Plug-and-Play Recipe Generation with Content Planning

Anonymous EMNLP submission

Abstract

Recent pre-trained language models have shown promising capability to generate fluent and realistic natural text. However, generating multi-sentence text with global content planning has been a long-existing research question. The current controlled text generation models cannot directly address this issue, as they usually condition on single known control attribute. We propose a low-cost yet effective framework that explicitly models content plans and optimizes the joint distribution of the natural sequence and the content plans in a plug-and-play post-processing manner. We evaluate our model with extensive automatic metrics and human evaluations and show that it achieves the state-of-the-art performance on the recipe generation task on Recipe1M+ dataset.

1 Introduction

Recent progress in large-scale language model pre-training has facilitated significant improvement in generating increasingly realistic text. Although this has been achieved on the surface-level fluency, it has been pointed that generating multi-sentence text with global constrains, or long-term planning is still far from being solved. Examples of such tasks are story continuation with logical coherency (Nye et al., 2021; Sinha et al., 2019), and recipe generation with step-by-step planning (Marin et al., 2019).

These issues cannot be ameliorated by simply increasing the size of model parameters or the scale of pre-training data, as suggested by LeCun (2022). Adding to this, the current controlled text generation methods cannot directly tackle those issues neither, for example, CTRL (Keskar et al., 2019), which trains a class-conditional language model, and PPLM (Dathathri et al., 2019), which re-ranks the language model predictions by an attribute model. They usually share a common setup of optimizing conditional distributions $P(\boldsymbol{y}|a)$, where $\boldsymbol{y}$ is the text sequence and a is the desired control attribute. Typical examples of control attribute include sentiment (Ghosh et al., 2017), topic (Tang et al., 2019) and formality (Wang et al., 2019). Therefore, we identify the research gap for the current controlled text generation models to generate multi-sentence text with long-term content planning.

Motivated by previous research in cognitive science (Evans, 2003), Nye et al. (2021) pointed out that the reasoning of a neural-based model should consist of two systems, i.e., the system 1 makes intuitive and associative responses, and the system 2 makes deliberative and logical decisions. With greatly increased capabilities, large language models have become sufficiently competent to act as the system 1. However, we argue that, to address the aforementioned research gap, it is vital to empower the language models with the ability to make logical decisions, i.e., predict content plans.

In contrast with the existing methods that optimize the conditional distributions, we propose a novel framework which explicitly models the content plan $\boldsymbol{c}$ and optimizes the joint distribution $P(\boldsymbol{y}, \boldsymbol{c})$ in a *plug-and-play* manner. Figure 1 depicts an overview of our approach. Specifically, our proposed framework consists of (i) a content planner predicting the content plan; and (ii) a sequence generator, based on pre-trained language models, that generates the output following the content plan. The content plan steers the generation process through a lightweight and plug-and-play style plan classifier. The sequence generator does not need to be trained with plan-specific data, which means adapting our framework to other Natural Language Generation (NLG) tasks is cheap and efficient.

In this study, we comprehensively evaluate our proposed approach on the recipe generation task with the widely-used Recipe1M+ benchmark (Marin et al., 2019). The experimental re-

