Generating Recipe Ingredients and Instructions with Controlled Text Generation

Stanford CS224N Custom Project

Benjamin Randoing
Department of Mechanical Engineering
Stanford University
bar39@stanford.edu

Justine Breuch
Department of Computer Science
Stanford University
jbreuch@stanford.edu

Kerrie Wu
Department of Computer Science
Stanford University
kerriewu@stanford.edu

Abstract

Recipe generation from recipe titles only is currently unsolved, as state of the art models require both recipe titles and ingredients lists for instruction generation (Lee et al., 2020). This project investigates if a number of different architectures such as Long Short-Term Memory (LSTM) encoder-decoders, LSTM decoders, or Transformer-based decoders, can produce meaningful ingredient lists when given recipe titles only. The recipe titles and generated ingredients are then passed into an existing recipe instruction generation framework to produce cooking instructions (Liu et al., 2022). Our best ingredient generation model yielded qualitatively coherent ingredients lists with BLEU score 11.2 and F1 score 8.9, however, the BLEU and ROUGE-L scores for the final recipe instructions with ingredients from our selected transformer decode were 3.4 and 22.7. The baseline plug-and-play recipe instruction generation framework, relying on RecipeGPT and ground truth recipe title and ingredients demonstrates BLEU and ROUGE-L scores of 13.73 and 39.1 respectively for instruction generation. Since BLEU and ROUGE-L performance are influenced by n-gram matching and order, further evaluation would be required with metrics such as Semantic Textual Similarity (STS) to evaluate the meaning of the produced ingredients in thee context of each recipe.

1 Key Information to include

Mentor: Siyan Li, External Collaborators: None, Sharing project: No, Sharing late days: Yes

2 Introduction

Although natural language generation models have become increasingly fluent, it is still difficult to control their output for specific qualities or content. Much work has been done on controlling output with single specific attributes, for example sentiment (Ghosh et al., 2017), but controlling multi-sentence text output with long-term content planning is still an unsolved problem. One task that captures this specific need of long-term content planning over multiple sentences is recipe generation with instructions (Marín et al., 2018). Existing recipe generation models require both a title and ingredients list to provide recipe instructions (Liu et al., 2022). However, it is often easier for a user to provide only a recipe title as input. This project explores designing an ingredients-list generating model provided a recipe title to serve as functional inputs for recipe instruction generating models.

The work of Liu et al. (2022) presents one existing model that generates recipes by creating a stage plan with DistilBERT using the combination of recipe title and ingredient inputs prior to recipe

text generation with GPT-2. The stage planner DistilBERT is a Transformer model that is based on BERT architecture and is designed to be more efficient. It achieves this by employing knowledge distillation during pre-training, which reduces the size of a BERT model by 40 %. Additionally, the model leverages the pre-training inductive biases of larger models through a triple loss that combines language modeling, distillation, and cosine-distance losses.

The approaches we tried for generating ingredient lists from recipe titles included a number of different architectures such as Long Short-Term Memory (LSTM) encoder-decoders, LSTM decoders, and Transformer-based decoders. The recipe titles and produced ingredients are then passed into an recipe instruction generation framework to produce cooking instructions (Liu et al., 2022). The recipe generation the ingredients generated in this project resulted in a decrease in the BLEU and ROUGE-L performance of the output recipes. The BLEU score decreased from 13.73 to 3.41 at best, and the ROUGE-L decreased from 39.1 to 22.7 at best.

3 Related Work

To the best of our knowledge, there is not much existing academic literature on generating recipe ingredients and instructions from recipe titles only. RecipeGPT can generate ingredients, but requires both the recipe title and recipe instructions as input (Lee et al., 2020). The majority of existing recipe generation models require both ingredient title and ingredients. For these methods, there is much work on controlling recipe instruction generation. Some approaches have included modifying the architecture of and then retraining pretrained language models, including CTRL (Keskar et al., 2019), and POINTER (Zhang et al., 2020), which are effective but require a lot of computational resources and task-specific labeled data (Liu et al., 2022). There are also finetuning methods such as ParaPattern (Bostrom et al., 2021) and prefix-tuning (Li and Liang, 2021), which require less computation and often perform adequately, but cannot enforce hard constraints on the outputs directly (Liu et al., 2022). Finally, there are post-processing methods such as PPLM (Dathathri et al., 2019), FUDGE (Yang and Klein, 2021), and neurologic decoding (Lu et al., 2020), which use a separate guiding module to control output. These methods require the least computational resources, and are flexible in design because the guidance module is separate from the pretrained language model (Liu et al., 2022). In our paper, we utilize the plug and play recipe generation method (Liu et al., 2022) because it achieves state of the art performance and is open sourced. We also focus on experimenting with different model architectures, training methods, and simple decoding methods (top-k, top-p, beam search) for ingredient generation.

4 Approach

Our very first approach began as an encoder-decoder LSTM with Luong multiplicative attention (Luong et al., 2015) trained as a machine translation task between recipe title and recipe ingredients. However, we quickly re-architected our model as the outputs were largely nonsensical after training over several epochs. We transitioned to using next-token prediction with a two-pronged approach: LSTM and transformer models.

4.1 Text Generation Task Architecture

4.1.1 LSTM

For the LSTMs, we experimented with three separate models: Two LSTM decoder models with block size 16 and 64 and one LSTM encoder-decoder model with Luong multiplicative attention as a single linear layer in order to target ingredient outputs based on the most substantive aspects of the recipe title. For all models we optimized cross-entropy loss:

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{1}$$

The decoder models comprised of an embedding layer of size 1024, 2 hidden layer LSTM with a hidden state size of 1024 and a dropout rate of 0.5. The last fully connected layer yielded a distribution over the entire vocabulary of size 29,058.

