Assistive Recipe Editing through Critiquing

Diego Antognini
École Polytechnique Fédérale de Lausanne
diego.antognini@epfl.ch

Shuyang Li
University of California, San Diego
shl008@ucsd.edu

Boi Faltings
École Polytechnique Fédérale de Lausanne
boi.faltings@epfl.ch

Julian McAuley
University of California, San Diego
jmcauley@ucsd.edu

ABSTRACT

There has recently been growing interest in the automatic generation of cooking recipes that satisfy some form of dietary restrictions, thanks in part to the availability of online recipe data. Prior studies have used pre-trained language models, or relied on small paired recipe data (e.g., a recipe paired with a similar one that satisfies a dietary constraint). However, pre-trained language models generate inconsistent or incoherent recipes, and paired datasets are not available at scale. We address these deficiencies with RecipeCrit, a hierarchical denoising auto-encoder that edits recipes given ingredient-level critiques. The model is trained for recipe completion to learn semantic relationships within recipes. Our work's main innovation is our unsupervised critiquing module that allows users to *edit* recipes by interacting with the predicted ingredients; the system iteratively rewrites recipes to satisfy users' feedback. Experiments on the Recipe1M recipe dataset show that our model can more effectively edit recipes compared to strong language-modeling baselines, creating recipes that satisfy user constraints and are more correct, serendipitous, coherent, and relevant as measured by human judges.

1 INTRODUCTION

Individual preferences and dietary needs shape the types of recipes that home cooks choose to follow. Cooks must often accommodate the desire for versions of recipes that do not contain a specific ingredient (substitution—e.g., for food allergies) or *do* make use of particular ingredients (addition—e.g., to use up near-expiry items). We thus aim to build a system for *recipe editing* that accommodates fine-grained ingredient preferences.

Prior research in recipe editing has thus far focused on replacing individual ingredients from the ingredients list with substitutes [30] or recommending new recipes based on learned clusters of similar ingredients [23]. While individual ingredient substitution rules may work in a vacuum, such substitutions (e.g., tapioca flour and xanthan gum for all-purpose flour) often necessitate additional changes to the cooking procedure (i.e., directions) to function properly [12]. Other studies employed recommendation-based approaches. However, they suffer from data sparsity: there is an extremely large set of possible recipes that differ by a single ingredient, and many specific substitutions may not appear in recipe aggregators [18].

Recipe editing can be seen as a combination of recipe generation and controllable natural language generation [22]. It has recently been explored for creating recipes that satisfy a specific dietary constraint [12] or follow a specified cuisine [16]. On one hand, pre-trained language models have been used to create recipe directions given a known title and set of ingredients [4, 9, 10], but generated recipes suffer from inconsistency [12]. On the other hand, [12] proposed a method to build a paired recipe dataset, but face difficulties scaling due to the large set of possible recipes and dietary restrictions; people's preferences often lean to be even more specific, on the ingredient-level (dislikes of certain ingredients or allergies).

In this work, we address the above challenges and propose RecipeCrit, a denoising-based model trained to complete recipes and learn semantic relationships between ingredients and instructions. The novelty of this work relies on an *unsupervised* critiquing method that allows users to provide ingredient-focused feedback iteratively; the model substitutes ingredients and also re-writes the recipe text using a generative language model. While existing methods for controllable generation require paired data with specially constructed prompts [8] or hyperparameter-sensitive training of individual models for each possible piece of feedback [5], our unsupervised critiquing framework enables recipe editing models to be trained with arbitrary un-paired recipe data.

Experiments on the Recipe1M [21] dataset show that RecipeCrit edits recipes in a way that better satisfies user constraints, preserves the original recipe, and produces coherent recipes compared to state-of-the-art pre-trained recipe generators and language models. Human evaluators judge RecipeCrit's recipes to be more serendipitous, correct, coherent, and relevant to the ingredient-specific positive and negative feedback (i.e., critiques).

2 RECIPECRIT: A HIERARCHICAL DENOISING RECIPE AUTO-ENCODER

Previous methods to edit recipes focused on broad classes like dietary categories [12] and cuisines [16] and require paired corpora (which do not exist for fine-grained edits). We propose a method that does not require paired corpora to train and accommodates positive and negative user feedback on an ingredient-level.

