NLP1 Final Project
Recipe Regression & Generation using the Food.com Dataset

Khaled Mili Maxim Bocquillion Samy Yacef Mateo Lelong
Group 9 — EPITA

1. Introduction

As part of our final project in Natural Language Processing, we explore multiple modeling approaches on a dataset of cooking recipes [2]. Our goal is to solve different NLP tasks using this cooking dataset.

1.1. Outline

We develop several models to address the following tasks:

- **A regression task** aimed at predicting the total cooking time from the recipe's textual content
- **A generative task** focused on producing coherent cooking steps given a list of ingredients

In addition to model's implementation, we provide:

1. A detailed overview of our **pre-processing** pipeline
2. A description of **hyperparameter tuning** and the **training preparation**
3. A **benchmark** for each model across both tasks

1.2. Process

1. **Model Selection**: Choose suitable models for each task, considering how each contributes to our experimental goals.
2. **Metric Definition**: Select appropriate evaluation metrics adapted to each model and task.
3. **Training and Validation**: Train models using a validation set and monitor performance based on the chosen metrics.
4. **Hyperparameter Tuning**: Analyze the influence of key hyperparameters to identify the best-performing model in each category.
5. **Evaluation**: Assess the selected models on the test set using the test metrics.

2. Cooking Dataset Analysis

This dataset consists of ∼200K recipes and 700K recipe reviews covering 18 years of user interactions and uploads on food.com.

2.1. Generic information

Attribute	**Description**
Source	Kaggle - Food.com Dataset
Paper	https://arxiv.org/pdf/1909.00105
Authors	Shuyang Li (UCSD PhD researcher)
Format	CSV
Number of Recipes	180K
Temporal Coverage	02/25/2000 – 12/18/2018
Provenance	Food.com scraped with Requests/BeautifulSoup
Update Frequency	No updates in the last 5 years

Table 1. General information about the Food.com dataset

2.2. Features

Feature	**Description**	**Type**
`name`	Name of the recipe	String
`id`	Unique identifier	Integer
`minutes`	Minutes to prepare a recipe	Integer
`contributor_id`	Author of the recipe	Integer
`submitted`	Date of submission	Date
`tags`	Tags on the recipe	List of string
`nutrition`	Compact nutrition information	List of integer
`description`	Detailed description of the recipe	String
`steps`	Detailed instructions for the recipe	List of string
`n_steps`	Number of steps	Integer
`ingredients`	List of ingredients in the recipe	List of string
`n_ingredients`	Number of ingredients in the recipe	Integer

Table 2. Recipe dataset features and descriptions

2.3. Exploratory data analysis (EDA)

This section presents a comprehensive EDA on the whole dataset.

2.3.1. Recipe preparation time

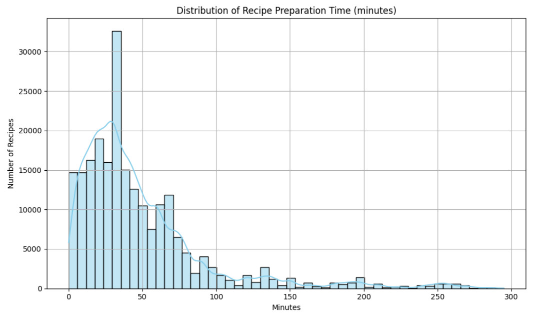


Figure 1. Distribution of recipe preparation time

Some recipes show outliers in preparation time, often due to the inclusion of non-active steps like cooling or marinating (refrigerating for 4 days). However, the majority of recipes fall within **a reasonable range of 15 minutes to 3 hours**.

2.3.2. Ingredients Analysis

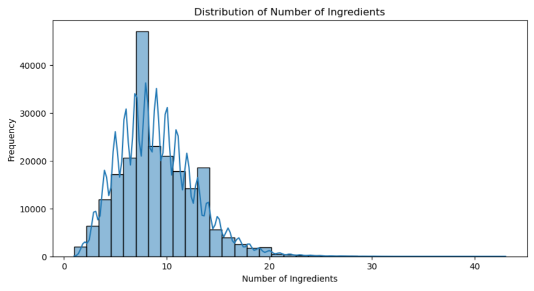


Figure 2. Distribution on the number of ingredients

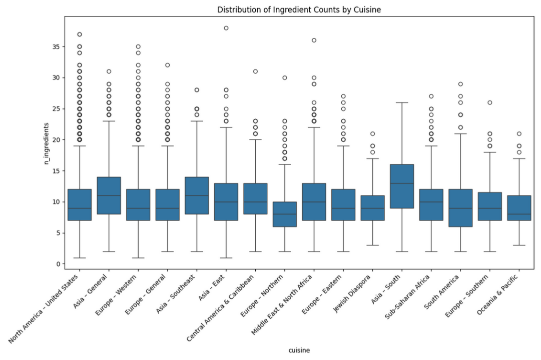


Figure 3. Number of ingredients in each kind of cuisine

The number of ingredients follows an approximately Gaussian distribution, centered around 7–8 ingredients. Recipes with more than 20 ingredients, which account for less than 0.5% of the dataset, will be treated as outliers and removed.

