Monte Carlo Tree Search for Recipe Generation using GPT-2

Karan Taneja[1,2], **Richard Segal**[2], **Richard Goodwin**[2]
[1] School of Interactive Computing, Georgia Institute of Technology, GA, US
[2] Computational Creativity Group, IBM Research, NY, US
karantaneja@gatech.edu, rsegal@us.ibm.com, rgoodwin@us.ibm.com

Abstract

Automatic food recipe generation methods provide a creative tool for chefs to explore and to create new, and interesting culinary delights. Given the recent success of large language models (LLMs), they have the potential to create new recipes that can meet individual preferences, dietary constraints, and adapt to what is in your refrigerator. Existing research on using LLMs to generate recipes has shown that LLMs can be fine-tuned to generate realistic-sounding recipes. However, on close examination, these generated recipes often fail to meet basic requirements like including chicken as an ingredient in chicken dishes. In this paper, we propose RecipeMC, a text generation method using GPT-2 that relies on Monte Carlo Tree Search (MCTS). RecipeMC allows us to define reward functions to put soft constraints on text generation and thus improve the credibility of the generated recipes. Our results show that human evaluators prefer recipes generated with RecipeMC more often than recipes generated with other baseline methods when compared with real recipes.

Introduction

With the vast number of cooking recipes available online and the success of large language models (LLMs) such as GPT-2 (Radford et al. 2019), researchers have investigated the automatic generation of recipes by fine-tuning LLMs on large datasets of food recipes (Bié et al. 2020; Lee et al. 2020). Automatic food recipe generation can be used in the creative process of recipe design where a chef can explore ingredient combinations, take inspiration for new recipes, write recipe drafts, or learn about flavor patterns. LLMs trained to generate recipes can help chefs by generating multiple possible recipes, completing incomplete ingredient lists and recipe instructions.

Large language models such as GPT-2 can be fine-tuned with large recipe datasets such as Recipe1M+ (Marin et al. 2021) and RecipeNLG (Bié et al. 2020) to generate reasonable-looking recipes. However, the quality of the output recipes is often limited due to the presence of repetitive text and inconsistencies between different parts of the recipe. There are two main reasons for this: (i) LLMs generate text by producing one token at a time and appending it to the existing text, resulting in a high focus on local coherence but a lack of attention to the long-term vision of the given context, and (ii) fine-tuned models are typically trained on smaller datasets compared to the original model, which can lead to subpar generalization over the target domain.

In this paper, we propose a method to sample from fine-tuned LLMs using Monte Carlo Tree Search (MCTS) and simple reward functions that put soft constraints on text generation. These constraints aim to eliminate the irregularities in generated recipes, improving their plausibility and making them more appealing to human evaluators. Our method does not require any additional training after the domain-specific fine-tuning of LLMs and can easily be wrapped over an API that exposes the next token probabilities. Our work takes inspiration from Chaffin, Claveau, and Kijak (2022) where a discriminator network is used as the reward function with MCTS to generate text conditioned on specific classes.

Figure 1 shows a real example of (i) an ingredients list generated from a given recipe name and (ii) recipe instructions generated from a recipe name and ingredients list. In the first example, while the recipe name mentions "Shrimp," the baseline method fails to add shrimp to the ingredients list. In contrast, our proposed method, RecipeMC, includes shrimp in the ingredients list as expected. In the second example, the baseline method does not employ "lemon zest" in recipe instructions while RecipeMC uses all the given ingredients as anticipated. Further, RecipeMC also meticulously generates the complete names "pepper Jack cheese" and "green onion" unlike the baseline method which used the shortened versions "grated cheese" and "onion".

This paper makes three main contributions:

1. We introduce RecipeMC, a text-generation process based on MCTS that controls the output text with simple manually-defined reward functions to softly constrain recipes generated by a fine-tuned LLM model.
2. We provide evidence that MCTS outperforms traditional sampling methods for recipe generation using common automatic evaluation metrics.
3. We conduct human evaluations that show that our generated recipes are often indistinguishable from human-created recipes and humans may prefer recipes generated by RecipeMC.

Next, we review the related literature to place our work in the context of previous research in food analysis and recipe generation. We then describe our proposed recipe

Recipe Name	John And Sarah's Best Sausage, **Shrimp**, Ham And Chicken Jambalaya
Ingredients (baseline)	4 celery ribs, chopped; 3-4 lbs chicken thighs; 1 teaspoon black pepper; 3 cups long grain white rice; 2 tablespoons garlic, chopped; 6 bay leaves; 1 teaspoon cayenne; 2 teaspoons salt; 1 lb smoked sausage; 1 teaspoon dried thyme; 1 cup green bell pepper, chopped; 1 cup ham, diced; 1 cup sweet onion, chopped; 2 cups chicken stock or 2 cups chicken broth; 4 cups water; 1 1/2 cups scallions, chopped; 8 tablespoons vegetable oil; 1 teaspoon oregano; 3 cups tomatoes, chopped (2 lb); 1 teaspoon white pepper
Ingredients (RecipeMC)	1 teaspoon dried oregano; 2 cups long grain white rice; 2 teaspoons cajun seasoning; 1 teaspoon dried basil; 1 teaspoon cayenne; 1/2 lb smoked ham, cut into 1/4 inch dice; 1 lb chicken, boneless, skinless, cut into 1/2 inch dice; 1 teaspoon dried thyme; 1 lb andouille sausage, cut into 1/4 inch slices; 1/2 lb **shrimp**, shelled, deveined and cut into 1/4 inch dice; 2 tablespoons olive oil; 2 teaspoons salt; 1 large onion, chopped; 4 cups chicken broth; 2 garlic cloves, minced

