Assistive Recipe Editing through Critiquing

Diego Antognini∗
IBM Research
diego.antognini@ibm.com

Shuyang Li†
Meta AI
lishuyang@meta.com

Boi Faltings
EPFL
boi.faltings@epfl.ch

Julian McAuley
UCSD
jmcauley@ucsd.edu

Abstract
Home cooks often have specific requirements regarding individual ingredients in a recipe (e.g., allergies). Substituting ingredients in a recipe can necessitate complex changes to instructions (e.g., replacing chicken with tofu in a stir fry requires pressing the tofu, marinating it for less time, and par-cooking)—which has thus far hampered efforts to automatically create satisfactory versions of recipes. We address these challenges with the RecipeCrit model that allows users to edit existing recipes by proposing individual ingredients to add or remove. Crucially, we develop an unsupervised critiquing module that allows our model to iteratively re-write recipe instructions to accommodate the complex changes needed for ingredient substitutions. Experiments on the Recipe1M dataset show that our model can more effectively edit recipes compared to strong language-modeling baselines, creating recipes that satisfy user constraints and humans deem more correct, serendipitous, coherent, and relevant.

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 focused on substituting individual ingredients in the ingredients list (Yamanishi et al., 2015) or recommending new recipes based on similar ingredients (Teng et al., 2012). Individual ingredient substitution rules (e.g., tapioca flour and xanthan gum for wheat flour) often necessitate additional changes to the cooking procedure to function properly (Li et al., 2022). 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 (Petrescu et al., 2021).

Recipe editing can be seen as a combination of recipe generation and controllable natural language generation (Shin et al., 2020), and has been explored to adapt recipes for broad dietary constraints (Li et al., 2022) and cuisines (Pan et al., 2020). Pre-trained language models have been used to create recipe directions given a known title and set of ingredients (Kiddon et al., 2016; Bosse-lut et al., 2018), but generated recipes suffer from inconsistencies. Li et al. (2022) instead build a paired recipe dataset, but face challenges scaling due to the large set of possible recipes and dietary restrictions; people often express even more specific ingredient-level preferences (e.g., 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 (Keskar et al., 2019) or hyperparameter-sensitive training of individual models for each possible piece of feedback (Dathathri et al., 2020), our unsupervised critiquing framework enables recipe editing models to be trained with arbitrary un-paired data. This generalizes recipe editing, unlike existing methods for controllable generation that rely on paired data with specially constructed
Cherry tomato confit

Ingredients
- 1 pint red cherry tomatoes
- ¼ cup extra virgin olive oil
- ...

Instructions
- Preheat oven to 325 degrees.
- Spread tomatoes and garlic on a sheet.
- ...

![Figure 1: RecipeCrit includes a recipe encoder and an ingredient and instruction decoders using the base recipe and target ingredients to edit cooking instructions.](image)

prompts (Keskar et al., 2019) or hyperparameter-sensitive training of individual models for each possible feedback (Dathathri et al., 2020).

Experiments on the RecipeLM (Salvador et al., 2017) dataset show that RecipeCrit edits recipes in a way that better satisfies user constraints, preserves the original recipe, and produces coherent recipes (i.e., recipe instructions are better conditioned on the ingredients list) 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 (Li et al., 2022) and cuisines (Pan et al., 2020) and require paired corpora (which do not exist for fine-grained edits). We propose RecipeCrit: a hierarchical denoising recipe auto-encoder that does not require paired corpora to train and accommodates positive and negative user feedback about ingredients (Figure 1).

RecipeCrit is divided into three submodels: an Encoder $E(\cdot)$, which produces the latent representation $z$ from the (potentially noisy) recipe; an ingredient predictor $C(\cdot)$, which predicts the ingredients $\hat{y}^{\text{ing}}$, and a decoder $D(\cdot)$, which reconstructs the cooking instructions $\hat{y}^{\text{ins}}$ from $z$ conditioned on the $\hat{y}^{\text{ing}}$.