2.3.3. Cuisine Type Analysis

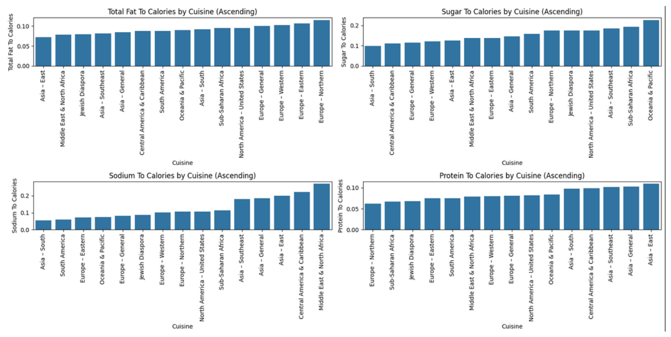


Figure 4. Nutrition information by type of cuisine

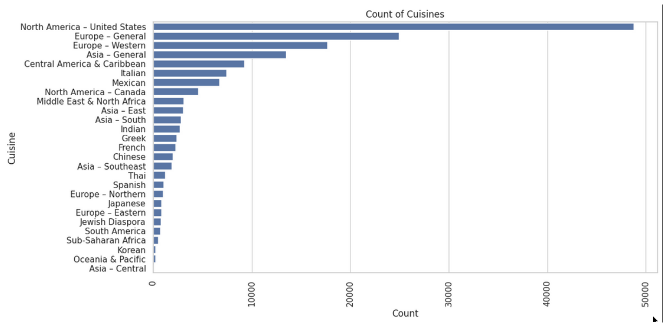


Figure 5. Recipes count by cuisine

We visualized the nutritional information of recipes by cuisine type and observed a significant imbalance in the distribution of cuisines. Due to this imbalance, **we decided not to use cuisine as a target for classification** and instead focused on a regression task predicting cooking time.

2.3.4. Steps Analysis

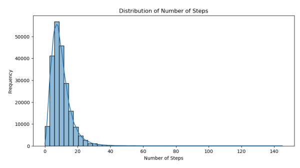


Figure 6. Distribution of the number of steps

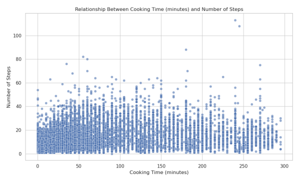


Figure 7. Cooking time vs. number of steps

The number of steps follows a roughly Gaussian distribution centered around 7. Outliers (20–40 steps) often reflect stylistic choices to write more detailed instructions, not increased complexity. As there's no standard for step granularity, step count is an unreliable metric and should be used with caution.

3. Pre Processing

This section explains all the preprocessing steps we applied to the dataset.

3.1. Removing columns and rows

The first step in our preprocessing was to remove non-relevant columns for our exploratory data analysis (EDA). For example, we excluded the *contributor_id* column (which identifies the author of the recipe) and the *submitted* column which indicates the submission date, as they did not serve our analytical objectives.

Next, we removed recipes with a total preparation time greater or equal than 5 hours, considering them outliers. This step eliminated approximately 10,000 recipes from the original dataset of around 231,000 entries.

3.2. Lemmatizer & Stop-Word Removing

We then lemmatized the text in the name column using the *WordNetLemmatizer* of the *nltk* library and the stopwords. Additionally, we used a custom list of over 400 irrelevant or noisy words, such as names like ”ashley” or ”aston”, to further clean the data.

Before Cleaning	After Cleaning
OH MY GOD ITS SO AMAZINGGGGG potatoes with chicken yummy yummy	potatoes with chicken

Table 3. Examples of name cleaned into concise title

3.3. Cleaning and Standardizing Instructions

We also performed preprocessing on the *steps_strings* column, which contains the step-by-step instructions for each recipe. This involved the following key tasks:

1. **Correcting Typos in Units**
   We manually corrected common misspellings, such as ‘”minteus”‘ instead of ‘”minutes”‘, to ensure consistency in unit recognition.
2. **Standardizing Units**
   To maintain uniformity, we converted various units to standard formats:
   - milliliters (mL) → liters (L)
   - inches → centimeters (cm)
   - fahrenheit → celsius (°C)
3. **Converting Imprecise Quantities**
   Imprecise or range-based quantities were normalized to approximate average values. Examples include:
   - $\frac{1}{2} \rightarrow 0.5$
   - ”2-3” or ”2 to 3” → 2.5
4. **Stemming Words**
   To further reduce variability and enhance text matching, we applied stemming to all words in the instructions. This helped standardize different forms of the same root word (e.g., ‘”chopped”‘, ‘”chopping”‘, ‘”chop”‘ → ‘”chop”‘).

3.4. Expanding Columns

After the general preprocessing, we decided (following the recommendation from RecipeNLG [1]) to expand the *nutrition* column into separate features for better insight into the food's composition.
The *nutrition* column was originally a list of values. We split it into the following individual columns: *calories*, *total_fat*, *sugar*, *sodium*, *protein*, *saturated_fat*, *carbohydrates*.