2.1 Model Overview

To leverage datasets of recipes comprising a title $\mathbf{x}^{ttl}$, an ingredients list $\mathbf{X}^{ing}$, and instructions $\mathbf{X}^{ins}$, we propose to train our **RecipeCrit** model as a denoising auto-encoder [26] for recipe *completion.* RecipeCrit is divided into three submodels: an Encoder $E(\cdot)$, which produces the latent representation $\mathbf{z}$ from the (potentially noisy) recipe; an ingredient predictor $C(\cdot)$, which predicts the ingredients $\hat{\mathbf{y}}^{ing}$, and a decoder $D(\cdot)$, which reconstructs the cooking instructions $\hat{\mathbf{y}}^{ins}$ from $\mathbf{z}$ conditioned on the $\hat{\mathbf{y}}^{ing}$.

Algorithm 1 Iterative Critiquing Gradient Update (Crit).

function CRITIQUE(latent vector $\mathbf{z}$, critiqued ingredient c, trained ingredients predictor C, decay coefficient ζ, patience P, a maximum number of iterations T, desired ingredients $\tilde{\mathbf{y}}^{ing}$)
  Set $\mathbf{z}_0 = \mathbf{z}^* = \mathbf{z}, \alpha_0 = 1$, best_val $= \infty$, patience $= 0, t = 1$;
  while patience $< P$ and $t < T$ **do**
    $\mathbf{g}_{t-1} = \nabla_{\mathbf{z}_{t-1}} \mathcal{L}_{ing}(C(\mathbf{z}_{t-1}), \tilde{\mathbf{y}}^{ing})$;
    $\mathbf{z}_t = \mathbf{z}_{t-1} - \alpha_{t-1} \frac{\mathbf{g}_{t-1}}{||\mathbf{g}_{t-1}||_2}$ and $\hat{\mathbf{y}}^{ing} = C(\mathbf{z}_t)$
    if $|\tilde{\mathbf{y}}_c^{ing} - \hat{\mathbf{y}}_c^{ing}| <$ best_val **then**
      best_val $= \hat{\mathbf{y}}_c^{ing}, \mathbf{z}^* = \mathbf{z}_t$, and patience $= 0$
    else
      patience $=$ patience $+ 1$
    $\alpha_t = \zeta \alpha_{t-1}$ and $t = t + 1$;
return $\mathbf{z}^*$;

2.1.1 Recipe Encoder $E(\cdot)$. We aim to build a powerful latent representation that accurately captures the different elements of a recipe. We compute text representations via the mean-pooled output of a transformer encoder [25]. While the title $\mathbf{x}^{ttl}$ comprises a single sentence, the ingredients $\mathbf{X}^{ing}$ and instructions $\mathbf{X}^{inst}$ are provided as lists of sentences. We encode the ingredients and instructions in a hierarchical manner using another transformer encoder to create fixed-length representations. Finally, we compute the latent representation $\mathbf{z}$ by concatenating the representations of each component and applying a projection followed by a tanh function:

$$\mathbf{z} = \tanh(\mathbf{W}[TRF(\mathbf{X}^{ing}) \parallel HTRF(\mathbf{X}^{ing}) \parallel HTRF(\mathbf{X}^{ins})] + \mathbf{b}), \quad (1)$$

where $\parallel$ is the concatenation, and $\mathbf{W}$, $\mathbf{b}$ the projection parameters.

2.1.2 Ingredient Predictor $C(\cdot)$. We treat ingredient prediction as multi-label binary classification. We follow [20] and build a normalized ingredient vocabulary I including an *EOS* token to determine the number of ingredients. We build a binary ingredient vector $\mathbf{y}^{ing}$ as the target for each recipe. Following [12, 20], we employ a set transformer. Instead of predicting the ingredients auto-regressively, we max-pool outputs across time-steps (ignoring the EOS token) into a single logit for each ingredient. Then, we uniquely sample ingredients until the *EOS* token is predicted. The loss function is:

$$\mathcal{L}_{ing}(C(\mathbf{z}), \mathbf{y}^{ing}) = -\sum\nolimits_i^{|I|} \mathbf{y}_i^{ing} \log \hat{\mathbf{y}}_i^{ing} - \lambda \mathbf{y}_{eos}^{ing} \log \hat{\mathbf{y}}_{eos}^{ing}, \quad (2)$$

where λ controls the impact of the *EOS* loss. At inference time, we return the top-k ingredients, where k is the first position with a positive *EOS* prediction.

