Project Final Report: MagicRecipe

Siyi He
siyih@usc.edu

Minhao Li
minhao@usc.edu

Yitian Yan
yitianya@usc.edu

Abstract

The MagicRecipe project aims to create a personalized dish recommendation system that leverages advanced language models to generate recipes and shopping lists tailored to user preferences and available ingredients. The system uses natural language processing to interpret user inputs and produce customized recipes. The project's goal is to develop a prototype model, specifically fine-tuning GPT-2, to address the challenge of automating the creative process of recipe development. The model's performance will be evaluated using a mixed-method approach, including automated metrics like ROUGE score, Ingredient Coverage and Relevance, and Diversity Score, as well as human evaluations on reasonableness, creativity, and coherence. The final report will document the development process, evaluation framework, and analysis of results, showcasing the model's ability to transform basic ingredient lists and dish names into detailed cooking instructions, as demonstrated in the provided example.

1 Introduction

Our project is centered on the challenge of generating accurate and creative recipes from specific inputs such as a list of ingredients and a recipe title. This task intersects the culinary arts and natural language processing (NLP), aiming to automate the creative process of recipe development. In the digital era, there is an increasing demand for personalized cooking experiences, with individuals seeking creative recipes tailored to their dietary preferences and the ingredients they have on hand. However, the ability to generate new, personalized recipes remains a significant challenge. For example, popular models such as GPT2.0 and T5 fail to generate detailed instructions for certain dishes. This gap underscores the need for advanced NLP models capable of understanding culinary context and generating suitable recipes, enhancing the cooking experience for both amateurs and professionals. Addressing this problem is not only beneficial for meeting the immediate needs of cooking enthusiasts but also serves as a catalyst for innovation in NLP, expanding the landscapes of creative content generation.

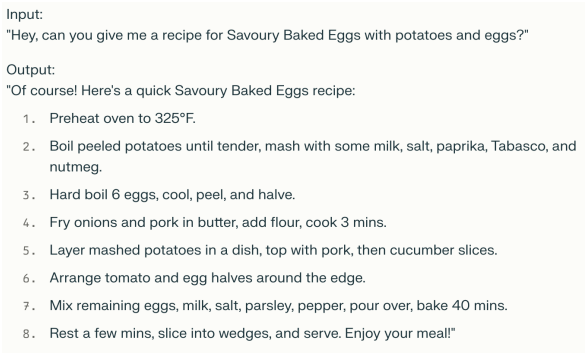


Figure 1: Our model's expected sample input and output

With the mentioned context, our primary objective is to develop a prototype model that is capable of producing coherent and creative recipes from a given title and list of ingredients. We present sample inputs and outputs in Figure 1. Given that both the input and output generate text sequences formatted in a dialogic style, we classify this as a sequence-to-sequence task, similar to question answering model. To realize our project goals, we propose a two-fold strategy focusing on the development of a recipe recommendation system and its subsequent evaluation.

In this paper, we aim to generate suitable and creative recipes by fine-tuning existing pre-trained models. We explore the current landscape of mainstream language models and evaluate their capabilities in text generation based on specific instructions. Preliminary investigations suggest that models such as GPT-2.0, Google Gemma, Bart, and Flan-T5 are good starting points. Specifically, GPT-2.0, a decoder-only model, can generate the next word based on preceding words within the same text, making it a suitable baseline for this question-answering task. In addition to GPT-2.0, models like Gemma, Bart, and Flan-T5, which are trained for handling a wide array of tasks including question answering, can be better candidates than GPT-2.0. These models, pre-trained on both supervised

and unsupervised tasks, and Bart, which incorporates a bidirectional encoder and an autoregressive decoder, are expected to outperform the simpler architecture of GPT-2.0 in terms of their capability of generating text[1]. Consequently, we use the un-fine-tuned GPT-2.0 as our baseline model. Although the other three models are better at general question-answering tasks, they still encounter difficulties in generating detailed cooking instructions, largely due to the fact that they are trained on diverse datasets intended for broad domain knowledge rather than specialized domains such as recipe generation. To improve their ability in understanding user inquiries and generating understandable instructions, we fine-tune these models on a dataset specifically tailored for recipe generation. We expect that the fine-tuned models will yield outputs that are more aligned with the ingredients specified in the input, as opposed to passively producing random words associated with cooking, as observed with the non-fine-tuned models.

The results of the fine-tuned models are evaluated against the evaluation data using various evaluation metrics and test sets, focusing on perplexity. In addition to these metrics, we also seek human evaluations by distributing questionnaires for individuals to evaluate text generation across four dimensions–reasonability, coherence, creativity, and conciseness. Overall, our results underscore how fine-tuning pre-trained models to learn specific domain knowledge can improve a model's performance in generating detailed and instructive recipes.