This allowed for more granular analysis and visualization of nutritional contents.

4. Regression on cooking time

4.1. Goal

The goal is to identify the most accurate model for estimating how long a recipe will take to prepare.

4.2. Models

1. **Linear Regression**: Serves as a starting point to compare more advanced methods.
2. **Bayesian Ridge Regression**: Introduces regularization helping us to prevent overfitting.
3. **Feedforward Neural Network (FFNN)**: A basic neural architecture that allows us to model non-linear relationships in the data.
4. **Gated Recurrent Unit (GRU) Neural Network**: Expected to outperform FFNNs for text-based tasks by capturing sequential dependencies, allowing us to evaluate the value of temporal modeling for this task.
5. **Transformer**: As a state-of-the-art architecture in NLP, it helps us see if the attention mechanisms can improve performance on this task.

4.3. Training

4.3.1. Preparation

We are selecting different features for our models:
- Numerical Features: *n_steps*, *n_ingredients*, *calories*, *total_fat*, *sugar*, *sodium*, *saturated_fat*, *carbohydrates*
- Text Features (TF-IDF or Word2Vec): *steps*, *ingredients*, *tags*

4.3.2. Hyperparameters Tuning

A preliminary test has identified the feedforward neural network as the most promising model. We employed Optuna to search for optimal hyperparameters from the following set:

- **Learning Rate**: **ReduceLROnPlateau** strategy was implemented to dynamically adjust the learning rate if the model plateaus during training.

- **Units**
- **Regularization Techniques**
  - **L2 Regularization (Weight Decay)**
  - **Dropout**

Following extensive optimization calculations, the optimal hyperparameters are:

Hyperparameter	Optimum Value
Learning Rate	0.0234
Dropout Rate	0.1852
L2 Regularization Rate	0.0881
Units in Hidden Layer	128

The bar plot illustrates the relevance of each hyperparameter in the optimization process, indicating that regularization parameters were slightly more influential.

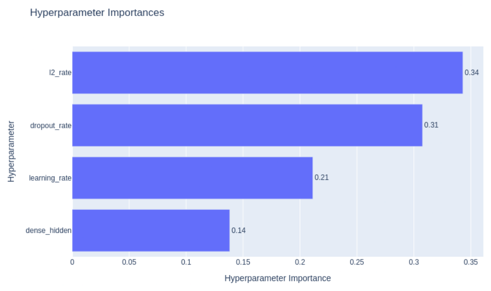


Figure 8. Importance of Different Hyperparameters

The higher dimensions plot, on the other hand, gives us an overview of how different settings of hyperparameters affect eh model performance.

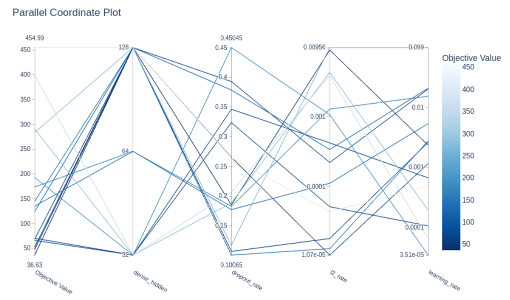


Figure 9. Parallel Coordinate Plot for FeedForward Hyperparameters Optimization

4.4. Evaluation

4.4.1. Evaluation Metrics

The following metrics have been selected:

- **Mean Absolute Error** (MAE): to help understand the average gap (in minutes) between predictions and actual recipe time
- **Mean Squared Error** (MSE): to identify whether the model is sensitive to outliers recipes
- **Coefficient of Determination** (R^2 Score)

4.4.2. Benchmark

Using TF-IDF Embeddings

Model	MAE	MSE	R^2	Train Time(s)
Linear Reg.	17.32	859.54	0.59	8
Bayes. Ridge	17.34	860.28	0.59	14
FFNN	**13.97**	657.75	0.68	253
GRU	14.54	**677.66**	**0.70**	3743

Using Word2Vec Embeddings

Model	MAE	MSE	R^2	Train Time(s)
Linear Reg.	23.67	1378.20	0.35	5
Bayes. Ridge	23.65	1377.69	0.35	10
FFNN	20.15	**1167.44**	**0.45**	110
GRU	**20.05**	1192.57	0.42	3052

As we used a pre-trained Transformer (DistilBERT) from Hugging Face, we did not use TF-IDF or Word2Vec. Instead, we used WordPiece embeddings, which are a lightweight version of BERT tokenization. The following metrics are the result of a single epoch of training that took 3 hours to run on our computers.

Model	MAE	MSE	R^2	Train Time(s)
Transformer	17.79	949.89	0.55	3672

4.4.3. Learning Curves

You can see that the model is not overfitting, as the training and validation errors are close to each other. However, the validation error seems to oscillate at the end of the training, which could indicate that the model is not learning anymore and may start to overfit. Thus our early stopping criterion has been reached and the training has been stopped.

