pretrained bert-base-uncased](#) as the base.
  - Added a dropout and fully connected layer for classification.
- Training:
  - Used Cross-Entropy Loss and AdamW optimizer.
- Evaluation:
  - Performance measured using `classification_report` on validation data



# CUISINE CLASSIFICATION

## DATASET DISTRIBUTION



## ANALYSIS: CUISINE CLASSIFICATION

- As shown the distribution of data in the previous page, **some cuisine classes has very low number of data points due to which the prediction made by the model for those classes is less accurate.**
- The input to the model may be direct ingredients or the whole recipe. **The model taking the entire recipe as input performed better than the model which takes only the ingredients to determine cuisine.**

Training Loss: 0.0594  
100%|██████████| 75/75 [00:12<00:00, 6.05it/s]

	precision	recall	f1-score	support
Afghan	0.00	0.00	0.00	0
African	0.50	1.00	0.67	3
American	0.00	0.00	0.00	0
Andhra	0.89	0.80	0.84	20
Appetizer	0.00	0.00	0.00	1
Arab	0.00	0.00	0.00	3
Asian	0.14	0.17	0.15	12
Assamese	1.00	0.50	0.67	2
Awadhi	0.58	0.70	0.64	10
Bengali Recipes	0.85	0.74	0.79	39
Bihari	0.67	0.67	0.67	3
British	0.00	0.00	0.00	3
Brunch	0.00	0.00	0.00	0
Burmese	0.00	0.00	0.00	0
Cantonese	0.00	0.00	0.00	1
Caribbean	0.00	0.00	0.00	0
Chettinad	0.91	0.67	0.77	15
Chinese	0.40	0.46	0.43	13
Coastal Karnataka	0.00	0.00	0.00	2
Continental	0.79	0.80	0.79	176
Coorg	0.33	1.00	0.50	1
Dessert	0.00	0.00	0.00	0
Dinner	0.00	0.00	0.00	0
...				
accuracy			0.61	1188
macro avg	0.39	0.37	0.37	1188
weighted avg	0.61	0.61	0.61	1188

# DIETARY CATEGORY CLASSIFICATION (RESULT)

---

- The model classifies the recipe in **9 different dietary categories**: High protein vegetarian, Vegetarian, Sugar Free Diet, Non Vegetarian, High Protein Non Vegetarian, Vegan, Diabetic Friendly, No Onion No Garlic (Sattvic), Eggetarian and Gluten Free.

BERT Model Performance:

Accuracy: 0.6781

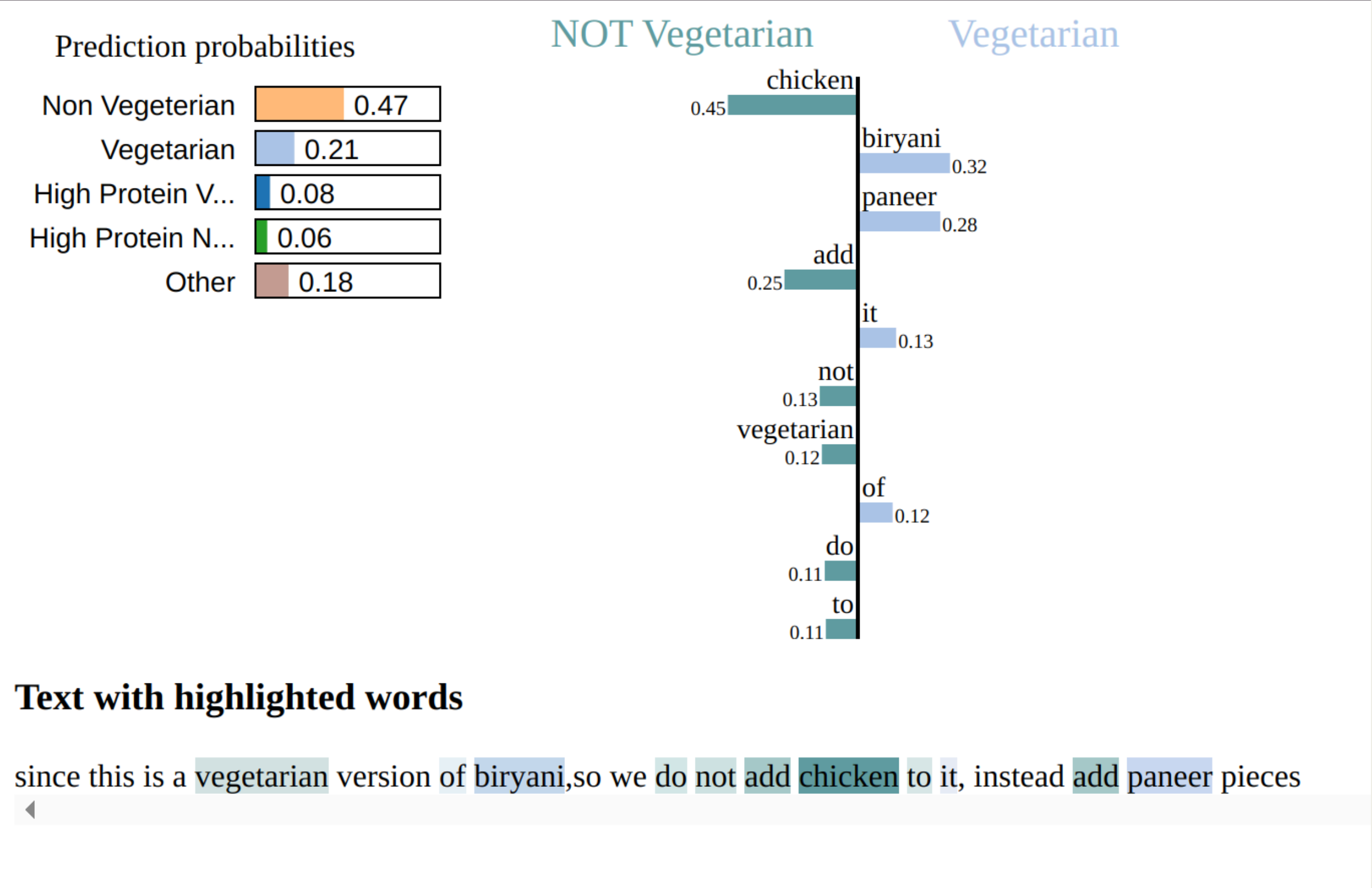
Classification Report:

	precision	recall	f1-score	support
Diabetic Friendly	0.00	0.00	0.00	5
Eggetarian	0.90	0.75	0.82	12
...				
accuracy			0.68	146
macro avg	0.33	0.36	0.34	146
weighted avg	0.68	0.68	0.67	146



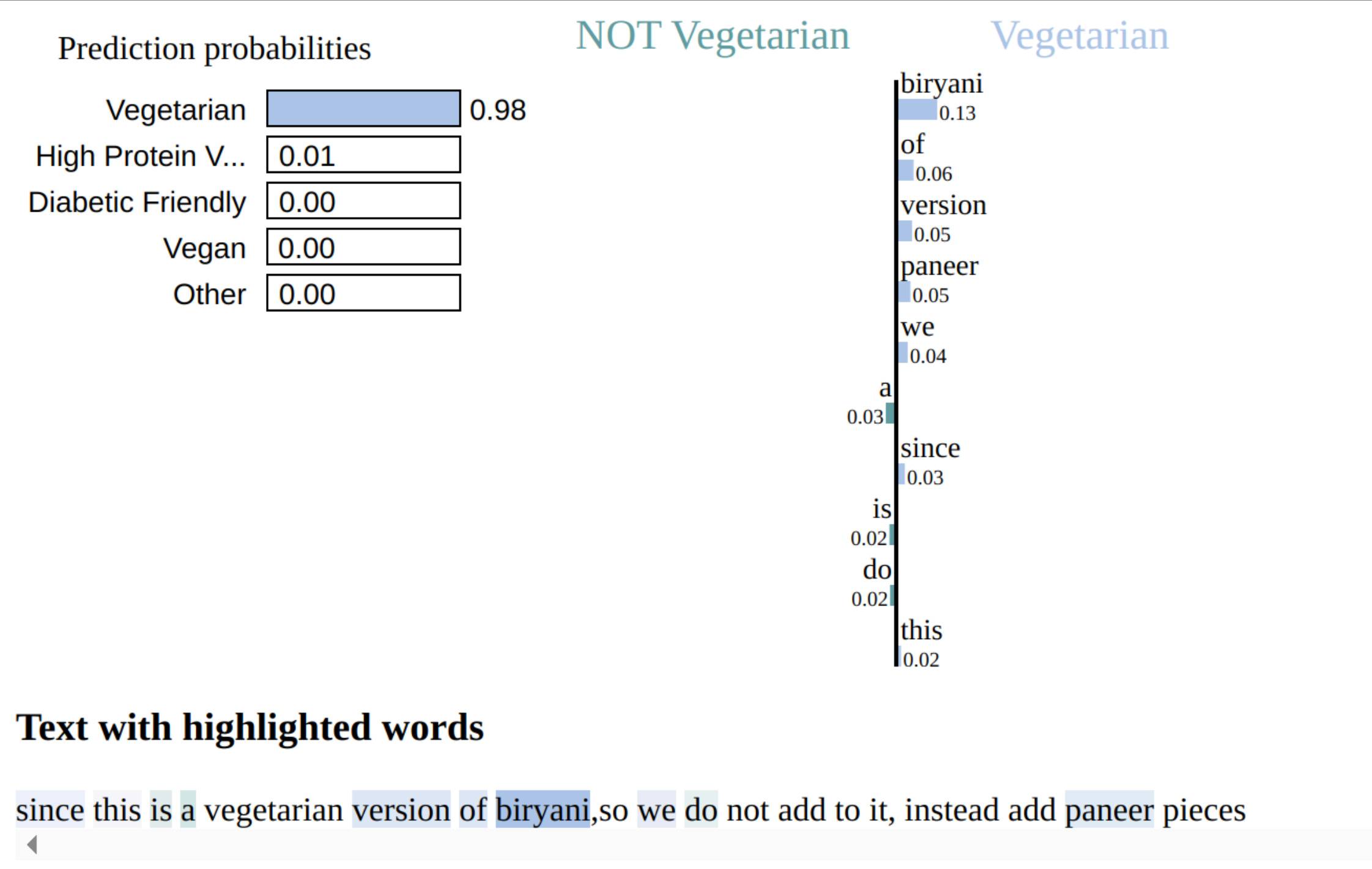
# WRONG PREDICTION OF DIETARY CATEGORY:

**Recipe text** = "since this is a vegetarian version of biryani,so we do not add **chicken** to it, instead add paneer pieces"



# IMPROVING PERFORMANCE : BY USING NEGATION AND SUBSTITUTION FILTERING

Filter out the “chicken” word and then predict the class



## DIFFICULTY CLASSIFICATION (RESULT)

---

- To train the model for difficulty prediction, we used a recipe dataset that has “**prep\_time**” and “**cook\_time**” for each recipe. So to label a recipe as easy, medium and hard , we calculated the **total\_time = sum of prep\_time + cook\_time**. Then used the following threshold value (in min): **0 to 30 = Easy, 30 to 60 = Medium, above 60 = Hard.**

Accuracy: 0.60

Classification Report:

	precision	recall	f1-score	support
easy	0.75	0.25	0.38	36
hard	0.50	0.13	0.21	30
medium	0.60	0.95	0.73	78
accuracy			0.60	144
macro avg	0.62	0.44	0.44	144
weighted avg	0.61	0.60	0.53	144



# NUTRITIONAL VALUE PREDICTION

---

- The goal is to estimate: **Total Fat, Saturated Fat, Cholesterol, Sodium, Total Carbohydrate, Dietary Fiber, Total Sugars, Protein.**
- We experimented with **three models** to predict nutritional values:
  - **Model 1:**
    - Ingredients one-hot encoded using MultiLabelBinarizer
    - Random Forest Regressor trained on binary vectors
  - **Model 2:**
    - Ingredient text embedded using BERT
    - Same regressor used on dense BERT vectors
  - **Model 3:**
    - Custom PyTorch network NutritionNN
    - Input: BERT-based recipe vector



## OBSERVATION

- All three models performed similarly with limited accuracy
- Realized that nutritional value prediction is unreliable using only ingredients

### Better Approach:

- A better approach is to use a nutritional database
- Compute nutrition mathematically based on ingredient quantities

```
MSE: 22507.98  
RMSE: 150.03  
MAE: 36.73
```

```
Accuracy within ±10:  
Overall: 59.93%  
Per nutrient:  
  Total Fat: 71.52%  
  Saturated Fat: 97.35%  
  Cholesterol: 27.81%  
  Sodium: 8.61%  
  Total Carbohydrate: 39.07%  
  Dietary Fiber: 100.00%  
  Total Sugars: 50.33%  
  Protein: 84.77%
```

```
MSE: 22507.98  
RMSE: 150.03  
MAE: 36.73
```

```
Accuracy within ±10:  
Overall: 59.93%  
Per nutrient:  
  Total Fat: 71.52%  
  Saturated Fat: 97.35%  
  Cholesterol: 27.81%  
  Sodium: 8.61%  
  Total Carbohydrate: 39.07%  
  Dietary Fiber: 100.00%  
  Total Sugars: 50.33%  
  Protein: 84.77%
```

```
MSE: 22507.98  
RMSE: 150.03  
MAE: 36.73
```

```
Accuracy within ±10:  
Overall: 59.93%  
Per nutrient:  
  Total Fat: 71.52%  
  Saturated Fat: 97.35%  
  Cholesterol: 27.81%  
  Sodium: 8.61%  
  Total Carbohydrate: 39.07%  
  Dietary Fiber: 100.00%  
  Total Sugars: 50.33%  
  Protein: 84.77%
```

## End-to-End Pipeline Architecture

**Here's how the system flows:**

Input: Raw recipe text

- » **Ingredient Extraction:** NER (rule-based + ML-based)
- » **Negation Filtering:** Filters out unused ingredients
- » **Text & Feature Embedding:** BERT + ingredient vectors
- » **Classification Models:** Cuisine, dietary, difficulty
- » **Nutrition Prediction:** Estimate macro nutrition
- » **Output:** A health chart summarizing all predictions

Thank you!

---