2 Related Work

The concept of utilizing language models for culinary applications aligns with an emerging interest in applying natural language processing (NLP) to diverse and practical tasks. In the realm of computational creativity in gastronomy, several works have demonstrated the utility of language models.

The introduction of advanced techniques in AI-driven culinary applications is exemplified by the work of Yu, Zang, and Wan in their study, "Routing Enforced Generative Model for Recipe Generation." (Yu et al., 2020) This paper pioneers the use of a routing algorithm to enhance recipe generation by modeling the intricate relationships between ingredients. Their approach effectively mitigates the complexity of ingredient interactions, enabling the generation of recipes that are not only creative and feasible but also tailored to user preferences. The model's ability to handle ingredient complexities and generate high-quality recipes is demonstrated through improvements in BLEU scores, F1 metrics, and human evaluations. This research sets a significant benchmark for our project. While our project does not directly employ the routing algorithm proposed by Yu, Zang, and Wan, the principles of handling complex ingredient interactions and focusing on user-specific recipe generation are highly pertinent. Our approach builds on the idea of ingredient-based personalization, aiming to enhance user engagement by generating cooking ideas that are both innovative and closely aligned with users' available resources and dietary constraints.

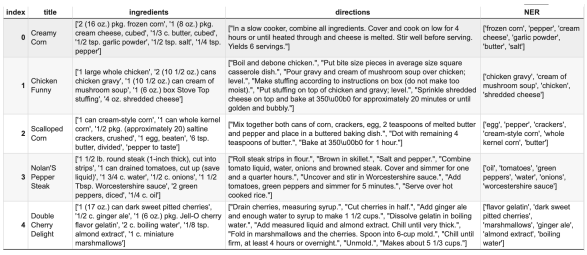
Majumder et al.(Majumder et al., 2019) address the challenge of generating personalized recipes by incorporating users' historical culinary preferences and partial knowledge of dishes. Their model uniquely applies an attention mechanism to merge technique- and recipe-level representations from a user's recipe history, effectively tailoring the recipe generation process to align with individual tastes and prior interactions. Their experiments, conducted on a substantial dataset of 180K recipes and 700K user interactions, demonstrate the model's capability to produce recipes that are not only plausible and coherent but also distinctly personalized, outperforming non-personalized baselines. This approach is particularly relevant to our project, which also seeks to personalize cooking ideas based on the ingredients users currently have at their disposal. While Majumder et al. focus on expanding recipe instructions from incomplete inputs using historical data, our project similarly aims to adapt recipe suggestions to fit the user's available pantry items and dietary preferences. By integrating insights from their methodology, we can enhance our system's ability to generate creative and personally relevant cooking ideas, thereby improving user engagement and satisfaction.

3 Dataset

For our project, MagicRecipe, we opted to utilize the Kaggle dataset "Recipe Recommendations" curated by Goomba16[2]. This dataset serves as the

[1] https://huggingface.co/docs/transformers/model_doc/bart

[2] https://www.kaggle.com/datasets/wilmerarltstrmberg/recipe-dataset-over-2m/data