Recipe Name	Crab-Stuffed Corn Muffins Casserole
Ingredients	1 1/2 tablespoons mayonnaise; 1 tablespoon minced green onion; cooking spray; 1 cup cornmeal; 1/4 cup grated pepper Jack cheese; 1/2 teaspoon baking soda; 1/2 cup grated pepper Jack cheese; 1/4 teaspoon Worcestershire sauce; 1/2 cup melted butter; 1 cup buttermilk; 8 ounces cooked crabmeat; 2 eggs; salt to taste; 1 cup all-purpose flour; 1 teaspoon Asian chili paste (sambal); 1 teaspoon fresh grated **lemon zest**; 1/2 teaspoon salt
Instructions (baseline)	Preheat oven to 350 degrees F (175 degrees C). Spray a 12-cup muffin tin with cooking spray. Whisk together the eggs, buttermilk, and melted butter in a bowl. Stir in 1 cup of the flour, 1/2 of the cornmeal, the baking soda, 1/4 of the salt, the grated cheese, onion, mayonnaise, chili paste, and Worcestershire sauce. Mix in the crabmeat [...]
Instructions (RecipeMC)	Preheat the oven to 400 degrees F (200 degrees C). Spray 12 muffin cups with cooking spray. Combine cornmeal, flour, baking soda, and salt in a large bowl. Whisk together buttermilk, butter, eggs, pepper Jack cheese, mayonnaise, chili paste, green onion, **lemon zest**, and Worcestershire sauce in a separate bowl. Stir into the dry ingredients [...]

Figure 1: Examples of inconsistencies in generated recipes. In the first example, an important ingredient "Shrimp" is missing from the baseline ingredients list. In the second example, baseline instructions do not use the ingredient "lemon zest". Our method, RecipeMC, also refers to the complete names of the ingredients such as "pepper Jack cheese" and "green onion" rather than shortening them to "grated cheese" and "onion" respectively.

generation method RecipeMC including GPT-2 fine-tuning, MCTS, and the reward functions. This is followed by a section on experiments and results including the automatic evaluation of RecipeMC with three other baseline methods and a human evaluation study. Finally, we wrap up with a discussion, our ideas for future work, and concluding remarks.

Related Literature

Food and Recipe Analysis: Salvador et al. (2017) introduced the Recipe1M dataset containing over 1M recipes with 800K food images. With these recipes and corresponding food images, they trained text and image models to generate embeddings in a joint embedding space. These text and image models with their common embedding space were used to retrieve recipes from food images (im2recipe retrieval task) by matching the embedding of a given image with those of recipes in an existing database. Marin et al. (2021) extended this dataset to Recipe1M+ by further adding 13M food images. Min et al. (2017) proposed a Multi-Modal Multi-Task Deep Belief Network (M3TDBN) to learn from multi-modal content and multi-attribute information in the food domain for cuisine classification, recipe image retrieval, and ingredient and attribute inference from food images. Herranz, Min, and Jiang (2018) reviewed work involving multiple modalities in food analysis including text, images, location, and cuisine. Our paper focuses on coherent recipe text generation beginning with a recipe name to create ingredient lists and instructions.

Recipe Generation: Chef Watson (Varshney et al. 2019) was based on a Bayesian model over a knowledge-representation schema containing culinary information such as relations between ingredients, geolocations, and chemical composition of ingredients in terms of flavor compounds. Chef Watson could generate creative recipes whose quality was verified by professional chefs. Wang et al. (2020) proposed a method to generate recipes from food images using the Recipe1M dataset based on unsupervised extraction of paragraph structures and generating tree structures from images for a structure-aware generation. Bié et al. (2020) introduced the RecipeNLG dataset for recipe generation which was created by expanding the Recipe1M recipes with over one million new recipes. Lee et al. (2020) introduced RecipeGPT, a GPT-2 based recipe generation model and evaluation system with a user interface for examining

the quality and encouraging experimentation. Reusch et al. (2021) introduced RecipeGM which generated recipes from a given list of ingredients (without quantities) using a hierarchical self-attention-based sequence-to-sequence model. This model was proposed by Fan, Lewis, and Dauphin (2018) for story generation where long-range dependencies are an important challenge. Overall, RecipeGPT performs consistently better than RecipeGM except at n-gram repetition, but note that RecipeGPT is a much bigger model as compared to RecipeGM. Antognini et al. (2022) developed a method to edit recipes through critiques where the latent representation of the recipe is modified using its gradient with respect to the critique. The critique is a list of desired ingredients in a recipe beginning with an initial recipe. In this paper, we propose RecipeMC which uses GPT-2 model for recipe generation with soft constraints on ingredient lists and instructions for improving the coherence and overall quality of the recipes. Our baseline methods use the same model as RecipeGPT, but use top-p sampling instead of top-k since it has been shown to consistently generate higher quality and more diverse text (Holtzman et al. 2020). We show that RecipeMC consistently performs better than these baselines on all metrics and human evaluation.