2.1.3 Instruction Decoder $D(\cdot)$. The last component consists of generating the cooking instruction steps. We use a transformer decoder and aim to learn to generate coherent recipes. Therefore, in addition to conditioning the decoder on $\mathbf{z}$ and the previously generated outputs $\hat{\mathbf{y}}_{1:t-1}^{ins}$, we also condition on the ingredients $\hat{\mathbf{y}}^{ing}$. Specifically, we encode the ingredients using an embedding layer $A(\cdot)$ and concatenate their representations with the recipe latent representation $\mathbf{z}$. We train using teacher-forcing and cross-entropy:

$$\mathcal{L}_{ins}(\mathbf{z}, A(\hat{\mathbf{y}}^{ing})) = -\sum\nolimits_t \mathbf{y}_t^{ins} \log \hat{\mathbf{y}}_t^{ins}. \quad (3)$$

2.1.4 Training. Taking inspiration from masked language modeling [6] and masked span modeling [7, 29], we replace random sentences in the lists of ingredients and instructions with <mask> tokens. The model takes as input the masked recipe and is trained to generate the full, un-masked recipe. It thus learns to complete a recipe using the masked ingredients, which are processed according to the masked steps. We train our model in two stages. First, we train RecipeCrit and minimize the loss $\mathcal{L}_{ing}$. In the second stage, we freeze the encoder $E(\cdot)$ and minimize the decoding loss $\mathcal{L}_{ins}$ using the ground truth ingredients $\mathbf{y}^{ingr}$ and teacher forcing [28].

2.2 Unsupervised Critiquing

The purpose of editing is to refine the recipe based on the user's feedback and the predicted ingredients $\hat{\mathbf{y}}^{ing}$. We denote $\tilde{\mathbf{y}}^{ing}$ the vector of desired ingredients. We find that the simplest way of incorporating user feedback—forcing inclusion/exclusion in the ingredients list before generating instructions—often cannot satisfy user preferences due to weak conditioning between predicted ingredients and generated instructions. Thus, in RecipeCrit we turn to a critiquing method that modifies the recipe representation $\mathbf{z}$ before using the updated representation to jointly generate the edited ingredients and instructions. Specifically, users add a new ingredient c by setting $\hat{\mathbf{y}}_c^{ing} = 1$ or remove existing ones using $\hat{\mathbf{y}}_c^{ing} = 0$.

Inspired by success in editing the latent space in text style transfer and recommendation [1, 2, 27], we modulate $\mathbf{z}$ using the gradient of the ingredient predictor (w.r.t. $\mathbf{z}$ instead of the model parameters) such that the new predicted ingredients $\hat{\mathbf{y}}^{ing}$ are close to the desired ingredients $\tilde{\mathbf{y}}^{ing}$ (see Alg. 1). Unlike prior work, we avoid using stopping criteria based on the L1-norm between $\tilde{\mathbf{y}}^{ing}$ and $\hat{\mathbf{y}}^{ing}$ and a threshold. Due to the large set of ingredients, a maximum difference of 1.0 in one of the dimension will not significantly affect the L1-norm. Instead, we propose to compute the absolute difference between $\tilde{\mathbf{y}}_c^{ing}$ and $\hat{\mathbf{y}}_c^{ing}$. Since the optimization is nonconvex, we improve the convergence by using an early stopping mechanism instead of a threshold (see Section 3.4). Our gradient-based approach is unsupervised and can update the full recipe latent representation, reflecting how adding or removing an ingredient can necessitate adjustments to other ingredients and cooking steps as well.

3 EXPERIMENTS

3.1 Experimental Settings

3.1.1 Dataset. We assess our model on the Recipe1M dataset of 1M recipes (title, ingredients list, and cooking instructions) crawled from cooking websites [21]. We filter out recipes with more than 20 ingredients or steps, creating training, validation, and test splits with 635K, 136K, and 136K recipes, respectively. The average recipe comprises 9 ingredients and 166 words. We follow [20] and build a set of 1,488 ingredients. For critiquing, we select 20 ingredients to be critiqued among the most and the least popular ingredients across the training split. For each critique, we randomly sample 50 recipes that contain the critiqued ingredient and 50 that do not contain it. In total, the dev and test sets each contain 2,000 samples.

3.1.2 Baselines. We first compare our proposed RecipeCrit model to RecipeGPT [10]. It consists of the GPT-2-based [19] pre-trained language model fine-tuned in a multi-task manner on Recipe1M.