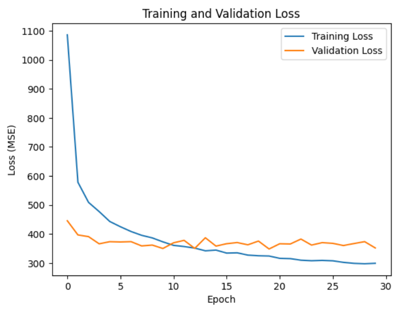


Figure 10. Training & Validation loss by epoch on GRU-based regression model

4.5. Discussion

4.5.1. Model Post-Hoc Interpretation

We plotted the residuals of the best-performing model (Feedforward Neural Network) to examine potential patterns. The residuals are mostly concentrated around zero, indicating good model performance. However, several outliers suggest that some recipes are inherently more difficult to predict.

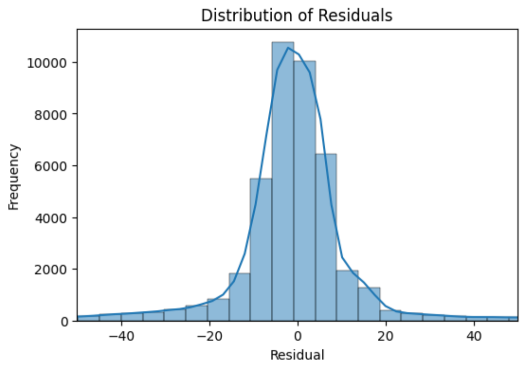
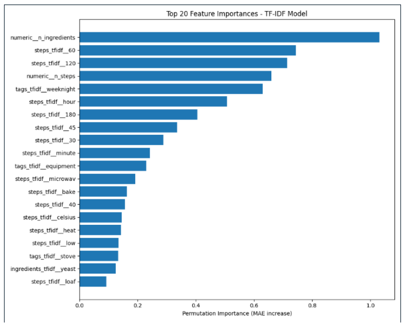


Figure 11. Distribution of residuals error on GRU-based regression model

To analyze the behavior of our feedforward model using TF-IDF, we examine the top 20 most important features using *permutation importance*, measured by the increase in Mean Absolute Error (MAE) when the values of a feature are randomly rearranged.

The most impactful feature is the number of ingredients (*numeric_n_ingredients*), followed by the number of steps (*numeric_n_steps*), and various TF-IDF terms corresponding to durations mentioned in the instructions (e.g., *60*, *120*, *180*, *hour*, *minute*) as well as keywords (e.g., *bake*, *celsius*, *stove*). This suggests the model heavily relies on explicit numerical time values and structured textual metadata (e.g., tags like *weeknight*, *equipment*) to make predictions.



4.5.2. Comparison with True Values

As you can see in the figure below, the GRU model's predictions are generally close to the true values, with some exceptions. It tends to underestimate the preparation time for recipes with very long preparation times, which is likely due to the limited amount of data available for these cases.

Some recipes with relatively short preparation times (less than 40 min) are overestimated, but most of the recipes in that timezone are well predicted.

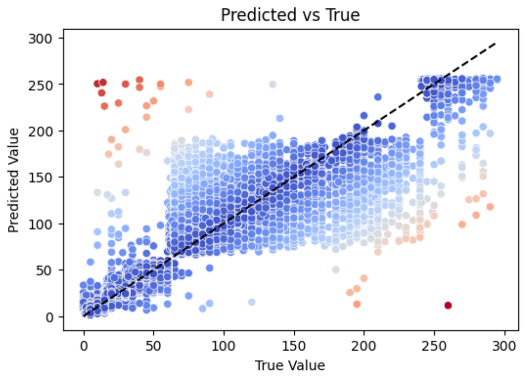


Figure 12. Predicted vs. True cooking times using GRU-based regression model

4.5.3. Error Analysis & Critics

- **Patterns in Prediction Errors:** Some errors are systematically associated with certain feature combinations.
- **Challenging Recipe Types or Time Ranges:** Specific types of recipes (e.g., with long preparation times or rare ingredients) are harder to predict.
- **Possible Causes:** Dataset imbalance, insufficient representation of edge cases, and limitations in embedding generalization may contribute to errors.
- **Embedding Issue:** As the TF-IDF directly reflects the importance of words in the dataset, while Word2Vec focuses on semantic similarity and context. For regression tasks (time prediction in our case), the presence or absence of certain keywords (captured by TF-IDF) may be more predictive than semantic similarity.

4.5.4. Model Strengths and Weaknesses

Model	**Strengths**	**Weaknesses**
Linear Reg.	Simple, interpretable, fast to train	Assumes linear relationships, sensitive to outliers
Random Forest	Handles non-linear data well, robust to overfitting, works with missing data	Can be slow with large datasets, less interpretable
Bayesian Ridge	Handles multicollinearity, provides uncertainty estimates	Assumes linearity, may be sensitive to prior selection
Feedforward NN	Can model complex non-linear patterns	Risk of overfitting, less interpretable
GRU	Suitable for sequential data, fewer parameters than LSTM	Difficult to train, requires significant computational resources
Transformer	Handles long-range dependencies, parallelizable	Requires large datasets, complex architecture, computationally expensive

4.5.5. Combining Models for Improved Performance

- **Bagging method**: We trained 8 Feedforward RNNs on subsets of our dataset but did not achieve better results compared to a single model. The R2 score (**0.8017**) and MAE (**10.5095**) did not improve significantly. The random subsets used for training each model may not have been representative of the entire dataset. Surprisingly, the predictions from each model were consistently worse than those of the single model.
- **Stacking method**: We first trained a TFIDF model on the text columns, but it performed poorly on its own. Instead, we used it to generate embeddings for other models, since many linear models can't handle text directly.