**Recipe Encoder $E(\cdot)$** We build a powerful latent representation that captures the different elements of a recipe via the mean-pooled output of the representation of each sentence using a Transformer encoder (Vaswani et al., 2017). We provide the title $x^{\text{ttl}}$, ingredients $X^{\text{ing}}$, and instructions $X^{\text{inst}}$ as raw text input. While the title $x^{\text{ttl}}$ comprises a single sentence, the ingredients $X^{\text{ing}}$ and instructions $X^{\text{inst}}$ are provided as lists of sentences; we use raw recipe texts directly as input, removing the need of a pre-processing step. We encode the ingredients and instructions in a hierarchical manner using another Transformer to create fixed-length representations. We compute the latent representation $z$ by concatenating the representations of each component and applying a projection followed by a tanh function: $z = \text{tanh}(W[HTRF(x^{\text{ttl}}) \| HTRF(X^{\text{ing}}) \| HTRF(X^{\text{inst}})])$, where $\|$ is the concatenation, and $W$, $b$ the projection parameters.

**Ingredient Predictor $C(\cdot)$** We treat ingredient prediction as multi-label binary classification over an ingredient vocabulary $I$, with a boolean target vector $y^{\text{ing}}$ representing ingredients in a ground truth recipe. We use a Set Transformer (Salvador et al., 2019) to decode ingredients, pooling ingredient logits over time-steps to compute binary cross-entropy loss against the target; we also employ an EOS token to predict the ingredient set cardinality:

$$\mathcal{L}^{\text{ing}}(\cdot) = - \sum_i |I| y_i^{\text{ing}} \log \hat{y}_i^{\text{ing}} - \lambda y_{\text{eos}}^{\text{ing}} \log \hat{y}_i^{\text{ing}}$$

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

**Instruction Decoder $D(\cdot)$** The last component generates cooking instructions using a Transformer decoder. We condition the decoder on $z$, previously generated outputs $\hat{y}^{\text{ins}}_{1:t-1}$, and ingredients $\hat{y}^{\text{ing}}$. Specifically, we encode the ingredients using an embedding layer $A(\cdot)$ and concatenate their representations with the recipe representation $z$. We train using teacher-forcing and cross-entropy:

$$\mathcal{L}^{\text{ins}}(z, A(\hat{y}^{\text{ing}})) = - \sum_t y_t^{\text{ins}} \log \hat{y}_t^{\text{ins}}.$$
mask random ingredients and instruction sentences in model input, and task our model to generate the full recipe. We train our model in two stages: first minimizing ingredient prediction loss $L_{\text{ing}}$; then freezing the encoder and optimizing for instruction decoding loss $L_{\text{ins}}$ using ground truth ingredients.

**Unsupervised Critiquing** We aim to refine a recipe based on the user’s feedback and the predicted ingredients $\hat{y}^{\text{ing}}$. We denote $\tilde{y}^{\text{ing}}$ the vector of desired ingredients. Simply incorporating user feedback by explicitly including/removing ingredients before generating instructions often cannot satisfy user preferences due to weak conditioning between predicted ingredients and generated instructions. In RecipeCrit we turn to a critiquing method that modifies the recipe representation $z$ before using the updated representation to jointly generate the edited ingredients and instructions. Specifically, users add a new ingredient $c$ by setting $\tilde{y}_c^{\text{ing}} = 1$ or remove some using $\tilde{y}_c^{\text{ing}} = 0$.

Inspired by success in editing the latent space in text style transfer and recommendation (Antognini et al., 2021a; Wang et al., 2019), we first compute the gradient with respect to $z$:

$$g_{t-1} = \nabla_{z_{t-1}} L_{\text{ing}}(C(z_{t-1}), \tilde{y}^{\text{ing}}).$$

Then, we use the gradient to modulate $z$ such that the new predicted ingredients $\hat{y}^{\text{ing}}$ are close to the desired ingredients $\tilde{y}^{\text{ing}}$:

$$z_t = z_{t-1} - \alpha_{t-1} g_{t-1} / ||g_{t-1}||_2.$$

Prior work stopped updating $z$ when $||\tilde{y}^{\text{ing}} - \hat{y}^{\text{ing}}||_1 < \epsilon$ for some threshold $\epsilon$. We instead propose to compute the absolute difference $|\tilde{y}_c^{\text{ing}} - \hat{y}_c^{\text{ing}}|$. Since the optimization is nonconvex, we improve convergence by using an early stopping mechanism. Our 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. Pseudo-code is available in the App.