The encoder-decoder model mirrored the same embeddings, however, they were fed into a 2 hidden layer LSTM encoder with a fully connected layer. The decoder featured a block size of 64, a single attention layer with block size of 16, dropout layer of 0.5 and a final projection layer producing distributions for a sequence length of 64.

4.1.2 Transformer

The second architecture investigated was a transformer-based text generation model. For the implementation of the model, we used the GPT-like implementation provided in the assignment 5 model (Hewitt and Khurana). In this model, the input tokens are encoded using both a word-token and position embedding. Then, the embedded inputs pass through a sequence of transformer blocks. Each transformer block consists of layer normalization of the input, followed by a self-attention layer, layer normalization, and finally a multi-layer perceptron block consisting of two linear layers with a RELU activation function applied after the first linear layer. All connections are residual connections within and between the transformer blocks. The final output head is a linear projection from the embedding size to the vocabulary size, which is fed into softmax to produce a probability distribution over the possible output tokens in the entire vocabulary.

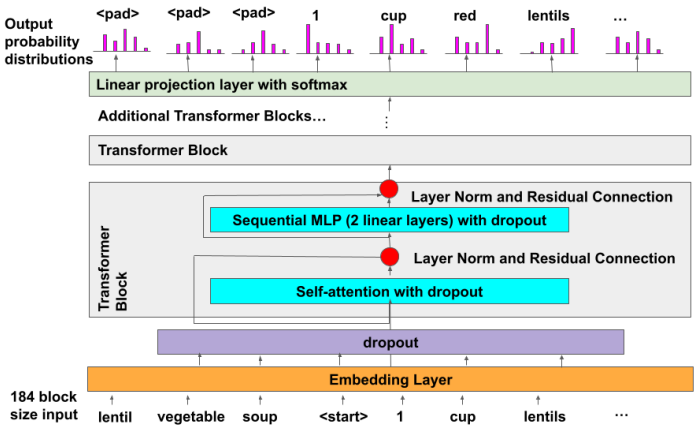

Figure 1: Diagram of the transformer architecture ((Hewitt and Khurana)) that we used, with inputs/outputs shown for our finetuning setup.

4.1.3 Training Approaches

4.2 LSTM

The LSTMs decoders were trained on next-token prediction for block sizes of 16 and 64. The titles were suffixed with a special token <title-end> and individual ingredients were split by <ingr-end>, which qualitatively improved outputs. All examples were joined together and chunked into sequences of 16 and 64.

```
Input: ["<sos>, "Teriyaki", "chicken", ..., "with", "miso"],
Target: ["bokchoy]

Input: ["<title-end>", "3", "chicken", "breasts", "<ingr-end>",
    ..., "1", "teaspoon"],
Target: ["paprika"]
```

The encoder-decoder model with attention, however, set input length equal to target length and tokens in the target were merely shifted by 1 (see transformer below). Notably, the encoder-decoder model relied on teacher-forcing to selectively provide the decoder the target token from $t-1$, rather than the predicted token, with a probability 0.5 during training.

4.3 Transformer

We trained the transformer models for next-token prediction using two different methods. In the first method (Transformer A), we constructed a custom dataset of input and target token sequences using the Recipes1M+ dataset. Each example corresponded to one recipe, and the input consisted of the tokenized recipe title, followed by the tokenized ingredients. An example is in the appendix. In the second method (Transformers B and C), we pretrained the model on a span corruption objective, similar to what was used in assignment 5 (Hewitt and Khurana). Details and an example are in the

appendix. For finetuning, the input was the tokenized sequence of the recipe title and ingredients, and the target was the same sequence shifted by one, with the title tokens replaced by pad tokens. An example below:

```
Input: [oprah ' s pomegranate martini <start> 12 cups pomegranate
    juice <ingr_end> 2 ounces absolute citrus vodka <ingr_end> ...
     <end> <pad>  ...]
Target: [<pad> <pad> <pad> <pad> <start> 12 cups pomegranate juice
     <ingr_end> 2 ounces absolute citrus vodka <ingr_end> ... <end
    > <pad> <pad> ...]
```

4.3.1 Decoding Approaches for Ingredient Prediction as Text Generation

When decoding, we prompted both the LSTM and transformer models with the recipe title only, and then constructed the generated ingredients list with top-k sampling, top-p sampling, and both combined. For the transformer, we also tried beam search from the model's output distribution. When combining top-k and top-p decoding, we select the top k results, and then sample from the results within the top-p percentile of the reduced set. To find the best-performing hyperparameters for decoding, we conducted a grid search across top-k values (1, 3, 5), top-p values (0.3, 0.9), and temperatures (0.8, 0.9, 1). For beam search, we tried three different beam sizes (2, 5, 10). We then selected the top-performing methods for further evaluation using F1 and BLEU scores, along with qualitative human review of the model outputs.

4.4 Baseline Recipe Plug and Play

Since our ultimate goal is to feed ingredient predictions into the plug-and-play framework, we ran a baseline experiment based on an established recipe generation model `https://github.com/williamLyh/RecipeWithPlans` using the existing dataset title and ingredients rather than our own (Liu et al., 2022). This uses the DistilBERT model as the planning stage classifier to label individual recipe instruction sentences as a particular stage ("Pre-processing", "Mixing", "Cooking", etc) (Sanh et al., 2019). Then it uses a BART model, a denoising auto-encoder for pretraining sequence-to-sequence models, Lewis et al. (2019) to consume a recipe title and ingredients list and produce an outline of recipe stages. Finally, a GPT-2 model, fine-tuned on Recipe1M+ dataset (Marín et al., 2018) populates the recipe with natural language cooking instructions conditioned on the stage (Liu et al., 2022).