index	title	ingredients	directions	NER
0	Creamy Corn	['2 (16 oz.) pkg. frozen corn', '1 (8 oz.) pkg. cream cheese, cubed', '1/3 c. butter, cubed', '1/2 tsp. garlic powder', '1/2 tsp. salt', '1/4 tsp. pepper']	['In a slow cooker, combine all ingredients. Cover and cook on low for 4 hours or until heated through and cheese is melted. Stir well before serving. Yields 6 servings.']	['frozen corn', 'pepper', 'cream cheese', 'garlic powder', 'butter', 'salt']
1	Chicken Funny	['1 large whole chicken', '2 (10 1/2 oz.) cans chicken gravy', '1 (10 1/2 oz.) can cream of mushroom soup', '1 (6 oz.) box Stove Top stuffing', '4 oz. shredded cheese']	['Boil and debone chicken.', "Put bite size pieces in average size square casserole dish.", "Pour gravy and cream of mushroom soup over chicken; level.", "Make stuffing according to instructions on box (do not make too moist).", "Put stuffing on top of chicken and gravy; level.", "Sprinkle shredded cheese on top and bake at 350\u00b0 for approximately 20 minutes or until golden and bubbly."]	['chicken gravy', 'cream of mushroom soup', 'chicken', 'shredded cheese']
2	Scalloped Corn	['1 can cream-style corn', '1 can whole kernel corn', '1/2 pkg. (approximately 20) saltine crackers, crushed', '1 egg, beaten', '6 tsp. butter, divided', 'pepper to taste']	["Mix together both cans of corn, crackers, egg, 2 teaspoons of melted butter and pepper and place in a buttered baking dish.", "Dot with remaining 4 teaspoons of butter.", "Bake at 350\u00b0 for 1 hour."]	['egg', 'pepper', 'crackers', 'cream-style corn', 'whole kernel corn', 'butter']
3	Nolan'S Pepper Steak	['1 1/2 lb. round steak (1-inch thick), cut into strips', '1 can drained tomatoes, cut up (save liquid)', '1 3/4 c. water', '1/2 c. onions', '1 1/2 Tbsp. Worcestershire sauce', '2 green peppers, diced', '1/4 c. oil']	['Roll steak strips in flour.", "Brown in skillet.", "Salt and pepper.", "Combine tomato liquid, water, onions and browned steak. Cover and simmer for one and a quarter hours.", "Uncover and stir in Worcestershire sauce.", "Add tomatoes, green peppers and simmer for 5 minutes.", "Serve over hot cooked rice."]	['oil', 'tomatoes', 'green peppers', 'water', 'onions', 'worcestershire sauce']
4	Double Cherry Delight	['1 (17 oz.) can dark sweet pitted cherries', '1/2 c. ginger ale', '1 (6 oz.) pkg. Jell-O cherry flavor gelatin', '2 c. boiling water', '1/8 tsp. almond extract', '1 c. miniature marshmallows']	['Drain cherries, measuring syrup.", "Cut cherries in half.", "Add ginger ale and enough water to syrup to make 1 1/2 cups.", "Dissolve gelatin in boiling water.", "Add measured liquid and almond extract. Chill until very thick.", "Fold in marshmallows and the cherries. Spoon into 6-cup mold.", "Chill until firm, at least 4 hours or overnight.", "Unmold.", "Makes about 5 1/3 cups."]	['flavor gelatin', 'dark sweet pitted cherries', 'marshmallows', 'ginger ale', 'almond extract', 'boiling water']

Figure 2: The first five tuples in the complete dataset, including four key attributes: "title," "directions," "ingredients," and "NER"

cornerstone of our model development and evaluation. It encompasses four key attributes: "title," "directions," "ingredients," and "NER" (Ingredients without amounts and brand information), which are pivotal for our model training and testing phases.

3.1 Dataset Cleaning and Splitting

In addition to the core attributes, the dataset includes supplementary information such as "link" (URL to the respective dish) and "site" (the domain from which the data was gathered). While these details enrich the dataset, our primary focus lies on leveraging the fundamental attributes to drive our model's performance and efficacy. So, these supplementary attributes were dropped as the first step of data cleaning, resulting in data with shape (2231142, 4).

We made the strategic decision to downsize our dataset, motivated by the necessity to optimize computational resources and streamline model training and testing processes. By randomly discarding half of the dataset's tuples, our aim is to achieve a harmonious equilibrium between data volume and computational feasibility, all while ensuring that the reduced dataset retains the essential characteristics of the original corpus. As a result of this curation process, our complete dataset now boasts a shape of (1231142, 4). The first five tuples in the complete dataset are shown in Figure 2 as an example.

With the complete dataset in hand, we employed the widely-used 80-10-10 splitting technique to partition our data for training, development, and testing purposes. Accordingly, the first 80% of the dataset, amounting to (984913, 4) tuples, was allocated for the training dataset. Similarly, 10% of the dataset, totaling (123114, 4) tuples, was earmarked for development, while the remaining 10%, comprising (123115, 4) tuples, was set aside for testing. After splitting, we facilitated ease of access and convenience by exporting all sets to CSV files. This step ensures seamless integration with various data processing and analysis tools. Each CSV file corresponds to its respective dataset—training, development, and testing—providing a structured and accessible format for further model development and evaluation.

Due to our limited GPU and RAM resources, we reduce the size of the training set by selecting only the first 10,000 examples, resulting in a new training set of size (10,000, 4). Similarly, the test set is downsized to include the first 1,000 examples. During training, the training set is further divided into two subsets, one used for actual training and the other for evaluation, maintaining an 8:2 ratio. Consequently, 8,000 examples are passed into our pre-trained models for fine-tuning, and the remaining 2,000 examples are used to evaluate model performance after each epoch throughout the training process.

To train a model that can generate dish recommendations and cooking instructions, we further preprocess our training, development, and test sets to make sure each dataset contains just two columns—one corresponds to the user's question which is our input text, and the other corresponds to the answer to the user's question which is our target text. We craft prompts for different scenarios.