Another advantage of our approach is the possibility to update multiple ingredients simultaneously: adding or removing an ingredient might affect other ones as well and thus, a local-based stopping criteria allows such a change.

3 Experiments

**Dataset** We assess our model on the Recipe1M (Salvador et al., 2017) dataset of 1M recipe texts. Each recipe contains a title, a list of ingredients, and a list of cooking instructions. We filter out recipes with more than 20 ingredients or steps, creating train, val, and test splits with 635K, 136K, and 136K recipes, respectively. The average recipe comprises 9 ingredients and 166 words. We follow Salvador et al. (2019) 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 train set. For each critique, we randomly sample 50 recipes that contain the critiqued ingredient and 50 that do not.

**Baselines** We compare our proposed RecipeCrit architecture against large language models trained using our denoising objective. We fine-tune BART (Lewis et al., 2020), an encoder-decoder language model trained to denoise documents, as well as RecipeGPT (Lee et al., 2020), a decoder-only language model pre-trained on Recipe1M to predict ingredients and cooking steps. To demonstrate the necessity of our denoising approach, we also compare against PPLM (Dathathri et al., 2020), a recent method for controllable generation from language models that leverages sets of desired and undesired sequences (for ingredient addition and substitution, respectively). All models use greedy decoding.

**Metrics** We evaluate edited recipes via metrics that reflect user 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 and base recipe ingredients list. Next, the recipe must satisfy the user’s specific ingredient feedback—we report the percentage of edited recipes that properly include/exclude the target ingredient (Success Rate). Finally, the recipe must be coherent: able to be followed and internally consistent. As an ingredient constraint can be satisfied in many ways, we follow Kiddon et al. (2016) and measure coherence via precision, recall, and F1-score of ingredients mentioned in the generated steps compared to the predicted ingredients. This verifies that the recipe itself relies on the listed ingredients.

**Training Details** For fair comparison, we compare similar-sized models. RecipeCrit uses an encoder and decoder with 4 Transformer layers, 4 attention heads, and hidden size of 512. We randomly mask 50% of the ingredients and instructions during training, and tune them on the validation set using random search. We give more details in App. B.
Table 1: Critiquing performance: success rate of adding/removing an ingredient, IoU and F1 ingredient scores, and the Precision, Recall, and F1 of ingredients in cooking instructions.

| Model       | % Succ. | IoU  | F1  | Prec. | Rec. | F1Add |
|-------------|---------|------|-----|-------|------|-------|
| Add         |         |      |     |       |      |       |
| RecipeGPT   | 33.2    | 65.4 | 78.7| 56.7  | 69.0 | 62.2  |
| PPLM        | 34.4    | 60.9 | 72.7| 53.0  | 63.0 | 57.6  |
| BART        | 41.1    | 70.5 | 82.8| 61.5  | 61.1 | 61.3  |
| RecipeCrit  | 66.3    | 74.5 | 85.4| 73.7  | 74.4 | 74.1  |
| Remove      |         |      |     |       |      |       |
| RecipeGPT   | 91.1    | 37.2 | 52.9| 38.4  | 54.6 | 45.0  |
| PPLM        | 92.3    | 61.3 | 32.6| 47.2  | 53.5 | 50.2  |
| BART        | 95.4    | 55.7 | 73.3| 57.6  | 61.6 | 59.5  |
| RecipeCrit  | 95.8    | 68.8 | 80.7| 74.0  | 74.5 | 74.2  |

RQ1: Recipe Editing via Critiquing. We evaluate whether our models can edit recipes by creating new ingredient sets and corresponding recipe instructions when faced with positive and negative feedback: an ingredient that must be added or removed (substituted) from the recipe to create a new version. For ingredient substitution, we mask the critiqued ingredient and all steps that reference it as denoising inputs; for addition, we use the full base recipe. For RecipeGPT and BART, we filter the predicted ingredients lists to exclude/include the target ingredient. For PPLM, we provide the target ingredient as a bag of words to steer generation, using RecipeGPT as the base generative model. RecipeCrit uses our iterative critiquing framework (Section 2) to accommodate user feedback.