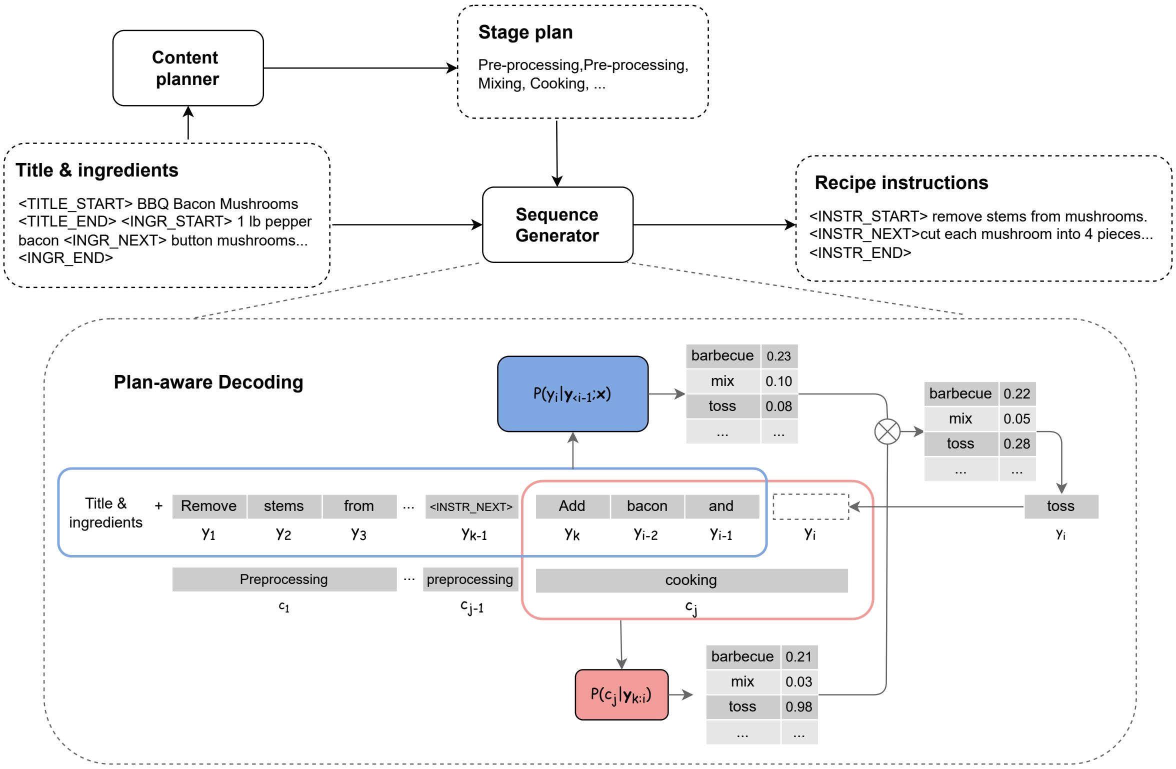

Figure 1: Model overview. The upper half demonstrates our framework. Firstly, the title and ingredients are used to predict the stage plan by the content planner module. Then, the sequence generator module, guided by the stage plan, generates the recipe instructions. The bottom half illustrates one step of the plan-aware decoding. In the given example, the current stage is 'cooking'. The language model (blue) outputs unconditional probabilities based on all previous context and inputs. The stage classifier (red) computes the probabilities of the current partial sentences belong to the current stage 'cooking'.

sults demonstrate that our approach significantly outperforms previous state-of-the-art (SOTA) as judged by both automatic and human evaluations. In particular, the results show that the recipes generated by our model are more accurate and highly controllable.

In summary, we conclude our contributions as two-fold: Firstly, we identify the current research gap and propose a novel framework that generates text with content plans in a plug-and-play manner. Secondly, we conduct extensive experiments to show that our framework achieves SOTA performance on the widely-used benchmark Recipe1M+.

2 Background and Related Works

2.1 Controlled Text Generation

Controlled Text Generation (CTG) refers to tasks of generating natural text conditioned on given controlled attributes. CTG approaches that leverage transformer-based Pre-trained Language Model (PLM) could be classified into three categories according to their required computation resources (Zhang et al., 2022). We provide a brief overview of these three categories.

Retraining. These methods usually modify the original architecture of PLMs and retrain them for a specific downstream task. For example, CTRL, proposed by Keskar et al. (2019), is a representative that trains a language model with task-specific control codes for each type of text corpus. Another work is POINTER, proposed by Zhang et al. (2020), which stacks the architecture of insertion-based transformer (Chan et al., 2019) in a hierarchical manner to enforce hard lexical constrains during text generation. This type of methods could control the generated text effectively, but may negatively affect generalization of the PLM. They also usually have high computational footprint, and large-scale task-specific annotated data.

Fine-tuning. These methods require partial or full fine-tuning of a PLM for each individual target attribute. For example, Bostrom et al. (2021) proposed ParaPattern which fine-tunes BART-based models (Lewis et al., 2020a) to generate text via applying different logical operations to premise inputs. Ribeiro et al. (2021) fine-tunes PLMs to control the generation from different types of graphical data. The prefix-tuning, proposed by Li and Liang (2021), only optimizes a task-specific vector (prefix), while freezing the rest of PLM, to control the domain of generation. Fine-tuning PLMs based on a small amount of labelled data for the specific downstream task has achieved competitive performance. However, fine-tuning based methods usually steer the PLM from the side of input, which means it could be hard to enforce hard constrains on the outputs directly.

Post-processing. These methods usually do not require task-specific data to fine-tune the PLM, but require decoding algorithms to re-rank the generated text in a post-processing manner. As a representative work, PPLM, proposed by Dathathri et al. (2019), uses gradients from an attribute discriminant model to steer the text generation. FUDGE, proposed by Yang and Klein (2021), weights the decoding probabilities with an attribute predictor which takes partial sequence as input. Su et al. (2022b) proposed Contrastive Search decoding, which encourages diversity by penalizing repetitive tokens. Lu et al. (2021) proposed NeuroLogic Decoding, which enforces the generation to satisfy a set of pre-defined hard lexical constrains. MAGIC, proposed by (Su et al., 2022a), applies an image relevance discriminator to guide the generation process with visual information. This type of methods are usually computationally cheap and flexible, because they have a separate guiding module. The increasing number of parameters of the PLM would not affect the complexities of the methods. Our approach falls into this category of methods.

2.2 Generation with Plan

From the perspective of CTG tasks, attributes to control during generation include sentiment, topic, style, formality, story structure, content plan, among others. For example, Ghosh et al. (2017) proposed Affect-LM, which extend an LSTM language model by conditioning on pre-defined affect categories and strength. Fu et al. (2018) investigated the task of learning paper-news title style transfer from non-parallel data. Li et al. (2020) proposed the framework of SongNet which studies rigid format control to generate poems or songs that obey pre-defined rhyming schemes.

Previous works in controlling the generation with content planning are mainly focusing on the task of data-to-text generation and always taking *schema selection* and *ordering* as content plans.[1] For example, Moryossef et al. (2019) separates planning from neural text realization and takes the most probable traversal of RDF graph trees as content plan. Zhao et al. (2020) employs a GCN encoder to order the nodes of input RDF data as content plan. Su et al. (2021) proposed Plan-then-Generate, which treats orders of tabular schema as plans and then plans are encoded along with linearized data as inputs to a generative model. However, those methods require graphical or tabular input data and can only model content planning based on the data schema, which limits their domain of application.