Constrained Text Generation: LLMs, such as GPT-2, struggle with maintaining the context of the prompt when generating structured responses. Researchers have explored constrained text generation using different methods. Zhang et al. (2020) and Hsieh, Lee, and Lim (2021) propose insertion-based transformer models to impose hard constraints that ensure the inclusion of given entities in the output text. To impose these constraints, these models progressively add tokens between the given entity tokens to generate text but also inadvertently constrain the order in which tokens are produced. Chaffin, Claveau, and Kijak (2022) propose a method for controlling generation using MCTS and a discriminator model as the reward function to generate text conditioned on a given discriminator class. This imposes a soft constraint on the output text, unlike insertion-based methods that enforce hard constraints. Our work takes inspiration from this paper but uses simpler manually-defined reward functions instead of a discriminator model. We linearly combine these reward functions to impose several soft constraints that encourage coherence and text quality in the recipes. This can be used in future research to allow human-in-the-loop collaborative recipe generation where humans prescribe the reward functions.

Recipe Generation

GPT-2 (Radford et al. 2019) is an LLM based on the transformer architecture (Vaswani et al. 2017) and can generate text conditioned on an initial prompt using next token prediction. GPT-2 was trained on a large corpus with millions of web pages. Fine-tuning a pre-trained LLM for specific tasks has been shown to have a significant advantage over training a model from scratch (Radford et al. 2018). For our method, RecipeMC, we fine-tuned a GPT-2 model using a large corpus of recipes collected from the internet. Each recipe contains three components: recipe name, ingredients list, and recipe instructions. Figure 2 shows the recipe format we use to fine-tune our language model. Here, <|startofname|> and <|endofname|> are special tokens used to denote the start and end of the recipe name respectively. Similar tags are also used for denoting the ingredient and instruction sections. We fine-tuned the GPT-2 model using the RecipeNLG dataset (Bié et al. 2020) which contains over 2.1 million recipes. We cleaned the dataset to remove text containing unwanted information such as text advertisements, empty or short ingredients, and instructions.

```
<|startofname|>Recipe Name<|endofname|>
<|startofingr|>
  Ingredient 1; Ingredient 2; ...;
  Ingredient n
<|endofingr|>
<|startofinst|>
  Instruction 1. Instruction 2. ...
  Instruction m.
<|endofinst|>
```

Figure 2: Recipe format used to fine-tune GPT-2. Special tokens define the start and end of each recipe section.

Prompt:

```
<|startofname|>
Chocalate Chip Cookies
<|endofname|>
```

Response:

```
<|startofingr|>
1/2 cup butter; 1/2 cup sugar;
1 large egg; 2 cups all-purpose flour;
1 cup semi-sweet chocolate chips;
<|endofingr|>
```

Figure 3: In the NAME→INGR task the system is given a recipe name and asked to generate a list of ingredients.

Similar to Lee et al. (2020), we also experimented with a multi-task learning setup where a model is trained to generate multiple possible orderings of name, ingredients, and instructions instead of only the default name-ingredient-instruction ordering. We did not see a decrease in perplexity over the test set, but rather a small increase, and decided to use the simpler single-order setup described in Figure 2.

For evaluation purposes, we split the recipe generation problem into two separate tasks. Figure 3 shows the NAME→INGR task where we prompt the system with the recipe name and ask the system to generate a list of ingredients. Figure 4 shows the NAME+INGR→INST task where we prompt the system with the recipe name and ingredients, and ask the system to generate the recipe instructions. We have split the task in this manner to simplify the evaluation of the two distinct types of structured text present in recipes. Note that running these two tasks one after another allows us to generate complete recipes from just the recipe name.

LLMs have a tendency to repeat text, especially the previous sentence at any point. These repetitions have a self-

Prompt:

```
<|startofname|>
Chocalate Chip Cookies
<|endofname|>
<|startofingr|>
1/2 cup butter; 1/2 cup sugar;
1 large egg; 2 cups all purpose-flour;
1 cup semi-sweet chocolate chips;
<|endofingr|>
```

Response:

```
<|startofinst|>
Preheat oven to 350F.  Combine all
ingredients in mixing bowl.  Mix in
chocolate chips. Place on baking sheet
and bake for 10 minutes.
<|endofinst|>
```

Figure 4: In the NAME+INGR→INST task the system is given a recipe name and a list of ingredients. The system is then asked to generate cooking instructions.

reinforcing effect — the probability of repeating a sentence successively increases with each repetition (Xu et al. 2022). Penalizing repetitions during inference can help mitigate this problem to some extent. Two common methods for inference time mitigation include (i) strictly disallowing n-gram repetitions and (ii) exponentially penalizing token repetitions (Shirish Keskar et al. 2019). Both these methods modify the token probabilities of LLMs during inference. A second major limitation of LLMs is their inability to reliably model long-range dependencies which leads to inconsistent text. For example, ingredients mentioned in the ingredients list may never be used in the generated recipe instructions (see Figure 1). Indiscriminately penalizing outputs for repetitions can further aggravate this problem because some repetitions, such as ingredients in recipes, naturally arise owing to their structure.