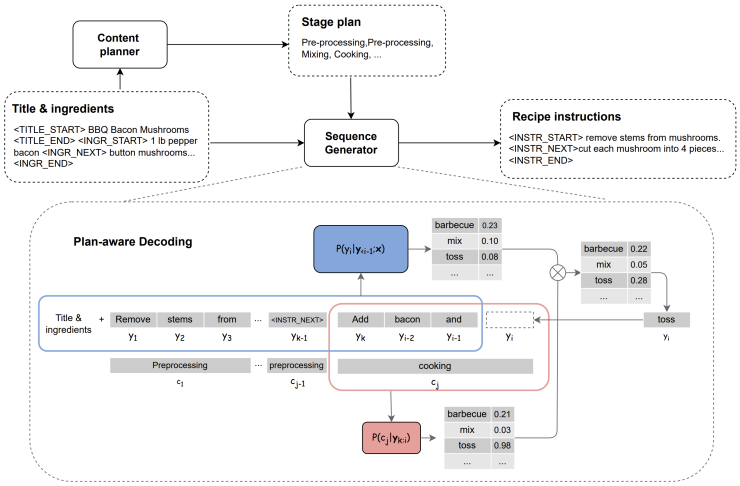

Figure 2: Baseline Architecture for Recipe Generation (Liu et al., 2022)

5 Experiments

5.1 Data

The Recipe1M+ dataset has served as a seminal source of data for projects within this domain (Marín et al., 2018). The Recipe1M+ dataset is the primary dataset for training the ingredient and recipe generation tasks in this project. The dataset includes over 1 million recipes, constituted of lists of ingredients and sequential cooking instructions. Each data entry, as depicted in Appendix A, provides ingredients with quantities, a source url, recipe title, and instructions. The dataset is split into three partitions which we used accordingly: train (720,639 examples), validation (155,036 examples), test (154,045 examples).

5.1.1 Data Preprocessing

We used only the recipe title and ingredient fields of the Recipes1M dataset. We cleaned the dataset by splitting quantities from the measurements and ingredients and removing extraneous descriptive text between parentheses (for example, "15g chicken (see note below)" becomes "15 g chicken" after cleaning). We then tokenized the dataset on a word level and then constructed a vocabulary using all of the tokenized recipe titles and ingredients with a minimum frequency of 5. Special characters that we included are described in the Training approaches section and shown within the input/target pair examples. The total vocabulary size was 29058.

5.2 Evaluation method

5.2.1 Ingredient Generation Evaluation

For ingredient evaluation, we used F1 scores and BLEU scores with a maximum n-gram length of 3 and equal weighting across all ngram lengths. When calculating scores, we removed special tokens (such as padding, start, end, and ingredient delimiting tokens). We choose to measure BLEU scores with a max n-gram length of 3 because individual ingredient entries are commonly 3 tokens long (eg "3 cups flour"), and as a way to measure more complex lexical similarity than token-level F1 scores. F1 scores are useful as a completely order-agnostic measurement.

5.2.2 Recipe Generation Evaluation

Quantitative evaluation of the plug-and-play recipe generation includes the BLEU and ROUGE-L scores. Qualitatively, the recipes are read to determine their fluency, quality, completeness, and coverage of input ingredients.

5.3 Experimental details

5.4 LSTM

To evaluate the ingredient generation for the LSTM, we used the model fine-tuned on next-token prediction to generate ingredients based on title input. Two models were trained on batch sizes of 128, block sizes of 16 and 64, respectively, so for a given title, the model sampled until a max length of size 184 to capture both title and ingredients. All model variants had learning rate $(1e-4)$, dropout rate (0.5), gradient norm clipping (max norm 1), and the Adam optimizer. Both models were trained on 100,000 examples using the cross-entropy loss between predicted ingredients and ground-truth items for 22 and 17 epochs, respectively, at a learning rate of 0.001. The attention model's loss, however, converged by the third epoch using a repetition penalty γ of 0.1. Experiments were run for both top-k (1, 3, 5) and top-p (0.3, 0.9) decoding with temperature sampling (0.8, 0.9, 1).

5.5 Transformer

We used a block size of 184 for the transformer, truncating the recipe title to 30 tokens and target ingredients to 150 tokens in each example, inclusive of special delimiting tokens; these values were selected to fit most of the examples in the dataset without increasing the computational complexity of the transformer excessively. The remainder of the block accomodates the additional special tokens that we used to indicate the start of the ingredients list, the end of the ingredients list, and mask tokens for the pretraining objective. Examples were padded to the block length, if shorter than the

block length. We used AdamW as the optimizer and a cross-entropy loss. In general, we stopped training once the training loss had converged.

We trained three different Transformer models: Transformer A, Transformer B, and Transformer C. A table of their hyperparameters is below.

Model	layers	attn heads	embed size	learning rate	pretrain epochs	finetune epochs
A	4	8	256	$3e-4$	None	1
B	4	8	256	$3e-4$	1	$\frac{1}{10}$
C	8	8	1024	$4.5e-4$	1	1

Table 1: Transformer model configurations and training hyperparameters. For transformers A and B, we used default hyperparameters from assignment 5 (Hewitt and Khurana). For Transformer C, we conducted a hyperparameter search over the learning rate and dropout rates applied at the embedding layer, attention outputs, and within transformer layers, but found the original dropout rates (0.1) to be best.

5.6 Results

5.6.1 LSTM Ingredient Generation

Model config and training	# Decoding parameters	F1	BLEU
LSTM decoder block size 16	$k = 1$	10.93	5.34
LSTM decoder block size 64	$k = 1$	11.16	5.96
LSTM decoder block size 64	$k = 1$ and $p = 0.3$	11.31	6.17
LSTM decoder block size 64	$k = 1$ and $p = 0.3$ and $t = 0.9$	11.67	6.40
LSTM with attention and title	$k = 5$ and $p = 0.3$ and $t = 0.9$	12.4	7.2

Table 2: Results of training for the best decoding hyperparameters, evaluated on the test set.