2.3 Recipe Generation

Recipe generation refers to the task of generating recipe instructions from food images or textual ingredients and food title. Because recipes have the natural step-by-step sequence flow, sentence-level content planning is desired in order to generate high quality recipes.

Previous works tackled this issue in many directions. Chandu et al. (2019) treated this problem as a Visual Story Telling task and built a dataset containing images and text for each intermediate step. The recipe instructions are generated from the sequential images. Kiddon et al. (2016) models global coherence of the recipes by maintaing an ingredient checklist dynamically. During generation, a language model is encouraged to refer to the checklist item. Bosselut et al. (2018) tracks ingredient entity with a recurrent memory module and explicitly models actions as a set of per-defined state transforming operations. The recipes are then interpreted as structured collections of ingredient entities executed upon by cooking actions. (Majumder et al., 2019) investigated the task of personalized recipe generation. The user's previously consumed recipes are encoded and attended by recipe name and ingredients to generate complete instructions. However, they require complicated input data format, and sophisticated planning templates.

Recipe1M+, introduced by Marin et al. (2019), is an extension of Recipe1M (Salvador et al., 2017) and contains over 1M textual recipes and ingre-

[1] Schema selection and ordering depend on input data structure, e.g. selecting and re-ordering the cells of tabular data or the nodes of graphical data.

dients and 13M corresponding food images. The dataset has been used for versatile tasks, such as image-recipe retrieval (Chen et al., 2017), multi-modal embedding learning (Min et al., 2017), and recipe generation (Salvador et al., 2019). The RecipeGPT, proposed by H. Lee et al. (2020), fine-tuned a GPT-2 as a backbone generation model, taking only recipe titles and ingredients as input and recipe instructions as output. The NeuroLogic Decoding (Lu et al., 2021) takes the same setup, while enforcing hard lexical constrains on the occurrence of the ingredients. We follow the setup of these works and only consider the textual components of the Recipe1M+.

To the best of our knowledge, for the task of CTG with content planning, there has been no previous attempt neither on dataset with more flexible format such as recipe, nor with a plug-and-play post-process method.

3 Methodology

3.1 Overview

Figure 1 depicts our proposed framework. Given the recipe title and ingredients, the planner module (§3.2) first predicts the most probable content plan. The predicted content plan then guides the generation of the sequence generator (§3.3) via a lightweight and plug-and-play operation. Below, we elaborate the details of the proposed approach.

3.2 Content Planner

A carefully designed plan schema could be vital for systems that require sophisticated controls in neural models. By examining recipe instructions, we observe the fact that they share a common structure of sequence of step-by-step stages and there are natural patterns behind those stage sequences. Therefore, we treat a content plan as a sequence of stages, where some stages could be of the same types. We define 7 types of instruction stage based on the processing step of the food, including:

- **Pre-processing** means the preparations of ingredients or cooker.
- **Mixing** includes actions of combining one or more ingredients together.
- **Transferring** is for the actions of moving or transferring food or intermediate food to a specific place.
- **Cooking** represents the actual cooking actions, which could be very different from recipe to recipe.
- **Post-processing** usually refers to the following up actions after the 'cooking' stage, such as 'cooling down', 'garnish'.
- **Final** refers to the last few actions before serving the food or the serving action itself.
- **General** includes the rest of actions which cannot be classified into the above categories.

Stage Types	Keywords
Pre-processing	Peel, beat, rinse, prepare ...
Mixing	Mix, add, combine, blend ...
Transferring	Move, put, pour, place ...
Cooking	Fry, bake, cook, boil, grill ...
Post-processing	Cool, shake, garnish, cover ...
Final	Serve, yield, wrap, enjoy ...
General	Uncovered or ambiguous verbs

Table 1: Seven stage types and example keywords for each stage type.

As recipe instructions are usually sentences led by action verbs, an assumption is made that the stage types of the instructions are decided by their main action verbs. For each type of stage, we assign a set of exclusive stage-specific action verbs, as shown in Table 1. For example, the 'cooking' stage includes actions such as 'fry', 'bake', 'boil' and so on. We built a rule-based system that automatically tags recipe instructions with stage labels according to the pre-defined verb sets. We tag the instructions from train set of Recipe1M+, which contains around 710K recipes, with the stage labels and refer to them as silver labels. In Appendix A.3, we elaborate more implementation details of the rule-based stage tagging system. In §4.2, we evaluate the quality of the silver labels with human annotations on an evaluation subset. By this way, we can obtain the content plan of a recipe, which is the sequence of the stage labels of that recipe.

After acquiring the content plan $\boldsymbol{c} = \{c_1, c_2, ..., c_{|\boldsymbol{c}|}\}$ using our rule-based system, the distribution of $\boldsymbol{c}$ is then modelled by the content planner as $P(\boldsymbol{c}|\boldsymbol{x})$, where $\boldsymbol{x}$ is the given recipe title and ingredients, and c_j belongs to one of the seven stage types defined in as Table 1. Specifically, given the recipe title and ingredients, we use a Seq2seq model, i.e. BART (Lewis et al., 2020b), to model the content plan as

$$P(\boldsymbol{c}|\boldsymbol{x}) = \prod_{j=1}^{|\boldsymbol{c}|} p_{\theta_c}(c_j|\boldsymbol{c}_{<j}; \boldsymbol{x}), \qquad (1)$$

where θ_c is the parameters of the content planner. The main assumption of our modelling choice is that the content plan, i.e. cooking procedure, could be mostly determined once the target food and ingredients are known.