4.6. Conclusion

In this benchmark, we evaluated multiple regression models for predicting recipe preparation time. The **Feedforward Neural Network** emerged as the best-performing model, with the lowest MAE and MSE while maintaining a reasonable training time. The **Gated Recurrent Unit** model also performed well but was significantly slower to train and did not provide the expected performance boost.

While linear models can provide a good baseline, more complex models like neural networks are better suited for capturing complex relationships between the content of the recipe and its preparation time.

5. Generation on cooking steps

5.1. Goal

The goal is to predict the cooking steps of a recipe using only **its ingredients**.

5.2. Models

1. **N-gram Model**: It serves as our baseline to evaluate the benefit of more complex sequence models.
2. **Feedforward Neural Network (FFNN)**: Processes text as a fixed-size input using embeddings, allowing us to verify how well non-sequential architectures perform.
3. **Recurrent Neural Network (RNN)**: A sequential model useful for modeling the structure of cooking instructions.
4. **RNN with GRU Layer**: Enhances the RNN to better handle long-term dependencies and solve the vanishing gradient issues. It helps us evaluate the impact of improved memory.

5.3. Training

5.3.1. Preparation

We use only the *steps* and *ingredients* features for this task.

To prepare the data, we follow a specific format. An input consisting of:

```
[ing1 ing2], [w1 w2]
```

is transformed into:

```
<s> ing1 ing2 <STEPS> w1 w2 </s>
```

Our objective is to predict the next token in a recipe's step-by-step instructions, using the list of ingredients as contextual input. Each training sample is composed of an input sequence and its corresponding next token. Special tokens such as 'STEPS', 's', and '/s' are included during training to guide the model in generating coherent steps and recognizing when to terminate the sequence. These tokens are removed from the output after generation.

Sample	**Input** (x)	**Target token** (y)
x_1	`<s> ing1 ing2`	`<STEPS>`
x_2	`ing1 ing2 <STEPS>`	`w1`
x_3	`ing2 <STEPS> w1`	`w2`
x_4	`<STEPS> w1 w2`	`</s>`

Table 4. Example training samples for next-token prediction using special tokens and fixed context size of 3.

5.3.2. Tokenization & Stemmer

The table below represents the usage of some combination of tokenizer & stemmer on each type of model.

Model	**Lancaster**	**BPE**	**Keras**	**AutoTokenizer**
NGram	✓	✓	✓	
FFNN	✓	✓	✓	
Simple RNN			✓	✓
GRU-based RNN			✓	✓

5.3.3. Hyperparameters Tuning

Ngram: N, smoothing, alpha

Training was performed on Kaggle Hub, as the N-gram model requires over 15GB of RAM to load its context grams on our dataset. Due to this heavy memory usage, training time is not meaningful on its own and should only be compared within the same class of models.

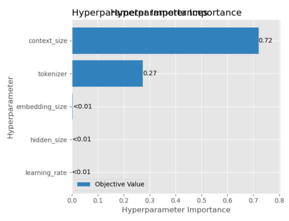
Prediction on 10 samples in the val. set:

tokenizer	**N**	**Smoothing**	**Val. Acc**	**Train Time(s)**
keras	3	✓	0.31	23
keras	5	✓	0.32	84
keras	8	✓	0.32	181
bpe 80	5	✓	0.68	130
bpe 80	5		0.68	125
bpe 80	8	✓	0.74	332

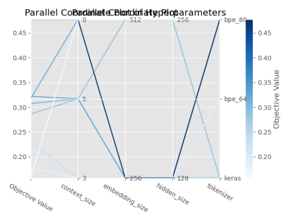
Enabling smoothing and adjusting the alpha parameter does influence performance. However, the NGram model remains quite primitive, scaling N to match the dataset size leads to overfitting and exponentially increases memory usage (27 GB of RAM for NGram BPE 80 with context of 8) and training time. The chosen model will be **NGram BPE 80 with N=5**.