The LSTM decoders performed worst of all approaches. The block size of 16 performed worse than that of block size 64. The task itself requires sequences that are often longer than 64, therefore we expected that a smaller sequence size would lead the model to stray further from accurate ingredient predictions. Results may improve with even larger sequence lengths, however, given the limitations of LSTMs and long sequences, we saved additional experimentation for the transformer, which limited our ability to test this hypothesis. Decoding with top p at $p = 0.3$ and scaling temperature to 0.9 did improve BLEU scores ≈ 1, however, led to diminishing returns by $p = 0.9$. While we ran top-k decoding for both LSTM decoders, it only yielded largely incoherent recipes at a BLEU score of 0. Perhaps with increased number of examples this may have improved since we limited our training to 100,000 examples.

5.6.2 Transformer Ingredient Generation

Results for transformer ingredient generation show that increasing model size and using masked pretraining, together, increase model performance. We did not find that our pretraining objective, on its own, increased performance, and perhaps actually made it decrease. This might be because our pretraining objective was very similar to the finetuning objective and used the same data; it did not introduce new data sources, while optimizing the model for a different task than the final objective. Future work may include a pretraining objective that includes recipe instruction text to allow for learning richer token embeddings for ingredients. Increasing model size may have helped the model learn more nuanced ingredient embedding representations, and also build more complex relationships between different ingredients and recipe titles. The consistent performance across train, validation, and test sets indicate that the model is not overfit, and further increasing model size/complexity may bring greater gains in performance. During hyperparameter tuning for Transformer C we also found that the lowest amounts of regularization we tested brought the best training results. The BLEU scores are low for all models, which is partially expected because ingredients lists should be evaluated as an order-agnostic list of ingredients, and the F1 scores are low as well, likely because the tokens used may be synonymous but not identical (eg: "teaspoons" vs "tsp.").

Transformer	Decoding params			Test		Val		Train	
	k	p	t	F1	BLEU	F1	BLEU	F1	BLEU
A	10	0.3	0.8	30.6	9.3	30.3	9.0	30.1	9.1
B	3	0.3	0.9	29.9	8.9	29.2	8.9	29.9	9.0
C	8	0.9	0.8	34.7	11.2	34.7	11.1	34.2	10.8

Table 3: Results of training for the best decoding hyperparameters with the transformer models, evaluated on 1000 examples from each of the training, validation, and tests sets. Beam search did not perform as well as top-k and top-p decoding methods, so we did not evaluate beam search as a decoding method further but some sample results are included in the appendix.

5.6.3 Quantitative Results for Recipe Generation

The findings indicate that the performance of recipe generation models, as measured by both BLEU and ROUGE-L scores, decrease om 13.73 to 3.41 and 39.1 to 22.7 at best, respectively, when generated ingredients are employed. This outcome was anticipated due to the fundamental differences in the order and phrasing of the ingredients. The decrease in performance may be attributable to the inaccuracies associated with the ingredient generation process, such as missing ingredients or varied quantities. Additionally, the performance decline may also be attributed to the limitations of the evaluation metrics employed. Specifically, the reliance of BLEU score on n-gram overlap and ROUGE-L score on word order may adversely impact the performance of recipe generation models when generated ingredients are used.

The expected recipe outputs adhere to the order of Recipe1M+ ingredients, which may not be preserved in the ingredient generation pipelines. While the semantic meaning of the ingredients may remain intact, the exact format and order may vary, leading to a degradation in quantitative recipe generation performance. These findings underscore the importance of carefully considering the evaluation metrics employed and the specific characteristics of the generated ingredients when assessing the performance of recipe generation models.

Ingredient Generating Model	Decoding Parameters	BLEU	ROUGE-L
Baseline Recipe1M+ Ingredients	N/A	13.73	39.1
Transformer B Ingredient Generation	$k = 3\ p = 0.3\ t = 0.9$	3.41	22.7
Transformer B Ingredient Generation	$k = 5\ p = 0.9\ t = 0.9$	3.20	22.6

Table 4: BLEU and ROUGE-L for Recipe Generation

6 Analysis

6.1 LSTM Ingredient Generation

Below is an example of ingredients generated by the LSTM decoder, block size 64 with $p = 0.3$ and $t = 0.9$.

```
Output: noodle if desired | 2 cans beans, drained well | 1
    tablespoon tomato paste | 2 cups low sodium chicken broth | 5
    ounce large spanish onions | 3 tablespoons chopped jalepeno
    peppper | 1 cup arugula, roughly chopped | 1 cup long grain
    milk, ...
Target: 1 cup lentils | 12 onion | 1 tomatoes | 2 medium carrots |
     2 stalks celery | 3 tablespoons extra virgin olive oil, or 3
    tablespoons vegetable oil, 1 teaspoon garlic powder, 1
    teaspoon fresh ginger root juice | 4 cups vegetable broth | 3
    cups water, ...
```

With LSTM decoders of size 16 and 64 (even when decoding with top $p = 0.3$), the ingredients grow less coherent as predictions move further away from the title. This accords with the architecture of the model: though LSTMs maintain longer memory than RNNs they still underperform with longer sequences. The title becomes less relevant as the list of ingredients grows as it's moving further away from the beginning of the input. When decoding with top-k, the results were still coherent food items but didn't accord with each other or the title. As the example in the appendix shows, straying from the most-likely token ($k = 3$) leads to more variable ingredients that also stray too far from the title to produce meaningful results and lead to more repetition, which is reflected by a BLEU score of 0. The encoder-decoder did not fare much better, achieving just a ≈ 1 point increase in BLEU score. This is likely due to the block size limitation that the model did not have access to the title after token 64. When decoding by k greater than 1, the outputs grew far worse and both models lost the <ingr-end> tokenization to separate out ingredients, given the lack of thematic conciseness in other experimentations, it made sense that adding variation worsened outcomes. Ultimately, we don't believe LSTMs are the best architecture for this task given their sequence length limitations.