3.3 Plan-Aware Decoding

Given the recipe title and ingredients $\boldsymbol{x}$, and the content plan $\boldsymbol{c}$, we formulate the conditional distribution of recipe $\boldsymbol{y}$ by following the Bayes rule as

$$\begin{aligned} P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{c}) &= \prod_{i=1}^{|\boldsymbol{y}|} p(y_i|\boldsymbol{y}_{<i};\boldsymbol{x},\boldsymbol{c}) \\ &\propto \prod_{i=1}^{|\boldsymbol{y}|} p_{\theta_g}(y_i|\boldsymbol{y}_{<i};\boldsymbol{x}) \cdot p_{\theta_f}(c_j|\boldsymbol{y}_{k:i}), \end{aligned} \tag{2}$$

where c_j refers to the stage that the current partial sequence $\boldsymbol{y}_{k:i}$ belongs to. The θ_f is an off-the-shelf stage classifier which predicts the probability distribution over 7 stage classes by taking the partial sequence $\boldsymbol{y}_{k:i}$ as input. It should be noted that we assume the probability of the current stage label should only depend on the partial sequence that belongs to the current stage.

During inference, based on Equation 2, the selection of the output token at step i follows

$$y_i = \underset{y_i \in V_S}{\operatorname{argmax}} \, p_{\theta_g}(y_i|\boldsymbol{y}_{<i};\boldsymbol{x})^{(1-\alpha)} \cdot p_{\theta_f}(c_j|\boldsymbol{y}_{k:i})^{\alpha}, \tag{3}$$

where α is a hyper-parameter that regulates the importance of two terms. V_S is the set of top-S predictions from the sequence generator's probability $p_{\theta_g}(\cdot|\boldsymbol{y}_{<i};\boldsymbol{x})$ and S is set as 5 by default. We use the sequence generator's predictions on subset V_S to approximate the predictions over the total vocabulary. With this approximation, the stage classifier only needs to be applied upon S candidates, therefore assuring the computational efficiency.

In this work, we fine-tune a GPT-2 (Radford et al.) on the train set of our evaluated benchmark Recipe1M+ to make it the sequence generator. To acquire the stage classifier, we fine-tune a lightweight DistilBERT (Sanh et al., 2019) on the partial recipe instructions with the silver stage labels we obtain as described in §3.2.

Intuitively, our approach can be deemed as utilizing the stage classifier as a re-ranking step on the top S candidates predicted by the sequence generator. An example is shown in Figure 1. The sequence generator first predicts probabilities across all the vocabulary, in which the word 'barbeque' has the highest likelihood, then the stage classifier re-ranks the predictions based on the current stage label 'cooking' and assigns the highest probability to 'toss'.

We note that using a partial sequence stage classifier to guide the decoding shares a similar idea with the previous study, i.e. FUDGE (Yang and Klein, 2021). However, in contrast to FUDEG, our approach works on discriminating 7-class planning stages rather than only supporting binary attributes. In addition, to ensure the structural fluency of the generated recipe, we also control the generation from the perspective of global content planning, rather than focusing on one single control attribute.

3.4 Advantages and limitations

In this section we discuss the theoretical advantages and limitations of our framework.

We highlight the advantages that: (i) We control the generation process in the plug-and-play manner, with no need to fine-tune the large language model in the sequence generator module with plan-specific data. In another words, given an off-the-shelf stage classifier and content planner, our framework is training-free. (ii) The stage classifier and content planner are both lightweight models compared to the sequence generator and can be fine-tuned with non-parallel data. (iii) Because the content plan schema is designed by humans, our framework can effectively inject human knowledge of the constrain patterns explicitly into the generation process.

We also point out the limitations of our framework: (i) The overall performance of our model depends on the manually designed plan schema, which cannot be perfect, as it is based on authors' experience. There are many cases where the stage of the instructions could be ambiguous. For example, in a real recipe, it is possible for 'slice the steak' to be both type of 'pre-processing' or 'post-processing'. It is hard for humans to decide whether 'add salt and pepper' is a special case of 'seasoning' which is type of 'pre-processing', or type of 'mixing'. Another example is 'pour milk and mix well', which contains two verbs from two stages. (ii) As pointed out by Zhang et al. (2022), guided re-ranking algorithm, as a type of post-processing controlled text generation methods, suffers the problem of relatively low control strength, compared with the methods based on fine-

Metrics	Planner
Uni-gram	69.4
Bi-gram	42.3
Tri-gram	16.9
Exact match	39.0

Table 2: Planner module evaluation results. Match rate accuracy (in percentage) for uni-gram, bi-gram, tri-gram and exact match, between predicted and reference plans

tuning or retraining.