Table 1: Reconstruction performance. We report the IoU and F1 ingredient scores, and the Precision, Recall, and F1 scores of ingredients in predicted instructions w.r.t. predicted ones.

	Ingr. Scores		*Predicted Instr.*		
Model	**IoU**	**F1**	**Prec.**	**Rec.**	**F1**
RecipeGPT	73.47	84.67	61.17	72.55	66.38
BART	76.70	86.44	61.47	64.72	63.05
RecipeCrit (Ours)	**78.63**	**88.23**	**68.16**	**73.03**	**70.51**

Given the recipe title, it predicts the ingredients. Then, it generates the cooking steps conditioned on the title and the ingredients. Similar to RecipeCrit, we fine-tune BART-based [11] language models using the denoising objective. All models use greedy decoding.

3.1.3 Metrics. To evaluate edited recipes, we use metrics that reflect users' preferences. First, a user wants a recipe similar to the base recipe—we measure *ingredient fidelity* via IoU (Jaccard distance) and F1 scores between the edited ingredients list and the base recipe ingredients list. Next, the recipe must satisfy the user's specific ingredient feedback (i.e., adding/removing an ingredient). We thus report the *success rate*: the percentage of edited recipes that include/exclude the target ingredient in the instructions, respectively. Finally, the recipe must be *coherent*: able to be followed and internally consistent. As an ingredient constraint can be satisfied in many ways, we avoid n-gram metrics like BLEU [17] and ROUGE [13] that require references [3, 12, 15, 20]. We instead follow [9] and measure coherence via precision, recall, and F1-score of ingredients mentioned in the generated steps compared to the predicted ingredients. This verifies that a model conditions on the edited ingredients and that the recipe itself relies on the listed ingredients.

3.1.4 Training Details. To seek fair comparisons, we keep a similar number of parameters across all models. We set the embedding and attention dimension to 512. RecipeCrit contains an encoder and decoder each with four transformer layers with 4 attention heads and 512 hidden dimensions. We use a batch size of 32, dropout of 0.2, and Adam with learning rate 0.0001. For the baselines, we use RecipeGPT-based and BART-based models. For all models, we randomly mask 50% of the ingredients and instructions during training, and tune them on the validation set using random search.

3.2 RQ1: Ingredient & Recipe Reconstruction

As baseline recipe generation models are unable to perform editing, we train all models using our de-noising recipe completion task. To evaluate their generalization performance, we ask the models to reconstruct recipes from the unseen test set, with results shown in Table 1. We measure how well each model can infer the missing ingredients given the partial recipe context (IoU and F1 ingredient scores), as well as how coherent the reconstructed recipes are—the precision, recall, and F1 score of ingredients mentioned in the generated instructions compared to the predicted ingredients.

RecipeCrit outperforms baselines in both measures. In particular, we find a significant improvement in ingredient mention precision, indicating that RecipeCrit better constrains its generated recipe directions based on the predicted ingredients list. Meanwhile,

Table 2: Critiquing performance. We report the success rate of recipes satisfying the constraint (i.e., add or remove an ingredient), the IoU and F1 ingredient scores, as well as the Precision, Recall, and F1 of ingredients in cooking instructions.

			Ingr. Scores		*Predicted Instr.*		
	Model	**% Succ.**	**IoU**	**F1**	**Prec.**	**Rec.**	**F1**
Add	RecipeGPT	33.20	65.44	78.74	56.68	68.99	62.23
	BART	41.10	70.46	82.78	61.45	61.12	61.28
	RecipeCrit	**66.25**	**74.51**	**85.44**	**73.68**	**74.42**	**74.05**
Remove	RecipeGPT	91.05	37.22	52.93	38.35	54.57	45.04
	BART	95.35	55.65	73.27	57.61	61.55	59.52
	RecipeCrit	**95.75**	**68.79**	**80.72**	**73.96**	**74.51**	**74.23**

Table 3: Human evaluation of edited recipes in terms of best-worst scaling for Serendipity, Correctness, Coherence, and Relevance. $*$ denotes a significant difference compared to RecipeCrit (post hoc Tukey HSD test [24] with $p < 0.05$).