FFNN: *context*, *embedding_dim*, *hidden_dim*, *droppout*, *tokenizer*, *learning_rate* (auto downscaled by tf – fixed at 0.1)

Tokenizer	Emb/Hid	Context	Acc.	Val. Acc
keras 50K	256/128	3	0.21	0.19
keras 50K	256/128	5	0.32	0.32
keras 50K	512/256	8	0.29	0.28
bpe 64	256/128	3	0.15	0.12
bpe 64	256/128	5	0.24	0.15
bpe 64	256/128	8	0.12	0.13
bpe 80	256/128	3	0.28	0.30
bpe 80	256/128	5	0.25	0.24
bpe 80	256/128	8	**0.44**	**0.41**



(a) FFNN hyperparameter importance



(b) FFNN hyperparameter parallel coordinates

Figure 13. Analysis of FFNN hyperparameter tuning

It is clear that the most influential hyperparameters are the **context size** and the **tokenizer** (which determines the vocabulary). On the other hand, increasing the layer dimensions has relatively little impact on the results. We observed better results with a balanced vocabulary size (bpe 80 tokenizer), suggesting that using either a too sparse or too dense final layer leads to poorer performance. The selected model will be **FFNN BPE 80, with 256 E / 128 H dimensions, 8 of context**.

Simple RNN

Tokenizer	Emb/Hid	Epochs	Acc.	Val. Acc
keras 10K	256/128	2	0.06	0.06
keras 10K	256/128	5	0.10	0.10
keras 10K	256/128	10	0.31	0.29

GRU-based RNN

Tokenizer	Emb/Hid	Epochs	Accuracy	Val. Acc
keras 10K	256/128	2	0.12	0.10
keras 10K	256/128	5	0.19	0.18
keras 10K	256/128	10	0.41	0.39
keras 10K	256/128	20	0.36	0.35

Either for the simple RNN or the GRU-based RNN, the hyperparameter that was conclusive is the number of epochs. For both models, selecting less than 5 epochs was not sufficient to obtain a decent accuracy and generation. The impact of epochs was obvious, as selecting 20 would also decrease the accuracy, especially for the GRU-based RNN. We ended up with 10 epochs as an empirical decision for the trainings which was the ideal value for our use case, combining an acceptable training time, and a decent accuracy.

Moreover, the variations on the temperature to make the probability distribution more uniform during the evaluation step did not impact considerably our models quality. Semantically, the results were quite the same.

5.4. Evaluation

5.4.1. Evaluation Metrics

The following metrics have been selected:

- **Ingredients Coverage**: it is a custom metric representing the ratio of output / intput ingredients that allows to see to what extend the generated recipe respects the constraints
- **BLEU Score**: exact matching and penalizes short outputs, making it better when fluency and precision matter
- **ROUGE Score**: rewards recall and partial matching, better for tasks like summarization or when it's important to cover key content.

5.4.2. Benchmark

Model	Ingredients Cov.	BLEU-1	ROUGE-L
FFNN	0.05	0.10	0.17
Simple RNN	0.10	0.10	0.16
GRU-based RNN	0.57	0.16	0.24

Table 5. Performance comparison of generative models

5.4.3. Results interpretation

NGram tend to overfit by memorizing the recipes rather than generating creative or generalized content. Interestingly, we also observed a specific type of error: some recipes contained repetitive loops, resulting in cyclic instructions.

```
Ingredients:  ['winter squash', 'mexican
seasoning', 'mixed spice', 'honey',
'but', 'olive oil', 'salt']
Generated Recipe:
make a choic and proceed with recip
depend on size of squash cut into half or
fourth remov seed for spici squash drizzl
oliv oil or melt butter over each cut
squash piec season with mexican season
mix ii for sweet squash drizzl melt honey
but grate piloncillo over each cut squash
piec season with mexican season mix ii
for sweet squash drizzl melt honey but
grate piloncillo over each cut squash
piec season with mexican season mix ii
for sweet squash drizzl melt honey but
grate piloncillo over eac
```

FFNN Due to an imbalance in token distribution, the model often predicts the same token repeatedly. It begins to capture grammatical structure around 30% accuracy, but its limited context window makes it impractical for generating coherent recipes. The model includes unknown ingredients and fails to capture the sequential nature of instructions, where certain steps should logically precede others.

```
Ingredients:  ['ground moose', 'egg'
'brown sugar', 'hot sauce', 'prepared
mustard', 'ketchup', 'barbecue sauc',
'salt and pepper']
Generated Recipe:
in a small bowl combin the flour and the
salt and the sugar and the sugar and the
sugar and in a bowl with a sharp knife.
Cut the appl into 2.54 cm piec and place
in a bowl in a small bowl combin the
egg and the egg and the egg and the egg
and the egg.  In a bowl and add the rest
of the ingredi and mix well add the egg
and mix well add in a bowl combin the
flour and the salt and the sugar and the
sugar and the in the center of the pan.
And cook for about 30.0 minute or in a
bowl and add the rest of the ingredi and
mix well add the egg and mix well add
cook the chicken and cook for about 30.0
minute or until the veget are tender and
the sauc is thick
```

Using **Batch Normalization**, **Dropout**, and changing **the optimizer** slightly improved the model's accuracy, but these adjustments were not enough to overcome the inherent limitations of the architecture.