Monte Carlo Tree Search

MCTS is a search algorithm commonly used in AI agents for playing strategy games such as Chess, Go, and Checkers (Świechowski et al. 2022). It balances exploitation and exploration to efficiently search a large search space to find the path that maximizes a user-provided reward function. In text generation, MCTS can take a long-term view of the text generation process because it evaluates multiple possible paths emanating from the text prompt (Chaffin, Claveau, and Kijak 2022). The algorithm works by maintaining a partial search tree of all possible sentences as shown in Figure 5. The root node r of the tree represents the initial prompt $x_{1:r}$. The children of each node represent the possible continuations of the current sentence. MCTS starts with just the root node and then expands one node in each iteration of the algorithm. The figure shows the state of the algorithm after two iterations have been completed, and the root and "together" nodes have been expanded. The figure also demonstrates how the third iteration proceeds. In each iteration, MCTS

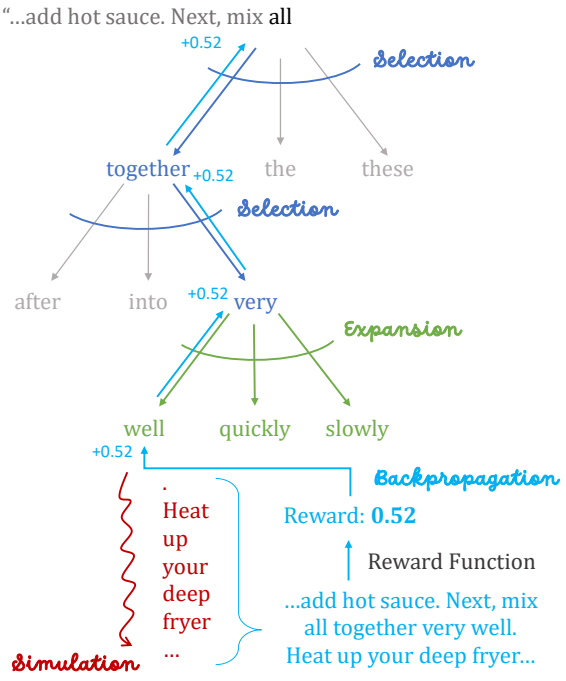

Figure 5: An illustrative example of RecipeMC in action for generating instructions. The initial prompt is "...add hot sauce. Next, mix all". The four MCTS steps Selection, Expansion, Simulation and Backpropagation are shown and color coded.

performs the following main steps:

1. **Selection:** Starting from the root node, MCTS iteratively selects a child node to explore until it reaches a leaf node which will be expanded in the next step. The node selection is based on maximizing a variant of PUCB (Predictor + Upper Confidence Bound) (Rosin 2011; Silver et al. 2017) over the child nodes:

$$\text{PUCB}(i) = Q(i) + c \cdot p(x_i|x_{1:i-1}) \frac{\sqrt{N}}{n_i + 1}$$

where the exploitation term $Q(i)$ is the average score from generating token x_i given $x_{1:i-1}$. $Q(i)$ is initialized with 0 whenever a node is created in the expansion step. It is updated for all selected nodes during backpropagation. The second term, c, is a constant that controls the weight of exploration and exploitation; n_i is the count of times that the child node i was visited; and $N = \sum_i n_i$ is the total number of iterations. The term $p(x_i|x_{1:i-1})$ is the predictor probability that serves as a prior. RecipeMC uses the output probabilities from the fine-tuned GPT-2 model for $p(\cdot)$. Note that PUCB(i) is also well-defined for unexplored child nodes that have $n_i = 0$. In Figure 5, the selection algorithm first picks "together" because it maximizes PUCB among the three options. It then selects "very" because it also maximizes PUCB. The selection step ends at "very" as it is a leaf node.

2. **Expansion:** From the selected leaf node l, we expand the tree by adding child nodes corresponding to the top-k tokens predicted by the fine-tuned GPT-2 model. We initialize the prior for each child by normalizing the probabilities over the top-k tokens to sum to one. In Figure 5, three nodes corresponding to the words "well", "quickly" and "slowly" are added. We used top-k here to limit the tree size as top-p will lead to an indeterministic size.
3. **Simulation:** We perform standard top-p sampling from the selected leaf node to generate the next t tokens giving us a text sequence $x_{1:l+t+1}$. In Figure 5, first, the leaf node "well" is selected using PUCB. Then, top-p sampling is used from "well" to generate the sequence "[period] Heat up your deep fryer [...]."
4. **Backpropagation:** We calculate the reward for text sequence $x_{1:l+t+1}$ and update aggregate scores $Q(\cdot)$ and increment $n_{(\cdot)}$ starting from node $l+1$ to the root node r. In Figure 5, the scores $Q(\cdot)$ are updated to accumulate a reward of 0.52, and $n_{(\cdot)}$ is incremented for the words "well", "very", "together", and the root node.

Finally, after repeating the above four steps Z times, one can decide the next token at the root node by choosing the child node with the highest $Q(r+1)$ or the node with the highest n_{r+1}. In our work, we select the node with the highest $Q(r+1)$ as the next token at the root. We used $Z = 20$, $c = 1$, $k = 50$, $p = 0.9$, and $t = 30$ without any further fine-tuning. To generate ingredients or instructions, this process is repeated until the corresponding end tag is reached.

Reward Functions

While previous MCTS work on LLMs has used discriminator models to guide the MCTS search (Chaffin, Claveau, and Kijak 2022), we use hand-designed soft constraints implemented by simple reward functions.

The reward functions discussed below use a predefined list of common ingredients such as milk, eggs, butter, chicken, etc. which we call *constituents*. To create the constituents list, we applied the NYT Ingredient Phrase tagger[1] (NYT-IPT) over the ingredient phrases in the RecipeNLG dataset. The NYT-IPT allows tagging of quantities, units, ingredient names, and comments within ingredient phrases. We extracted the set of ingredient names from all ingredient phrases in the ingredients lists and filtered them to remove any constituents with non-alphabet characters or stop words ("and", "or", etc.). Further, we filtered out constituents that could be decomposed into other constituents present in the list. The final list has 2,122 constituents.