6.2 Transformer Ingredient generation

Subjectively, we find that our transformer models are capable of producing ingredients that are valid food items and approximately related to the recipe title most of the time and outperform LSTMs. The best model (Transformer C) is often able to cover ingredients mentioned in the title and presents common ingredient pairings together. For example, below is one sample recipe title, and predicted/target pairs in which the model outputted realistic results (formatted for readability).

```
Input: butternut squash soup or bisque (roasting method)
Output: 1 whole butternut squash, peeled, seeded, and diced | 1 whole
    onion, peeled and diced | 1/2 teaspoons salt | 1/2 teaspoons black
     pepper | 1/2 teaspoons cinnamon | 1 cup water | 1 cup chicken
    broth or bouillon | 1/4 cups heavy cream or half-and-half | 1/4
    cups butter
Target: 1 small butternut squash, peeled, diced | 1 medium onion,
    large dice | 1 tablespoon olive oil | 1 apple, skinned, large dice
     | 48 ounces chicken broth | 1 cup half-and-half | 2 tablespoons
    parsley, fresh, chopped | 1 tablespoon thyme, fresh, chopped |
    crouton | sour cream
```

We found that our transformer models all made similar errors, which were common across all three different model configurations and hyperparameters for decoding methods. These are detailed in table 5. In summary, some of the errors are due to dataset errors, which could be resolved by cleaning the data to remove web scraping errors such as advertisement text and extraneous numerical quantities. Ingredient relevance and coverage based on the title could be improved by decreasing k, p, t and increasing model size; however there was a tradeoff where decreasing k, p, and t seemed to increase frequency of ingredient repetition. Increasing beam size increased F1 and BLEU scores (see Appendix), and decreased ingredient repetition, suggesting that with a large-enough beam size, results could further be improved. However, because beam search is slow and computationally expensive with large beam size compared to sampling, we did not experiment with larger beam sizes than 10. There was no combination of hyperparameters we explored that completely eliminated the repetition problem, so we think that this is something inherent to the model architecture, training method, and decoding strategies that we used.

Error	Example	Explanation/Analysis
Repeated Ingredients	Output: [**1 tsp . dried thyme | 1 tsp . dried thyme | 1 tsp . dried thyme** ...]	Increasing k, p, and t values and model size reduced repetition.
Two sequential quantities	Output: [**1 12** cups all-purpose flour ..]	Web scraping errors (present in Recipes1M+ dataset).
Inappropriate quantities	Output: [**14 cup** olive oil | 2 lbs skinless chicken pieces]	Some of this is inherent to the problem setup because recipe titles do not include a quantifier for the number of portions. However the ingredient ratios are still incorrect; this is a limitation of the approach we used.
Predicting advertisement text	Output: [1 medium onion , chopped **king sooper ' s 1 lb for $0 . 99 thru 02/09** ...]	Web scraping errors (present in Recipes1M+ dataset).
Missing ingredients mentioned in title	Input: [leek , potato , and bacon casserole] **Output does not include leek or bacon**.	Reducing k, p, and t (temperature) and increasing model size reduced these errors.
Inappropriate ingredients	Input: [spanish mussels vinaigrette] Output: [1/2 cup **lager-style pickles**...]	Reducing k, p, and t (temperature) and increasing model size greatly reduced these errors.

Table 5: Analysis of some errors common to transformer models.

6.3 Analysis for Plug and Play Recipe Generation

The recipe generation was compared qualitatively for the "Steak Asparagus Wraps" recipe created using the Recipe1M+ baseline ingredients and transformer generated ingredients. Table 4 depicts the expected recipe from the Recipe1M+ dataset. The transformer model produced the ingredient list below provided the recipe title. The ingredients from the transformer are missing the steak and marinade, and added an onion. The corresponding recipe generated below provides reasonable steps to prepare the ingredients provided, suggesting the generated ingredients provide a reasonable format for the recipe generation to provide interpretable recipes. However, the absence of key ingredient presents a weakness in the current transformer ingredient outputs.

Recipe Title	Ingredients (Transformer)	Ingredient Source	Generated Results
Steak and Asparagus Wraps	2 cups sliced fresh asparagus , trimmed and cut into 1-inch pieces 1 lb . asparagus , ends trimmed , cut into 1-inch pieces 1 cup sliced fresh asparagus , trimmed and cut into 1/2-inch pieces 1 cup chopped onion 1/2 cup kraft classic ranch dressing , divided large flour tortillas , cut into 1/2-inch cubes	Recipe1M+	Pour marinade over steak in resealable plastic bag. Seal bag; turn to evenly coat steak with dressing. Refrigerate at least 30 min. to marinate. Meanwhile, preheat grill to medium heat. Remove steak from marinade; discard bag and marinade. Grill steak 13 to 16 min. or until medium doneness (160F), turning occasionally. Meanwhile, steam asparagus 8 to 10 min. or until crisp-tender. Cut steak into thin strips. Place 1 to 2 steak strips and about 2 asparagus spears on each tortilla strip roll up. Secure with frilled toothpicks or red bell pepper ties, if desired. Serve warm.
		Transformer Generator	Cook and stir asparagus and asparagus in 1/4 cup dressing in large skillet on medium heat 5 to 7 min. or until crisp-tender. Add onions; cook and stir 2 min. Stir in 1/4 cup dressing; cook and stir 1 min. Add asparagus mixture; mix lightly. Spoon onto tortillas; roll up.