Model	**Ser.**	**Cor.**	**Coh.**	**Rel.**
RecipeGPT	-0.09^*	-0.09^*	-0.04^*	-0.12^*
BART	0.00	0.02	-0.03^*	0.01^*
RecipeCrit (Ours)	**0.06**	**0.03**	**0.06**	**0.08**

RecipeGPT and BART both tend to mention new ingredients in the recipe text even if they are not included in the ingredients list. As we see in Section 3.3, this is problematic because such models can include ingredients in the recipe instructions even if a user has specified a dislike or allergy.

Such text-to-text models capture the distribution of *language* well, producing fluent-sounding text. However, their lower scores for ingredient completion and recipe text coherence suggest that RecipeGPT and BART cannot distinguish how recipes are procedural texts with internal consistency, compared to generic text documents.

3.3 RQ2: Recipe Editing via Critiquing

Next, we evaluate whether RecipeCrit can edit recipes given positive and negative feedback about specific ingredients. In this setting, we select an ingredient that must be added or removed from the recipe to create a new version. The models then predict a new list of ingredients and recipe directions using those predicted ingredients. For ingredient substitution, we mask the critiqued ingredient and all steps mentioning that it, providing the resulting text as input to the models. For ingredient addition, we provide the full base recipe ingredients and instructions as input. To satisfy user feedback using the baseline models, we filter the predicted ingredients lists to exclude/include the target ingredient depending on the type of feedback. For our model, we use our iterative critiquing framework from Section 2.2 to accommodate user feedback.

We show results for constraint satisfaction (success rate), ingredient fidelity (ingredient scores), and recipe coherence (predicted instructions) in Table 2. RecipeCrit out-performs both RecipeGPT and BART across all metrics for ingredient addition and removal. While both baselines take advantage of pre-trained language models, they

Table 4: Comparison of a cherry tomato confit recipe with its edited versions to include "kale" as an additional ingredient. RecipeCrit preserves the concept of tomato confit and adds kale, but BART disregards the base recipe to make a cocktail.

Cherry tomato confit (base recipe)	BART	RecipeCrit (Ours)
clove, oil, pepper, rosemary, salt, tomato	**kale**, cachaca, cream, ice, juice, liqueur, pineapple, rum, strawberries, sugar, water ~~clove~~, ~~oil~~, ~~pepper~~, ~~rosemary~~, ~~salt~~, ~~tomato~~	clove, **kale**, oil, pepper, rosemary, salt, tomato
1) preheat oven to 325 degrees 2) spread tomatoes and garlic out on a baking sheet 3) drizzle with oil, and sprinkle with rosemary, crushed red pepper, a large pinch of salt and several grinds of pepper. 4) bake until tomatoes are wrinkled and fragrant, about 45 minutes, shaking pan once or twice. 5) transfer tomato pan to a rack to cool. 6) discard garlic. 7) use confit in salads, or serve over dishes like pasta, fresh ricotta cheese or polenta.	1) place ice cubes in a cocktail shaker. 2) add pineapple juice, coconut liqueurs, cachacca, cream and rum ; shake well add crushed ice to a collins glass. 3) add **kale** and strawberries to shaker ; shake. 4) strain drink into glass over crushed ice. 5) garnish with a strawberry and a slice of pineapple. 6) place pineapple garnish on rim of glass, and serve. 7) note : if you don't have coconut lillet, you can use a combination	1) heat oven to 350 degrees. 2) place tomatoes in a large bowl. 3) drizzle with olive oil and sprinkle with rosemary, salt and pepper. 4) toss to coat. 5) spread in a single layer on a baking sheet. 6) roast for 40 minutes. 7) remove from oven and let cool for 10 minutes. 8) place **kale** in a large bowl. 9) add tomatoes and garlic. 10) toss to combine.

cannot successfully incorporate user feedback during editing. We find in preliminary experiments that constrained decoding methods [5] lead to degenerate output with significant repetition and nonsensical phrases. Meanwhile, forcing ingredient lists to omit or contain specific ingredients has little impact on the generated recipe instructions—even when the desired ingredient is manually inserted into the ingredients list, RecipeGPT and BART mention using the ingredient only in 33% and 41% of generated instructions.

Our model architecture and gradient-based critiquing method (see Section 2.2) leads to a stronger influence of the edited ingredients on recipe instructions. By directly modifying the recipe latent representation that is then attended over during step generation, RecipeCrit achieves 30-50% relative improvement in success rate for adding ingredients and 20-65% relative improvements in coherence (F1 score between predicted ingredients and ingredients mentioned in the instructions) for both addition and removal. Meanwhile, baselines tend to ignore many ingredients in the ingredient list when generating new recipe directions.