In the first scenario, users may ask for recipes based on their available cooking ingredients. For this case, the input text should be structured as "I have <NER>. Can you give me some cooking ideas?" and the target text as "You can make <title>. Here's the instruction: <directions>", where NER, title, and directions are placeholders to be populated with data from the original dataset.

The second scenario addresses requests for specific dishes, where the input text is "How to make <title>?" and the target text follows the same format as the first scenario: "You can make <title>. Here's the instruction: <directions>".

To accommodate both scenarios, we split each dataset evenly. One half follows the format of the first scenario, and the other half is structured according to the second scenario. This approach

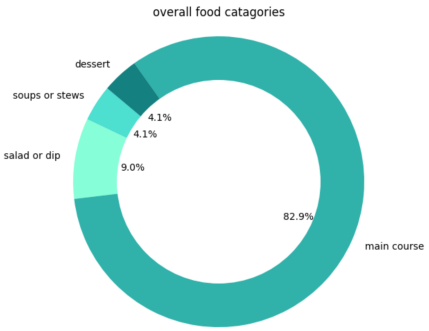


Figure 3: Pie chart representing the proportions of recipes within categories: 'soups or stews,' 'salad or dip,' 'main course,' and 'dessert'

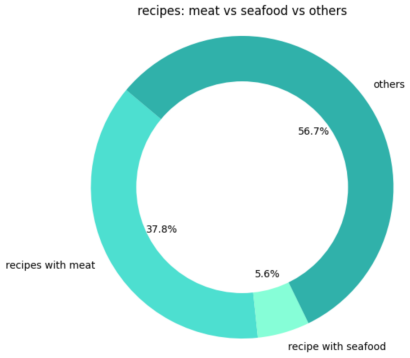


Figure 4: Pie chart representing the proportions of 'recipes with meat,' 'recipes with seafood,' and 'others'

allows our model to handle a variety of user input regarding recipes and cooking instructions.

Once the model is fine-tuned with the training set and optimized with a set of hyper-parameters, its performance is assessed against the test set–a set of unseen data which contains up to 1,000 examples. With this test set, perplexity is calculated and compared across different models.

3.2 Exploratory Data Analysis

In our preliminary analysis, we focused on two key aspects of the dataset: recipe categories and protein content distribution. The objective was to gain insights into the distribution and composition of recipes across different categories and protein types, with the intention of better understanding our dataset and its distribution. By examining these aspects, we aimed to uncover underlying patterns and trends that could inform subsequent model development and refinement in the MagicRecipe project.

3.2.1 Recipe Categories Analysis

We analyzed the distribution of recipes across various categories, including 'soups or stews,' 'salad or dip,' 'main course,' and 'dessert.' The pie chart (Figure 3) illustrates the proportion of recipes within each category, providing a visual representation of recipe distribution. Our analysis reveals that over half of our recipes belong to the main course category, underscoring its prominence within our dataset and highlighting potential areas for focused development and refinement.

3.2.2 Protein Content Distribution

Furthermore, we categorized recipes based on their protein content, distinguishing between 'recipes with meat,' 'recipes with seafood,' and 'others'. The pie chart (Figure 4) showcases the distribution of recipes across these protein categories, allowing for a comprehensive understanding of protein usage within the dataset. Our analysis indicates that general meat, excluding eggs, constitutes 37.8% of the recipes, emphasizing its prevalence and significance in recipe composition.

3.2.3 Recipe with Popular Meat Analysis

Following the protein content distribution, we extended our analysis to focus on recipes containing popular meat types. We examined recipes featuring pork, lamb, chicken, and beef and compared them with recipes containing eggs. The resulting comparison is depicted in the bar graph (Figure 5), where we observe a significant presence of recipes with eggs (95832), followed by chicken (60452), beef (9832), pork (7069), and lamb (1832). Understanding the distribution of recipe categories and protein content can provide insights into users' cuisine preferences. Also, recognizing the popularity of certain ingredients or categories can enable the model to offer customization options. Users could input their dietary preferences, allergies, or ingredient preferences, and the generator could tailor recipes accordingly.

4 Initial Result

We developed two baseline models for our investigation: a basic string matching algorithm and a more advanced GPT-2.0 model, each tailored to work with our collection of recipes.

4.1 String-match model

The preliminary model operates on a simple principle of keyword matching. It identifies key ingre-

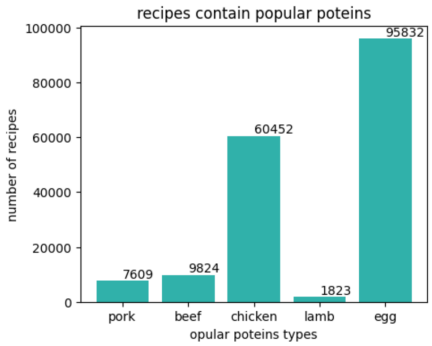


Figure 5: Bar graph showing number of recipes with popular meat types

dients in a user's query—for instance, "what dish can I make with tomato, potato, and egg?"—and matches these keywords against the ingredients listed under the "ingredients" attribute. If a dish's ingredient list contains all mentioned keywords, the model considers it a viable option and outputs its title and recipe. This model, while basic and not requiring training, offers a quick method for navigating the dataset but often suggests recipes that might require unavailable ingredients from the user.