Table 6: Analysis of Recipe Generation Example

7 Conclusion

We find that generating recipe ingredients and recipe instructions given a title only is feasible using LSTM and Transformer-based text generation models, however more work is needed to achieve ingredient generation from recipe titles that are **consistently** realistic, cover the ingredients relevant to the title, are not repetitive, and have correct quantities.

Although we did some regex cleaning of the dataset, we found that some web scraping errors were prevalent enough to make it to the final model. These included advertisement text that would show up in predictions, and extraneous numerical quantities for the ingredients. In the future, it would be good to clean the data to remove these errors. We would also want to strip out titles that don't provide meaningful direction (i.e. "Chocolate bonanza"). These models would also benefit from pre-trained recipe word embeddings that have a rich representation of which foods relate well together.

The LSTM would benefit from a more complex attention scheme, like forcing increased attention on the title or removing ingredient quantities in order to make examples more concise given the constraint of a small sequence size.

Additionally, the transformer training pre-training and fine-tuning methods only exposed the model to the database examples, while at test time, the model is making predictions based on model-generated examples. This is a significant distribution shift especially when p, k, t are large. It could be beneficial to try training the model on model-generated samples occasionally during training to address this. A more structured complex decoding method may be worth exploring in the future as well. We could train a model to predict a high-level content plan of ingredients based on the recipe title (including number of ingredient and type of ingredients), and then use another, separate ingredient classification model to re-rank the transformer generation model's outputs conditioned on the ingredient content plan. It would be similar to the modular controlled generation method used in Plug and Play article (Liu et al., 2022), which the authors noted as reducing the frequency of repeated instructions compared to other decoding approaches. To address the inappropriate quantity errors, it would be interesting to see if we can train on data labeled with desired portions, forgo predicting quantities altogether, or assess if the predicted ingredient quantity ratios are correct. Finally, a better evaluation metric for recipe ingredient generation would be useful, as neither BLEU score nor F1 score fully capture the semi-order-agnostic nature of ingredient lists.

The quantitative evaluation of recipe generation using BLEU score relies heavily on n-gram matching, which may not always reflect the quality of the generated text. It also doesn't capture more complex aspects of language such as coherence, overall meaning, and fluency. ROUGE-L is designed to be more robust to differences in word order and vocabulary; however, it is still limited by its reliance on lexical overlap. Investigating a Semantic Textual Similarity (STS) evaluation metric may be more indicative of true recipe performance regardless of differences in word choice, syntax, and other linguistic features from the Recipe1M+ dataset. It measures the degree of similarity between two tests based on semantic meaning. This would also help remedy the dependence on ingredient order, since erring from the ground-truth order does not in fact indicate a degredation in quality, however lowers n-gram accuracy, which is reflected by our model's lower BLEU scores.

8 Contributions

Justine worked on the data parsing and pipelines for all models and implemented all LSTM variations and worked on decoding methods. Ben worked on implementing the recipe generation pipeline and conducting recipe generation evaluation experiments with the ingredient generation models established by Kerrie and Justine. The experiments involved generating stage plans for the ingredient sets, processing ingredient inputs, and evaluating recipe outputs. Kerrie worked on implementing, evaluating, and hyperparameter searching for the initial LSTM with attention NMT approach and the transformer-based ingredient generation approaches. All team members contributed to researching + discussing approaches and writing up the final report and poster.

References

Kaj Bostrom, Xinyu Zhao, Swarat Chaudhuri, and Greg Durrett. 2021. Flexible generation of natural language deductions. In *Conference on Empirical Methods in Natural Language Processing*.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2019. Plug and play language models: A simple approach to controlled text generation. *ArXiv*, abs/1912.02164.

Sayan Ghosh, Mathieu Chollet, Eugene Laksana, Louis-Philippe Morency, and Stefan Scherer. 2017. Affect-lm: A neural language model for customizable affective text generation. In *Annual Meeting of the Association for Computational Linguistics*.

John Hewitt and Ansh Khurana. Cs224n 2022-23 winter assignment 5.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *ArXiv*, abs/1909.05858.

Helena H. Lee, Ke Shu, Palakorn Achananuparp, Philips Kokoh Prasetyo, Yue Liu, Ee-Peng Lim, and Lav R. Varshney. 2020. RecipeGPT: Generative pre-training based cooking recipe generation and evaluation system. In *Companion Proceedings of the Web Conference 2020*. ACM.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190.

Yinhong Liu, Yixuan Su, Ehsan Shareghi, and Nigel Collier. 2022. Plug-and-play recipe generation with content planning. *ArXiv*, abs/2212.05093.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Neurologic decoding: (un)supervised neural text generation with predicate logic constraints. In *North American Chapter of the Association for Computational Linguistics*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Javier Marín, Aritro Biswas, Ferda Ofli, Nick Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. 2018. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:187–203.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Kevin Yang and Dan Klein. 2021. Fudge: Controlled text generation with future discriminators. *ArXiv*, abs/2104.05218.

Pencheng Yin, Sahil Chopra, Vera Lin, and Siyan Li. Cs224n 2022-23 winter assignment 4.

Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and Bill Dolan. 2020. Pointer: Constrained progressive text generation via insertion-based generative pre-training. In *Conference on Empirical Methods in Natural Language Processing*.