RNN The model generated grammatically valid sequences with minor orthographic errors. However, the coherence and semantic integrity of the output were limited. A representative output generated using a simple RNN is provided below:

```
Ingredients:  ['flour', 'sugar',
'butter', 'eggs', 'vanilla']
Generated Steps:
in a larg bowl , mix the yeast and sugar
, then beat in the sugar and pour mixtur
into a bowl place a small amount of
milk into the cheesecak while you might
hold your finger fold in the <OOV> and
fill with the applesauc to make the
avgolemono cream:  in the 6 corner befor
cut the end in the prepar pastri seal
the pie plate take out the end of the
phyllo and sprinkl more crumb mix with a
second layer of bake soda bake the second
15.0 minute don't over them <eos> the
'apples'] should be jiggli
```

Although the syntax is correct, the generated content lacks logic and culinary relevance, making it unusable in practice. This led us to examine the tokenization process as a possible cause of the incoherent output.

We employed the `Tokenizer` class from Keras to preprocess the input text. The keras's vocabulary construction appeared to be suboptimal. A significant proportion of tokens were marked as `<OOV>` (out-of-vocabulary), indicating possible issues with the size or quality of the vocabulary. This may have led to a deterioration in generation quality due to information loss during tokenization.

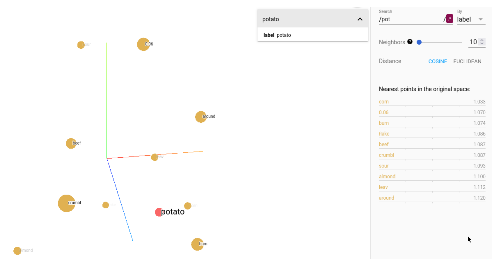
Gated Recurrent Unit (GRU) layer The model improved performance marginally, as reflected by the increased BLEU and ROUGE scores. The generated recipes became more coherent and complete but still failed to meet the standard required for real-world usability.

Further refinements in tokenization—such as character-level encoding, subword units (BPE), or domain-specific vocabulary may be required to enhance the content of generated recipes.

5.5. Discussion

5.5.1. Model Post-Hoc Interpretation

Using Tensorboard, we proposed an interpretation of our embdedding layer on a **trained FFNN 256/128, context 8 with keras tokenizer**. We plotted several tokens and analyzed their similarities using a 3D projection of a compressed space with T-SNE.



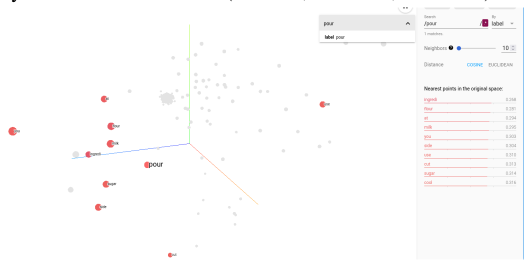
potato
label potato

Search /pot By label
Neighbors 10
Distance COSINE EUCLIDEAN

Nearest points in the original space:

corn	1.033
0.06	1.070
burn	1.074
flake	1.086
beef	1.087
crumbl	1.087
sour	1.093
almond	1.100
leav	1.112
around	1.120

(a) Similar words to "potato" identified through the embedding layer trained with a FFNN (256/128, 8 context, keras tok.)



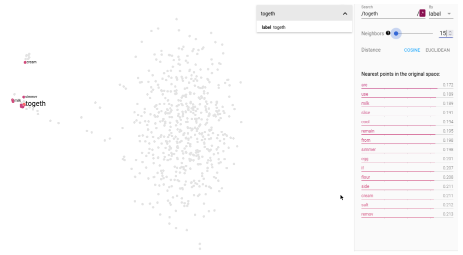
pour
label pour

Search /pour By label
1 matches.
Neighbors 10
Distance COSINE EUCLIDEAN

Nearest points in the original space:

ingredi	0.268
flour	0.281
at	0.294
milk	0.295
you	0.303
side	0.304
use	0.310
cut	0.313
sugar	0.314
cool	0.316

(b) Similar words to "pour" identified through the embedding layer trained with a FFNN (256/128, 8 context, keras tok.).



togeth
label togeth

Search /togeth By label
Neighbors 15
Distance COSINE EUCLIDEAN

Nearest points in the original space:

are	0.172
use	0.189
milk	0.189
slice	0.191
cool	0.194
remain	0.195
from	0.198
simmer	0.198
egg	0.201
if	0.207
flour	0.208
side	0.211
cream	0.211
salt	0.212
remov	0.213

(c) Similar words to "together" identified through the embedding layer trained with a FFNN (256/128, 8 context., keras tok.)

- The instruction "pour" is associated with "milk", "ingredients", and "sugar", which seems relevant in the context of a culinary dataset, as these are **common items being poured during food preparation**. This highlights the model's ability to **context-aware associations** between actions and objects.
- Ingredient "potato" is associated with "beef", "crumble", "burn" and "sour" which is quite interesting. This suggests that the embedding layer **captures both ingredient co-occurrence and contextual usage** from the dataset. It reflects not only common **culinary pairings** (beef & potato) but also **cooking sensory attributes** ("burn" and "sour") that appears in some recipes.
- The word "together" is associated with "are", "use", "milk", and "slice". While some words like "milk" and "slice" make sense in a cooking context, others like "are" and "use" are more general. This suggests the embedding captures both recipe-related and grammatical patterns, which might show **some mixing of meanings**.