4.2 GPT-2.0 model

The second model explores the capabilities of pre-trained models, specifically using GPT-2.0 as the starting point for our recipe generator. We evaluate the model's performance with the test set. Since GPT-2.0 is a decoder-only model that replicates the input as the output during training, it requires an additional data preprocessing procedure to tackle sequence-to-sequence tasks. The data preparation procedure includes combining the question (input) and the answer (target) into a single sentence, separated by special tokens to distinguish between the input and target. Additionally, each sentence is marked with a beginning-of-sentence token and concluded with an end-of-sentence token to differentiate between different examples. After these data preprocessing steps, the data is tokenized using the built-in tokenizer from Hugging Face. Upon evaluating the model across 64 batches, the calculated perplexity we get is 7.7742e29, which indicates GPT-2.0's non-ideal performance in generating recipes and cooking instructions. Although this model offers faster inference compared to the string matching approach, it tends to yield less accurate dish recommendations and repetitive cooking instructions. This can be explained by its decoder-only architecture that limits the model's ability to understand the input before generating an output; furthermore, this vanilla model has not been fine-tuned with a domain-specific dataset that would enable it to generate more detailed and reasonable recipes and cooking instructions.

5 Experiments

5.1 Gemma

Gemma is a family of lightweight, state-of-the-art open models developed by Google, built using the same research and technology as the Gemini models. These versatile text-to-text models are available in English, with open weights, pre-trained variants, and instruction-tuned variants. Gemma models come in two sizes: 2B and 7B parameters, each designed for different applications and hardware requirements. The 2B models are suitable for mobile devices and laptops, while the 7B models are intended for desktop computers and small servers. The Keras 3.0 multi-backend feature enables seamless compatibility with JAX, TensorFlow, and PyTorch, empowering developers to effortlessly choose and switch frameworks depending on the task at hand. Given Gemma's lightweight architecture, impressive performance, and flexibility for fine-tuning, it is well-suited for a wide range of natural language processing tasks.

To fine-tune the Gemma 2B model, we utilize the Keras 3.0 framework and load the pre-trained model weights from Kaggle. We employ the Low-Rank Adaptation (LoRA) technique, which efficiently adapts the model to new tasks by training only a small set of additional parameters. Specifically, we set the LoRA rank to 4, striking a balance between model performance and computational efficiency. The learning rate is set to 5e-5, and a weight decay of 0.01 is applied to prevent overfitting.

During the fine-tuning process, we use top-k sampling with k=5 to generate diverse and coherent responses. The model's performance is evaluated using three metrics: loss (sparse categorical cross-entropy), perplexity, and sparse categorical accuracy. These metrics provide insights into the model's learning progress and its ability to predict the next token in a sequence. The fine-tuning dataset consists of 8,000 examples, evenly split between two prompt structures (4,000 each). This

approach ensures that the model learns to respond to a variety of input patterns. The fine-tuning process is conducted for a single epoch, considering the dataset size and the model's quick adaptation capabilities.

The training is performed on an NVIDIA V100 GPU, which provides the necessary computational power to efficiently fine-tune the Gemma 2B model. The entire fine-tuning process takes 3,501 seconds, which is equivalent to 58 minutes and 21 seconds.

By leveraging the LoRA technique, carefully selecting hyperparameters, and using a diverse fine-tuning dataset, we successfully adapt the Gemma 2B model to perform specific tasks while maintaining its lightweight architecture and impressive performance.

After fine-tuning the Gemma 2B model using the aforementioned techniques and hyperparameters, we evaluate its performance on a held-out test set. The model yields an evaluation loss of 0.716, indicating that it has effectively learned to minimize the difference between its predictions and the ground truth. However, the perplexity value of 2,590,969.25 appears to be unusually high, suggesting that the model may be struggling to assign high probabilities to the correct next tokens in the sequence. This could be due to the complexity of the task or the need for further optimization of the model architecture and hyperparameters.

Despite the high perplexity, the model achieves a sparse categorical accuracy of 0.6287, which means that it correctly predicts the next token in the sequence approximately 62.87 percent of the time. This accuracy score demonstrates that the fine-tuned Gemma 2B model has learned to generate relevant and coherent responses to the given prompts, albeit with room for improvement.