4 Experiments

In this section we evaluate our method from three aspects: The performance of planner module, the performance of the stage classifier and the performance of the recipe generation. The details of how they are trained and evaluated are explained in details in §4.1, §4.2 and §4.3 respectively. In Appendix A.4, we demonstrate a few generation examples of our model and the baselines.

We pre-processed the Recipe1M+ dataset by firstly filtering out too short instructions (instructions with less than 3 words, for example 'combine all') which are usually trivial, and truncating recipes with too many instructions at the length of 15, because recipes with too many instructions usually include irrelevant information due to data scraping errors. By this way, about 9% of the original Recipe1M+ are filtered out and the resulting dataset are used for all the evaluation experiments below.

4.1 Planner Evaluation

The content planner module predicts the sequence of stage plans from the given recipe title and ingredients. We took the sequences of the silver stage labels as reference plans and finetuned a seq2seq model, BART base version. The silver labels are generated through the automatic tagging system described in §3.2. We evaluate the content planner module on the test set of the Recipe 1M+.

The exact match rate is the percentage of matched stages in their exact positions of the reference plan. As shown in Table 2, our planner module achieves 39% accuracy. Additional to this, because we are comparing two plan sequences, n-gram match rates are also important indicators to measure how good underlying patterns are learnt. We show that for uni-gram and bi-gram we achieved relatively high match rates at 69.4% and 42.3% and for tri-gram we got 16.9%. This drop shows that our planner can learn the patterns between two successive instructions to an acceptable level, but the patterns among three successive instructions become hard to predict.

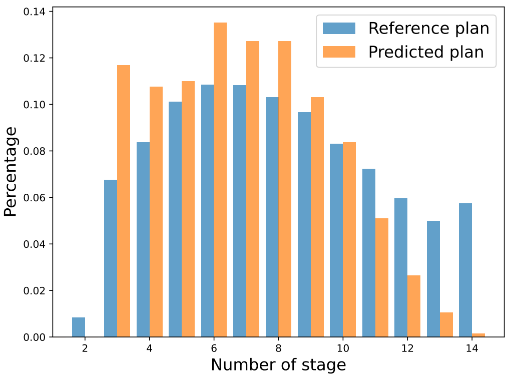

Figure 2: Histograms of the length of the predicted and reference plans.

Model	Accuracy
Stage classifier	56.3
Silver label	62.7

Table 3: Stage classifier evaluation results. The accuracy (in percentage) of the predictions of our stage classifier and automatically tagged silver stage labels, compared with manually labeled gold annotations.

To further illustrate the performance of the content planner module, we also compare the distribution of lengths of the predicted and reference stage plans. As shown in Figure 2, their histograms show similar bell-shape and the percentage of their mismatching is around 29.1%, which we consider as acceptably low. The main source of this, we believe, is due to the heavy tail of the distribution at length of 15. As explained in Appendix A.1, this is caused by the truncation of recipe instructions during pre-processing.

4.2 Stage Classifier Evaluation

A stage classifier predicts the stage label for the each given full or partial recipe instruction. It is implemented by fine-tuning a DistilBERT with partial instructions with the silver stage labels. The partial instructions can be obtained by truncating the instructions from the train set of the Recipe1M+ at random positions. The size of the resulting partial instructions training set is around 4.9M. In this section, we evaluate the performance of our stage

Model	BLEU	ROUGE-L	Plan Match	Coverage	Extra
RecipeGPT, top-k	11.5	34.8	26.0	59.0	24.0
RecipeGPT, beam	12.2	37.1	24.0	63.1	21.9
NeuroLogic	11.8	38.2	21.8	**67.1**	22.5
Our model	**13.9**	**39.1**	**40.6**	65.4	**20.7**
Our model, oracle	14.3	40.6	39.2	65.8	22.0

Table 4: Experimental results for our models and the baselines. *Oracle* version of our model represents the plan-aware decoding is guided by the reference plan. All models are evaluated on the same evaluation subset.

Method	Fluency	Quality
RecipeGPT, beam	4.07	0.68
Proposed model	**4.34**	**0.88**
NeuroLogic	3.87	0.59
Proposed model	**4.27**	**0.85**
Our model, oracle	**4.31**	**0.81**
Our model	4.28	0.76

Table 5: Human evaluations on the generated recipes. A/B test style pairwise comparisons.

classifier.

We construct a evaluation set by randomly sampling 300 examples from the recipes instructions in the test set. Then we asked three human annotators to manually annotate the instructions with stage labels following the guidance we provided and the intuition of their own. The human annotations are referred as gold labels. In Table 3, we evaluate the stage classifier and the rule-based tagging system with the gold labelled evaluation set. The stage classifier achieves accuracy of 56.3%, while its upper bound, the silver labels, has accuracy of 62%. We consider this is an acceptable performance, because this is a 7-class classification problem and there is subjective understanding of the imperfect plan schema, as expalined in §3.4.

4.3 Recipe Generation

In this section, we evaluate our plan-aware decoding method with both automatic metrics and human evaluations, and compare the performance with two types of baseline methods. The sequence generator of our model is a base version of GPT-2, finetuned on the train set of the Recipe1M+ dataset. We preprocessed the recipe data with special separation tokens, as shown in the example in Figure 1, which details are explained in Appendix A.1. Both the sequence generator of our model and baselines are fine-tuned on the processed data.