We show results for constraint satisfaction (success rate), ingredient fidelity, and recipe coherence (predicted instructions) in Table 1. RecipeCrit outperforms baselines across all metrics for ingredient addition and removal. While our baselines take advantage of pre-trained language models, they cannot successfully incorporate user feedback during editing. PPLM-guided constrained decoding is not only two orders of magnitude slower than our denoising models (3min vs. 1s per recipe), but we observe poor fidelity and frequent incoherent instructions (e.g., repetition). 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 ingredient list, RecipeGPT and BART mention using the ingredient only in 33% and 41% of generated instructions.

Our model and gradient-based critiquing method 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 those 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 creates edits 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 perform a qualitative human evaluation of our edited recipes via Mechanical Turk, asking the user: how pleasantly surprised they were (Serendipity); whether the recipe respected their feedback (Correctness); how easy the recipe was to follow (Coherence); and whether the recipe resembled the original recipe (Relevance). We uniformly sampled 800 edited recipes (400 for adding and 400 for removing) across the ingredients to critique and showed them in random order. The annotators judged the edited recipes using best-worst scaling (Louviere et al., 2015) with scores normalized to $[-1, +1]$. Table 2 shows that our edited recipes are largely preferred on all criteria. Our results highlight that critiquing improves the coherence of generated recipes and their resemblance to the original ones.

Case Study. Table 3 shows a sample of our best-performing baseline (BART) and RecipeCrit editing the “cherry tomato confit” recipe to include “kale”. While both edited recipes include kale, RecipeCrit stays faithful to the user’s preference for “tomato confit” 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 instead that ignores the base recipe: it’s a drink rather than food, sweet rather than savory, and ig-
Cherry tomato confit (base recipe)
clove, oil, pepper, rosemary, salt, tomato

1) preheat oven to 325 degrees
2) spread tomatoes and garlic on a 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.
5) transfer tomato pan to a rack to cool.
6) discard garlic.

1) place ice cubes in a cocktail shaker.
2) add pineapple juice, coconut liqueurs, cachaca, cream and rum; shake well and strain drink into glass over crushed ice.
3) garnish with strawberry and pineapple.

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; coat.
4) spread in a single layer on a sheet.
5) roast for 40 minutes.
6) remove and let cool for 10 minutes.
7) toss kale with tomatoes and garlic.

Table 3: Comparison of a cherry tomato confit recipe with its edited versions to include “kale” as an additional ingredient. RecipeCrit proposes tomato confit with kale, but BART disregards the base recipe to make a cocktail.
References

Diego Antognini and Boi Faltings. 2021. Rationalization through concepts. In Findings of the Association for Computational Linguistics: ACL 2021, pages 761–775, Online. Association for Computational Linguistics.

Diego Antognini, Claudiu Musat, and Boi Faltings. 2021a. Interacting with explanations through critiquing. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pages 515–521. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Diego Antognini, Claudiu Musat, and Boi Faltings. 2021b. Multi-dimensional explanation of target variables from documents. In Proceedings of the AAAI Conference on Artificial Intelligence, (AAAI 2021), volume 35, pages 12507–12515.

Jasmijn Bastings, Wilker Aziz, and Ivan Titov. 2019. Interpretable neural predictions with differentiable binary variables. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 2963–2977, Florence, Italy. Association for Computational Linguistics.

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.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In NAACL-HLT, pages 4171–4186. Association for Computational Linguistics.

Mandar Joshi, Danqi Chen, Yinhua Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. Trans. Assoc. Comput. Linguistics, 8:64–77.

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.

Chloe Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In EMNLP.

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.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Rationalizing neural predictions. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 107–117, Austin, Texas. Association for Computational Linguistics.

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, pages 7871–7880. Association for Computational Linguistics.

Shuyang Li, Yufei Li, Jianmo Ni, and Julian J. McAuley. 2022. SHARE: a system for hierarchical assistive recipe editing. In EMNLP, pages 11077–11090. Association for Computational Linguistics.

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

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), pages 94–97.