Since NAME→INGR and NAME+INGR→INST tasks have very different outputs, they require different reward functions. The reward functions discussed below were designed to address several structural shortcomings we found in recipes that were generated without MCTS. For instance, several generated recipes without MCTS failed to include the key ingredient that defines the recipe such as not including chicken as an ingredient in "Chicken Masala." In RecipeMC, we used the following three reward functions for NAME→INGR task:

- **Name & Ingredients Coherence:** This function rewards the model for using ingredients names present in the recipe name. For example, for the recipe *Chocolate Apple Pie*, the function rewards outputs with ingredients *Chocolate* and *Apple*. We first search for constituents in the recipe name. If $z > 0$ constituents are found, we search these z constituents in the generated ingredients list and, say, z_f are found. The reward value is $z_f/z \in [0, 1]$ if $z > 0$, and 1 otherwise.
- **Constituents Repetition Penalty:** This function penalizes any repetition of constituents in the ingredient list. Let p be the sum of the number of times a constituent is repeated. Note that we do not count the first occurrences. Similarly, let q be the sum of the number of times the ingredient phrases separated by ";" are repeated. Then, the reward is given by $e^{-p-q} \in (0, 1]$.
- **Closing Ingredients List:** This function rewards the `<|endofingr|>` token generation. If the tag is found, the reward is 1, and 0 otherwise.

The total reward q is defined as the weighted sum of the reward functions ($q = \sum_i w_i r_i$) where weight $w_i \in (0, 1)$ is assigned such that $\sum_i w_i = 1$. This ensures $q \in [0, 1]$ since each function value $r_i \in [0, 1]$. We used the weights 0.30, 0.45, and 0.25 for the above three functions respectively without any hyper-parameter fine-tuning.

Similarly, we also use three reward functions for the NAME+INGR→INST task:

- **Ingredients & Instructions Coherence:** Similar to *Name & Ingredients Coherence*, this function rewards the use of constituent names present in the ingredients list when generating recipe instructions. We first search for constituents in the ingredients list. If $z > 0$ constituents are found, we search these z constituents in the generated instructions (say z_f are found). The reward value is $z_f/z \in [0, 1]$ if $z > 0$, and 1 otherwise.
- **Special Characters Repetition Penalty**: We observed the model tends to repeat some characters like "!" and "-" because they may be repeated in some training recipe instructions. If the sum of occurrences of these characters is s, the reward is given by $e^{-s/S} \in (0, 1]$. We used $S = 3$ to avoid excessive penalization for using these characters.
- **Closing Recipe Instructions:** This functions rewards the `<|endofinst|>` tag. If the tag is found, the reward is 1, and 0 otherwise.

For generating instructions, we used the weights 0.50, 0.20 and 0.30 for the above three functions respectively without any hyper-parameter fine-tuning.

Experiments and Results

We create a LLM for our experiments by fine-tuning GPT-2 on our cleaned RecipeNLG data. We use the same LLM for all our experiments. We separately evaluate RecipeMC on the NAME→INGR task and the NAME+INGR→INST task.

[1] https://github.com/nytimes/ingredient-phrase-tagger

Table 1: Automatic evaluation results for NAME→INGR task using different sampling methods. The down arrow (↓) indicates that lower is better.

Sampling Method	Coherence	F_1-Score	Perplexity↓	ROUGE-1	ROUGE-2	BLEU	Repetition↓
Ground Truth	0.451	-	2.934	-	-	-	0.667
Top-p	0.443	0.572	4.173	0.457	0.200	0.155	1.724
+ No 4-gram Repetition	0.444	0.562	5.150	0.456	0.198	0.144	1.641
+ Repetition Penalty	0.413	0.548	6.754	0.407	0.135	0.115	0.711
RecipeMC	**0.513**	**0.597**	**3.961**	**0.505**	**0.242**	**0.210**	**0.192**

We compare RecipeMC with following three baseline sampling methods commonly used with LLMs:

- **Top-p Sampling:** Top-p sampling, also known as nucleus sampling (Holtzman et al. 2020), uses tokens with the highest probabilities that cumulatively add up to the nucleus size p and zeroes out the probability of other tokens. This is an adaptive version of top-k sampling where exactly k tokens with the highest probabilities are considered independent of their cumulative probability. This method has been shown to generate more diverse and interesting text than other greedy approaches.
- **Top-p Sampling with Repetition Penalty:** To prevent the repetition of tokens, the output logit values of repeated tokens are divided by a parameter $\theta > 1$ and the distribution is re-normalized(Shirish Keskar et al. 2019). We use the recommended value $\theta = 1.2$ along with top-p sampling as a baseline.
- **Top-p Sampling with No n-gram Repetitions**: This baseline method also uses top-p sampling but forbids repetition of n-grams. To ensure that no n-gram is repeated, we can search for the last n-1 generated tokens in the sequence generated so far and find the list of tokens that follow them. These tokens should not be generated to ensure that there are no n-gram repetitions. This method enforces a strict constraint unlike the repetition penalty but it allows repetitions of n-1-grams or smaller sequences without any penalization. We used $n = 4$ in our experiments to allow for some margin in repetition.

Automatic Evaluation

We compare RecipeMC with the three baselines on several standard metrics. We used 1,000 test recipes with ground-truth ingredient lists and instructions and compare the generated ingredient lists and instructions for each method with the ground truth. For some metrics where ground truth is not required, we also report the values for ground truth as an oracle reference. For the NAME→INGR task, we gave the recipe name from each test recipe as a prompt and sampled the LLM's output ingredients list using all three baseline methods and RecipeMC till the model generated the end tag. Similarly, for the NAME+INGR→INST task, we gave the recipe name and ingredients from each test recipe as a prompt and sampled the output instructions. The results from the automatic evaluation for NAME→INGR and NAME+INGR→INST tasks are summarized in Table 1 and 2 respectively.