4.3.1 Baselines

RecipeGPT: The RecipeGPT, proposed by H. Lee et al. (2020), fin-tuned a base version of GPT-2 with train set of the Recipe1M+ dataset on the recipe generation task. During generation, it employs two types of decoding methods, top-k sampling and beam search. We re-implement the RecipeGPT as a representative of finetune-based methods and set the sampling candidate number and the beam size as 5.

NeuroLogic decoding: The NeuroLogic decoding, proposed by Lu et al. (2021), is a post-processing method which can be applied to different generative backbone models. It tries to search for optimal output sequences that also satifsy a set of pre-defined lexical constrains. The constrains enforce certain words to appear or not appear in the generated sequences. In this work, we choose the base version of GPT-2 as the underlying generative model and set the constrains such that all ingredients from the inputs should appear in the generated sequences. The beam size is set as 5.

4.3.2 Metrics

Automatic Metrics: We use two widely-used metrics to assess surface similarities, BLEU score (Papineni et al., 2002) and ROUGE-L (Lin and Hovy, 2002), with 1 reference recipe per test example. To measure the control abilities of the models, we measure the plan match rate, which is the average percentage of the stage plan of the generated recipes that agree with the input stage plan. The stage plans of the generated recipes are also labelled by the rule-based stage tagging system described in §3.2. In Table 4, we refer the plan match rate as Plan Match.

We also explicitly measures the average percentage of coverage of the given ingredients and the percentage of hallucinated ingredients. We refer to them as coverage and extra respectively in Table 4. The details of how they are calculated are in

Appendix A.2

Human evaluations To make a similar comparison, we follow the human evaluation setup in previous works such as the FUDGE (Yang and Klein, 2021) and the PPLM (Dathathri et al., 2019). We run A/B test style human evaluations to compare our model with the baselines on fluency and quality. For each pairwise comparison between models, we ask evaluators to evaluate 100 pairs of generated recipes given input title and ingredients. The evaluators were asked to rate the fluency, in Likert scale from 1 to 5, and the quality of the generated recipes. For the quality, the evaluators need to decide, by their intuition, which recipe can reproduce the food described by the given title (recipe A, recipe B, neither or both). In Table 5, we report quality as the percentage of the recipes that can reproduce the food.

4.3.3 Results

We create an evaluation subset by randomly sampling 4000 examples from the test set of the Recipe1M+, because the test set contains over 120K recipes, which is too large to evaluate decoding algorithms. All evaluation experiments are conducted on this same subset. The scores presented in Table 4 are the results averaged over 5 random seeds.

Apart from the the aforementioned baselines, we also evaluate the oracle version of our model, which takes the reference stage plans as the guidance rather than the ones predicted by the content planner module. We show that our model outperforms all three baselines in all metrics except the ingredient coverage. The differences are statistically significant for BLEU, ROUGE-L and Plan Match with $p < 0.01$. For the percentage of hallucination ingredients, the difference is weakly significant ($p < 0.1$). The significantly better performance on BLEU and ROUGE-L justifies that our model can produce recipes with better surface similarities by injecting the knowledge of content plans. The NeuroLogic decoding achieves highest ingredient coverage, which, in our opinion, is because it explicitly priorities the hard constrain of occurrence of the ingredients over surface fluency. It is worth noting that our models, including the oracle version, generally achieve significantly higher plan match rate than all the baselines. This shows that they can effectively control the generation process to generate recipes which follow the given content plans.

For the human evaluation, our model outperforms two baselines in pairwise comparisons on both fluency and recipe quality. We observe that, with the help of the stage plan, our model can produce much less repeated, irrelevant or redundant instructions, compared with the baselines. Furthermore, by explicitly conditioning on stage plans, the recipes generated by our model are considered of better quality, which means they are easier for human readers to follow successfully. We show a few generation examples in the case study in Appendix A.4. The oracle version has superior performance over our normal model, which suggests that better stage plans can effectively provide human reader better reading experience and more helpful guidance.

5 Conclusion

In this paper, we identify the research gap for the current controlled text generation models to generate text with sentence-level content planning, and propose a framework that optimizes the joint distribution of the natural sequence and the content plans in a lightweight plug-and-play manner. Our framework outperforms the previous works and achieve a new state of the art on the recipe generation task on the Recipe1M+. Through extensive evaluations with both automatic metrics and human evaluations, we show that the recipes guided by explicit content plans, are more accurate and controllable.

References

Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018. Simulating action dynamics with neural process networks. In *International Conference on Learning Representations*.

Kaj Bostrom, Xinyu Zhao, Swarat Chaudhuri, and Greg Durrett. 2021. Flexible generation of natural language deductions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6266–6278.

William Chan, Nikita Kitaev, Kelvin Guu, Mitchell Stern, and Jakob Uszkoreit. 2019. Kermit: Generative insertion-based modeling for sequences. *arXiv preprint arXiv:1906.01604*.

Khyathi Chandu, Eric Nyberg, and Alan W Black. 2019. Storyboarding of recipes: Grounded contextual generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6040–6046, Florence, Italy. Association for Computational Linguistics.

Jing-jing Chen, Chong-Wah Ngo, and Tat-Seng Chua. 2017. Cross-modal recipe retrieval with rich food attributes. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1771–1779.