Human Evaluation

We have established that RecipeCrit produces recipes that better satisfy user constraints (as expressed via critiques), more closely resemble the user's original preferences (base recipe), and make better use of the predicted ingredients (ingredient coherence). We next turn to qualitative human evaluation of edited recipes. We perform a human evaluation using Amazon Mechanical Turk to judge the quality of the generated recipes on four dimensions: Serendipity—how pleasantly surprised a user is by the edits; Correctness—whether the recipe respects the user's feedback; Coherence—how easy it is to follow the recipe; and Relevance—whether the recipe resembles the original (base) recipe found by the user. We uniformly sampled 800 edited recipes (400 for adding and 400 for removing) across the selected ingredients to critique and showed them in random order. The annotators judged the edited recipes using best-worst scaling [14] . We report a score ranging from -1 to +1. Table 3 shows that our edited recipes are largely preferred on all criteria. These results highlight that critiquing improves the coherence of generated recipes and their resemblance to the original versions.

Case Study We present a representative sample of recipes generated by our best-performing baseline (BART) and RecipeCrit in Table 4. Here we see a base recipe for "cherry tomato confit" to which the user wanted to add "kale". We next present the edited recipes according to BART and RecipeCrit. The user's original preference is for a tomato confit. While both edited recipes technically satisfy the requirement of having kale in the recipe, RecipeCrit stays more faithful to the user's preference while incorporating the new feedback: it makes a slightly different tomato confit but uses kale as the "fresh" or salad part of the dish. However, BART generates a cocktail recipe that ignores the base recipe: it's a drink rather than food, sweet rather than savory, and ignores tomatoes altogether. This aligns with the results of the human evaluation.

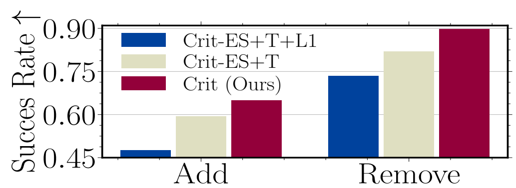

Figure 1: Critiquing algorithm comparison.

3.4 RQ3: Variants of Critiquing Algorithms

Since optimization for our critiquing method (Alg. 1) is nonconvex, there is a risk that the edited latent representation $\mathbf{z}^*$ stays in a local minimum. Thus, we experiment with different stopping criteria for our iterative gradient-based critiquing strategy. We denote our model *Crit*, which uses early stopping. One variant uses instead a threshold τ (Crit-ES+T) as a stopping criterion: $|C(\mathbf{z}_t^*)_c - \tilde{\mathbf{y}}_c^{ing}| < \tau$. The second variant extends the latter by using L1 norm on the whole vectors (Crit-ES+T+L1). We repeat the experiments in Section 3.3 for each stopping criteria, with success rates shown in Fig. 1. We note that an L1-based stopping criterion is suboptimal because of the high dimensionality of the ingredients. Using the absolute difference between $C(\mathbf{z}_t^*)_c$ and $\tilde{\mathbf{y}}_c^{ing}$ considerably improves the success rate (+25% relative gain for add and +12% for remove). Finally, we note that using early stopping further increases the success rate (+10%) for both adding and removing an ingredient.

4 CONCLUSION

In this paper, we presented RecipeCrit, a denoising-based model to edit cooking recipes. We first trained the model for recipe completion to learn semantic relationships between the ingredients and the instructions. The novelty of this work relies on the user's ability to provide ingredient-focused feedback. We designed an unsupervised method that substitutes the ingredients and re-writes the recipe text accordingly. Experiments show that RecipeCrit can more effectively edit recipes compared to strong baselines, creating recipes that satisfy user constraints and are more serendipitous, correct, coherent, and relevant as measured by human judges.

REFERENCES

[1] Diego Antognini and Boi Faltings. 2021. Fast Multi-Step Critiquing for VAE-Based Recommender Systems. In *Fifteenth ACM Conference on Recommender Systems* (Amsterdam, Netherlands) *(RecSys '21)*. Association for Computing Machinery, New York, NY, USA, 209–219. https://doi.org/10.1145/3460231.3474249

[2] Diego Antognini, Claudiu Musat, and Boi Faltings. 2021. Interacting with Explanations through Critiquing. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Zhi-Hua Zhou (Ed.). International Joint Conferences on Artificial Intelligence Organization, 515–521. https://doi.org/10.24963/ijcai.2021/72 Main Track.