To further enhance the model's performance, we could explore techniques such as increasing the size of the fine-tuning dataset, adjusting the learning rate and weight decay, or experimenting with different prompt structures. Additionally, analyzing the model's outputs and error patterns could provide valuable insights into areas where the model struggles and guide future fine-tuning efforts.

5.2 Bart

Developed by Facebook AI, BART (Bidirectional and Auto-Regressive Transformers) combines autoencoder and autoregressive elements in a Transformer architecture. Described by Lewis et al. (Lewis et al., 2019), it is pre-trained via a denoising process to excel in tasks like text generation, translation, and comprehension. This robust model reconstructs the original text from corrupted versions, showcasing its versatility across various language processing applications. BART's denoising pre-training equips it to handle incomplete inputs effectively, crucial for generating recipes from partial ingredient lists. Its autoregressive decoder produces linguistically fluent and structurally coherent recipes, ideal for transforming simple ingredients into detailed culinary instructions. The adaptability and strong pre-trained base of BART allow for efficient fine-tuning on our culinary dataset, enhancing its capability to generate personalized and innovative recipes.

In our project, we choose to fine-tune the facebook/bart-base model instead of its larger counterpart, facebook/bart-large. Although bart-large generally yields better training performance, its high resource consumption significantly impacts usability. Specifically, bart-large demands extensive RAM and CPU memory, making it impractical for our computing environment. By using bart-base, we can efficiently manage memory resources while maintaining a good balance between performance and operational feasibility. This model allows for a larger batch size of 4 during training and evaluation, compared to only 1 for the bart-large under the same hardware constraints.

In configuring the training parameters for our model, we utilize the Seq2SeqTrainingArguments from the Hugging Face Transformers library to optimize our training process. We direct the output of our training sessions to ./results and stored logs in ./logs, updating every 10 steps to monitor progress. The model trains for three epochs, with each training and evaluation batch set to four samples per device, balancing efficiency with our computational limits. To ensure a smooth adjustment to the optimal learning rate of 3e-4, we incorporate 500 warm up steps, which also aids in model convergence. We implement a weight decay of 0.01 for regularization to prevent overfitting. To handle our GPU memory constraints while effectively

increasing the training batch size, we use eight gradient accumulation steps. Additionally, enabling mixed precision training (fp16=True) allows us to speed up the training without significantly impacting model accuracy. Finally, setting the evaluation accumulation steps to one allows for more frequent updates during model evaluation, ensuring more immediate feedback on performance adjustments.

In the training of our BART model, we achieve a training loss of 1.628 and an evaluation loss of 0.719 after three epochs, demonstrating effective learning and strong generalization capabilities. The lower evaluation loss underscores the model's ability to generalize to new data, essential for its practical deployment in generating accurate and coherent recipes. These results validate our training strategy and the efficiency of our computational resource management. Additionally, we employ specific generation parameters to refine the text output, setting max length at 200, enabling sampling with do sample, and using controls 'temperature' at 0.72, and a repetition penalty of 1.4 to enhance the quality and variety of the generated recipes. This comprehensive approach highlights the effectiveness of our model configuration in balancing performance with computational efficiency, proving its suitability for the recipe generation task.

5.3 Flan-T5

T5 is an encoder-decoder model pre-trained on a mix of supervised and unsupervised tasks, where each task is transformed into a text-to-text format[3]. T5 can be utilized for various applications including translation, summarization, and question answering. Flan-T5 is an enhanced version of T5, fine-tuned on thousands of additional tasks. By incorporating instruction fine-tuning, Flan-T5 has demonstrated a better performance compared to T5 in both zero-shot and few-shot settings (Chung et al., 2022). Additionally, Flan-T5 allows developers to fine-tune the model for specific needs due to its highly customizable design. Given Flan-T5's encoder-decoder architecture, which makes it suitable for sequence-to-sequence tasks, and its flexibility for fine-tuning, we believe Flan-T5 is ideally suited for the question-answering scenario. We expect it to generate sensible recipes and cooking instructions that align with the user input.

We start our experiment by loading the pre-trained Flan-T5 model from Google, which offers variants including Flan-T5-small, Flan-T5-base, and Flan-T5-large. Considering the capabilities of our hardware, specifically the GPU and RAM, we select Flan-T5-base, which contains 250 million parameters and requires 990 MB of memory, which can be accommodated by the NVIDIA A100 GPU (Chung et al., 2022). The model is then fine-tuned using our preprocessed and tokenized training set, which includes 8,000 examples and an evaluation set of 2,000 examples. For performance evaluation, we utilize the ROUGE metric, which measures the similarity between the reference text and the generated text, aligning well with our dataset that includes ground-truth references[4].