A Appendix: A Single Recipe1M+ Dataset Entry


```
{"ingredients": [{"text": "6 ounces penne"}, {"text": "2 cups Beechers
     Flagship Cheese Sauce (recipe follows)"}, {"text": "1 ounce
    Cheddar, grated (1/4 cup)"}, {"text": "1 ounce Gruyere cheese,
    grated (1/4 cup)"}, {"text": "1/4 to 1/2 teaspoon chipotle chili
    powder (see Note)"}, {"text": "1/4 cup (1/2 stick) unsalted butter
    "}, {"text": "1/3 cup all-purpose flour"}, {"text": "3 cups milk
    "}, {"text": "14 ounces semihard cheese (page 23), grated (about 3
     1/2 cups)"}, {"text": "2 ounces semisoft cheese (page 23), grated
     (1/2 cup)"}, {"text": "1/2 teaspoon kosher salt"}, {"text": "1/4
    to 1/2 teaspoon chipotle chili powder"}, {"text": "1/8 teaspoon
    garlic powder"}, {"text": "(makes about 4 cups)"}], "url": "http
    ://www.epicurious.com/recipes/food/views/-world-s-best-mac-and-
    cheese-387747", "partition": "train", "title": "Worlds Best Mac
    and Cheese", "id": "000018c8a5", "instructions": [{"text": "
    Preheat the oven to 350 F. Butter or oil an 8-inch baking dish."},
     {"text": "Cook the penne 2 minutes less than package directions
    ."}, {"text": "(It will finish cooking in the oven.)"}, {"text": "
    Rinse the pasta in cold water and set aside."}, {"text": "Combine
    the cooked pasta and the sauce in a medium bowl and mix carefully
    but thoroughly."}, {"text": "Scrape the pasta into the prepared
    baking dish."}, {"text": "Sprinkle the top with the cheeses and
    then the chili powder."}, {"text": "Bake, uncovered, for 20
    minutes."}, {"text": "Let the mac and cheese sit for 5 minutes
    before serving."}, {"text": "Melt the butter in a heavy-bottomed
    saucepan over medium heat and whisk in the flour."}, {"text": "
    Continue whisking and cooking for 2 minutes."}, {"text": "Slowly
    add the milk, whisking constantly."}, {"text": "Cook until the
    sauce thickens, about 10 minutes, stirring frequently."}, {"text":
     "Remove from the heat."}, {"text": "Add the cheeses, salt, chili
    powder, and garlic powder."}, {"text": "Stir until the cheese is
    melted and all ingredients are incorporated, about 3 minutes."},
    {"text": "Use immediately, or refrigerate for up to 3 days."}, {"
    text": "This sauce reheats nicely on the stove in a saucepan over
    low heat."}, {"text": "Stir frequently so the sauce doesnt scorch
    ."}, {"text": "This recipe can be assembled before baking and
    frozen for up to 3 monthsjust be sure to use a freezer-to-oven pan
     and increase the baking time to 50 minutes."}, {"text": "One-half
     teaspoon of chipotle chili powder makes a spicy mac, so make sure
     your family and friends can handle it!"}, {"text": "The
    proportion of pasta to cheese sauce is crucial to the success of
    the dish."}, {"text": "It will look like a lot of sauce for the
    pasta, but some of the liquid will be absorbed."}
```


B Appendix: B Transformer training input and output pairs

Here is one sample input-target pair use for training Transfomer A, untokenized for readability. We used special tokens "→" and "←" for delimiting the start and end of ingredients, a "<recipe_start>" token, and "<start>" and "<end>" tokens to indicate the start and end of the ingredients list.

```
Input: [<recipe_start> saskatoon berry jam <start> → 4 cups saskatoon berries ,
    crushed ← → 4 tablespoons lemon juice ← ... <end> <pad> <pad>...]
Target: [saskatoon berry jam <start> → 4 cups saskatoon berries , crushed ← → 4
     tablespoons lemon juice ← ... <end> <pad> <pad> <pad> ...]
```

For pretraining Transformers B and C, we used a span corruption objective with masking. Similar to assignment 5 (Hewitt and Khurana), We selected the masked portion with uniformly randomly selected start locations and lengths between $\frac{1}{4}$ and $\frac{3}{4}$ of the total tokenized title and ingredient length. The masked portion is replaced with the "<mask>" token, and after the end of the original input sequence, we append the <mask> token again, followed by the masked portion of the sequence. The target is the input shifted by one token to the left. An example is below.

```
Input: [spiced rice <start> 1 tbsp vegetable oil <ingr_end> 1 <mask> cloves <
    ingr_end> ... <end> <mask> clove garlic , smashed <ingr_end> 2 <pad> ...]
```

```
Target: [rice <start> 1 tbsp vegetable oil <ingr_end> 1 <mask> cloves <ingr_end>
     ... <end> <mask> clove garlic , smashed <ingr_end> 2 <pad> <pad> ...]
```

C Appendix: C Beam Search Hyperparameter tuning results

For Beam Search we used an implementation adapted from that used in assignment 4 (Yin et al.)

beam_size	f1	bleu
2	23.133989	6.401306
5	26.261698	7.979147
10	25.562274	7.480659

Figure 3: Transformer B beam search hyperparameter tuning results

beam_size	f1	bleu
2	30.256144	7.729223
5	32.774792	9.660656
10	33.193706	10.132679