Coherence: We compare the *coherence* of different methods by comparing constituents present in the generated ingredients list and the recipe name, and between instructions and the ingredients list. This definition is inspired by the definition of coherence given by Lee et al. (2020). For NAME→INGR task, we define coherence with *Name & Ingredients Coherence* function defined earlier. Similarly, for NAME+INGR→INST task, we define coherence with the *Ingredients and Instructions Coherence* function. The results in Tables 1 and 2 show that RecipeMC achieves the highest coherence, surpassing even the ground-truth recipes. These results confirm that the Coherence reward functions have the desired effect of improving the coherence of the recipes generated with MCTS.

F_1-Score: (only for NAME→INGR task) In order to compare the quality of generated ingredients with respect to ground truth, we search for constituents in the ground-truth ingredients list and calculate the average precision and recall for the generated ingredients list. The F_1-Score is the harmonic mean of average precision and recall. The results in Table 1 show that RecipeMC has the highest F_1-score among all methods confirming that the *Name & Ingredients Coherence* reward also leads to higher ingredients accuracy with respect to the ground truth.

Perplexity: The perplexity of a sample, defined with respect to a language model, measures the surprise of the model in seeing the given example. It is defined as the exponent of cross-entropy over the sequence of tokens in a given text. Even though LLMs are trained to minimize the cross-entropy loss, the text-generation process or the inference method can influence the perplexity of a sampled text. For this metric, we only consider the perplexity of the generated part of the output and mask the output probability for the prompt text. The result in Tables 1 and 2 show that RecipeMC generates recipes with the lowest perplexity, but still more than that of the test dataset. This improvement over other baseline methods is because MCTS allows us to look ahead before the next token generation and to avoid tokens that later lead to poor outputs with lower reward values. It is interesting to note that our method did not explicitly reward lower perplexity, but its low perplexity is a consequence of the soft constraints imposed by the reward functions and the additional search performed by MCTS.

ROUGE & BLEU: ROUGE or Recall-Oriented Understudy for Gisting Evaluation (Lin 2004) is a set of metrics to evaluate the quality of text in summarization and machine-translation tasks. It measures the quality of output text by

Table 2: Automatic evaluation results for NAME+INGR→INST task using different sampling methods. The down arrow (↓) indicates that lower is better.

Sampling Method	Coherence	Perplexity↓	ROUGE-1	ROUGE-2	BLEU
Ground Truth	0.486	4.115	-	-	-
Top-p	0.709	7.948	0.338	0.102	0.067
+ No 4-gram Repetition	0.690	8.441	0.339	0.103	0.069
+ Repetition Penalty	0.416	11.680	0.301	0.072	0.044
RecipeMC	**0.768**	**7.337**	**0.362**	**0.115**	**0.080**

measuring the overlap, i.e. recall, precision, and accuracy of n-grams between the output text and reference texts. We use ROUGE-1 and ROUGE-2 F_1 values to measure unigram and bigram overlap between the generated recipe texts and the original recipe texts. BLEU or Bilingual Evaluation Understudy (Papineni et al. 2002) metric was proposed to measure the quality of machine-translation systems and has been shown to have a high correlation with human judgment. It combines the precision of n-grams where $n = 1, 2, 3, 4$, and a brevity penalty for generating output text shorter than reference text. We measure the BLEU score of output ingredients and instructions by comparing them to the original ones in the test set. The results in Tables 1 and 2 show that RecipeMC generates recipes closest to the ground-truth recipes. Note that the same GPT-2 model was used for each method, but the sampling process used by RecipeMC led to higher-quality ingredient lists and instructions.

Repetition: (only for NAME→INGR task) Repetition is defined as the average number of repetitions of constituents in the ingredients list. Zero value indicates that no constituents were repeated in the ingredients list. Table 1 shows that RecipeMC leads to minimum repetitions, but the repetition value for RecipeMC is even lower than that of the ground-truth recipes. This may indicate that we are over-penalizing the constituent reuse in the ingredients list.

Output Length: The average character length of output ingredient lists and instructions are reported in Table 3. We observe that ground-truth recipes have the shortest length. RecipeMC has the shortest length among all sampling methods since we reward the generation of the end tag. It is interesting to note that top-p sampling with repetition penalty has the shortest length among the baseline methods for ingredient lists but a much higher length than the other baselines for instructions. We observed on inspection that the repetition penalty led to a quicker generation of the end tag for ingredients but created a *blabbering* effect for instructions where the model generates extremely elaborate instructions and occasional unrelated text about alternate or complementary recipes using some ingredients that were not present in the given ingredients list.

Overall, RecipeMC outperformed the baseline methods while balancing several objectives which led to good-quality recipes. Using top-p sampling without any constraints outperformed other baseline methods except for its highly repetitive text as constituents were repeated 1.7 times on average as compared to 0.7 times for ground truth. The top-p sampling baselines with No 4-gram Repetition and Repetition Penalty led to lower repetitions as compared to top-p sampling, but it reduced coherence, F_1-Score, ROUGE, and BLEU values and increased perplexity. RecipeMC achieved the best of both worlds with higher F_1-score, ROUGE, and BLEU values when compared to ground-truth recipes with the least repetitive text. The smaller average length of output for RecipeMC (Table 3) also confirmed that it was able to succinctly capture more relevant information. As discussed next, our human evaluations also confirmed that humans prefer recipes generated by RecipeMC.