Diana Andreea Petrescu, Diego Antognini, and Boi Faltings. 2021. Multi-step critiquing user interface for recommender systems. In Fifteenth ACM Conference on Recommender Systems, RecSys ’21, page 760–763, New York, NY, USA. Association for Computing Machinery.

Amaia Salvador, Michal Drozdzal, Xavier Giró-i-Nieto, and Adriana Romero. 2019. Inverse cooking: Recipe generation from food images. In CVPR.

Amaia Salvador, Nicholas Hynes, Yusuf Aytar, Javier Marin, Ferda Oflı, Ingrid 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.

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, pages 4222–4235. Association for Computational Linguistics.

ChunYuen Teng, Yu-Ru Lin, and Lada A. Adamic. 2012. Recipe recommendation using ingredient networks. In WebSci.
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc.

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, volume 32. Curran Associates, Inc.

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.

Mo Yu, Yang Zhang, Shiyu Chang, and Tommi Jaakkola. 2021. Understanding interlocking dynamics of cooperative rationalization. In Advances in Neural Information Processing Systems, volume 34. Curran Associates, Inc.
Table 4: 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.

| Model       | Ingr. Fidelity | Predicted Instr. |
|-------------|----------------|------------------|
|             | IoU | F1  | Prec. | Rec. | F1  |
| RecipeGPT   | 73.5 | 84.7 | 61.2  | 72.6 | 66.4 |
| BART        | 76.7 | 86.4 | 61.5  | 64.7 | 63.1 |
| RecipeCrit  | 78.6 | 88.2 | 68.2  | 73.0 | 70.5 |

A Ingredient & Recipe Reconstruction

As baseline recipe generation models are unable to perform editing, we train all models using our denoising 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 4. 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 list.

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, 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, this is problematic because such models can include ingredients in the recipe steps even if users have specified dislikes or allergies.

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.

B Additional Training Details

We use a batch size of 32, dropout of 0.2, and Adam with learning rate 0.0001. For the baselines RecipeGPT and PPLM, we reuse the official code from the authors. For BART, we employ the HuggingFace library.

B.1 Hardware / Software

- **CPU**: 2x Intel Xeon E5-2680 v3, 2x 12 cores, 24 threads, 2.5 GHz, 30 MB cache;
- **RAM**: 16x16GB DDR4-2133;
- **GPU**: 1x Nvidia Titan X Maxwell;
- **OS**: Ubuntu 18.04;
- **Software**: Python 3.6, PyTorch 1.6.1, CUDA 10.2.
**Algorithm 1** Iterative Critiquing Gradient Update (Crit).

1: function CRITIQUE(latent vector $z$, critiqued ingredient $c$, trained ingredients predictor $C$, decay coefficient $\zeta$, patience $P$, a maximum number of iterations $T$, desired ingredients $\hat{y}^{\text{ing}}$)

2: Set $z_0 = z^* = z$, $\alpha_0 = 1$, best_val = $\infty$, patience = 0, $t = 1$

3: while patience < $P$ and $t < T$ do

4: $g_{t-1} = \nabla_{z_{t-1}} L_{\text{ing}}(C(z_{t-1}), \hat{y}^{\text{ing}})$

5: $z_t = z_{t-1} - \alpha_{t-1} \frac{g_{t-1}}{||g_{t-1}||^2}$ and $\hat{y}^{\text{ing}} = C(z_t)$

6: if $|\hat{y}^{\text{ing}}_c - \hat{y}^{\text{ing}}_c| < \text{best_val}$ then

7: best_val = $\hat{y}^{\text{ing}}_c$, $z^* = z_t$, and patience = 0

8: else

9: patience = patience + 1

10: $\alpha_t = \zeta \alpha_{t-1}$ and $t = t + 1$

11: return $z^*$;

---

Table 5: Comparison of a cherry tomato confit recipe with its edited versions to include “kale” as an additional ingredient. None of those baselines add the critique ingredient. RecipeGPT disregards the base recipe. PPLM generates a recipe in the same style than the base one, but three times longer and with a lack of coherence.

| Cherry tomato confit (base recipe) | RecipeGPT |
|-----------------------------------|-----------|
| clove, oil, pepper, rosemary, salt, tomato | cream, cheese, margarine, milk, onion, pepper, salmon, salt, tea, clove, rosemary, tomato |

1) saute green peppers and onions in margarine.
2) add cream cheese and stir until melted.
3) add velveeta, milk and corn.
4) heat until melted, stirring occasionally.
5) add salt.
6) garnish with thin strips of red pepper and cilantro.
7) serve with crackers.