Figure 4: Transformer C beam search hyperparameter tuning results

D Appendix: D Additional Recipe Generation Comparisons

Recipe Title	Ingredients	Ingredient Source	Generate Results
Lentils Vegetable Soup	1 cup lentils 1 onion , chopped 1 cup chopped onion 12 cup chopped carrot 1 tablespoon olive oil 1 tablespoon minced garlic 2 cups vegetable stock 1 teaspoon dried oregano 1 teaspoon dried thyme 1 teaspoon dried thyme 12 cup chopped celery 12 cup chopped carrot 1 cup chopped zucchini 12 teaspoon salt 14 teaspoon pepper	Recipe1M+	Saute the onions, carrots, celery with olive oil by gradually adding half cup of water until soft. Add to the saute onions, diced tomatoes, hot red pepper powder and stir for five minutes; add 3 cups of water, leave it to boil. Rinse the lentils with cold water. To the boiling sauce, add lentils, ginger juice, white pepper and two cups of vegetable broth; reduce the heat to medium; cook for 30 minutes or until the lentils are very soft. Using a cooking stick, puree the soup in its pot or transfer to a food processor, blend the lentils until smooth; and bring it back to its saucepan. Add the rest of the vegetable broth or water as need; then in a lower heat stir the soup for 10 minutes. Add salt and black pepper. Remove from heat. Garnish it with fresh parsley, basil or cheese of your choice. Serve it hot or cold. Keep it in the fridge.
		Transformer	in a large pot, bring lentils, lentils, onion, salt, and carrot to a boil, stirring occasionally. reduce heat to low and simmer for 20 minutes. in a large saucepan, saute garlic in olive oil until tender. add vegetable stock, oregano, thyme, celery, carrots, zucchini, salt, and pepper. bring to boil, reduce heat to low, and simmer for 20 minutes. add lentils and simmer for 10 minutes.

E Appendix: E LSTM analysis

Full example of Vegetable Lentil Soup:

Top-k decoding k = 3:

```
['dry', 'white', 'wine', '<ingr_end>', '1', 'tablespoon', 'chopped', 'fresh', 'flat-
    leaf', 'parsley', '<ingr_end>', '1', 'tablespoon', 'thinly-sliced', 'fresh', '
    basil', 'leaf', '<ingr_end>', 'salt', ',', 'to', 'taste', '<ingr_end>', '3', '
    teaspoons', 'freshly', 'squeezed', 'lemon', 'juice', '<ingr_end>', '12', 'cup',
```

'heavy', 'cream', '<ingr_end>', '1', 'tablespoon', 'chopped', 'fresh', '
tarragon', '<ingr_end>', '2', 'tablespoons', 'butter', '<ingr_end>', 'available
', 'chives', ',', 'for', 'topping', '<ingr_end>', 'mini', 'marshmallows', ',', '
for', 'topping', '<ingr_end>', '4', 'manicotti', ',', 'cooked', 'to', 'only', '
firm-tender', 'with', 'kitchen', 'string', '<ingr_end>', '12', 'slices', 'of', '
homemade-type', 'white', 'bread', ',', 'each', 'slice', 'toasted', 'and', 'cut',
'into', 'a', 'decorative', 'shape', '<ingr_end>', '4', 'slices', 'of', '
italian', 'sausage', ',', 'prosciutto', 'removed', 'and', 'cut', 'into', 'a', '
decorative', 'shape', '<ingr_end>', '4', 'slices', 'aged', 'sharp', 'cheddar', '
cheese', ',', 'grated', '<ingr_end>', '2', 'tablespoons', 'butter', ',', '
melted', '<ingr_end>', '1', 'teaspoon', 'salt', '<ingr_end>', '1', 'small', '
egg', ',', 'beaten', '<ingr_end>', '10-', '1/2', 'ounces', ',', 'weight', '
ready-made', 'rolled', 'puff', 'pastry', '<ingr_end>', '1', 'pound', 'asparagus
', ',', 'ends', 'trimmed', '<ingr_end>', '1', 'cup', 'melted', 'butter', ',', '
as', 'an', 'accompaniment'].

Top-p decoding p = 3:

['6', 'large', 'green', 'onions', '<ingr_end>', '10', 'garlic', 'cloves', ',', '
minced', '<ingr_end>', '4', 'ounces', ',', 'weight', 'feta', ',', 'cubed', '<
ingr_end>', '1', 'pound', 'shelled', 'and', 'deveined', 'shrimp', ',', 'halved',
'lengthwise', '<ingr_end>', '14', 'cup', 'chopped', 'fresh', 'parsley', '<
ingr_end>', '14', 'cup', 'chopped', 'parsley', '<ingr_end>', 'salt', '&', '
freshly', 'ground', 'black', 'pepper', '<ingr_end>', '14', 'cup', 'chopped', '
parsley', '<ingr_end>', '4', 'tablespoons', 'sour', 'cream', '<ingr_end>', '1',
'tablespoon', 'chopped', 'fresh', 'tarragon', '<ingr_end>', '2', 'tablespoons',
'chopped', 'fresh', 'basil', '<ingr_end>', '12', 'teaspoon', 'salt', '<
ingr_end>', '14', 'teaspoon', 'ground', 'black', 'pepper', '<ingr_end>', '2', '
cups', 'grated', 'monterey', 'jack', 'cheese', '<ingr_end>', 'avocado', ',', '
cut', 'in', 'wedges', '<ingr_end>', 'additional', 'salsa', 'verde', 'cheese',
',', 'grated', '<ingr_end>', '12', 'cup', 'cheddar', 'cheese', ',', 'grated', '<
ingr_end>', '12', 'cup', 'colby', 'cheese', ',', 'grated', '<ingr_end>', '12', '
cup', 'colby-monterey', 'jack', 'cheese', ',', 'grated', '<ingr_end>', '14', '
cup', 'parmesan', 'cheese', '<ingr_end>', '1', 'bunch', 'green', 'onion', ',', '
chopped', '<ingr_end>', '1', 'tablespoon', 'ground', 'black', 'pepper', '<
ingr_end>', '1', 'teaspoon', 'smoked', 'paprika', '<ingr_end>', '1', 'teaspoon',
'ground', 'thyme', '<ingr_end>', '1']