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. In *International Conference on Learning Representations*.

Jonathan St BT Evans. 2003. In two minds: dual-process accounts of reasoning. *Trends in cognitive sciences*, 7(10):454–459.

Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. 2018. Style transfer in text: Exploration and evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

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 *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 634–642.

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*, pages 181–184.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

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

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas. Association for Computational Linguistics.

Yann LeCun. 2022. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020b. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Piji Li, Haisong Zhang, Xiaojiang Liu, and Shuming Shi. 2020. Rigid formats controlled text generation. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 742–751.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *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)*, pages 4582–4597.

Chin-Yew Lin and Eduard Hovy. 2002. Manual and automatic evaluation of summaries. In *Proceedings of the ACL-02 Workshop on Automatic Summarization*, pages 45–51.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Neurologic decoding:(un) supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4299.

Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. 2019. Generating personalized recipes from historical user preferences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5976–5982.

Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. 2019. 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.

Weiqing Min, Shuqiang Jiang, Shuhui Wang, Jitao Sang, and Shuhuan Mei. 2017. A delicious recipe analysis framework for exploring multi-modal recipes with various attributes. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 402–410.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019. Step-by-step: Separating planning from realization in neural data-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2267–2277, Minneapolis, Minnesota. Association for Computational Linguistics.

Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M Lake. 2021. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34:25192–25204.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners.

Leonardo FR Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. 2021. Investigating pretrained language models for graph-to-text generation. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pages 211–227.

Amaia Salvador, Michal Drozdzal, Xavier Giro-i Nieto, and Adriana Romero. 2019. Inverse cooking: Recipe generation from food images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

Amaia Salvador, Nicholas Hynes, Yusuf Aytar, Javier Marin, Ferda Ofli, Ingmar Weber, and Antonio Torralba. 2017. Learning cross-modal embeddings for cooking recipes and food images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

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

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L Hamilton. 2019. Clutrr: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4506–4515.

Yixuan Su, Tian Lan, Yahui Liu, Fangyu Liu, Dani Yogatama, Yan Wang, Lingpeng Kong, and Nigel Collier. 2022a. Language models can see: Plugging visual controls in text generation. *arXiv preprint arXiv:2205.02655.*

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. 2022b. A contrastive framework for neural text generation. *arXiv preprint arXiv:2202.06417.*

Yixuan Su, David Vandyke, Sihui Wang, Yimai Fang, and Nigel Collier. 2021. Plan-then-generate: Controlled data-to-text generation via planning. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 895–909.

Hongyin Tang, Miao Li, and Beihong Jin. 2019. A topic augmented text generation model: Joint learning of semantics and structural features. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5090–5099.

Wenlin Wang, Zhe Gan, Hongteng Xu, Ruiyi Zhang, Guoyin Wang, Dinghan Shen, Changyou Chen, and Lawrence Carin. 2019. Topic-guided variational auto-encoder for text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 166–177.

Kevin Yang and Dan Klein. 2021. Fudge: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3511–3535.

Hanqing Zhang, Haolin Song, Shaoyu Li, Ming Zhou, and Dawei Song. 2022. A survey of controllable text generation using transformer-based pre-trained language models. *arXiv preprint arXiv:2201.05337.*

Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and William B Dolan. 2020. Pointer: Constrained progressive text generation via insertion-based generative pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8649–8670.

Chao Zhao, Marilyn Walker, and Snigdha Chaturvedi. 2020. Bridging the structural gap between encoding and decoding for data-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2481–2491, Online. Association for Computational Linguistics.

A Appendix

A.1 Details of preprocessing

We then further processed the recipes by adding special separation tokens, as shown in the example in Figure 1. The separation tokens include $< TITLE\_START >$ and $< TITLE\_END >$ to wrap the recipe title, $< INGR\_START >$, $< INGR\_END >$ and $< INGR\_NEXT >$ to wrap and split the recipe ingredients. Similarly, $< INSTR\_START >$, $< INSTR\_END >$ and $< INSTR\_NEXT >$ are used to wrap and split the recipe instructions. There is no leaking of the stage label information from the separation tokens.

A.2 Details of metrics computing

To identify the ingredients in the generated recipes, we first create a list of total input ingredients in the Recipe1M+ dataset and then identify the ingredients in the recipes by string match. The hallucination percentage is the number of hallucinated ingredients over the total number of ingredients in the input. Ingredients that are not included in the input, but included in the total ingredient list, are considered as hallucinated. It is worth noting that because of the limitations of string match, which cannot deal with plural, quantifier, synonym and etc, the coverage and hallucination percentages are not perfect. Therefore, they are better interpreted as rough indicators and used to compare between models parallelly.

A.3 Rule-based stage label tagging system

In this section, we elaborate how we implement the rule-based tagging system. To process on instruction, firstly we use the tokenizer from Python package Spacy (Honnibal and Montani, 2017) to identify all the verbs by checking the Part-Of-Speech (POS) tag of each token. Then we remove the verbs that are in the clauses by identifying punctuation and conjunction words. If, by this point, there are more than one verbs left, we always keep the first verb as the main verb and tag the instruction base on this verb. We tag the stage label by looking up which stage type the main verb belongs to, as shown in the example keywords in Table 1.