In addition to the ROUGE score, the training parameters such as learning rate, batch size, and the number of epochs are carefully selected to optimize the model's learning. We find that a learning rate of 3e-5, a batch size of 8, and two epochs result in a reduction in training loss, and a weight decay of 0.01 helps prevent overfitting. Over the course of two training epochs, we observe that the training loss decreases from 1.4610 to 1.3430, the validation loss from 0.9487 to 0.9389, and rouge score from 0.1753 to 0.1758, indicating progressive improvement in the model's ability to understand user's question and generate recipes.

Training Loss	Validation Loss	Rouge1
1.343000	0.938856	0.175800

Once fine-tuning of the Flan-T5-base model is complete, the model is saved and subsequently reloaded for testing purposes to generate recipes and instructions based on user input. To ensure the model's output is detailed, coherent, and readable, we incorporate several inference parameters. Instead of selecting the most likely next token, the model is configured to sample from a probability distribution. To further improve coherence and readability, we implement top-k sampling, restricting the sample pool to the k most likely tokens. Additionally, to motivate creativity in the outputs, we adjust the temperature setting to control the randomness of the output. We also notice that the generation can be repetitive without enforcing the repetition penalty. Therefore we set repetition penalty to 1.3 to stop the text from repeating itself. The specific

[3] https://huggingface.co/docs/transformers/en/model_doc/t5

[4] https://en.wikipedia.org/wiki/ROUGE_(metric)

parameters used for recipe generation are detailed in the following table:

do_sample	top_k	temperature	repetition_penalty
True	50	0.7	1.3

6 Evaluation

6.1 Auto-evaluation

To compare the performance of our fine-tuned models, we employ an automated evaluation metric: perplexity. Perplexity is a widely used measure in natural language processing that assesses the quality of a language model by quantifying how well it predicts the next token in a sequence. A lower perplexity value indicates that the model is more confident in its predictions and better captures the patterns and structure of the language.

We choose to use perplexity as our primary evaluation metric instead of other common metrics such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation) or BLEU (Bilingual Evaluation Understudy). While ROUGE and BLEU are effective for evaluating the quality of generated text against a set of reference texts, they are more suitable for tasks with a limited range of correct answers, such as machine translation or text summarization. In contrast, generative language models like our fine-tuned models are designed to generate diverse and open-ended responses. The inherent creativity and variability in the model's outputs make it challenging to define a fixed set of reference texts for comparison. Perplexity, on the other hand, directly measures the model's ability to predict the next token in a sequence, which is a fundamental aspect of language modeling. By assessing how well the model has learned the patterns and structure of the language, perplexity provides a more appropriate evaluation metric for generative models.

In our evaluation, we calculate the perplexity scores for out fine-tuned Gemma 2B model, Flan-T5 Base model, and BART model. By comparing the perplexity values across these models, we can gauge the effectiveness of our fine-tuning process and determine whether our model has improved upon the pre-trained version.

gemma 2b	Bart	flan t5 base
2.0458	2.0914	2.6158

The perplexity scores for the three models - Gemma 2B (2.0458), BART (2.0914), and Flan-T5 Base (2.6158) - provide insights into their language modeling capabilities. Gemma 2B and BART demonstrate strong performance, with Gemma 2B being slightly more confident in predicting the next token in a sequence. Flan-T5 Base, on the other hand, exhibits a higher perplexity score, suggesting it may struggle more in generating coherent and contextually relevant responses compared to the other two models. While these perplexity scores offer a comparative measure, they should be considered alongside other evaluation metrics and human judgments to comprehensively assess the models' performance in specific natural language processing tasks.

By using perplexity as a consistent metric across the three models, we aim to provide a standardized comparison of their language modeling capabilities. This comparison will help us identify the strengths and weaknesses of our fine-tuned models and guide future improvements to enhance its performance in various natural language processing tasks.

6.2 Human Evaluation

To assess the quality of recipes generated by our model, we implement a human evaluation system. This evaluation is crucial as recipe generation does not have objective right or wrong answers, making standard automated metrics insufficient for fully capturing the nuances of recipe quality. Our approach involves a Recipe Rating Form, which is distributed to evaluators who are asked to rate generated recipes based on four specific criteria:

- Reasonability: whether the recipe appears logical and reasonable, including the appropriateness of the ingredients and steps.
- Coherence: whether the recipe is well-organized and easy to follow, with steps that logically flow from one to the next.
- Creativity: rate the creativity and originality of the recipe, looking for interesting variations on existing recipes.
- Conciseness: determine the conciseness of the recipe, focusing on the utility of the information provided and the absence of redundancy.