PPLM

| oil, pepper, tomato, clove, rosemary, tomato | oil, pepper, tomato, clove, rosemary, tomato |
|---------------------------------------------|---------------------------------------------|
| 1) preheat the oven to 400 degrees f. | 1) preheat the oven to 400 degrees f. |
| 2) place a large roasting pan in the center of the oven. | 2) place a large roasting pan in the center of the oven. |
| 3) heat the olive oil in a large skillet over medium heat until hot. | 3) heat the olive oil in a large skillet over medium heat until hot. |
| 4) add the tomatoes and cook for about 3 minutes or until the skins begin to split. | 4) add the tomatoes and cook for about 3 minutes or until the skins begin to split. |
| 5) place in the oven for about 30 to 40 minutes until the skins begin to blister and shrivel. | 5) place in the oven for about 30 to 40 minutes until the skins begin to blister and shrivel. |
| 6) place the tomatoes in cold water to cool, then remove the skins. | 6) place the tomatoes in cold water to cool, then remove the skins. |
| 7) in a bowl, combine the tomatoes and their juices, salt, and pepper, tossing well to coat evenly. | 7) in a bowl, combine the tomatoes and their juices, salt, and pepper, tossing well to coat evenly. |
| 8) remove the tomatoes, place them on a sheet tray, and bake for about 45 minutes, until the skins begin to blister and shrivel. | 8) remove the tomatoes, place them on a sheet tray, and bake for about 45 minutes, until the skins begin to blister and shrivel. |
| 9) the tomatoes should still be juicy, and the skins should be very wrinkly. remove the tomatoes from the oven and let cool. | 9) the tomatoes should still be juicy, and the skins should be very wrinkly. remove the tomatoes from the oven and let cool. |
| 10) remove the skins from the tomatoes and discard the skins. transfer the tomatoes to a clean food processor. | 10) remove the skins from the tomatoes and discard the skins. transfer the tomatoes to a clean food processor. |
| 11) blend the tomatoes with the roasted garlic, and add the peppercorns in the last 3 minutes of the cooking time, just until you can no longer process the tomatoes into a paste. place a sheet of parchment paper | 11) blend the tomatoes with the roasted garlic, and add the peppercorns in the last 3 minutes of the cooking time, just until you can no longer process the tomatoes into a paste. place a sheet of parchment paper |
| 12) on a clean work surface. | 12) on a clean work surface. |
| 13) place a sheet of the phyllo on the work surface. | 13) place a sheet of the phyllo on the work surface. |
| 14) (keep the rest of the oil covered with damp paper towel. | 14) (keep the rest of the oil covered with damp paper towel. |
| 15) brush the sheet of phyllo with oil. repeat with two more sheets of phyllo to form the second sheet. | 15) brush the sheet of phyllo with oil. repeat with two more sheets of phyllo to form the second sheet. |
| 16) you should have four layers. | 16) you should have four layers. |
| 17) if you have extra sheets left, place on a baking sheet and toast in oven with the oven door propped open until the sheets become golden brown. | 17) if you have extra sheets left, place on a baking sheet and toast in oven with the oven door propped open until the sheets become golden brown. |
| 18) remove the phyllo sheets from the oven and place on a clean work surface. cut each layered sheet along the scored lines into quarters. | 18) remove the phyllo sheets from the oven and place on a clean work surface. cut each layered sheet along the scored lines into quarters. |
| 19) place 1 heaping tablespoon of the tomato-peppercorn mixture in the center of each of four phyllo sheets. | 19) place 1 heaping tablespoon of the tomato-peppercorn mixture in the center of each of four phyllo sheets. |
| 20) fold the phyllo over into a triangle and place on a parchment-lined baking sheet. bake at 400 degrees f for 7 to 8 minutes, until lightly browned. | 20) fold the phyllo over into a triangle and place on a parchment-lined baking sheet. bake at 400 degrees f for 7 to 8 minutes, until lightly browned. |
| 21) cool slightly before removing from the baking sheet, and serve warm. | 21) cool slightly before removing from the baking sheet, and serve warm. |
| Table 6: Comparison of a baked parmesan fish recipe with its edited versions to include “cream” as an additional ingredient. RecipeCrit incorporates the cream into a single-batter style fry, while BART only uses cream as a “serve with sour cream” addition to the original recipe. RecipeGPT disregards the base recipe to make a pizza. PPLM generates a completely different and incoherent recipe, and it suffers from text degeneration. |
| --- |