Table 3: Average character length of the ingredients list and instructions for different sampling methods.

Method	Ingredients	Instructions
Ground Truth	167	240
Top-p	247	485
+ No 4-gram Repetition	248	484
+ Repetition Penalty	233	545
RecipeMC	190	441

Human Evaluation

To evaluate each method on NAME→INGR task using human evaluation, we created a *Recipe Turing Test* where a human evaluator looks at two possible ingredient lists, the real one and the generated one, for a given recipe name, and their task is to **choose the generated ingredients list** among these. For a fair comparison, we uniformly shuffled the real and generated recipes to show them on the left or right side. Also, evaluators were not aware that four different methods were being evaluated. We perform a similar test for NAME+INGR→INST task where human evaluators are asked to **choose the generated instructions** based on a recipe name and the associated ingredients list.

We randomly sampled 50 recipes for each of the four methods and created a total of 200 binary-choice questions. The evaluators see 10 randomly-chosen questions (without replacement) for NAME→INGR task and 5 questions for NAME+INGR→INST task on a separate test. Note that all evaluators did not take both the tests. The results of the human evaluation for the two tasks are shown in Tables 4 and 5. *Real* and *Gen.* columns count the number of times the real and generated recipes were selected by the evaluator. P(Incorrect) is the probability that humans incorrectly identified the real recipe as the generated one. This is calculated

Table 4: Human evaluation results on the NAME→INGR task. Data was collected from 147 evaluators who answered 10 questions each. Evaluators were asked to identify the generated ingredients (**Gen.**), hence P(**Incorrect**) = P(**Real**) = **#Real / (#Real + #Gen.**).

Method	**Real**	**Gen.**	P(**Incorrect**)
Top-p	175	185	0.4861
+ No 4-gram Repetition	179	200	0.4723
+ Repetition Penalty	183	180	0.5041
RecipeMC	201	167	**0.5462**
Overall	738	732	0.5020

Table 5: Human results on the NAME+INGR→INST task. Data was collected from 83 evaluators who answered 5 questions each. Evaluators were asked to identify the generated ingredients (**Gen.**), hence P(**Incorrect**) = P(**Real**) = **#Real / (#Real + #Gen.**).

Method	**Real**	**Gen.**	P(**Incorrect**)
Top-p	51	42	0.5484
+ No 4-gram Repetition	67	62	0.5194
+ Repetition Penalty	36	65	0.3564
RecipeMC	57	35	**0.6196**
Overall	211	204	0.5084

as follows:

$$P(\text{Incorrect}) = \frac{\#\text{Real}}{\#\text{Real} \ + \ \#\text{Generated}}.$$

The results in Tables 4 and 5 show that human evaluators are most likely to believe that RecipeMC ingredient lists and instructions are human-generated when compared to those produced by the other baselines. Human evaluators believed recipes generated by RecipeMC to be human-generated more often than the next best method, top-p sampling with Repetition Penalty, for NAME→INGR task with a p-value of 0.128. Similarly, RecipeMC outperforms top-p sampling on the NAME+INGR→INST task with $p = 0.164$. While we did not observe statistically significant ($p < 0.05$) improvements with RecipeMC results, we observe that results from human evaluation correlate well with results from automatic evaluation presented in Tables 1 and 2.

We also observed that top-p sampling with repetition penalty performed at par with other baselines for NAME→INGR task but performed worse than other baselines on the NAME+INGR→INST task with $p = 0.007$. This can be attributed to its unusually long instructions as discussed earlier and shown in Table 3.

Juxtaposing these results with average lengths shown in Table 3, we observe that human evaluators prefer recipes with shorter length among the generative methods, but do not prefer the shortest recipes, i.e the real recipes, over RecipeMC for both ingredients and instructions. This confirms that humans did not heavily rely on text length. Assuming that human evaluators select outputs at random when recipes are equally good, it is surprising to see that the ingredient lists generated by RecipeMC are perceived to be more human-like than the original (human-written) recipes with $p = 0.042$ and instructions are perceived to be more human-like than original recipes with $p = 0.014$. We believe that this is because (i) RecipeMC repeats ingredients far less than the original recipes, as shown in Table 1, and (ii) it uses complete ingredient names consistently, as shown in Figure 1, and indicated by coherence values higher than ground truth shown in Tables 1 and 2.

Discussion and Future Work

MCTS and manually defined reward functions provide an effective way to control the ingredients and instruction generated by LLMs without requiring additional training. This approach offers flexibility and can enable users to generate recipes that adhere to specific constraints. It can also be valuable for interactive recipe editing. Users can present partial recipes to the system and ask the system to fill in the blanks. Personal constraints such as being sugar-free, low sodium, or vegetarian can be added to the reward function. Users can iteratively prompt the system with different combinations of ingredients and collaboratively create new recipes. We plan to explore interactive recipe generation in future work to measure novelty and creativity through user studies with amateur and professional chefs.

The ability of MCTS to look ahead during the generation of each token coupled with heuristic-based reward functions interestingly led to lower perplexity and higher similarity to ground truth recipes without directly optimizing for these properties. This finding is interesting from the perspective of text generation and suggests that using MCTS with LLMs may be applicable to a wider set of applications beyond structured text generation.