Each recipe is rated on a scale from 1 to 5 for these criteria, with multiple models' outputs being compared to gauge performance. This human-centric

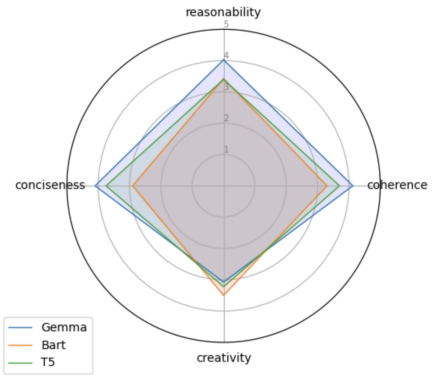


Figure 6: Comparative Evaluation of Gemma, Bart, and T5 Models on Recipe Generation: Scores are averaged across two prompts and evaluated on reasonability, coherence, creativity, and conciseness

approach ensures a comprehensive evaluation of our model's ability to generate practical, innovative, and user-friendly recipes.

We test three models—Gemma, Bart, and T5—on two prompts: 'How to make a cookie?' and 'I have salmon. Give me some ideas and instructions, please.' Each model generates responses that are evaluated based on reasonability, coherence, creativity, and conciseness by 20 college students who are independently living on their own and cooking on a daily basis. The results, depicted in the radar chart (Figure 6), represent the average scores across these dimensions.

In the human evaluation of recipe generation, each model showcased distinct strengths and weaknesses. Gemma performed consistently well across all criteria, scoring highest in coherence (4.125) and conciseness (4.1), and showing strong reasonability (4.025), but lagging slightly in creativity (3.075). This suggests that while Gemma is excellent at producing logical and well-structured recipes, it may be less innovative compared to the others. Bart, on the other hand, displayed a balanced profile with its highest score in creativity (3.5), reflecting its ability to generate novel and varied culinary ideas. However, it scored lower in coherence (3.3), conciseness (2.9), and reasonability (3.425), indicating some challenges in producing consistently clear and logical recipes. T5 presented a varied performance, excelling in conciseness (3.75) and scoring well in coherence (3.7) and creativity (3.225), but it was slightly behind in reasonability (3.4). This profile suggests that T5 is effective at generating coherent and compact recipes, although it may occasionally struggle with logical structuring compared to Gemma.

These detailed scores, illustrated in the radar chart, underline the unique capabilities and areas for improvement of each model in the complex task of automated recipe generation.

7 Conclusion

In conclusion, the MagicRecipe project successfully developed a personalized dish recommendation system that leverages advanced language models to generate recipes and shopping lists tailored to user preferences and available ingredients. By fine-tuning pre-trained models such as Gemma 2B, BART, and Flan-T5, the system demonstrated its ability to interpret user inputs and produce customized, coherent, and creative recipes.

The evaluation of the fine-tuned models using perplexity scores revealed that Gemma 2B and BART outperformed Flan-T5 in terms of language modeling capabilities. However, human evaluations provided a more comprehensive assessment, considering factors such as reasonability, coherence, creativity, and conciseness. Gemma showcased strengths in coherence and conciseness, while Bart excelled in creativity, and T5 demonstrated a balanced performance across all criteria.

The implications of this research are significant for both the culinary and NLP domains. The MagicRecipe system offers a novel approach to automating the creative process of recipe development, catering to the growing demand for personalized cooking experiences. Moreover, the project's success in fine-tuning language models for a specific domain highlights the potential for applying similar techniques to other creative and practical applications.

Future research could focus on further optimizing the models' performance, expanding the dataset, and incorporating additional features such as dietary preferences and nutritional information. Ultimately, the MagicRecipe project lays the groundwork for innovative NLP solutions that can transform the landscape of culinary creativity and enhance the cooking experience for users worldwide.

8 Code and Data

We put all our code in a google drive. Here is the link to google drive:
https://drive.google.com/drive/folders/

1URI9vOtsnKXOcfgZaQGna8mCu9jaHT49?usp=sharing

We get the dataset from Kaggle. Here is the link to the dataset:
https://www.kaggle.com/datasets/wilmerarltstrmberg/recipe-dataset-over-2m/data

References

Hyung Won Chung et al. 2022. Scaling instruction-finetuned language models. https://doi.org/10.48550/arxiv.2210.11416.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 2019 Conference of the Association for Computational Linguistics (ACL)*.

Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. 2019. Generating personalized recipes from historical user preferences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhiwei Yu, Hongyu Zang, and Xiaojun Wan. 2020. Routing enforced generative model for recipe generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3797–3806, Online. Association for Computational Linguistics.