[3] Ashutosh Baheti, Alan Ritter, Jiwei Li, and Bill Dolan. 2018. Generating More Interesting Responses in Neural Conversation Models with Distributional Constraints. In *EMNLP*. https://doi.org/10.18653/v1/d18-1431

[4] Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018. Simulating Action Dynamics with Neural Process Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJYFzMZC-

[5] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=H1edEyBKDS

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[7] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Trans. Assoc. Comput. Linguistics* 8 (2020), 64–77. https://transacl.org/ojs/index.php/tacl/article/view/1853

[8] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *CoRR* abs/1909.05858 (2019). arXiv:1909.05858 http://arxiv.org/abs/1909.05858

[9] Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally Coherent Text Generation with Neural Checklist Models. In *EMNLP*. https://doi.org/10.18653/v1/d16-1032

[10] Helena H. Lee, Shu Ke, 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 *WWW*. https://doi.org/10.1145/3366424.3383536

[11] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. 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, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[12] Shuyang Li, Yufei Li, Jianmo Ni, and Julian J. McAuley. 2021. SHARE: a System for Hierarchical Assistive Recipe Editing. *CoRR* abs/2105.08185 (2021). arXiv:2105.08185 https://arxiv.org/abs/2105.08185

[13] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[14] Jordan J. Louviere, Terry N. Flynn, and A. A. J. Marley. 2015. *Best-Worst Scaling: Theory, Methods and Applications*. Cambridge University Press.

[15] Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian J. McAuley. 2019. Generating Personalized Recipes from Historical User Preferences. In *EMNLP*. https://doi.org/10.18653/v1/D19-1613

[16] Yuran Pan, Qiangwen Xu, and Yanjun Li. 2020. Food Recipe Alternation and Generation with Natural Language Processing Techniques. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*. 94–97. https://doi.org/10.1109/ICDEW49219.2020.000-1

[17] 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*. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[18] Diana Andreea Petrescu, Diego Antognini, and Boi Faltings. 2021. Multi-Step Critiquing User Interface for Recommender Systems. In *Fifteenth ACM Conference on Recommender Systems* (Amsterdam, Netherlands) *(RecSys '21)*. Association for Computing Machinery, New York, NY, USA, 760–763. https://doi.org/10.1145/3460231.3478886

[19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog* (2019).

[20] Amaia Salvador, Michal Drozdzal, Xavier Giró-i-Nieto, and Adriana Romero. 2019. Inverse Cooking: Recipe Generation From Food Images. In *CVPR*. https://doi.org/10.1109/CVPR.2019.01070

[21] 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*.

[22] Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *EMNLP*. Association for Computational Linguistics, 4222–4235. https://doi.org/10.18653/v1/2020.emnlp-main.346

[23] ChunYuen Teng, Yu-Ru Lin, and Lada A. Adamic. 2012. Recipe recommendation using ingredient networks. In *WebSci*. https://doi.org/10.1145/2380718.2380757

[24] John W Tukey. 1949. Comparing individual means in the analysis of variance. *Biometrics* (1949), 99–114.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[26] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008 (ACM International Conference Proceeding Series, Vol. 307)*, William W. Cohen, Andrew McCallum, and Sam T. Roweis (Eds.). ACM, 1096–1103. https://doi.org/10.1145/1390156.1390294

[27] Ke Wang, Hang Hua, and Xiaojun Wan. 2019. Controllable Unsupervised Text Attribute Transfer via Editing Entangled Latent Representation. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/8804f94e16ba5b680e239a554a08f7d2-Paper.pdf

[28] Ronald J. Williams and David Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Comput.* 1, 2 (June 1989), 270–280. https://doi.org/10.1162/neco.1989.1.2.270

[29] Dongling Xiao, Yu-Kun Li, Han Zhang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2021. ERNIE-Gram: Pre-Training with Explicitly N-Gram Masked Language Modeling for Natural Language Understanding. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 1702–1715. https://doi.org/10.18653/v1/2021.naacl-main.136

[30] Ryosuke Yamanishi, Naoki Shino, Yoko Nishihara, Junichi Fukumoto, and Aya Kaizaki. 2015. Alternative-ingredient Recommendation Based on Co-occurrence Relation on Recipe Database. In *KES*. https://doi.org/10.1016/j.procs.2015.08.138