The model **does not clearly distinguish between different types of words** indicating that the embedding captures co-occurrence patterns, but **not the functional roles of words**.

5.5.2. Comparing with True Values

The scatter plot below shows token predictions on 10,000 samples using a feedforward neural network. It highlights that common tokens are predicted more accurately, suggesting the model may be **overfitting to frequent patterns**.

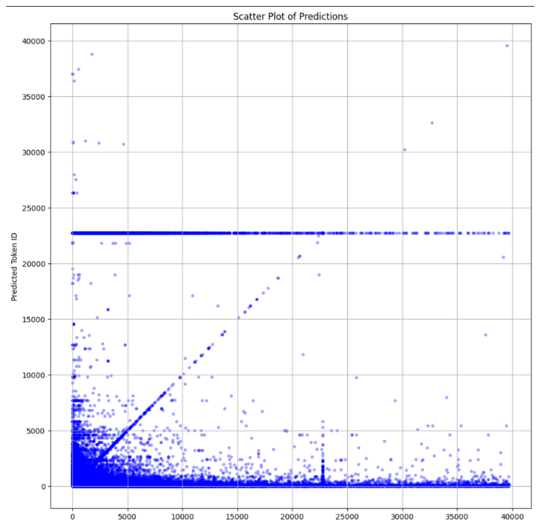


Figure 15. Predicted vs. True token using FFNN 256/128 (average of 0.44 accuracy)

Token	ID	Accuracy	Misclassified Count
the	0	0.73	15057
and	1	0.73	14780
to	4	0.48	13453
in	3	0.50	13407
a	2	0.57	11962
stir	16	0.08	7813
cook	13	0.17	7588
/s	22747	0.47	5405
put	107	0.00	1798
olive oil	119	0.00	275
flo	164	0.00	466
continu	227	0.00	828

Table 6. Token stats with ID, Accuracy, Count

5.5.3. Model Strengths & Weaknesses

Model	Strengths	Weaknesses
N-gram	Fast training for $N \leq 5$	High memory footprint, Fixed context size, slow generation, lacks grammatical consistency, cyclic generation error
Feedforward Neural Network	Generates grammatically consistent text	Fixed context size, large memory footprint
Simple RNN	Easy to implement, and good grammatical structure	Poor semantic consistency, low ingredient coverage, slow to train
RNN with GRU Layer	Improved ingredient coverage, better grammatical and semantic coherence	Insertion of unknown ingredients, OOV errors, slow to train

Table 7. Comparison of generative models: strengths and weaknesses

5.5.4. Error Analysis & Critics

The use of the **BLEU metric** in our case is very questionable as it was originally designed for machine translation, where word-for-word and phrase-for-phrase correspondence is expected. But in recipe generation:

- Multiple valid recipes can exist for the same ingredients.
- Phrases like *fry the onions until golden* vs. *sauté onions until soft* may be semantically equivalent, but BLEU would penalize them heavily.

So 2 perfectly good recipes using the same ingredients and achieving the same cooking goal can get low BLEU scores just because of wording.

5.6. Conclusion

Several models were trained and gave promising results based on training and test metrics. However, when generating recipes, the results showed clear limitations. The output often lacked structure, missed some ingredients or even added ones that weren't in the list.

Recipe generation turned out to be harder than expected. Simpler models like NGram and FFNNs couldn't handle the order of steps or how ingredients should be used.

RNN and GRU models showed promising potential, but their high training time and memory usage limited training over many epochs, leading to only average performance. The out-of-vocabulary problem also remained an issue, reducing the model's accuracy.

It became clear that our evaluation metrics (BLEU score especially) were not well-suited to the task.

For future work, more advanced models like fine-tuned LLMs should be explored, as they can better understand context and produce more coherent instructions. Adding some cooking knowledge or rules could also help improve the quality of generated recipes.

6. Conclusion

In this paper, we proposed a methodology to analyze the Food.com dataset through two main tasks: predicting cooking time and generating recipe steps from a list of ingredients.

A comprehensive exploratory data analysis (EDA) was presented to gain insights into the dataset and inform feature selection and preprocessing. Based on this analysis, we introduced a detailed preprocessing strategy to match the dataset's characteristics and our target tasks.
For each task,

- several models were **trained and evaluated**, with the best-performing models selected based on validation metrics obtained through a train/validation split and **hyper-parameter tuning**.
- a **detailed benchmark** was provided, highlighting model performance using appropriate evaluation metrics.
- an **interpretation of the results** has been proposed
- **evaluation and discussion** outlined the strengths and limitations of the different techniques and tokenization strategies used

Overall, our study shows that it is possible to model structured cooking data, while also highlighting some of the challenges involved.

References

[1] Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, and Agnieszka Lawrynowicz. RecipeNLG: A cooking recipes dataset for semi-structured text generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28, Dublin, Ireland, 2020. Association for Computational Linguistics. 3

[2] Shuyang Li. Food.com recipes and interactions, 2019. 1