Conclusions

We presented a new sampling method, RecipeMC, that combines LLMs, MCTS, and custom reward functions to generate recipes that are often indistinguishable from human recipes for the same dish. We have shown that our method outperforms common text-generation approaches for LLMs for this task on a variety of automatic generation metrics. We conducted a *Recipe Turing Test* and found that users preferred RecipeMC ingredients about 55% of the time and RecipeMC generated instructions 62% of the time as compared to human-generated ingredients and instructions. These evaluations show that MCTS combined with manually-defined reward functions can be an effective tool for recipe generation with LLMs such as GPT-2.

Author Contributions

Author 1 was in charge of writing the manuscript, planning the study, building the system, and setting up the programs for automatic and human evaluation. Author 2 ran the experiments and compiled and analyzed the results. All three authors contributed to algorithmic design, writing the manuscript, planning the study, and conducting the human evaluation.

References

Antognini, D.; Li, S.; Ai, M.; Faltings, B.; and Mcauley, J. 2022. Assistive Recipe Editing through Critiquing. *arXiv:2205.02454*.

Bié, M.; Gilski, M.; Maciejewska, M.; Taisner, W.; Wi´sniewski, D. W.; and Ławrynowicz, A. 2020. RecipeNLG: A Cooking Recipes Dataset for Semi-Structured Text Generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, 22–28.

Chaffin, A.; Claveau, V.; and Kijak, E. 2022. PPL-MCTS: Constrained Textual Generation Through Discriminator-Guided MCTS Decoding. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2953 - 2967*, 2953–2967.

Fan, A.; Lewis, M.; and Dauphin, Y. 2018. Hierarchical Neural Story Generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, volume 1, 889–898. Association for Computational Linguistics (ACL).

Herranz, L.; Min, W.; and Jiang, S. 2018. Food recognition and recipe analysis: integrating visual content, context and external knowledge. *arXiv:1801.07239*.

Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; Choi, Y.; and Allen, P. G. 2020. The Curious Case of Neural Text DeGeneration. In *Proceedings of the International Conference on Learning Representations*.

Hsieh, L.-H.; Lee, Y.-Y.; and Lim, E.-P. 2021. ENCONTER: Entity Constrained Progressive Sequence Generation via Insertion-based Transformer. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, 3590–3599.

Lee, H.; Shu, K.; Achananuparp, P.; Prasetyo, P. K.; Liu, Y.; Lim, E. P.; and Varshney, L. R. 2020. RecipeGPT: Generative Pre-training Based Cooking Recipe Generation and Evaluation System. In *WWW-20 - Companion Proceedings of the World Wide Web Conference*, 181–184. Association for Computing Machinery.

Lin, C.-Y. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out: Association for Computational Linguistics*, 74–81.

Marin, J.; Biswas, A.; Ofli, F.; Hynes, N.; Salvador, A.; Aytar, Y.; Weber, I.; and Torralba, A. 2021. Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43(1):187–203.

Min, W.; Jiang, S.; Sang, J.; Wang, H.; Liu, X.; and Herranz, L. 2017. Being a supercook: Joint food attributes and multimodal content modeling for recipe retrieval and exploration. *IEEE Transactions on Multimedia* 19(5):1100–1113.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, 311–318.

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving Language Understanding by Generative Pre-Training. *OpenAI*.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI*.

Reusch, A.; Weber, A.; Thiele, M.; and Lehner, W. 2021. RecipeGM: A Hierarchical Recipe Generation Model. In *Proceedings of the IEEE 37th International Conference on Data Engineering Workshops*, 24–29. Institute of Electrical and Electronics Engineers Inc.

Rosin, C. D. 2011. Multi-armed Bandits with Episode Context. *Annals of Mathematics and Artificial Intelligence* 61:203–230.

Salvador, A.; Hynes, N.; Aytar, Y.; Marin, J.; Ofli, F.; Weber, I.; and Torralba, A. 2017. Learning Cross-modal Embeddings for Cooking Recipes and Food Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Shirish Keskar, N.; Mccann, B.; Varshney, L. R.; Xiong, C.; Socher, R.; and Research, S. 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *arXiv:1909.05858*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; Van Den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature* 550(7676):354–359.

Świechowski, M.; Godlewski, K.; Sawicki, B.; and Mańdziuk, J. 2022. Monte Carlo Tree Search: a review of recent modifications and applications. *Artificial Intelligence Review* 56(3):2497–2562.

Varshney, L. R.; Pinel, F.; Varshney, K. R.; Bhattacharjya, D.; Schörgendorfer, A.; and Chee, Y.-M. 2019. A Big Data Approach to Computational Creativity. *IBM Journal of Research and Development* 63(1):1–7.

Vaswani, A.; Brain, G.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. In *Proceedings of the Conference on Neural Information Processing Systems*.

Wang, H.; Lin, G.; Hoi, S. C.; and Miao, C. 2020. Structure-Aware Generation Network for Recipe Generation from Images. In *Proceedings of the European Conference on Computer Vision*, volume 12372 LNCS, 359–374. Springer Science and Business Media Deutschland GmbH.

Xu, J.; Liu, X.; Yan, J.; Cai, D.; Li, H.; and Li, J. 2022. Learning to Break the Loop: Analyzing and Mitigating Repetitions for Neural Text Generation. In *Proceedings of the Conference on Neural Information Processing Systems*.

Zhang, Y.; Wang, G.; Li, C.; Gan, Z.; Brockett, C.; and Dolan, B. 2020. POINTER: Constrained Progressive Text Generation via Insertion-based Generative Pre-training. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 8649–8670. Association for Computational Linguistics (ACL).