FIRE: Food Image to REcipe generation

Prateek Chhikara[1,2], Dhiraj Chaurasia[1,2], Yifan Jiang[1,2], Omkar Masur[1], and Filip Ilievski[2,3]
[1]University of Southern California, USA, [2]Information Sciences Institute, USA
[3]Vrije Universiteit Amsterdam, Netherlands
{pchhikar,chaurasi,yifjia}@isi.edu, omasur@usc.edu, f.ilievski@vu.nl

1 Hyper-parameters

Title Generation Model: For title generation, we utilized the BLIP Salesforce/blip-image-captioning-base[1] model, which was trained for 20 epochs with a batch size of 24 and a learning rate of 10^{-5}. To evaluate the performance on the validation set, we opted to employ string similarity metrics, specifically the longest common subsequence (LCS), instead of traditional loss functions. This decision was motivated by the superior results achieved by the LCS-based model in comparison to loss based validation function.
Ingredients Generation Model: We train the model for 100 epochs with a batch size of 150, utilizing an Adam optimizer with a learning rate of 10^{-4}. At each epoch, we decrease the learning rate by 0.01%. The input image was resized to a dimension of 224×224×3 (299×299×3 for InceptionV3), while the ingredients vocabulary size was set to 1488, with a corresponding embedding size of 512.
Recipe Generation Model: We fine-tune the base T5 (220M parameters) with a batch size of 12 and learning rate of 3×10^{-4} for 30 epochs using an AdamW optimizer. We have `max source length=50` and `max target length=512` considering the average length of titles, ingredients, and cooking instructions. During generation, we use beam search with `num beams=4`, `length penalty=1`, and `repetition penalty=2.5`.

2 Resource Details

We used a hardware configuration consisting of four Quadro RTX 8000 GPUs, each equipped with 48 GB GDDR6 memory, to train the Image to Title and Ingredients model. The GPU computations were supported by an Intel(R) Xeon(R) Gold 5217 CPU running at 3.00GHz, which offered 32 processors. To train the Cooking Instructions model, we employed eight NVIDIA RTX A5000 GPUs, with each GPU having 24 GB GDDR6 memory. The GPU acceleration was complemented by an Intel (R) Xeon(R) Gold 5215 CPU operating at 2.50GHz, featuring 40 processors.

3 Dataset

As of June 2023, the Recipe1M dataset is not publicly available. However, we accessed the data using the following URL: http://im2recipe.csail.mit.edu/dataset/login/.

4 Pilot Experiment

4.1 Recipe Customization

Table 1 shows the detailed experimental results on the average over seven human annotators.

1. **Efficacy**: The customization is implemented correctly, and the modified recipe successfully achieves the desired taste or flavor profile based on the prompt.
2. **Coherence** The modified recipe maintains a coherent structure and flow, ensuring that the added or altered ingredients make sense within the context of the original recipe.

[1]https://huggingface.co/Salesforce/blip-image-captioning-base

3. **Soundness** The instructions are clear and unambiguous. A user can easily follow the steps to recreate the customized recipe.
4. **Proportions and Measurements** The proportions and measurements of the modified ingredients are appropriate and maintain a balance with the original recipe.

Table 1: Results of human evaluation on recipe customization

Metric	Efficacy	Coherence	Soundness	Proportions and Measurements
Average Score	3.59	3.76	3.76	3.54
Inter-annotator agreement	0.90	0.92	0.89	0.78

Survey Link:https://bit.ly/fire-customization

4.2 Recipe to Code Generation

We conducted a survey to assess the quality of the code generated from recipes. Each instance in the survey was evaluated based on three questions. Firstly, we determined whether all ingredients were correctly converted to code. Secondly, we examined the accuracy of converting cooking instructions to code. Lastly, we assessed whether cooking instruction parameters were appropriately converted. The questions asked are as follows.

1. **Ingredients:** How well are the ingredients incorporated into the code?
2. **Cooking instructions:** How well are the cooking instructions translated into code?
3. **Step parameter:** How well are the step parameters of cooking instructions translated into code?

Participants were asked to rate each question on a scale from 0 to 5, where **0: Extremely poor** - The conversion of ingredients is completely inadequate. **1: Low quality** - Only a few ingredients are converted correctly, and most are missing. **2: Fair quality** - Some ingredients are accurately incorporated, but there are instances of misleading information or unclear details. **3: Good quality** - A significant portion of the ingredients are accurately incorporated. **4: Very good quality** - All ingredients are mostly incorporated correctly, with only minor errors. **5: Excellent quality** - Every single ingredient is flawlessly incorporated, ensuring accurate representation. Table 2 shows the detailed experiment results on the average over seven human annotators.

Table 2: Results of human evaluation on recipe to code generation.

Metric	Ingredients	Cooking instructions	Step parameter
Average Score	4.47	4.29	4.27
Inter-annotator agreement	0.83	0.75	0.79

Survey Link:https://www.bit.ly/fire-code-generation

Moreover, we conduct a pilot experiment on a sample of 200 recipes. To describe cooking details, we predefine 30 cooking operation functions, such as heat, cut, and boil, with eight function parameters, including time and tools. The defined set covers 84% of operations that appear in the dataset.

5 Prompting Design for Downstream Applications

5.1 Recipe Customization

Our test on recipe customization covers a wide range of topics, following is the list of prompts used in each topic:

1. **Ingredient adjustment**: I (don't) like {ingredient}, can you remove/add that for me?
2. **Detail addition**: I am new to cooking, can you expand more details for the recipe?
3. **Taste adjustment** I (don't) like {taste} food, can you change the taste of the recipe?
4. **Calories adjustment** Can you reduce/increase the calorie content of the food?
5. **Cooking time adaptation** Can you provide a more convenient version that can be done in {time} minutes?

5.2 Recipe to Code Generation

For the conversion of the recipe to Python code, we formulate explicit rules for the demonstration and employ well-defined prompts to provide clear guidance to the LLMs. For each recipe 'r' with a list of cooking instructions $\{c_1, c_2, \ldots, c_n\}$, its code format conform following format:

```
def main():
    #instruction
    #$c_1$
    #$c_2$
    ...
    #$c_n$

    def $f_1$():
        $h_1$ = {operation}(input, (tool),(how),(temperature))
    def $f_2$():
        $h_2$ = {operation}(input, (tool),(how),(temperature))
    ...
    def $f_n$():
        $h_n$ = {operation}(input, (tool),(how),(temperature))
```

The code script begins with comments on cooking instructions. Each cooking instruction 'c' will become function name 'f' in the code script by adding an underscore between each word within cooking instructions. For example, "*Preheat oven to 350 degrees Fahrenheit.*" will become $Preheat\_oven\_to\_350\_degrees\_Fahrenheit$. For each function definition, we will extract the operation as the function object and input ingredients as parameters. The parameter set will also consider tools, cooking methods, and temperature if they are adaptable. For the same example, "*Preheat oven to 350 degrees Fahrenheit.*", the function object in the definition will be h_1 = Preheat(tool = oven, temp = 350 degrees F).