**Baked parmesan fish (base recipe)**
- **catfish**, cheese, egg, flour, milk, paprika, pepper, salt
- 1) preheat oven to 350 degrees.
- 2) rinse fillets and pat dry with paper towel.
- 3) in a pie plate, combine parmesan cheese, flour, paprika, salt and pepper.
- 4) in another pie plate, combine milk and beaten egg.
- 5) dip fillet in egg mixture and then coat with parmesan cheese mixture.
- 6) place in a well greased 9 x 13 pan.
- 7) bake uncovered for 25-30 minutes or until fish easily flakes with a fork.

**BART**

| cream, egg, fillets, flour, garlic, milk, paprika, parsley, pepper, salt, thyme |
| --- |
| 1) preheat oven to 350 degrees. |
| 2) rinse fillets and pat dry with paper towel. |
| 3) in a pie plate, combine parmesan cheese, flour, paprika, salt and in another pie plate or bowl, combine milk and beaten egg. |
| 4) dip fillet in egg mixture and then coat with parmesan cheese mixture. |
| 5) place in a well greased 9 x 13 pan. |
| 6) bake uncovered for 25-30 minutes or until fish easily flakes with a fork. |
| 7) remove from oven and let stand for 5 minutes. |
| 8) serve with sour cream and lemon wedges. |
| 9) enjoy! |

**RecipeCrit (Ours)**

| catfish, cheese, cream, egg, flour, paprika, pepper, salt, milk |
| --- |
| 1) preheat oven to 350 degrees f ( 175 degrees c ). |
| 2) grease a baking dish. |
| 3) rinse fillets and pat dry with paper towel. |
| 4) in a medium bowl, mix together the cream cheese, sour cream, flour, salt, paprika, and pepper. |
| 5) beat in the eggs. |
| 6) dip fillets |
| 7) pour the mixture into the baking dish. |
| 8) bake in the preheated oven until the fish flakes easily with a fork. |
| 9) enjoy! |

**RecipeGPT**

| crusts, jalapeno, mushrooms, oil, popcorn, sauce, tomato |
| --- |
| 1) combine barbecue sauce and tomato sauce in bowl; set aside. |
| 2) toss mushrooms with oil; bake in 425 degrees f standard oven 5 min. |
| 3) or until softened. |
| 4) place crusts on sheet pan. |
| 5) spread each crust with 1/2 cup sauce mixture. |
| 6) top evenly with mushrooms, jalapenos, shrimp and cheese. |
| 7) bake 7 min. |
| 8) or until cheese is melted and golden brown. |
| 9) slice each pizza into 8 pieces. |
| 10) serve with remaining barbecue sauce mixture. |
| 11) serve with pickled jalapeno slices. |

**PPLM**

| basil, bread, butter, cod, garlic, eggplant, oil, onions, parmesan, parmiggiano, parsley, pepper, salt, tomatoes |
| --- |
| 1) preheat oven to 400 degrees. |
| 2) wash the potatoes and put them in a saucepan with the cream. |
| 3) bring cream to a boil and add the parmiggiano-cream cheese and the cream and cream. |
| 4) let simmer, stirring often, to make a soft creamy cream and thick cream cheese cream. |
| 5) season with salt and pepper set aside. melt the cream cheese and cream together and cream together. |
| 6) beat in the buttercreamcreamcreamecreamecreatcream. |
| 7) creamcreamecreamecreamecreamcreamecreamecreamecreamecreamecreamcreame. |