A.4 Case study: Recipe Generation Example

In Table 6, we show generation examples of three recipes. Instructions labeled in red show undesired characteristics.

In the first example, the ‘Eggs Barbados’, the recipes generated by baselines, RecipeGPT and NeuroLogic, both have the instruction ‘Preheat oven to 350 degrees f.’. We consider this instruction is irrelevant because the oven is never used in the rest of the recipes.

In the second example, the ‘Our Favourite Wheat Bread’, the recipe generated by RecipeGPT shows undesired repetitive patterns on the instructions ‘Punch down and let rise again.’.

In the third example, the ‘Pomegranate Ice Tea’, the recipe generated by RecipeGPT is inaccurate by ‘Serve immediately’, because it lacks the step of ‘Pour over ice’ or ‘add ice’.

Recipe Title	Ingredients	Model	Generated Results

Eggs Barbados	1 bunch green onions;
	1 package cream cheese;
	1 tablespoon butter;
	12 whole eggs;
	1 tablespoon milk;
	salt and pepper to taste
		RecipeGPT	Preheat oven to 350 degrees f. Slice the green onions in half lengthwise. Slice the cream cheese in half lengthwise. Melt the butter in a large skillet over medium heat. Add the green onions and saute until tender. Add the cream cheese, eggs, milk and salt and pepper to taste. Cook until the eggs are set. Serve immediately.

		NeuroLogic	Preheat oven to 350 degrees f. Slice the green onions and the cream cheese in half lengthwise. Melt the butter in the pan over medium heat. Add the green onions and saute until tender. Add the cream cheese, eggs, milk and salt and pepper to taste. Cook until the eggs are completely cooked.
		Ours	Slice the green onions into 1/4 inch slices. Melt the butter in a frying pan. Add the chopped onions and saute until they are soft. Add the cream cheese and mix well. In a separate bowl, beat the eggs and milk. Add the eggs to the cream cheese mixture and mix well. Add the green onions and salt and pepper to taste. Drop by spoonfuls onto a greased baking sheet. Bake at 350 degrees for 20 minutes.
		Reference	Chop your green onions and break up the cream cheese into little chunks. Melt the butter in your frying pan on medium-low heat. While the butter melts, beat the eggs with the milk until combined. Add your salt and pepper. pour the egg mixture into the pan. Dont stir up the eggs too much, just occasionally scrape the bottom to mix up the cooked egg with the runny stuff. Once the eggs are mostly cooked but still look a little wet, turn off the heat. You dont want overcooked eggs, theyre rubbery and icky! As soon as you turn off the heat, mix the cream cheese chunks and chopped green onions into the eggs. Let it sit for just a few seconds, so that the heat of the pan melts the cream cheese.

Our Favourite Wheat Bread	1 tablespoon dry yeast;
	2 cups warm water;
	14 cup soft butter;
	18 cup molasses;
	14 cup honey;
	1 teaspoon salt;
	3 cups whole wheat flour;
	2 cups white flour.
		RecipeGPT	Dissolve yeast in warm water. Add butter, molasses, honey, and salt. Mix well. Add whole wheat flour and white flour. Knead until smooth and elastic. Cover and let rise in a warm place for 1 hour. Punch down and let rise again. Punch down again and let rise again. Punch down again and let rise again. Punch down again and let rise again. Punch down again and let rise again. Punch down again and let rise again. Punch down again and let rise again. Punch down again and let rise again

		NeuroLogic	Dissolve yeast in warm water. Add water, butter, molasses, honey, and salt. Mix well. Add whole wheat flour and white flour. Mix well. Knead until smooth and elastic. Cover and let rise in a warm place for 1 hour. Bake for 1 hour.
		Ours	Dissolve yeast in warm water. Add butter, molasses, honey, and salt. Add whole wheat flour and white flour. Mix until smooth. Cover and allow to rise in a warm place until doubled in size. Punch down and let rise again. Pre-heat oven to 350 degrees. Grease a loaf pan. Pour in batter. Bake for 1 hour.
		Reference	Dissolve yeast in warm water. Combine butter, molasses, honey, and salt. Add yeast mixture. Gradually add flour. Turn onto floured surface and knead until smooth. Place in greased bowl and let rise until double. Let dough rest a few minutes. Shape into 2 loaves. Place in greased bread pans. Let rise about 1 hour. Bake at 375 for 35-40 minutes.

Pomegranate Iced Tea	2 ounces freshly squeezed pomegranate juice or
	2 ounces store-bought pomegranate juice;
	5 ounces tea;
	1 ounce simple syrup;
	lemon wedge.
		RecipeGPT	Combine the pomegranate juice, tea, simple syrup, and lemon wedge in a pitcher. Serve immediately.

		NeuroLogic	Combine the pomegranate juice, tea, simple syrup, and lemon wedge in a pitcher. Add ice and stir well.
		Ours	Combine the pomegranate juice, tea, simple syrup and lemon wedge in a pitcher. Add ice and stir well. Pour into a chilled glass and garnish with a lemon wedge.
		Reference	Prepare tea and cool. Mix the rest of the ingredients. Pour over ice in an iced tea glass. Garnish with lemon wedge.

Table 6: Case Study. Generation examples of 3 recipes by our model and baselines. Instructions labeled in red are considered problematic.